Energy-Aware Scientific Workflow Scheduling

by

Mehul Vikas Warade

BSoftEng(Hons)

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Deakin University

August, 2025

CANDIDATE DECLARATION

I certify the following about the thesis entitled **Energy-Aware Scientific Workflow Scheduling** submitted for the degree of **Doctor of Philosophy**.

(a) I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment is given.

(b) The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

(c) That if the work(s) have been commissioned, sponsored or supported by any organisation, I have fulfilled all of the obligations required by such contract or agreement.

(d) That any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.

(e) All research integrity requirements have been complied with.

I certify that I am the student named below and that the information provided in the form is correct

Full Name: Mehul Vikas Warade

Signed: 

Date: August 15, 2025

Acknowledgments

This thesis would not have been possible without the support of many people, and I would like to thank all those who have helped me along the way, knowing that I will never be able to fully express my appreciation.

I want to express my deepest gratitude to my supervisors - Associate Professor Kevin Lee, Associate Professor Chathu Ranaweera, and Professor Jean-Guy Schneider - for their exceptional guidance, unwavering support, and insightful feedback throughout my doctoral journey. Their expertise, patience, and encouragement have been instrumental in shaping both this research and my development as a researcher. The countless hours of discussions, their constructive criticism, and their willingness to share their knowledge have profoundly influenced this work.

I am particularly grateful to Deakin University for supporting this research through the Deakin University Postgraduate Research Scholarship (DUPRS). Additionally, I extend my sincere appreciation to the university for providing the necessary computational infrastructure and resources, without which the evaluation of this work would not have been possible. The technical support staff deserves special mention for their assistance in maintaining and troubleshooting the systems used for my experiments.

I would also like to acknowledge my fellow research colleagues at the School of Information Technology for the stimulating discussions, collaborative atmosphere, and friendship that made this challenging journey more enjoyable. Our exchange of ideas and mutual support has contributed significantly to broadening my perspective on various research problems.

I am forever indebted to my loving parents, Mr. Vikas Warade and Dr. Nisha Warade, for their unconditional love, understanding, and encouragement throughout this demanding period. Their upbringing has been vital in my achieving everything in life. I would never be in this place without the support of my parents and loving grandparents, Mr. Laxman Warade and Mrs. Sunita Warade. Their belief in my abilities, even during the most challenging times, has been a constant source of strength. To my parents, who instilled in me the value of education and perseverance, your sacrifices and support have made this achievement possible.

Finally, I am deeply grateful to my friends who have been a vital source of motivation, laughter, and moral support throughout this journey. Your encouragement during difficult times, celebration of small victories, and occasional distractions provided the necessary balance in my life, reminding me of the world beyond research.

This thesis serves as a testament to the collective support of all the individuals and institutions that have contributed to my academic and personal growth over the years.

List of Publications

The author has published the following research papers as a result of the work conducted for this thesis. The published papers have undergone full peer review.

Journal:

1. Warade, M., Schneider, J.-G., & Lee, K. (2022). Measuring the energy and performance of scientific workflows on low-power clusters. *Electronics*, 11(11), 1801. https://doi.org/10.3390/electronics11111801 **(Published)**

2. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2024). Energy and Scientific Workflows: Smart Scheduling and Execution. *Journal of Information Science and Engineering*, 40, 957–977. https://doi.org/10.6688/JISE.202409_40(5).0002 **(Published)**

3. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. A Multi-User Energy-Aware Scheduler for Scientific Workflows. *IEEE Transactions on Emerging Topics in Computing* **(Submitted)**

Conference:

1. Warade, M., Schneider, J.-G., & Lee, K. (2022). Towards energy-aware scheduling of scientific workflows. *Proc. GECOST*, pp. 93–98. https://doi.org/10.1109/GECOST55694.2022.10010634 **(Published)**

2. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2022). Energy aware adaptive scheduling of workflows. *Proc. ISPA/BDCloud/SocialCom/SustainCom*, pp. 562–570. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00078 **(Published)**

3. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2023). Monitoring the energy consumption of Docker containers. *Proc. COMPSAC*, pp. 1703–1710. https://doi.org/10.1109/COMPSAC57700.2023.00263 **(Published)**

4. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2023). Optimizing workflow execution for energy consumption and performance. *Proc. GREENS*, pp. 24–29. https://doi.org/10.1109/GREENS59328.2023.00011 **(Published)**

5. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. Optimisation of scientific workflows for energy consumption and performance. *Proc. ASE* **(Submitted)**

Abstract

High-performance computing (HPC) systems have emerged as indispensable infrastructure for advancing scientific research across various domains, with scientific workflows increasingly used to describe complex computations as structured compositions of interconnected computational tasks. These workflows facilitate sophisticated scientific applications ranging from climate modeling and genomic analysis to computational physics and materials science, leveraging parallel execution capabilities where independent tasks can be processed simultaneously across multiple computing resources. However, traditional HPC system design and resource management strategies have primarily focused on maximizing computational performance and minimizing execution time, often overlooking energy efficiency considerations. As scientific computations become increasingly complex and data-intensive, energy consumption has escalated to unprecedented levels, making energy costs a significant portion of total ownership expenses. The intricate nature of energy consumption in HPC environments involves numerous interdependent factors, including processor utilization, memory access patterns, network communication overhead, and cooling requirements, making theoretical energy estimation highly challenging and often inaccurate.

The work presented in this thesis proposes an approach to energy-aware scientific workflow scheduling through the development of a generic energy monitoring framework that provides standardized methodologies for measuring and analyzing energy consumption during computational tasks. The research demonstrates that significant energy savings can be achieved through novel intelligent scheduling algorithms and strategic hardware allocation techniques. These techniques are further enhanced by adaptive scheduling mechanisms that dynamically respond to changing system conditions and workload characteristics. The work further explores optimization-based approaches to automate the scheduling process, moving beyond traditional rule-based methods to achieve optimal task schedules that balance performance and energy efficiency objectives. The research aims to abstract the complexity of energy-aware scheduling from end users, allowing scientists to specify their energy-performance preferences. Based on the preferences, the system will then automatically handle the underlying resource optimization and task allocation decisions.

The validity and effectiveness of the proposed approach are established through experimental evaluation conducted across multiple scientific workflow applications and diverse cluster configurations. These experiments systematically validate each component of the framework, demonstrating measurable improvements in energy efficiency while maintaining acceptable performance levels. The experimental results provide empirical evidence of the framework's ability to adapt to various workflow characteristics and system configurations, establishing its practical applicability for real-world scientific computing environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The computational demands of modern science continue to grow exponentially. This transformation is primarily driven by the increasing need for computational power to solve complex problems across diverse scientific domains, from climate modeling and genomic sequencing to particle physics simulations and astronomical data analysis [1, 2]. Modern scientific research relies heavily on sophisticated computational methods that necessitate the development of advanced hardware architectures and software frameworks [3, 4]. These frameworks are capable of handling the enormous scale and complexity of contemporary scientific problems [5]. *Scientific workflows* are an attempt to meet this growing demand for complex computational solutions by harnessing the power of parallel computation [6]. Scientific workflows help reduce computational complexity by abstracting it as a series of multiple smaller tasks that can be executed independently [7]. The concept of scientific workflows can be applied to a wide range of computations as they are not specifically tailored for any individual application [8]. Scientific workflows are being used in mainstream computation as a viable alternative to traditional computing processes [9]. Additionally, scientific workflows are seen as a potential improvement over traditional computing by exploiting the parallel computing capabilities of modern computing architectures [10].

Structurally, scientific workflows comprise a series of small tasks [4]. These tasks are computed individually and independently. The tasks include the input data and the actual computation using the input data to generate some output. This output is then considered as input to future tasks. Scientific workflows are executed by workflow engines that manage the data, task dependencies, and reporting [11, 12, 13]. Scientific workflows provide a useful abstraction, allowing scientists

to efficiently execute computations without worrying about optimizing computing resources [14]. Workflow engines also enable scientists to use centrally managed cluster infrastructure without having to understand the details of the underlying infrastructure [15]. Due to this abstraction, and since most scientists do not manage their infrastructure, it becomes difficult for them to understand the environmental impact of their computation [16].

In this thesis, the current state of scientific workflow execution and the existing scheduling techniques for it are examined. An energy-aware framework is developed to investigate the impact of different scheduling and allocation techniques on the computation. The research also investigates the smart scheduling decisions that can be implemented to reduce the energy consumption of calculations without compromising their performance. The focus of this thesis is to aid scientists in making energy-aware decisions regarding their computations. The users do not need to understand the intricacies of their computation, and just provide their preference for the execution of their computation.

The remainder of this chapter is structured as follows. Section 1.2 provides an introduction to the reasons behind the emergence of scientific workflows and explains how they provide an improvement over traditional computation practices. The techniques currently used for scientific workflow computation are also presented. Section 1.3 presents the reasons behind the demand for energy-aware execution of workflows. Section 1.4 presents the aim and motivation of the research. The section also presents the contributions of this thesis to the domain of scientific workflow computing. Finally, Section 1.5 presents the structure of the thesis.

1.2 The Emergence of Scientific Workflow Computation

The increasing complexity of scientific computations and the limitations of traditional monolithic applications have driven the development of workflow-based approaches to scientific computing [4, 14]. Scientific workflows represent the shift from single-purpose applications to modular and reusable computational frameworks that can address the diverse and evolving needs of modern scientific research [7]. In this section, the emergence of workflows as a fundamental approach to organizing and executing complex scientific computations in high-performance computing environments is explored.

Scientific workflows have emerged as a natural response to the growing sophistication of scientific problems and the need for more flexible, maintainable, and scalable computational solutions [8]. The traditionally developed monolithic applications proved inadequate to meet the demands of the growing complexity and requirements of these computations [9]. Workflows provide a structured methodology for decomposing complex scientific problems into manageable, interconnected components that can be developed, tested, and executed independently while

maintaining the overall consistency of the scientific computation [11].

Section 1.2.1 presents the execution of scientific workflows from the perspective of High-Performance Computing and the advantages that can be achieved. Section 1.2.2 presents the different domains in which scientific workflows can be applied, and Section 1.2.3 presents the current approaches in the domain of scientific workflow computation. Finally, Section 1.2.4 presents the challenges and limitations of executing these workflows.

1.2.1 High Performance Computing and Workflows

The integration of workflows with high-performance computing (HPC) systems has created new opportunities for scalable and efficient scientific computation [15, 5]. HPC environments provide the computational resources necessary to execute large-scale scientific workflows, while workflows provide the organizational framework needed to utilize these resources [17] effectively. This synergy has enabled researchers to tackle computational problems of unprecedented scale and complexity, leading to significant advances in fields ranging from climate science and genomics to particle physics and materials research [18].

High-performance computing systems are particularly well-suited to workflow execution due to their parallel processing capabilities and sophisticated resource management systems [6, 19]. Modern HPC systems can execute multiple workflow tasks simultaneously across hundreds or thousands of processing cores, significantly reducing the time required to complete complex scientific computations [10]. The distributed nature of HPC systems also provides natural support for the modular structure of workflows, allowing different workflow components to be executed on different computing nodes or even different computing facilities [20].

The development of workflow management systems (WMS) designed explicitly for HPC environments has further enhanced the effectiveness of workflow-based scientific computing [21, 13]. These systems provide sophisticated scheduling algorithms that can optimize the mapping of workflow tasks to available computing resources, taking into account factors such as data locality, communication overhead, and resource availability [22]. Advanced workflow management systems also provide fault tolerance mechanisms that can automatically recover from hardware failures or software errors, ensuring the reliable execution of long-running scientific workflows [23].

1.2.2 Applications of Workflows

Scientific workflows have found widespread application across numerous scientific domains like bioinformatics, astrophysics, seismology, and simulations [4]. They are versatile and effective in addressing diverse computational challenges. In bioinformatics, workflows are extensively used for genomic analysis pipelines that process raw sequencing data through multiple stages of quality control, alignment, variant calling, and annotation [2]. These bioinformatics workflows involve the integration of different software tools and databases. Tools for sequence process and quality controls with downstream analysis such as FastQC [24], Bowtie2 [25], ANNOVAR [26] and DESeq2 [27] makes the workflow approach essential for managing the complexity of modern genomic analysis. Climate modeling represents another domain where workflows have proven invaluable. Climate simulations typically involve the coupling of multiple physical models representing different components of the Earth system, such as the atmosphere, oceans, land surface, and ice sheets [4, 28, 17]. Workflows are best suited for this kind of complex simulation and managing the complex data exchange between the different models. The exploration of varying parameter combinations can be effectively done due to the parallel characteristics of the workflow execution.

In astronomy and astrophysics, workflows are used to process and analyze large datasets from ground-based and space-based observatories [1, 29]. These workflows involve complex data analysis that calibrates the raw observations, removes instrumental artifacts, and extracts scientific measurements. The scale of modern astronomical surveys, which can generate terabytes (TBs) of data per night, makes workflow-based approaches essential for automated data processing and analysis. Microbiology and Drug discovery workflows represent another vital application domain, combining molecular modeling, database searches, and machine learning approaches to identify promising pharmaceutical compounds. These workflows integrate diverse computational tools and databases to screen large libraries of potential drug compounds, predict their biological activity, and optimize their chemical properties.

1.2.3 Current Approaches of Scientific Workflow Computing

Contemporary scientific workflow computing encompasses diverse methodologies and technologies that address varying priorities and use cases within the scientific computing community [7, 8]. These approaches can be categorized based on their target computing environments, workflow specification methods, execution strategies, and optimization objectives [4, 30]. Identifying the opportunities for improvement in these approaches requires an understanding of these approaches.

Performance-oriented approaches have traditionally dominated scientific workflow computing, focusing on minimizing execution time, maximizing throughput, and optimizing resource utilization

to achieve the fastest possible completion of scientific computations [22, 31]. These approaches employ specialized scheduling algorithms that analyze workflow dependencies to identify bottlenecks, load balancing techniques that distribute work evenly across available resources, and data locality optimization strategies that minimize data movement overhead [17, 32]. Adaptive scheduling algorithms are also gaining popularity as they can respond to dynamic system conditions [23, 33]. The primary limitation of purely performance-focused approaches is their tendency to maximize resource utilization and throughput without considering the associated implications on other factors of the computation, such as cost or accuracy [34].

Energy and cost reduction approaches have gained significant importance as the scale of computations has grown and operational expenses have become substantial portions of the total cost associated with the execution [16, 35]. These approaches employ techniques to reduce the energy consumption of the computation at minimal cost to the performance [36, 37]. Techniques such as Dynamic Voltage and Frequency Scaling (DVFS) have been an effective way to achieve lower energy consumption for a computation by adjusting the processor operating parameters such as frequency and operating energy supply [38, 39]. Resource consolidation strategies concentrate workloads on a few resources to reduce the number of active systems [40, 41]. Cloud computing cost optimization methods leverage the variable pricing and resource configuration flexibility to achieve the lowest cost of the computation [42, 43]. These approaches can achieve significant energy and cost savings, but they often require accepting longer execution times or reduced computational throughput, a trade-off that must be managed [44].

Multi-objective optimization approaches recognize that scientific workflow execution involves balancing multiple competing objectives and seek to find optimal solutions that consider performance, energy consumption, cost, reliability, and quality-of-service requirements simultaneously [45, 46]. These techniques utilize different optimization algorithms that can identify the optimal trade-offs between different objectives [47, 48]. It also allows users to express their preferences explicitly in terms of weighted objective functions [33, 49]. They employ constraint-based formulations that treat some requirements as hard constraints while optimizing other constraints [50, 51]. Advanced multi-objective approaches also incorporate fuzzy logic for handling uncertainty and machine learning techniques for improving optimization effectiveness based on historical execution data [52, 53]. The primary challenge for multi-objective approaches lies in their computational complexity and the difficulty of determining appropriate trade-offs between conflicting objectives in real-world deployment scenarios [54].

1.2.4 Challenges and Limitations of Executing Workflows

Despite their many advantages, executing scientific workflows presents significant challenges and limitations that must be carefully considered in their design and implementation [14, 8]. One of the primary challenges is the complexity of workflow development and maintenance [4, 55].

Creating effective workflows requires expertise in multiple areas, including the underlying scientific domain, computational methods, distributed computing, and workflow management systems. This multidisciplinary requirement can create barriers to adoption, particularly for researchers who lack extensive computational backgrounds.

Performance optimization represents another significant challenge in workflow-based computing [22, 55]. While workflows provide natural opportunities for parallelization, achieving optimal performance requires careful consideration of task granularity, data locality, and resource allocation [32, 56]. Workflows with many small tasks may suffer from scheduling overhead and communication costs, while workflows with large tasks may not fully utilize available parallel resources. Finding the right balance requires a detailed understanding of both the workflow structure and the target computing environment.

The performance or execution time of a scientific computation depends on many factors such as the architecture of computing resources, the operating system used, the environment used for parallel programming, and the model used by the environment [57]. While developing or choosing a platform to execute a scientific computation, all these factors need to be considered. There are many complex interactions between all these factors, and it is difficult to consider them all [57].

Data management challenges become particularly acute in large-scale scientific workflows that process massive datasets [14, 55]. The movement and storage of intermediate data products can become a significant bottleneck, particularly when workflow tasks are distributed across multiple computing sites [55, 17]. Network bandwidth limitations, storage constraints, and data transfer costs must all be considered in workflow design and execution planning.

Error handling and fault tolerance remain significant challenges, particularly for long-running workflows executing in distributed environments [23, 58]. While workflow management systems provide various mechanisms for dealing with failures, determining the appropriate level of fault tolerance and implementing effective recovery strategies requires careful consideration of the trade-offs between reliability, performance, and resource consumption [58, 59]. Effective workflow execution in HPC environments requires sophisticated resource management strategies that can consider these varying requirements while maintaining high system utilization.

Resource allocation in HPC environments presents unique challenges for workflow execution [19, 60]. Unlike traditional batch jobs that typically request a fixed number of resources for a predetermined duration, workflows often exhibit dynamic resource requirements that change as the computation progresses [61]. Some workflow tasks may require large amounts of memory or specialized hardware accelerators, while others may be more compute-intensive or I/O-bound.

Energy consumption and cost management represent a significant emerging challenge in the field of scientific workflow execution [16, 35]. Executing a workflow involves complex involvement of the networking infrastructure, compute resources, and resource configurations,

creating challenges for effectively and accurately measuring energy consumption. The growing scale of scientific computations and rising energy costs have made energy efficiency a critical concern for research institutions and cloud computing providers [62, 63]. Measuring the energy consumption of the computation is a vital step in understanding and analyzing the energy and cost of a particular computation [64, 65]

1.3 Energy-Aware Scientific Workflow Scheduling

As computational demands have grown exponentially, so too has the energy consumption of high-performance computing systems [62, 63]. Modern HPC installations can consume megawatts of power, resulting in substantial operational costs and significant environmental impacts [65, 66]. Traditional workflow scheduling algorithms focus primarily on minimizing execution time, often leading to resource allocation strategies that maximize computational performance at the expense of energy efficiency [22, 35]. This has resulted in exploration of approaches that consider energy efficiency as a critical design and operational parameter [36, 37].

The modular structure of workflows provides natural opportunities for energy optimization through intelligent task scheduling and resource allocation [16, 67]. Different workflow tasks may have varying computational intensities, memory requirements, and communication patterns, leading to different energy consumption profiles [68, 64]. By understanding these characteristics, workflow schedulers can make informed decisions about task placement and resource allocation that minimize overall energy consumption while maintaining acceptable performance levels [42].

1.3.1 Current Approaches in Energy-Aware Workflow Scheduling

Existing approaches to energy-aware scientific workflow scheduling include a diverse range of methodologies and techniques that reflect different strategies for balancing performance and energy objectives. These approaches can be broadly categorized based on their optimization strategies, energy modeling techniques, and integration with existing workflow management systems. Understanding the strengths and limitations of current approaches is essential for identifying opportunities for advancement in this rapidly evolving field.

Heuristic-based scheduling algorithms represent one major category of current approaches, employing rule-based strategies to make scheduling decisions that consider both performance and energy criteria [69]. These algorithms typically use predefined rules or policies to guide resource allocation decisions, such as preferentially assigning tasks to more energy-efficient processors or consolidating workloads to minimize the number of active computing nodes [70]. Examples include energy-aware list scheduling algorithms that modify traditional critical path

scheduling to consider energy consumption, and cluster-based approaches that group similar tasks to improve energy efficiency through better resource utilization [71]. Mathematical optimization approaches constitute another important category, formulating energy-aware scheduling as constrained optimization problems that can be solved using techniques from operations research and mathematical programming. These approaches typically model workflow scheduling as multi-objective optimization problems that seek to minimize both execution time and energy consumption subject to various constraints such as task dependencies, resource capacities, and quality-of-service requirements [72, 73, 74]. Linear programming, integer programming, and evolutionary algorithms have all been applied to energy-aware workflow scheduling problems with varying degrees of success [75].

Dynamic voltage and frequency scaling (DVFS) based approaches leverage the ability of modern processors to adjust their operating voltage and frequency to trade off computational performance and energy consumption [39, 76]. These techniques are integrated into workflow scheduling algorithms to optimize energy consumption by adjusting processor performance levels based on task characteristics and timing requirements. Some approaches use predictive models to determine optimal DVFS settings for different workflow tasks, while others employ feedback control mechanisms to adjust processor performance during workflow execution dynamically [39].

Machine learning approaches have emerged as a promising direction for energy-aware workflow scheduling, using historical execution data and system characteristics to build predictive models that can guide scheduling decisions [77]. These approaches employ supervised learning techniques to predict the energy consumption of different scheduling decisions, reinforcement learning to continuously improve scheduling policies based on observed outcomes, or unsupervised learning to identify patterns in workload characteristics that can inform energy optimization strategies. Hybrid approaches combine multiple techniques to address different aspects of the energy-aware scheduling problem. For example, some systems use mathematical optimization for initial workflow mapping decisions and then apply heuristic techniques for runtime adaptation and adjustment [52]. Others combine machine learning approaches for energy prediction with mathematical optimization for schedule generation, leveraging the strengths of different methodologies to achieve better overall performance.

1.3.2 Limitations and Drawbacks of Current Approaches

Despite significant research effort and promising initial results, current approaches to energy-aware scientific workflow scheduling face several fundamental limitations that restrict their practical applicability [35, 37]. These limitations stem from various sources, including inadequate energy modeling capabilities, oversimplified optimization formulations, insufficient consideration of system dynamics, and limited integration with existing workflow management systems [16, 67].

Energy modeling limitations represent one of the most significant challenges facing current approaches [64, 65]. Many existing methods rely on simplified energy models that fail to capture the complex relationships between application characteristics, system utilization patterns, and actual energy consumption [68, 35]. These models often make unrealistic assumptions about hardware behavior, such as assuming linear relationships between processor utilization and power consumption, or neglecting the energy overhead associated with task scheduling and data movement [78, 79]. The heterogeneous nature of modern computing systems adds additional complexity to energy-aware workflow execution [31, 80]. Different types of computing resources, including general-purpose processors, graphics processing units (GPUs), and specialized accelerators, have vastly different energy consumption characteristics [81, 20].

Scalability issues pose another major limitation for many current approaches, particularly those based on mathematical optimization techniques [48, 47]. The computational complexity of multi-objective optimization problems grows rapidly with the size of workflows and the number of available computing resources, making it challenging to apply these approaches to large-scale scientific workflows that may contain thousands of tasks and execute on computing systems with thousands of processing cores [82, 83]. Even heuristic approaches struggle with scalability when they consider complex energy models or maintain detailed system state information [84].

Limited adaptability represents a significant drawback of many current approaches, which often make scheduling decisions based on static information about workflow characteristics and system capabilities [85, 23]. Real-world workflow execution involves dynamic conditions, including varying system load, changing resource availability, and unpredictable task execution times that can significantly impact both performance and energy consumption [55, 86]. Approaches that cannot adapt to these changing conditions may produce suboptimal results or fail when actual execution conditions differ significantly from initial assumptions [87, 61].

Integration challenges with existing workflow management systems limit the practical deployment of many energy-aware scheduling approaches [21, 13]. Most current research focuses on algorithmic development without adequate consideration of how these algorithms can be integrated into existing systems [4, 30]. This includes challenges related to energy monitoring infrastructure, compatibility with existing scheduling frameworks, and the need for extensive system modifications to support energy-aware scheduling capabilities [19, 60].

Validation and evaluation limitations affect many current approaches, which are often evaluated using simplified simulations or small-scale experiments that may not accurately reflect the complexity and scale of real-world scientific workflow execution [88, 89]. The lack of standardized benchmarks and evaluation methodologies makes it challenging to compare different approaches or assess their practical effectiveness [90, 91]. Many evaluations focus primarily on algorithmic performance metrics without adequate consideration of practical factors such as deployment complexity, system overhead, and user acceptability [92, 93].

1.4 Research Aims and Contributions

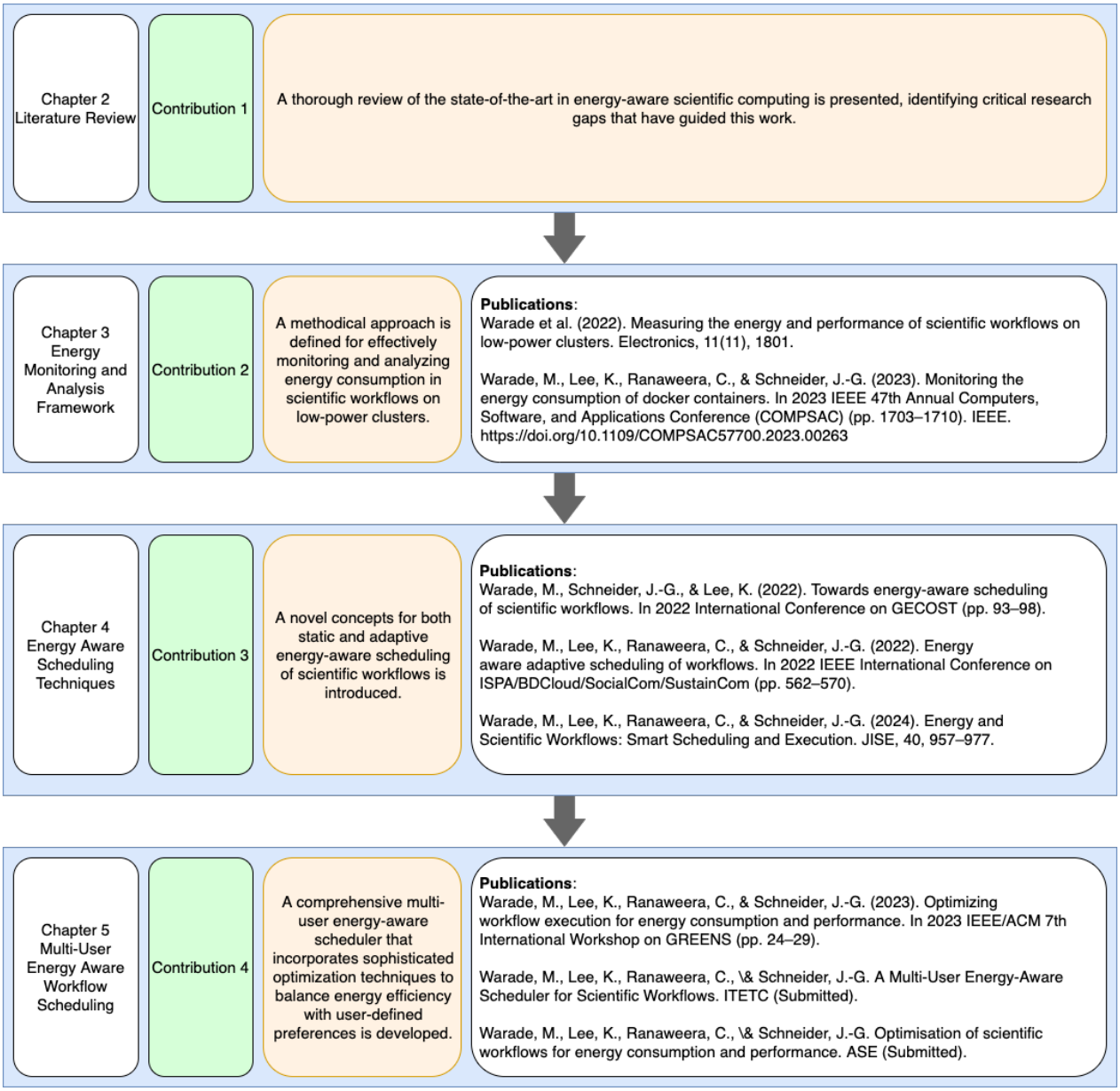


Figure 1.1: Mapping Publications to Contributions

The main aim of this thesis is **to investigate and address the critical challenge of energy consumption in scientific workflow execution by exploring energy-aware execution and innovative resource allocation strategies.** An approach for energy-aware scientific workflow scheduling is proposed as a solution that enables a significant reduction in energy consumption of high-performance computing systems while maintaining acceptable computational performance and scientific output quality in multi-user, multi-workflow environments with resource contention. In this section, the aim of this thesis and an overview of the research contributions are outlined.

Figure 1.1 illustrates the mapping between thesis contributions, chapters, and the published papers that form the basis of each chapter, along with a summary of each chapter. This thesis makes several significant contributions to the field of energy-aware scientific workflow scheduling:

1. **A Literature Review and Gap Analysis** (Chapter 2)
   A review of the state-of-the-art in energy-aware scientific computing is presented, identifying critical research gaps that have guided this work. This systematic analysis provides essential background knowledge and contextualizes the novel contributions of this research within the broader scientific community. The literature review establishes a foundation for understanding both existing approaches and the innovative solutions proposed in this thesis.

2. **A Framework for Energy Consumption Monitoring and Analysis** (Chapter 3)
   This contribution details a methodical approach for effectively monitoring and analyzing energy consumption in scientific workflows on compute clusters. The research identifies and characterizes key factors affecting energy consumption during scientific computations. This work is substantiated by comprehensive empirical data collected across diverse scientific workflows, providing valuable insights into the energy consumption patterns of scientific computing environments. The monitoring framework serves as a critical foundation for subsequent energy-aware scheduling strategies.

3. **Techniques and Mechanisms for Energy-Aware Scheduling** (Chapter 4)
   This contribution introduces novel concepts and mechanisms for both static and adaptive energy-aware scheduling of scientific workflows. The research presents theoretical foundations and algorithms that dynamically respond to system status, energy constraints, and computational progress. A proof-of-concept scheduler demonstrates the effectiveness of these proposed approaches, validating their practical application in real-world scientific computing environments.

4. **A Multi-User Energy-Aware Workflow Scheduling System** (Chapter 5)
   An approach to multi-user energy-aware scheduling is proposed that incorporates sophisticated optimization techniques to balance energy efficiency with user-defined preferences. This system effectively addresses contention between multiple workflows and users, proposing dynamic optimization strategies for resource allocation to minimize energy consumption while meeting performance requirements. Extensive evaluation demonstrates that the proposed approach successfully achieves the dual objectives of energy efficiency and computational performance across heterogeneous cluster environments.

Together, these contributions advance the state-of-the-art in energy-aware scientific workflow scheduling, providing both theoretical frameworks and practical approaches that enable more sustainable high-performance computing. The research demonstrates that optimized energy-aware

scheduling strategies can significantly reduce energy consumption while maintaining computational performance, thus meeting the diverse goals of both system administrators and scientific users.

The research contributes significantly to the advancement of scheduling algorithms through the introduction of optimization-based approaches that can automatically determine optimal schedules for energy-aware workflow execution. These proposed algorithms represent a substantial advancement over existing rule-based approaches by providing mathematically rigorous solutions to the multi-objective optimization problem inherent in energy-aware scheduling.

1.5 Structure of this Thesis

Figure 1.2 illustrates the structure of the thesis, showing how each chapter contributes to the overall work and providing brief chapter summaries. The remainder of this thesis is as follows:

Chapter 2 presents a comprehensive literature review that establishes the theoretical foundation for the research. The chapter begins with a cluster computing background, covering the evolution of HPC systems, modern cluster architectures, and energy consumption characteristics. It then examines related work in scientific workflow execution and scheduling techniques, reviewing existing workflow management systems, traditional scheduling algorithms, and performance optimization approaches. The chapter concludes by identifying open issues and research gaps, including limitations in current energy-aware scheduling, insufficient multi-user optimization, inadequate handling of resource contention, and a lack of standardized energy measurement frameworks.

Chapter 3 introduces the Energy Monitoring and Analysis Framework, presenting the first significant contribution of this thesis. The chapter proposes a framework for reliable energy monitoring of computations, detailing design principles for portable energy measurement, hardware-level power monitoring techniques, and software-based energy profiling methods. The experimental framework design, setup, and evaluation section describes the testbed configuration, benchmark selection, measurement protocols, and validation techniques used to assess the framework's effectiveness and portability across different HPC platforms.

Chapter 4 focuses on Energy Aware Scheduling Techniques, presenting the second significant contribution through the development of static and adaptive energy-aware schedulers for scientific workflow computations. The chapter provides a mathematical formulation of the energy-aware scheduling problem, presents both static and adaptive scheduling algorithms, and discusses multi-objective optimization approaches. The experimental evaluation demonstrates the implementation of proposed algorithms, their integration with workflow management systems, and comparative analysis with baseline schedulers using real-world application case studies.

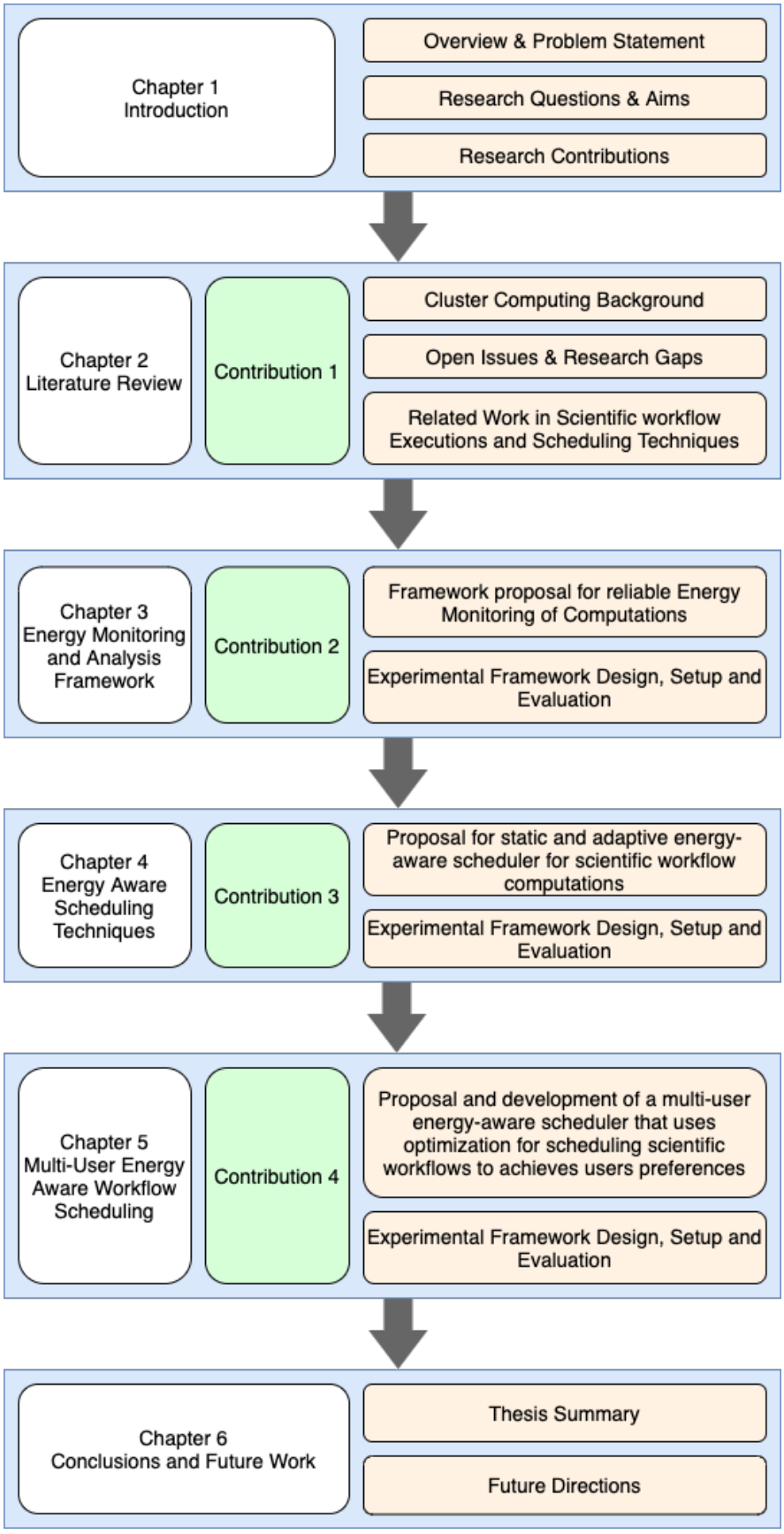


Figure 1.2: Thesis Structure

Chapter 5 addresses Multi-User Energy Aware Workflow Scheduling, presenting the third and fourth significant contributions of this thesis. The chapter proposes and develops a comprehensive multi-user energy-aware scheduler that uses optimization techniques to schedule scientific workflows while achieving individual user preferences. Key aspects include multi-user scheduling problem formulation, user preference modeling, fair resource allocation mechanisms, resource contention modeling, and multi-workflow coordination. The experimental evaluation section describes the multi-user testbed setup, workload generation, performance metrics for multi-user environments, and assessment of fairness, efficiency, and system scalability.

Chapter 6 provides conclusions and future work, summarizing the key achievements of this research and their implications for the scientific computing community. The thesis summary recapitulates the research objectives, validates the research hypotheses, and discusses the significance of results while acknowledging limitations of the proposed approaches. The future directions section outlines potential extensions to cloud and edge computing environments, integration with machine learning techniques, advanced energy prediction models, and long-term sustainability considerations for community adoption.

Chapter 2

Literature Review

2.1 Overview

This chapter provides a comprehensive review of the research landscape in the field of energy-aware scientific workflow scheduling, establishing the foundational knowledge and current state-of-the-art required for the research presented in this thesis. The chapter is organized into several main sections that progressively build understanding from fundamental concepts to identifying major research gaps and outlining the research contributions.

Section 2.2 examines the high-performance computing infrastructure that underlies scientific computing, including hardware components, software systems, and cluster architectures. Section 2.3 explores energy monitoring and measurement approaches in computing systems, covering both hardware-based and software-based monitoring solutions. Section 2.4 focuses on scientific workflow execution and management systems, including workflow models, management platforms, and resource schedulers. The related work section forms the core of the literature review. Section 2.5 reviews existing approaches to energy-aware computing and optimization, examining energy-aware workflow execution techniques and scheduling approaches for energy optimization. Section 2.6 analyzes research in experimental cluster computing and workflow automation that provides methodological foundations for energy-aware system evaluation.

The chapter concludes with critical analysis sections that identify gaps in current research. Section 2.7 analyzes the review findings to identify open issues and research opportunities in current approaches. Section 2.8 positions the contributions of this thesis within the broader research landscape. Finally, Section 2.9 provides a comprehensive summary of the key findings and their implications for energy-aware scientific workflow scheduling research.

Through this structured approach, this chapter establishes both the technical foundations and research context necessary for understanding the novel contributions presented in subsequent chapters. The review demonstrates that while significant progress has been made in individual areas such as energy monitoring, workflow scheduling, and optimization techniques, substantial gaps remain in developing integrated solutions that can effectively balance energy efficiency with performance requirements in realistic multi-user computing environments.

2.2 High Performance Computing Infrastructure

High-performance computing infrastructure forms the foundational layer upon which energy-aware scientific workflow scheduling operates [15, 5]. The energy consumption characteristics and computational capabilities of individual hardware and software components directly influence scheduling decisions and optimization strategies [65, 63]. This section provides an examination of the key infrastructure components that enable distributed scientific computing while establishing the technical foundation necessary for understanding energy-aware scheduling challenges and opportunities. Section 2.2.1 examines the hardware components that comprise modern cluster computing systems. Section 2.2.2 explores the software stack that enables coordination and management of distributed computing resources. Finally, Section 2.2.3 analyzes cluster computing architectures and scalability considerations, examining how different architectural choices impact both computational capabilities and energy consumption patterns.

2.2.1 Hardware Infrastructure for Cluster Computing

The hardware infrastructure of cluster computing systems forms the foundation for energy-aware scientific workflow scheduling, as the energy consumption characteristics of individual components directly influence scheduling decisions and optimization strategies [93, 94]. Understanding the energy profiles and computational capabilities of different hardware components is essential for developing effective energy-aware scheduling approaches that can balance performance requirements with energy efficiency objectives [68, 65].

Energy Considerations for Processors and Compute Units

Modern processors represent the primary computational resource and energy consumer in cluster computing systems. The Central Processing Unit (CPU) comprises a processing unit (PU) and a control unit (CU) that perform basic arithmetic, logic, control, and input/output operations. Traditional CPUs are typically implemented as microprocessors contained on a single integrated

circuit (IC) chip, with multi-core designs featuring two or more separate cores working together to achieve improved computational performance.

Processor architecture fundamentally determines both computational capabilities and energy efficiency characteristics [81, 20]. The two dominant instruction set architectures exhibit markedly different energy profiles. x86 processors, based on Complex Instruction Set Computing (CISC) principles, provide high performance for general-purpose computing but typically consume more energy due to their complex instruction decoding and execution mechanisms [80]. ARM processors, built on Reduced Instruction Set Computing (RISC) principles, achieve significantly higher energy efficiency through simplified instruction sets and optimized power management, making them increasingly popular in energy-constrained environments and large-scale deployments [78, 93].

The energy consumption characteristics of processors are fundamentally linked to their operational parameters, particularly clock frequency and voltage levels. Clock rate refers to the frequency at which a processor executes instructions [95] and synchronizes component operations [95]. Higher clock rates generally indicate faster processors and improved computing performance [96], but they also result in increased energy consumption. Dynamic Voltage and Frequency Scaling (DVFS) techniques enable runtime adjustment of processor energy consumption by modifying voltage and frequency settings. Overvolting increases clock rate and energy consumption for enhanced performance, while undervolting reduces both parameters to achieve energy savings at the cost of computational speed.

Multi-core processor architectures introduce additional complexity for energy-aware scheduling due to the need for inter-core communication and resource sharing. Cores can be tightly coupled through shared cache systems or loosely coupled through message passing or physically shared memory interfaces. The coupling mechanism significantly impacts both performance and energy consumption patterns, as tightly coupled cores may share power management domains while loosely coupled cores can often be managed independently.

Modern scientific computing environments employ diverse processor architectures that exhibit different computational capabilities and energy efficiency characteristics. Traditional multi-core processors provide general-purpose computing with sophisticated power management features including DVFS [39], power gating, and thermal management. Graphics Processing Units (GPUs) offer massive parallel processing capabilities for specific computational workloads but demonstrate different energy consumption patterns compared to traditional processors. Specialized accelerators, including Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs), provide highly efficient computation for specific algorithm classes but require careful consideration of their unique energy characteristics in scheduling decisions.

Memory and Storage Systems

Memory and storage subsystems significantly impact both performance and energy consumption in scientific computing environments, particularly for data-intensive computations involving large datasets and complex data access patterns. The energy overhead associated with data movement between different levels of the storage hierarchy must be carefully considered in energy-aware scheduling decisions.

Computer memory systems typically comprise Random Access Memory (RAM) for temporary information storage and Read-Only Memory (ROM) for permanent storage. In computational contexts, RAM serves as the primary working memory that the CPU utilizes for active computations, including synchronization, input/output operations, and intermediate results storage. The energy consumption of memory systems varies significantly based on access patterns, data volume, and the specific memory technologies employed.

Multi-core computing systems require sophisticated memory management approaches to enable effective resource sharing across cores. The shared memory model includes multiprocessors that communicate with each other through a shared or global memory, and the message passing model refers to a multi-computer environment where intercommunication is achieved through message passing interfaces (MPIs) [3]. Shared memory models allow multiple cores to communicate through a common memory space. In contrast, distributed memory systems implement Virtual Shared Memory (VSM) models that provide processors with an abstraction of shared memory across physically distributed nodes [3]. These different memory architectures exhibit distinct energy consumption patterns and require specific considerations in energy-aware scheduling algorithms [68, 65].

Cache memory systems provide high-speed data storage for frequently accessed information, offering faster access times than primary RAM at the cost of increased energy consumption per unit of storage. The cache hierarchy, typically including multiple levels (L1, L2, L3), creates complex energy trade-offs where cache hits reduce energy consumption by avoiding slower memory accesses. In contrast, cache misses result in increased energy overhead due to additional memory operations.

Storage systems in scientific computing environments range from traditional Hard Disk Drives (HDDs) to Solid-State Drives (SSDs) and emerging storage-class memory technologies. Each storage type offers different trade-offs between performance, capacity, and energy efficiency. Modern systems employ sophisticated memory hierarchies that include multiple cache levels, high-bandwidth memory, and non-volatile memory technologies, each contributing distinct energy consumption characteristics that must be considered in comprehensive energy-aware scheduling strategies.

Networking and Interconnect Infrastructure

High-performance networking and interconnect technologies enable the distributed execution of scientific workflows across multiple computing nodes while contributing significantly to overall system energy consumption. The energy characteristics of network infrastructure must be carefully considered in energy-aware scheduling decisions, as data movement and communication patterns directly impact both performance and energy efficiency.

Modern HPC networks employ high-bandwidth, low-latency interconnects such as InfiniBand [97], high-speed Ethernet, and proprietary solutions that provide the communication capabilities required for parallel scientific applications. InfiniBand represents one of the most widely adopted high-performance interconnect technologies, offering multiple generations with varying performance and energy characteristics [98]. Current implementations include HDR (High Data Rate) at 200 Gbps, EDR (Enhanced Data Rate) at 100 Gbps, and FDR (Fourteen Data Rate) at 56 Gbps per port, each with distinct energy consumption profiles [98].

InfiniBand's Remote Direct Memory Access (RDMA) capabilities enable direct memory-to-memory transfers between nodes without CPU intervention, significantly reducing communication energy overhead while improving application performance. High-speed Ethernet has evolved to compete with specialized HPC interconnects, with 25, 50, and 100 Gigabit Ethernet becoming increasingly common. Technologies such as RDMA over Converged Ethernet (RoCE) and iWARP provide RDMA capabilities over Ethernet infrastructure, combining performance benefits with cost and energy advantages [99].

Network topology choices significantly impact both performance and energy consumption characteristics. Fat-tree topologies provide non-blocking communication between nodes but require substantial switching infrastructure with proportional energy consumption [100]. Mesh and torus topologies can reduce switching requirements while maintaining good communication performance for regular communication patterns. Dragonfly networks balance low-diameter benefits with reduced hardware and energy requirements through sparse inter-group connectivity.

The integration of network monitoring and telemetry systems enables runtime optimization of network energy consumption through real-time monitoring of utilization, power consumption, and performance metrics. These systems provide valuable data for developing accurate energy models that account for complex interactions between network topology, communication patterns, and power consumption.

Power and Cooling Infrastructure

Power and cooling infrastructure represents one of the most critical aspects of cluster computing system design, directly impacting operational costs, system reliability, and environmental sustainability. Modern HPC clusters consume substantial electrical power, with petascale systems typically requiring 10-20 megawatts and exascale systems projected to consume 20-30 megawatts, generating significant heat that must be efficiently removed.

Power Distribution Units (PDUs) serve as the final stage of power distribution, converting facility-level voltages to computing node requirements while incorporating intelligent monitoring capabilities that provide real-time power consumption data. The efficiency of the power distribution chain significantly impacts overall system energy consumption, with typical data center systems achieving 85-90% efficiency from utility feed to computing components.

Thermal management presents significant challenges due to high power densities and concentrated heat generation. Traditional air cooling systems employ Computer Room Air Conditioning (CRAC) units, maintaining facility temperatures between 18-27°C with relative humidity levels between 40-60%. Advanced cooling technologies, including liquid cooling, immersion cooling, and direct-to-chip cooling, offer improved efficiency but require careful integration with energy-aware scheduling strategies.

Emerging technologies such as on-chip cooling and advanced materials promise to further improve the efficiency and effectiveness of HPC cooling systems. Silicon photonic and advanced packaging technologies may enable more efficient heat removal directly from semiconductor devices, potentially reducing the burden on facility-level cooling systems. The integration of artificial intelligence and machine learning into power and cooling management systems offers opportunities for more sophisticated optimization strategies. These systems can learn from historical patterns and adapt to changing conditions more effectively than traditional rule-based control systems, potentially achieving significant energy savings.

Single Board Computers and Edge Computing

Single Board Computers (SBCs) represent an emerging class of energy-efficient computing platforms that offer unique opportunities for energy-aware scientific computing. SBCs provide complete computing functionality on a single circuit board, typically implemented with ARM processors that deliver reasonable performance compared to their cost and power consumption [72, 101]. The evolution of SBC technology has resulted in approximately 20-fold performance improvements over predecessors within five years [93], making SBC clusters increasingly viable for scientific computing applications. While individual SBCs have computational limitations compared to traditional systems [93], clusters of SBCs can provide competitive performance with superior energy efficiency for appropriate workloads.

Single Board Computer	Family	CPU	Memory
Beagleboard-xm	Cortex A8	TI DM3730 512	512 MB
Beaglebone Black	Cortex A8	TI AM3358/9	512 MB
Pandaboard ES	Cortex A9	TI OMAP4460	1 GB
ODROID-xU	Cortex A7	Exynos 5 Octa	2 GB
Dragonboard	Cortex A53	Snapdragon 410	1 GB
Jetson TX-1	Cortex A57	Tegra X1	4 GB
Raspberry Pi 3	Cortex A53	Broadcom BCM2837	1 GB
Raspberry Pi 3B+	Cortex A53	Broadcom BCM2837B0	1 GB
Raspberry Pi 4B	Cortex A72	Broadcom BCM2711	1/2/4/8 GB
Raspberry Pi 5	Cortex A76	Broadcom BCM2712	2/4/8 GB
Orange Pi 5	Cortex A76/A55	Rockchip RK3588S	4/8/16 GB
Orange Pi 5 Pro	Cortex A76/A55	Rockchip RK3588S	4/8/16 GB
NVIDIA Jetson AGX Orin	Cortex A78AE	NVIDIA Tegra	32/64 GB
LattePanda 3 Delta	x86	Intel Celeron N5105	8 GB

Table 2.1: Comparison of widely used Single Board Computers

Table 2.1 presents commonly used SBCs, with the Raspberry Pi and ODROID series representing well-researched platforms. The latest Raspberry Pi 5 features an ARM Cortex-A76 CPU, up to 8GB RAM, and advanced connectivity options, while ODROID systems provide octa-core ARM processors with competitive specifications. These platforms offer laptop-equivalent capabilities in energy-efficient form factors suitable for distributed computing applications.

Edge computing is a decentralized approach to computing that moves processing power and data storage nearer to where data is generated, instead of depending entirely on remote centralized servers or cloud infrastructure [102, 93]. The integration of edge computing resources into scientific workflow execution requires new scheduling strategies that can effectively utilize heterogeneous computing environments while considering the energy implications of task placement and data movement decisions [103, 104]. Edge computing platforms typically offer lower absolute computational performance than traditional HPC systems but provide significantly better energy efficiency for appropriate workloads [103].

2.2.2 Software Infrastructure and Resource Management

The software infrastructure that enables distributed scientific computing comprises multiple layers of system software, middleware, and application frameworks that collectively determine the efficiency and energy consumption characteristics of computation. The effectiveness of energy-aware scheduling depends not only on hardware capabilities but critically on the software's ability to interact with hardware components in the most efficient manner possible. Incompatibilities between software and hardware can lead to performance degradation, system instabilities, and suboptimal energy utilization that affects overall computational effectiveness [105]. This section

aims to examine the key software technologies that support scientific workflow execution in distributed computing environments, from low-level operating system services to high-level application frameworks and scheduling systems.

Operating Systems and Power Management

Modern operating systems (OS) provide the fundamental platform for scientific computing, offering process management, resource allocation, and power management capabilities that directly impact energy consumption [106, 16]. Linux has emerged as the dominant operating system for cluster computing environments due to its open-source nature, customizability, and robust support for distributed computing [15, 5]. Popular Linux distributions used in cluster environments include Ubuntu Server, CentOS, and SUSE Linux Enterprise Server, each offering different trade-offs between stability, performance, and support ecosystems [93].

The Linux kernel provides sophisticated process scheduling mechanisms that can be optimized for energy-aware workloads [107]. The Completely Fair Scheduler (CFS) implements a red-black tree-based scheduling algorithm that provides good performance for interactive workloads, while alternative schedulers such as FIFO and Round Robin may be more appropriate for specific high-performance computing applications [19]. Memory management includes advanced features such as Non-Uniform Memory Access (NUMA) awareness, which is crucial for optimizing both performance and energy consumption on systems commonly used in cluster nodes [68].

Advanced power management features in traditional operating systems include CPU frequency scaling through the `CPUfreq` subsystem, which enables dynamic adjustment of processor operating frequencies based on current workload demands [38, 39]. The Linux kernel supports multiple CPU governors, including performance, powersave, ondemand, and conservative modes, each implementing different strategies for balancing performance and energy consumption [108, 109]. Idle state management through the `CPUidle` subsystem allows processors to enter low-power states during periods of inactivity, significantly reducing energy consumption when computational resources are not fully utilized [110]. Device power management frameworks enable coordinated power control across system components, including network interfaces, storage devices, and peripheral components [111]. The Linux kernel's runtime power management framework provides APIs for device drivers to implement sophisticated power management policies that higher-level energy management systems and scheduling frameworks can leverage [35, 37].

Virtualization and Container Technologies

Virtualization technologies add additional layers of abstraction that can both enable new optimization opportunities and introduce additional energy overhead that must be considered

in scheduling decisions [112, 113]. Hypervisor-based virtualization using technologies such as KVM, Xen, and VMware vSphere enables multiple virtual machines to share physical hardware resources while maintaining isolation between different workloads [40, 114].

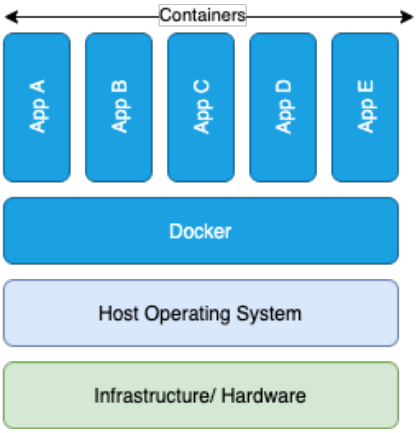


Figure 2.1: Container-based Docker Architecture

Container-based virtualization using technologies such as Docker, LXC, and Podman provides lightweight alternatives to traditional virtual machines by sharing the host operating system kernel while maintaining process and resource isolation [115, 116]. As illustrated in Figure 2.1, Docker enables multiple applications to run seamlessly on a single server, with each application and its dependencies isolated in its container, facilitating easy management and updates [117].

Container technologies typically impose lower overhead than virtual machines, making them attractive for high-performance computing applications where both performance and energy efficiency are critical [118, 119]. Docker containers provide infrastructure for lightweight, deterministic, and manageable isolated environments that enable agile computing resource utilization [120]. The lightweight nature of containers allows rapid deployment and scaling, which can be leveraged by energy-aware scheduling systems to dynamically adjust resource allocation based on current energy and performance requirements [121, 122]. Container orchestration platforms such as Kubernetes and Docker Swarm provide sophisticated mechanisms for managing containerized workloads across cluster environments [123, 124]. These platforms implement advanced scheduling algorithms that consider resource requirements, affinity rules, and quality of service constraints when placing containers on cluster nodes, providing opportunities for integration with energy-aware optimization strategies [125, 126].

Inter-node Communication and Middleware

Distributed scientific workflows require sophisticated communication middleware that enables efficient data exchange and coordination between workflow tasks executing on different computing nodes. The communication subsystem represents a critical component that significantly affects both performance and energy consumption characteristics of distributed applications.

The Message Passing Interface (MPI) standard provides a comprehensive framework for parallel computing that enables efficient communication and coordination between processes executing on distributed computing nodes. MPI defines standardized communication primitives, including point-to-point messaging, collective operations, and one-sided communications that form the foundation for most parallel scientific applications [127]. Multiple high-performance implementations are available, each optimized for different hardware architectures and energy efficiency characteristics.

MPICH provides a widely used open-source implementation that offers good performance across diverse hardware platforms and network interconnects. Open MPI emphasizes modularity and extensibility, enabling optimization for specific hardware configurations and communication patterns. Intel MPI Library provides a commercial implementation optimized for Intel processors and interconnect technologies. In contrast, MVAPICH is optimized explicitly for InfiniBand and high-speed Ethernet networks, implementing sophisticated algorithms for exploiting RDMA capabilities and reducing communication energy overhead. Language bindings for MPI enable parallel programming in multiple programming languages while maintaining performance and energy efficiency. MPI4Py provides Python bindings that allow rapid prototyping and development of parallel applications, while RMPI provides R language bindings for parallel statistical computing applications.

Storage Systems and Data Management

Distributed storage systems and parallel file systems provide the data management capabilities required for large-scale scientific workflows, with different storage architectures offering varying trade-offs between performance, reliability, and energy consumption. The storage subsystem significantly impacts both application performance and energy efficiency, as data movement operations can consume substantial energy and introduce performance bottlenecks.

Network File System (NFS) represents the most widely deployed distributed file system, providing transparent access to remote file systems across network connections. Modern NFS implementations include NFSv4, which incorporates advanced features such as strong security mechanisms, improved performance through protocol optimizations, and enhanced caching capabilities. NFSv4 implements sophisticated caching algorithms that can significantly reduce network traffic and improve application performance while reducing energy consumption through decreased data movement.

Data placement strategies significantly impact both performance and energy consumption of storage systems. Intelligent data placement can improve locality and reduce network traffic, while also enabling more effective power management through concentrated data access patterns. Caching and prefetching mechanisms can improve performance while providing opportunities for energy optimization through reduced storage device access frequency, enabling more aggressive power management strategies for storage hardware.

Parallel Programming Models and Energy Efficiency

The efficiency of scientific workflow execution depends heavily on the parallel algorithms and programming models used to implement individual workflow tasks. Different parallelization strategies exhibit different computational and communication patterns that lead to varying energy consumption characteristics.

Data parallelism represents the most common parallelization approach, where the same computational operations are applied to different portions of a large dataset. Data parallel algorithms typically exhibit regular communication patterns and good load-balancing characteristics, making them well-suited for energy-efficient execution. Task parallelism involves decomposing applications into independent tasks that can execute concurrently on different processing elements, requiring sophisticated load balancing mechanisms to achieve good performance and energy efficiency.

OpenMP provides a widely used programming model for shared memory parallel computing, enabling developers to add parallelism to existing sequential programs through compiler directives. The OpenMP runtime system includes sophisticated mechanisms for thread management, work distribution, and synchronization that can be optimized for different hardware architectures and energy consumption targets. MPI+OpenMP represents a popular hybrid programming model that combines distributed memory parallelism through MPI with shared memory parallelism through OpenMP. This approach can achieve good performance and energy efficiency on modern cluster systems with multi-core nodes, enabling optimization of both inter-node and intra-node parallelism while considering energy consumption at multiple system levels.

Resource Schedulers and Job Management Systems

Resource schedulers and job management systems provide the infrastructure for allocating computing resources to scientific workflows and managing their execution across distributed computing environments. These systems serve as critical middleware components, bridging the gap between complex scientific workflows and the underlying computational infrastructure while increasingly incorporating energy-aware features.

HTCondor represents a widely adopted high-throughput computing scheduler that implements a matchmaking mechanism pairing computational tasks with available resources based on their requirements and characteristics [128, 7]. HTCondor's Directed Acyclic Graph Manager (DAGMan) serves as a meta-scheduler that supervises workflow execution and submits tasks to HTCondor while handling dependencies between jobs [28]. The ClassAd mechanism allows for detailed specification of both resource capabilities and job requirements, enabling precise matching of tasks to appropriate computational resources.

SLURM Workload Manager focuses on highly scalable scheduling for large-scale clusters, implementing various scheduling algorithms, including backfill scheduling that allows smaller jobs to execute ahead of larger waiting jobs when doing so does not delay predicted start times. SLURM's architecture incorporates sophisticated mechanisms for handling node failures, power management, and quality of service guarantees, making it particularly suitable for large-scale scientific computing facilities requiring energy-aware optimization. PBS (Portable Batch System) and its variants implement sophisticated queue-based scheduling mechanisms where different queues can be configured with distinct scheduling policies, resource limits, and access controls. This approach allows organizations to implement complex scheduling policies that reflect specific needs, such as energy efficiency requirements or user prioritization, while maintaining fair resource allocation.

Kubernetes represents a specialized scheduler for containerized workloads that handles both task scheduling and the entire lifecycle management of containerized applications. Kubernetes implements a sophisticated scheduling framework considering factors such as resource requirements, node affinity, anti-affinity rules, and quality of service requirements, providing opportunities for integration with energy-aware scheduling strategies.

Traditional schedulers such as SLURM, PBS, and LSF focus primarily on performance optimization and resource utilization but increasingly incorporate energy-aware features. Advanced scheduling frameworks provide APIs and extensibility mechanisms that enable integration of custom energy-aware scheduling algorithms. The interaction between workflow management systems and underlying resource schedulers significantly affects the ability to implement effective energy optimization strategies, making the choice and configuration of these systems critical for energy-aware scientific computing environments.

2.2.3 Cluster Computing Architectures and Scalability

High-performance cluster computing represents the combination of integrated hardware and software technologies working together to deliver computational capabilities that exceed what different computing systems can achieve individually. This paradigm combines the distributed computational power of multiple interconnected nodes with sophisticated software coordination mechanisms to enable the solution of complex scientific problems. The effectiveness of cluster computing systems depends critically on optimizing interactions between hardware components, software layers, and application characteristics, requiring sophisticated management strategies that consider complex interdependencies between computational performance, energy efficiency, and system reliability. This section aims to provide background on the necessary cluster computing architecture required for High-Performance Computing and how the systems are integrated to achieve the scale necessary for the complex computation.

Cluster Computing Fundamentals and Energy Considerations

A cluster computing system consists of a collection of interconnected computers that work together to solve computational problems as a unified system while managing energy consumption across distributed resources. Unlike traditional multiprocessor systems, where multiple processing units share common memory and resources within a single machine, cluster computing employs a distributed memory architecture where each node maintains its local memory and communicates with other nodes through message passing over network interconnects.

The fundamental principle underlying cluster computing is the decomposition of large computational problems into smaller, independent tasks that can be executed concurrently across multiple nodes. This approach leverages Amdahl's Law, which states that the speedup of a parallel program is limited by the sequential portion of the computation, while also introducing energy optimization opportunities through intelligent task distribution and resource utilization strategies.

Modern cluster architectures typically employ commodity hardware components to achieve cost-effectiveness while maintaining high performance and energy efficiency. This approach contrasts with traditional supercomputing systems that relied on specialized, expensive hardware. The use of commodity components has democratized access to high-performance computing capabilities and enabled the development of energy-efficient clusters with varying scales and performance characteristics. The transition from sequential to parallel computing was necessitated by physical limitations of processor scaling, commonly referred to as the "power wall" and "memory wall", which constrained the ability to achieve performance improvements through increased clock frequencies alone [3]. These limitations have made energy-aware design and scheduling increasingly critical for achieving sustainable computational scaling.

Cluster Classification and Architectural Design

Cluster systems can be classified along several dimensions based on their architectural characteristics, deployment scenarios, and energy management approaches. Geographic distribution represents one primary classification criterion, distinguishing between tightly coupled clusters where nodes are located close within a single facility, and loosely coupled distributed systems where nodes may be geographically dispersed across different locations. Tightly coupled clusters, also referred to as multi-computer systems, typically reside within a single data center or computational facility. These systems benefit from high-bandwidth, low-latency interconnects such as InfiniBand or high-speed Ethernet, enabling efficient communication between nodes while allowing for centralized energy management strategies. The physical proximity of nodes allows for coordinated power and cooling infrastructure, simplifying both system administration and energy optimization efforts.

Distributed computing systems span multiple geographic locations and must contend with the inherent challenges of wide-area networking, including higher latency, lower bandwidth, and potential network partitions. These systems often employ hierarchical architectures where local clusters at each site are interconnected through wide-area networks, requiring sophisticated energy management strategies that account for both local and global optimization objectives. The choice between tightly coupled and distributed architectures depends on several factors, including computational workload characteristics, available network infrastructure, administrative requirements, cost considerations, and energy efficiency objectives. Applications with frequent inter-node communication and tight synchronization requirements generally perform better on tightly coupled clusters with centralized energy management. In contrast, embarrassingly parallel applications may be suitable for distributed execution with decentralized energy optimization.

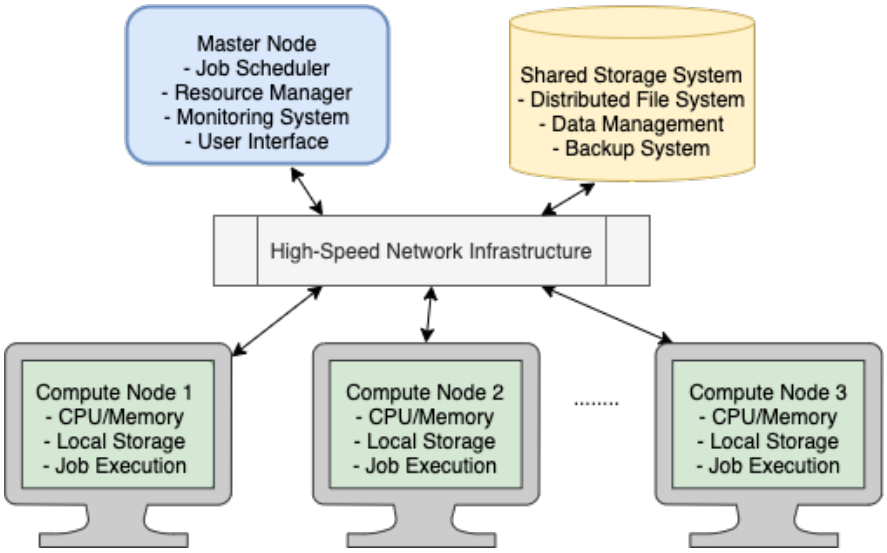


Figure 2.2: Distributed Compute Cluster Architecture

Figure 2.2 presents a distributed compute cluster system comprising nodes, networking infrastructure, and shared storage systems with integrated energy management capabilities. The master node functions as the cluster's control center, orchestrating job scheduling, resource allocation, energy monitoring, and system optimization, while compute nodes perform the actual computational tasks under energy-aware management policies. Data management within the cluster operates through distributed file systems and local storage, with energy-aware data placement and caching strategies to minimize data movement costs. This hierarchical arrangement enables efficient parallel processing of complex scientific workflows while maintaining energy efficiency through intelligent task distribution and resource management.

Modern computing clusters often incorporate heterogeneous specialized hardware, such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs), to accommodate diverse computational requirements while optimizing energy efficiency for specific workload types. This flexibility makes compute clusters essential infrastructure for a wide range of scientific applications, providing the necessary computational power, scalability, and

energy efficiency for advancing scientific research in an environmentally sustainable manner. The integration of energy monitoring and management capabilities throughout the cluster architecture enables sophisticated energy-aware scheduling and optimization strategies that can balance computational performance with energy consumption objectives, making these systems increasingly crucial for sustainable scientific computing initiatives.

Node Architecture and Hardware Optimization

The selection of appropriate hardware for cluster nodes represents a critical design decision that significantly impacts overall system performance, energy consumption, and cost-effectiveness. Each node in a cluster typically comprises standard computer components, including processors, memory, storage devices, and network interfaces. Still, the specific configuration and characteristics of these components must be carefully optimized for both computational requirements and energy efficiency.

Processor selection involves complex trade-offs between computational performance, power consumption, and cost. Modern clusters may employ homogeneous configurations where all nodes utilize identical processors to simplify energy management, or heterogeneous designs that incorporate different processor types optimized for specific computational tasks and energy profiles. The emergence of specialized accelerators such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) has further expanded the design space for energy-efficient cluster node architectures.

Memory subsystem design significantly impacts both cluster performance and energy consumption, particularly for memory-intensive applications. The amount of memory per node, memory bandwidth, and NUMA (Non-Uniform Memory Access) characteristics all influence application performance and energy efficiency. Modern memory technologies offer different trade-offs between capacity, performance, and power consumption that must be considered in energy-aware cluster design. Storage systems must balance capacity, performance, and energy considerations while providing appropriate data access patterns for intended workloads. The choice between different storage technologies, including traditional hard disk drives, solid-state drives, and emerging storage-class memory, significantly impacts both performance and energy consumption characteristics of cluster systems.

Network interface selection and configuration determine the communication capabilities between cluster nodes while contributing to overall system energy consumption. The bandwidth, latency, and message rate characteristics of the network interconnect directly impact both the scalability and energy efficiency of parallel applications. Modern clusters often employ multiple network interfaces to separate different types of traffic, enabling optimized energy management for different communication patterns.

Interconnection Networks and Communication Efficiency

The interconnection network serves as the communication backbone of cluster computing systems, enabling coordination and data exchange between distributed computational tasks while contributing significantly to overall system energy consumption. Network topology, bandwidth characteristics, and latency properties fundamentally determine both the types of applications that can be efficiently executed and the energy consumption patterns of cluster systems.

Network bandwidth represents a primary constraint on cluster performance for communication-intensive applications, while also directly impacting energy consumption through data movement costs. The bandwidth requirements of parallel applications vary significantly based on their communication patterns, with some applications requiring high aggregate bandwidth for bulk data transfers. In contrast, others prioritize low-latency messaging for frequent synchronization operations.

Communication patterns in cluster computing can be broadly categorized into point-to-point, collective, and one-sided communications, each with distinct energy consumption characteristics. Point-to-point communications involve direct message exchange between pairs of nodes and are commonly used for nearest-neighbor interactions in scientific simulations. Collective communications, including broadcast, reduction, and gathering operations, involve coordination among multiple nodes and can be optimized for energy efficiency through intelligent routing and aggregation strategies.

One-sided communications, enabled by Remote Direct Memory Access (RDMA) technologies, allow nodes to directly access memory on remote nodes without involving the remote processor, potentially reducing energy consumption by eliminating unnecessary CPU involvement in data transfers. The efficiency of these communication patterns depends on both the underlying network hardware and the software protocols used for message passing. Modern cluster interconnects employ various optimization techniques, including message aggregation, overlap of communication and computation, and adaptive routing algorithms to minimize both communication overhead and energy consumption while maximizing application performance.

Scalability and Energy Efficiency Challenges

Cluster computing systems must address fundamental scalability challenges that arise as the number of nodes increases, with energy consumption becoming an increasingly critical constraint alongside traditional performance metrics. These challenges include increased communication overhead, load balancing difficulties, fault tolerance requirements, and energy management complexity.

Communication overhead typically increases with cluster size due to the growing number of potential communication paths and increased likelihood of network contention, directly impacting both performance and energy consumption. Applications with high communication-to-computation ratios may experience diminishing returns as cluster size increases, particularly if communication patterns do not scale efficiently with the number of nodes or if energy costs of communication exceed computational benefits. Load balancing becomes increasingly challenging in large clusters due to heterogeneity of node performance, varying computational requirements of different tasks, and dynamic changes in system state, including thermal conditions and power constraints. Effective load balancing strategies must account for both static factors, such as node capabilities, and dynamic factors, such as current system utilization, energy consumption, and thermal state.

Fault tolerance requirements become more stringent as cluster size increases, since the probability of node or network failures grows with the number of components. Energy-aware fault tolerance strategies must balance the energy costs of redundancy and checkpointing against the energy costs of failure recovery and restart operations. System management complexity increases significantly with cluster size, requiring sophisticated tools and procedures for software deployment, configuration management, monitoring, and energy optimization. The administrative overhead of managing large clusters can become a significant factor in the total cost of ownership and operational efficiency, particularly when energy management requirements are considered.

2.3 Energy Monitoring in Computing Systems

Accurate measurement and monitoring of energy consumption represents the foundational requirement for effective energy-aware scheduling, necessitating sophisticated instrumentation and measurement infrastructure that can provide real-time visibility into the power consumption characteristics of cluster systems [64, 65]. The complexity of modern heterogeneous computing environments, with their diverse hardware components, dynamic workload patterns, and multi-level system hierarchies, requires comprehensive monitoring approaches that can capture energy consumption at multiple granularities and time scales [63, 66]. Energy-aware scheduling systems depend critically on the availability of accurate, timely, and actionable energy consumption data to make informed optimization decisions that balance computational performance with energy efficiency objectives [16, 67].

2.3.1 Hardware-Based Monitoring Solutions

Hardware-based energy monitoring solutions provide the most accurate measurements of actual power consumption by directly measuring electrical parameters at various points in

the power distribution hierarchy [94, 93]. These solutions offer the highest fidelity energy consumption data but require dedicated measurement hardware and integration with existing cluster infrastructure [92].

Power Distribution Units (PDUs) with integrated measurement capabilities represent the coarsest level of monitoring, providing facility-level or rack-level power consumption data [129]. Modern intelligent PDUs incorporate sophisticated measurement circuits that monitor voltage, current, and power factor at regular intervals, enabling calculation of real power consumption and identification of power quality issues. These devices typically achieve measurement accuracies of 1-3% of reading with sampling rates of 1-10 seconds, providing adequate temporal resolution for facility-level energy management applications [65]. Advanced PDUs provide per-outlet monitoring capabilities, allowing individual measurement of power consumption for each connected device [64]. This granularity enables correlation of energy consumption with specific computing nodes or infrastructure components, supporting more sophisticated energy management strategies. Network connectivity through standardized protocols such as Simple Network Management Protocol (SNMP) enables centralized collection and analysis of power consumption data across large cluster deployments.

Node-level power meters provide more granular monitoring by measuring power consumption at individual server or computing node levels [94]. These devices may be integrated into server power supplies, implemented as external measurement devices inserted between power supplies and electrical outlets, or incorporated into server management subsystems. In-line power meters typically achieve higher accuracy than PDU-based measurements, with typical accuracies of 0.5-1% of reading and sampling rates up to 1000 samples per second, enabling detailed analysis of power consumption dynamics [93]. Processor-integrated power monitoring features have become increasingly sophisticated in modern CPUs and GPUs, providing real-time power consumption information through hardware performance counters and specialized registers [130]. Intel's Running Average Power Limit (RAPL) interface allows access to power consumption data for different processor domains, including CPU cores, integrated graphics, and memory controllers [131]. NVIDIA's NVIDIA Management Library (NVML) provides similar capabilities for GPU power monitoring, enabling fine-grained tracking of accelerator power consumption with sampling rates up to several hundred hertz.

2.3.2 Software-Based Monitoring and Estimation

Software-based monitoring approaches leverage hardware performance counters, analytical power models, and estimation techniques to provide energy consumption information without requiring dedicated independent measurement hardware [68, 106]. These approaches are particularly valuable in environments where hardware-based monitoring is not available or where the cost and complexity of deploying measurement infrastructure is prohibitive [92].

Hardware performance counters, available in most modern processors, provide indirect indicators of power consumption through metrics such as instruction execution rates, cache miss rates, memory access patterns, and functional unit utilization [130]. These counters can be accessed through standardized interfaces such as the Performance Application Programming Interface (PAPI) or Linux perf subsystem, enabling development of sophisticated power estimation models based on dynamic performance characteristics [132]. Power modeling approaches use statistical techniques, machine learning algorithms, or analytical models to correlate performance counter values with actual power consumption measurements obtained from hardware-based monitoring systems [133]. These models can then be deployed to estimate power consumption in real-time based on performance counter readings, providing power consumption estimates with typical accuracies of 5-15% compared to hardware measurements, depending on the sophistication of the modeling approach and the stability of the workload characteristics [68].

Dynamic Voltage and Frequency Scaling (DVFS) state monitoring provides another source of power consumption information, as processor power consumption varies predictably with operating frequency and voltage according to well-established relationships [38, 39]. Software monitoring tools can track DVFS state transitions and use processor-specific power models to estimate power consumption based on current operating parameters, enabling power estimation with minimal measurement overhead [134]. Temperature-based power estimation techniques leverage the strong correlation between processor temperature and power consumption to provide power estimates based on thermal sensor readings [131]. These approaches can be particularly effective for GPU and accelerator monitoring, where thermal sensors may be more readily accessible than dedicated power measurement interfaces.

2.3.3 Hybrid Monitoring Approaches

Hybrid monitoring approaches combine hardware-based measurements with software-based estimation techniques to provide comprehensive energy monitoring coverage while optimizing the trade-offs between accuracy, cost, and deployment complexity [64, 92]. These approaches typically use hardware-based measurements to calibrate and validate software-based estimation models, enabling accurate power consumption estimates across larger portions of the cluster infrastructure [65].

Hierarchical monitoring architectures implement different monitoring approaches at different levels of the system hierarchy, using high-accuracy hardware-based monitoring at critical points while employing software-based estimation for comprehensive coverage [63]. For example, facility-level and rack-level measurements may be performed using PDU-based monitoring. At the same time, node-level and component-level estimates are derived from performance counter-based models calibrated against periodic hardware measurements [66]. Adaptive monitoring approaches dynamically adjust monitoring frequency and granularity based on system conditions, workload

characteristics, and energy management requirements [67]. During periods of stable operation, monitoring frequency may be reduced to minimize overhead, while periods of dynamic workload changes or energy optimization activities may trigger increased monitoring resolution.

2.3.4 Monitoring Overhead and Performance Impact

The overhead associated with energy monitoring systems can impact overall cluster performance and must be carefully considered when designing monitoring infrastructure for production computing environments [92, 93]. Monitoring overhead includes both the computational resources required for data collection and processing, as well as the network bandwidth consumed by monitoring data transmission and the storage resources needed for data archival [106]. Hardware-based monitoring generally imposes minimal direct overhead on the monitored systems, as the measurement functions are performed by dedicated hardware components that operate independently of the computational workloads [94]. However, the collection, processing, and transmission of monitoring data can consume significant computational and network resources, particularly for high-frequency monitoring of large clusters with thousands of nodes [64].

Software-based monitoring approaches typically impose higher overhead on monitored systems, as they require periodic execution of monitoring software that competes with production workloads for computational resources [130]. The frequency of monitoring data collection must be carefully balanced against the computational overhead to avoid significant performance degradation of scientific applications [132]. Network bandwidth consumption for monitoring data transmission can become substantial in large clusters with high-frequency monitoring, particularly when monitoring data must be transmitted over shared network infrastructure used for application communication [63]. Efficient data compression, intelligent aggregation, and selective filtering techniques can help reduce network overhead while maintaining adequate monitoring resolution for energy management applications [92].

The design of energy monitoring systems must carefully consider these trade-offs between monitoring fidelity, system overhead, and practical deployment constraints to ensure that monitoring infrastructure enhances rather than degrades overall system effectiveness for energy-aware scientific computing applications [16, 37].

2.4 Scientific Workflow Execution and Management

A scientific workflow is defined as a series of interconnected computational tasks executed in a specific structure to achieve a particular research objective [4]. The fundamental challenge lies in the observed trade-off between runtime performance and energy consumption across

multiple workflow types [92], where utilizing additional cluster resources often results in minimal computational improvements while incurring substantial energy overheads. Workflow engines execute these workflows while managing data flow, task dependencies, logging, and reporting [11, 12, 13], but most existing systems provide limited visibility into the environmental impact of computational decisions, making it difficult for scientists to optimize their workflows for both performance and energy efficiency. Understanding the characteristics and energy implications of different workflow execution approaches is essential for developing effective energy-aware scheduling strategies that can optimize both computational performance and energy consumption in realistic scientific computing scenarios.

2.4.1 Workflow Models and Composition

Scientific workflows are typically represented as directed acyclic graphs (DAGs) that capture the computational tasks and their dependencies, with different workflow models offering varying levels of expressiveness and energy optimization opportunities. The representation and modeling choices significantly impact the ability to implement energy-aware scheduling strategies and optimization techniques.

Workflow Representation and Structure

Scientific workflows are commonly described as directed acyclic graphs where vertices represent computational tasks and edges represent data or control dependencies [21]. A task within a workflow constitutes the smallest divisible unit of work [135], typically implemented as executable programs that perform computation on input data to generate output data. Tasks are blocked until required data becomes available or all preceding tasks are complete, creating opportunities for energy-aware scheduling optimization through intelligent task ordering and resource allocation.

The data processed by scientific workflows comprises structured, unstructured, binary, and text-based data structures [135], with varying computational and energy requirements for different data types and processing operations. Abstract workflow representations focus on the logical structure of computations without specifying implementation details, enabling portability across different execution environments while facilitating energy-aware optimization at the scheduling level. Concrete workflow representations include detailed specifications of computational requirements, resource constraints, and execution parameters that enable optimized scheduling and resource allocation. These detailed specifications provide the foundation for energy-aware scheduling systems to make informed decisions about task placement, resource allocation, and execution strategies based on both performance and energy consumption characteristics.

Workflow Design and Composition Considerations

The design and composition of scientific workflows require careful consideration of multiple factors, including computational requirements, data dependencies, error handling, resource constraints, and energy efficiency objectives. Effective workflow design begins with a thorough analysis of the scientific problem and the identification of distinct computational tasks that can be executed independently or with well-defined dependencies, maximizing opportunities for parallelization and energy optimization. Data flow design constitutes a critical aspect of workflow composition, as scientific workflows often involve processing large datasets through multiple computational stages with significant energy implications for data movement and storage. The workflow designer must carefully consider data formats, storage requirements, and transfer mechanisms to ensure efficient data movement between workflow tasks while minimizing energy consumption associated with data transfer operations.

Parameter management and configuration control enable workflows to be adapted to different scientific problems or computational environments while maintaining energy efficiency characteristics. Well-designed workflows provide clear interfaces for specifying input parameters, computational options, and output requirements, enabling energy-aware scheduling systems to optimize execution strategies based on specific workflow configurations and energy objectives. Error handling and fault tolerance represent significant challenges in workflow design, particularly for long-running workflows that may execute for extended periods. Energy-aware fault tolerance strategies must balance the energy costs of redundancy, checkpointing, and error recovery mechanisms against the energy costs of workflow restart operations, requiring sophisticated optimization approaches that consider both reliability and energy efficiency objectives.

2.4.2 Workflow Management Systems and Scheduling

Workflow Management Systems (WMS) serve as critical middleware that bridges the gap between complex scientific applications and distributed computing resources while providing opportunities for energy-aware optimization through intelligent orchestration of computational tasks, dependency management, and resource allocation strategies.

Workflow Management System Architecture and Capabilities

Modern workflow management systems provide comprehensive platforms for workflow specification, optimization, execution, and monitoring in distributed computing environments, with increasing support for energy-aware optimization features. These systems automate the orchestration of computational tasks, managing dependencies, data movement, and resource allocation while abstracting underlying computational infrastructure complexities from scientists.

The complexity of scientific workflows has necessitated sophisticated workflow management systems that can handle orchestration, execution, and monitoring of workflow-based computations. These systems provide essential services including task scheduling, resource allocation, data management, and error recovery, while offering opportunities for integration with energy-aware scheduling algorithms and optimization strategies.

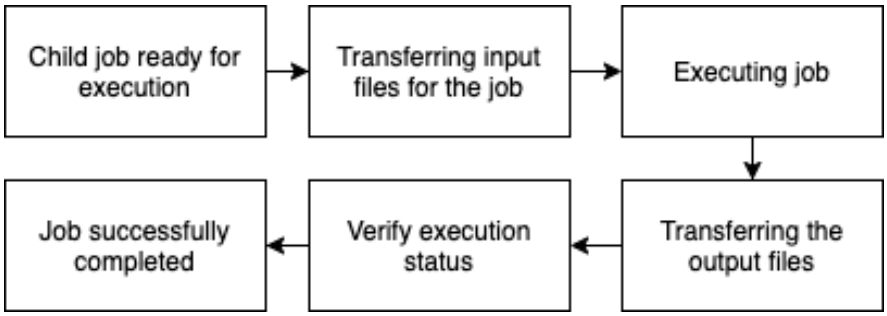


Figure 2.3: Life cycle of job execution in workflow management systems

Figure 2.3 presents the typical life cycle for each job in a workflow management system. The WMS ensures that job dependencies are satisfied, input data is available, and appropriate resources are allocated before scheduling job execution. This lifecycle provides multiple opportunities for energy-aware optimization, including intelligent data staging, resource selection based on energy efficiency characteristics, and coordination with cluster-wide energy management strategies.

Notable Workflow Management Systems

Several production-grade workflow management systems have emerged to address the diverse requirements of scientific computing while providing varying levels of support for energy-aware optimization:

- **Pegasus WMS**, developed by USC/ ISI, represents a leading workflow management system that provides sophisticated mapping and execution capabilities for large-scale scientific workflows [21]. It specializes in mapping abstract workflows to execution environments. Pegasus offers robust data management, fault tolerance, and performance optimization features, with an architecture that supports integration of energy-aware scheduling strategies [21]. The system provides comprehensive execution logging that enables detailed analysis of workflow execution patterns and energy consumption characteristics.

- **Apache Airflow** provides Python-based workflow authoring capabilities with strong scheduling features and extensive operator ecosystems [136]. Initially developed by Airbnb [136], Airflow offers flexibility for implementing custom energy-aware scheduling logic and integration with external energy monitoring systems.

- **Kepler** offers a rich graphical interface and an extensive library of reusable workflow components, focusing on scientific workflow composition and execution. The system provides opportunities for energy-aware optimization through component-level resource specification and execution monitoring capabilities.

- **Snakemake** has gained popularity in bioinformatics due to its Python-based workflow definition and built-in support for conda environments [137]. The system provides automatic parallelization capabilities that can be enhanced with energy-aware scheduling strategies [137].

The choice of workflow management system significantly impacts the ability to implement energy-aware scheduling strategies, as different systems provide varying levels of extensibility, monitoring capabilities, and integration opportunities with external energy management systems.

Advanced Workflow Management Features

Modern workflow management systems incorporate several advanced capabilities that can be leveraged for energy-aware optimization:

- **Workflow Optimization** techniques include task clustering for better resource utilization, data locality-aware scheduling, and resource usage prediction capabilities that can be enhanced with energy consumption modeling and optimization objectives.

- **Monitoring and Provenance** systems provide real-time workflow monitoring, detailed execution logs, and scientific provenance tracking that enable correlation of execution patterns with energy consumption data for optimization purposes.

- **Fault Tolerance** mechanisms include automatic job retry, alternative resource selection, and checkpoint/restart capabilities that must be balanced against energy efficiency objectives in energy-aware scheduling implementations.

2.4.3 Resource Schedulers and Computing Infrastructure

The integration of workflow management systems with underlying resource schedulers and computing infrastructure determines the effectiveness of workflow execution in distributed environments while providing the foundation for implementing energy-aware scheduling strategies and optimization techniques.

Resource Scheduler Architecture and Functionality

Resource schedulers serve as critical middleware components in distributed computing environments, bridging the gap between complex scientific workflows represented as directed acyclic graphs and the underlying computational infrastructure of clusters, grids, and clouds. These systems implement sophisticated scheduling algorithms to efficiently map tasks onto available resources while managing dependencies, data movement, and resource allocation.

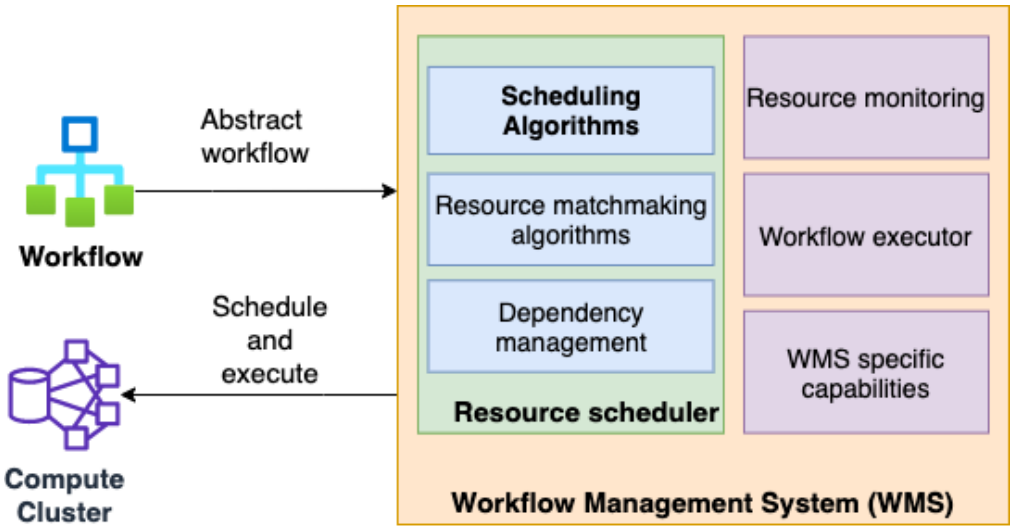


Figure 2.4: Integration between Scientific Workflow and Workflow Management Systems

Figure 2.4 provides a high-level view of the workflow execution lifecycle, showing how abstract workflows are submitted to workflow management systems, which then orchestrate, schedule, and execute them on compute clusters through resource schedulers. This architecture provides multiple intervention points for energy-aware optimization strategies.

Prominent Resource Scheduling Systems

Several widely used resource schedulers offer distinct approaches to resource management with varying degrees of support for energy-aware optimization:

- **HTCondor** [138] employs a matchmaking mechanism through its ClassAd system [128], enabling precise task-to-resource matching based on detailed specifications. HTCondor's Directed Acyclic Graph Manager (DAGMan) serves as a meta-scheduler that supervises workflow execution while handling dependencies between jobs [28]. The system's flexible matchmaking architecture provides opportunities for incorporating energy efficiency criteria into resource selection decisions.
- **SLURM Workload Manager** [19] focuses on highly scalable scheduling for large-scale clusters through backfill scheduling algorithms [107]. SLURM's architecture incorporates

mechanisms for handling node failures, power management, and quality of service guarantees, making it particularly suitable for energy-aware optimization in large-scale scientific computing facilities.

- **PBS (Portable Batch System)** [139] and its variants implement sophisticated queue-based scheduling mechanisms where different queues can be configured with distinct scheduling policies, resource limits, and access controls. This approach allows organizations to implement complex scheduling policies that reflect energy efficiency requirements while maintaining fair resource allocation.
- **Kubernetes** represents a specialized scheduler for containerized workloads that handles both task scheduling and the entire lifecycle management of containerized applications. Kubernetes implements a sophisticated scheduling framework considering resource requirements, node affinity, and quality of service requirements, providing opportunities for integration with energy-aware scheduling strategies.

Scheduling Algorithm Categories and Energy Implications

Task scheduling in distributed computing environments represents a critical component that significantly impacts both workflow execution performance and energy consumption. The fundamental goal of scheduling algorithms is to efficiently map tasks onto available computational resources while optimizing various objectives such as minimizing makespan, reducing costs, improving energy efficiency, and maximizing resource utilization. Different categories of scheduling algorithms offer varying optimization objectives and energy efficiency characteristics:

- **List-Based Scheduling Algorithms** such as the Heterogeneous Earliest Finish Time (HEFT) algorithm represent widely adopted approaches for workflow scheduling [140, 141]. HEFT operates through task prioritization and processor selection phases, with opportunities for incorporating energy efficiency criteria into both scheduling decisions.
- **Critical Path Scheduling** algorithms identify and prioritize tasks on the workflow's critical path, which represents the longest dependency chain determining minimum execution time. These algorithms can be enhanced with energy-aware optimization by considering both execution time and energy consumption in critical path analysis.
- **Priority-Based Scheduling** algorithms assign different priority levels to tasks based on various metrics, including deadline constraints, resource requirements, or user-defined importance. Energy efficiency criteria can be incorporated into priority calculations to balance performance and energy objectives.
- **Queue-Based Scheduling** algorithms, such as the First Come First Serve (FCFS), represent the most straightforward scheduling approach, where tasks are executed in the

order they arrive in the scheduling queue. While FCFS provides fairness and predictability, it may lead to poor resource utilization and increased makespan, especially when long-running tasks block shorter ones. Round Robin scheduling builds upon FCFS by allocating a fixed time quantum to each task in a circular manner. When a task's quantum expires, it is moved to the back of the queue, and the next task receives processor time. This approach ensures fair resource distribution and prevents task starvation.

- **Energy-Aware Scheduling** algorithms specifically incorporate energy efficiency considerations into scheduling decisions, monitoring and predicting energy consumption patterns while making scheduling decisions that balance performance requirements with energy conservation goals [35, 142, 143]. Modern scheduling algorithms increasingly incorporate energy efficiency considerations. These algorithms monitor and predict patterns, making scheduling decisions that balance performance requirements with energy conservation goals.

- **Cost-Aware Scheduling** algorithms focus on optimizing workflow execution within cost constraints, particularly relevant in cloud computing environments. These algorithms consider resource pricing models, data transfer costs, and storage expenses while making scheduling decisions. They often implement trade-off mechanisms between execution time and monetary cost, allowing users to specify their preferences regarding these competing objectives.

- **Adaptive scheduling** algorithms dynamically adjust their scheduling decisions based on real-time system conditions and workflow behavior. These algorithms monitor various metrics such as resource utilization, task completion times, and system load, modifying their scheduling policies accordingly. Machine learning techniques are increasingly being incorporated to predict resource requirements and optimize scheduling decisions based on historical execution data.

- **Hybrid and Multi-Objective Scheduling** algorithms often implement hybrid scheduling approaches that combine multiple algorithms to address diverse requirements. These hybrid schedulers may switch between different scheduling policies based on workflow characteristics, system conditions, or user preferences. Multi-objective scheduling algorithms simultaneously optimize multiple competing objectives, using techniques such as Pareto optimization to find balanced solutions that satisfy various performance, cost, and reliability requirements.

Case Study on how different Scheduling Algorithms affect the scheduling of tasks

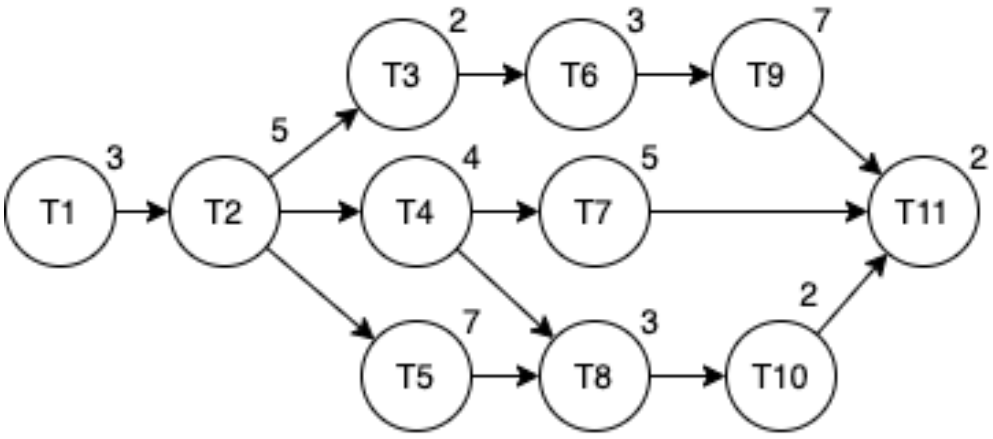


Figure 2.5: Example Workflow with Task Complexities

Figure 2.5 presents a workflow where nodes represent tasks with different computational complexities. Various scheduling algorithms prioritize different goals and use different approaches when processing such workflows. The Heterogeneous Earliest Finish Time (HEFT) algorithm is one of the most widely adopted list scheduling algorithms for workflow scheduling [140, 141]. HEFT operates in two primary phases: task prioritization and processor selection. HEFT's effectiveness stems from its ability to consider both task dependencies and resource heterogeneity. Calculating the upward rank and making sure that the dependencies are met, the scheduling sequence for the provided DAG via the HEFT algorithm will be:

$$T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T6 \rightarrow T9 \rightarrow T7 \rightarrow T8 \rightarrow T10 \rightarrow T11$$

Critical Path scheduling algorithms identify and prioritize tasks on the workflow's critical path, which is the longest dependency chain [144]. The scheduling sequence for the DAG provided in Figure 2.5 when using a critical path algorithm will be:

$$T1 \rightarrow T2 \rightarrow T3 \rightarrow T6 \rightarrow T9 \rightarrow T5 \rightarrow T4 \rightarrow T8 \rightarrow T10 \rightarrow T7 \rightarrow T11$$

First Come First Serve (FCFS) represents the most straightforward scheduling approach, where tasks are executed in the order they arrive in the scheduling queue. While FCFS provides fairness and predictability, it may lead to poor resource utilization and increased makespan, especially when long-running tasks block shorter ones. In the above scenario, the sequence will be:

$$T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T6 \rightarrow T7 \rightarrow T8 \rightarrow T9 \rightarrow T10 \rightarrow T11$$

Beyond these basic approaches, many other complex and characteristic-specific algorithms have been developed, such as cost-aware scheduling algorithms [125, 145, 146, 147, 53], energy-aware scheduling algorithms [35, 142, 143, 140, 148, 149, 126], and fault-tolerant scheduling algorithms [150, 151]. Other complex algorithms include adaptive scheduling algorithms [33, 23, 61, 59, 152], and Multi-Objective Scheduling Algorithms [153, 45, 154, 155].

Computing Libraries and Frameworks

Scientific workflows typically leverage specialized computing libraries and frameworks that provide optimized implementations of standard computational kernels and algorithms. High-performance computing libraries such as BLAS, LAPACK, and FFT implementations offer optimized algorithms for numerical computation with different performance and energy characteristics. Domain-specific frameworks such as bioinformatics toolkits, climate modeling libraries, and astronomy processing pipelines provide pre-built components that can be integrated into workflows. The energy efficiency of these libraries and frameworks significantly impacts the overall energy consumption of workflows that utilize them.

All of the previous software is used to set up the cluster and the computing processes [105]. Apart from these, individual libraries are required, which are dependent on the type of computing the cluster is doing. Computing libraries in the domain of vector and matrix operations include the Basic Linear Algebra Sub-Programs (*BLAS*), Automatically tuned Linear Algebra Software (*ATLAS*), *OpenBLAS* and Linear Algebra PACKage (*LA-PACK*). LA-PACK is heavily used by High-Performance Linpack (*HPL*) benchmark to calculate the efficiency of clusters [94, 156, 157]. Intel developed its own library called Math Kernel Library (*MKL*), which is heavily optimized to work alongside Intel processors to provide the best possible performance and efficiency. In the domain of image processing and graphic processing, the Open-CL framework is used to execute programs on GPUs and other digital signal processors [94]. Other commonly used libraries for computing are SciPy, NumPy, and Sklearn.

2.4.4 Autonomic Computing

Autonomic computing provides the theoretical and practical foundation for implementing adaptive energy-aware scheduling systems that can automatically adjust their optimization strategies based on changing system conditions and workflow characteristics. The Monitor-Analyze-Plan-Execute (MAPE) model provides a systematic approach for implementing self-managing systems that can optimize energy consumption while maintaining performance objectives [158].

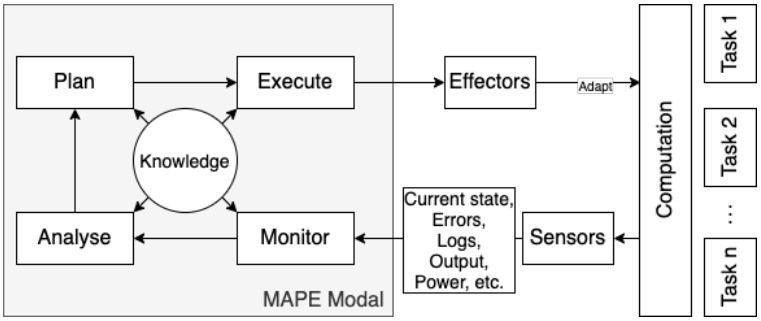


Figure 2.6: MAPE Model for Autonomic Computing

Figure 2.6 illustrates the MAPE model structure comprising monitoring, analysis, planning, and execution phases that operate in a continuous loop. This model provides the foundation for implementing adaptive energy-aware scheduling systems that can continuously optimize energy consumption based on real-time system conditions and workflow characteristics.

The **Monitor** component uses sensors to collect data from managed resources, including energy consumption data, performance metrics, and system state information. The **Analyze** component processes monitoring data to identify optimization opportunities, bottlenecks, and potential energy efficiency improvements. The **Plan** component develops specific adaptation strategies based on analysis results and predefined policies, while the **Execute** component implements planned adaptations through effectors that modify system behavior.

2.4.5 Optimization techniques for Scientific Workflow Execution

Optimization techniques are used to find the optimal set of parameters for a model that minimizes the error or maximizes the performance. It is essential to understand the different optimization techniques available to choose the most appropriate method for a specific problem or dataset. In this Section, a survey of various high-level optimization concepts and their examples is presented. The purpose of this survey is to provide an overview of different optimization techniques that are commonly used.

Discrete Optimizations

Discrete optimization is the process of finding the optimal solution from a finite set of discrete options. The following optimizations have been used to address the concerns of scientific workflow execution.

- **Linear Programming:** This is a method used to optimize a linear objective function, subject to constraints represented by linear equations or inequalities [50].
- **Simplex Method:** This is a popular algorithm for solving linear programming problems, which iteratively improves a solution by moving to adjacent vertices of the feasible region [159].
- **Interior-Point Methods:** This is a family of algorithms that solve linear programming problems by finding a solution that is close to the central point of the feasible region [160].
- **Gradient Descent:** This is an optimization algorithm that iteratively improves a solution by moving in the direction of the negative gradient of the objective function [161].

Metaheuristics Optimization

Metaheuristics optimization is a method of solving complex problems by using high-level procedures, or heuristics, that guide the search for an approximate solution.

- **Genetic Algorithm:** A genetic algorithm (GA) is a type of optimization algorithm inspired by natural selection. It uses techniques of reproduction, mutation, and selection to find the best solution for a given problem [162].
- **Grasshopper Algorithm:** The Grasshopper algorithm (GHA) is an optimization algorithm that uses the concept of swarm intelligence and the movement of grasshoppers inspires it. It uses a population of potential solutions that move randomly, and then converge to the optimal solution through a process of selection and imitation [163].
- **Particle Swarm Optimization (PSO):** Particle Swarm Optimization (PSO) is a population-based method that is modeled after the collective behavior of birds and fish, such as flocking and schooling. It uses a population of particles that move in the search space to find the optimal solution through a process of exploration and exploitation [164].
- **Cuckoo Search (CS):** Cuckoo Search (CS) is an optimization algorithm inspired by the behavior of cuckoos. It can be used to find an optimized scheduling solution by iteratively updating the position of each cuckoo in the population based on the energy consumption and performance of the computation [165].

Heuristic Approaches

Heuristic algorithms are a method of solving problems by using practical approaches or rules of thumb, rather than relying on a systematic, theoretical solution. The following are the widely used heuristics methods used for scientific workflow execution.

- **Hill Climbing:** Hill climbing is an optimization method that begins with an initial solution and gradually makes small adjustments to it to enhance the solution. This technique is commonly used to locate a nearby optimal solution [166].
- **Greedy Algorithm:** A greedy algorithm constructs a solution piece by piece, always opting for the next piece that provides the most evident and immediate benefit [167].
- **Beam Search:** Beam search is a method of searching for solutions that uses heuristics to examine a small subset of the most likely solutions at each step, rather than exploring all potential solutions [168].

- **Randomized Local Search:** Randomized Local Search is a heuristic optimization algorithm that starts with a random initial solution and iteratively applies random changes to it to find an improved solution [169].

Meta-heuristic Optimizations include population-based algorithms and are powerful optimization techniques, but do not guarantee finding the global optimum. This indicates that the achieved policy or output might not translate or always work optimally with other workflow executions. To mitigate this problem, some variants such as the Artificial Bee Algorithm (ABA) have been developed to find the global optimum [170]. These variants can be used to find generalized scheduling techniques that might work with different computations and resources.

Machine Learning

Machine learning optimization is the process of finding the best set of parameters for a model to minimize its error on a given dataset.

- **Reinforcement Learning:** Reinforcement learning can optimize scheduling decisions by gaining insight from the system's feedback. By analyzing the energy consumption and performance of the computation, it can continuously improve the scheduling decisions over time [171, 77].
- **Q-learning:** Q-learning is a form of reinforcement learning that can be utilized to identify the best scheduling decisions by studying the energy consumption and performance of the computation [171, 77].
- **Neural Networks:** Neural networks can be utilized to depict the correlation between scheduling decisions and energy consumption and performance of computation. They can be trained on past data to anticipate energy consumption and performance of new scheduling decisions [171, 77].
- **Decision Trees:** Decision trees can be used to depict the association between scheduling decisions and energy consumption and the performance of computation. Using historical data can enhance the accuracy of Decision Tree predictions [171, 77].

Multi-Objective optimization

Multi-objective optimization techniques are used to optimize scheduling solutions by taking into account multiple objectives, including energy consumption, accuracy, and performance. These techniques can find a set of optimal solutions that balance the various goals and make trade-offs between them.

- **Non-dominated Sorting Genetic Algorithm (NSGA):** NSGA is a genetic algorithm that is capable of dealing with multiple objectives. It can be used to find optimal solutions with minimal or no trade-offs between the different objectives [48].
- **Multi-objective Particle Swarm Optimization (MOPSO):** MOPSO is a particle swarm optimization algorithm that can handle multiple objectives simultaneously and can be used to identify the trade-offs between energy consumption and the performance of the computation [47].
- **Strength Pareto Evolutionary Algorithm (SPEA):** SPEA is an evolutionary algorithm based on the concept of Pareto Optimality, which states that a solution is considered optimal when no other solution can improve one of its objectives without compromising at least one of the other objectives [172].
- **Multi-objective Cuckoo Search (MOCS):** MOCS combines the Cuckoo Search algorithm with NSGA-II (Non-dominated Sorting Genetic Algorithm II) to handle multiple objectives in optimization [173].

It's important to note that these algorithms depend on multiple factors such as the characteristics of the problem, the available resources, and the desired level of accuracy and computational time. Some algorithms may work better for certain types of issues or settings than others, and it may be necessary to experiment with different algorithms to find the best solution.

2.4.6 Representative Scientific Workflows

Understanding the characteristics and energy consumption patterns of real-world scientific workflows is essential for developing and evaluating energy-aware scheduling strategies that can effectively optimize diverse computational workloads across different scientific domains.

Montage Astrophotography Workflow

Montage represents a widely-used software toolkit in astrophotography for combining Flexible Image Transport System (FITS) images of the sky into composite mosaics [29, 1]. A montage is a series of separate images that are combined to obtain a continuous sequence. Astrophotography makes use of this concept to create high-definition mosaics of different astronomical objects such as stars, and planets. Highly detailed images of small portions of the sky are obtained and combined to create mosaics of the sky. The toolkit preserves calibration and positional fidelity of original input images while providing a representative example of I/O-bound scientific workflows with complex dependency structures [64].

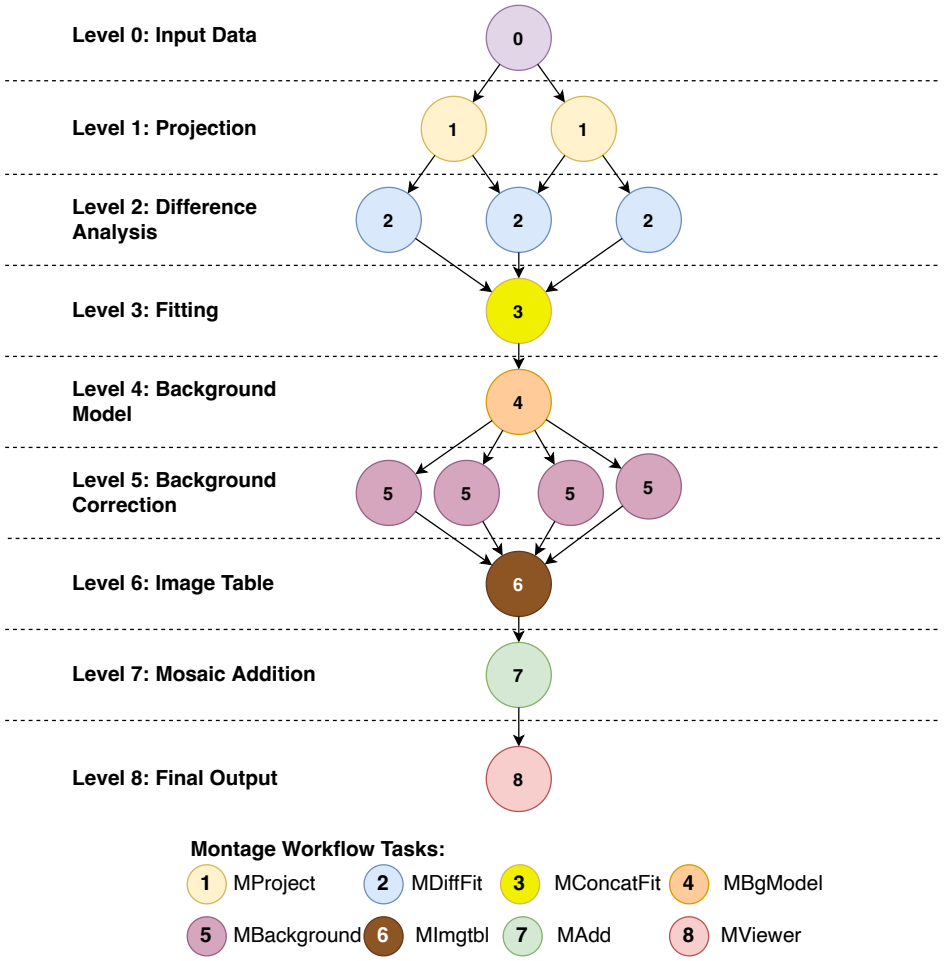


Figure 2.7: Montage workflow structure showing eight levels of task dependencies

Figure 2.7 illustrates the Montage workflow as a directed acyclic graph with eight distinct levels of jobs that exhibit dependencies from prior levels. The workflow structure includes computationally intensive tasks such as `mProject` operations alongside less intensive data management and coordination tasks, providing opportunities for energy-aware scheduling optimization through intelligent task placement and resource allocation.

Montage has been classified as an input/output-bound workflow [18] compared to other scientific workflows. Still, it exhibits varied internal complexity with different job types having different requirements for I/O, memory, and CPU resources. This heterogeneity provides an excellent testbed for evaluating energy-aware scheduling strategies across diverse computational patterns.

The Montage toolkit enables the generation of workflows with varying sizes depending on scientific requirements, specified by sky coverage area (in degrees) and color channels for final image generation. The varying size of a Montage workflow is defined in (i) *degrees* of the sky and (ii) the color channels from which the final images should be generated. Table 2.2 illustrates the number of jobs for different sizes of the Montage workflow, showing how computational requirements and energy consumption scale with problem size. The workflows listed are 0.5,

Montage Job	Workflow Size			
	0.5 Deg.	1 Deg.	1.5 Deg.	2 Deg.
mProject	12	48	108	192
mDiffFit	18	360	1,890	6,048
mConcatFit	3	3	3	3
mBgmodel	3	3	3	3
mBackground	12	48	108	192
mImgtbl	3	3	3	3
mAdd	3	3	3	3
mViewer	4	4	4	4
Total jobs	**58**	**472**	**2,122**	**6,448**

Table 2.2: Montage workflow size and number of jobs

1.0, 1.5, and 2 degrees; all containing three color channels of RGB. The scaling characteristics shown in Table 2.2 demonstrate how specific jobs remain constant regardless of workflow size. In contrast, others scale significantly, providing different optimization opportunities for energy-aware scheduling systems that can exploit these patterns.

Bioinformatics Genomic Analysis Workflow

The Bioinformatics workflow analyzes genomic data from the 1,000 Genomes Project [174, 2], focusing on mutation detection and cross-matching analysis for disease-related research. This workflow represents compute-intensive scientific computing with distinct bottleneck characteristics that create unique energy optimization opportunities. The purpose of this workflow is to analyze the data and cross-match the whole dataset for mutations. The workflow also identifies mutational overlaps to evaluate potential disease-related mutations. The extracted data, along with the mutation's sift scores (calculated by the Variant Effect Predictor [175]), can help researchers in discovering the exact mutation that is the cause of a specific disease in a person. The workflow also provides a visualization of the data for easier understanding and future analysis.

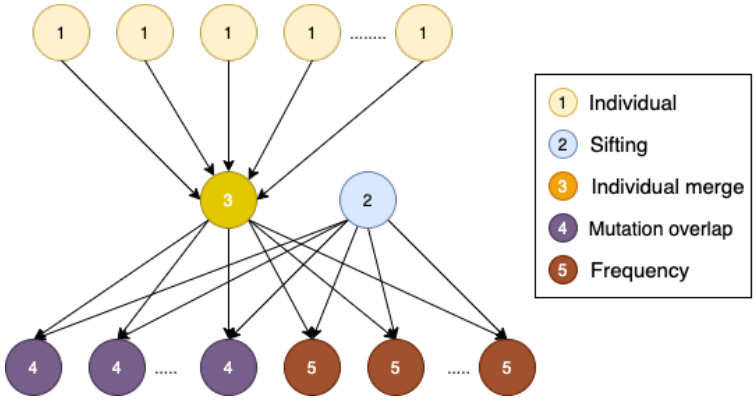


Figure 2.8: Bioinformatics workflow structure showing bottleneck characteristics

Figure 2.8 illustrates the Bioinformatics workflow as a directed acyclic graph with three primary levels of execution. The workflow exhibits a clear bottleneck in the `individual_merge` task that must complete before subsequent processing can proceed, creating opportunities for energy optimization through strategic resource allocation during different execution phases.

The workflow processes data from seven population cohorts to calculate, analyze, and report relationships between individuals and genetic variation. The computational intensity varies significantly across different workflow phases, with parallel `individual` tasks requiring substantial computational resources while the `individual_merge` bottleneck limits parallelization opportunities.

Bioinformatics Job	**Workflow Size**		
	Small (10k)	**Medium (20k)**	**Large (30k)**
`individual`	50	50	50
`sifting`	1	1	1
`individual_merge`	1	1	1
`frequency`	7	7	7
`mutation_overlap`	7	7	7

Table 2.3: Bioinformatics workflow size and number of jobs

Table 2.3 shows that job counts remain relatively constant across different data sizes, with the primary scaling occurring in the computational intensity of individual tasks rather than the number of tasks. This characteristic provides different optimization opportunities compared to the Montage workflow, emphasizing the importance of task-level energy optimization rather than workflow-level parallelization strategies.

Workflow Characteristics and Energy Optimization Opportunities

Both Montage and Bioinformatics workflows exhibit specific characteristics that create distinct opportunities for energy-aware scheduling optimization. Montage demonstrates significant task heterogeneity with computationally intensive `mProject` operations alongside less intensive coordination tasks, enabling energy optimization through intelligent task-to-resource matching based on computational requirements and energy efficiency characteristics. The Bioinformatics workflow presents bottleneck management challenges where parallel computation phases are followed by sequential consolidation operations, creating opportunities for dynamic resource management that can power down unused resources during bottleneck phases while maintaining computational capabilities for parallel execution phases.

Understanding these workflow-specific characteristics is essential for developing energy-aware scheduling strategies that can effectively optimize energy consumption while maintaining

performance requirements across diverse scientific computing applications. The combination of different computational patterns, dependency structures, and scaling characteristics provides a testbed for evaluating the effectiveness of energy-aware scheduling approaches in realistic scientific computing scenarios.

2.5 Related Work in Energy-Aware Computing and Optimization

Energy-aware computing represents a critical research domain that addresses the growing environmental and economic challenges associated with large-scale computational systems. This research area consists of methodologies, algorithms, and systems designed to optimize energy consumption while maintaining acceptable levels of performance and scientific quality. As the scale and complexity of scientific workflows continue to grow, the demand for computational resources has led to increased energy consumption that negatively impacts both operational costs and environmental sustainability. Understanding existing approaches to energy-aware computing provides essential foundations for developing energy-aware scheduling systems that can effectively balance multiple competing objectives in realistic computing environments.

2.5.1 Energy Modeling and Measurement Techniques

Modeling involves using mathematical tools and techniques to represent and analyze real-world situations or systems. This process is essential for understanding, predicting, and optimizing solutions [22]. Minimizing the makespan of a computation has always been a topic of interest for researchers. Many mathematical models have been proposed to determine the time it takes for a calculation to execute on a resource [22, 34, 176, 49]. These models utilize the known speed of resources as the number of machine instructions they can execute per second and approximate the makespan by dividing the length of the task by the speed of the resource [177, 178]. This formulation works very accurately for single-core machines, though it requires additional considerations for multi-core architectures where parallel execution affect performance.

Modern multi-core machines are more complex and require more sophisticated modeling approaches. Time overheads due to caching and context switching between cores increase the complexity of accurately modeling make-span calculations. Attempts to model the makespan required for a computation on heterogeneous environments have been done [179, 83, 180]. These models facilitate the development of more sophisticated scheduling algorithms that can accommodate the complex performance characteristics of modern computing architectures. Modeling energy consumption in computing systems presents significant challenges due to the

absence of standardized measurement approaches and the sensitivity of energy usage to even minor hardware or software modifications [64]. Traditional research approaches have typically considered only two operational states: idle and full load, assuming linear energy consumption patterns between these extremes [181, 182, 92]. However, this simplified model fails to capture the complexity of modern multi-core systems.

Traditional energy modeling approaches have typically considered only two operational states: idle and full load, assuming linear energy consumption patterns between these extremes [181, 182, 92]. However, this simplified model fails to capture the complexity of modern multi-core systems that exhibit non-linear energy consumption characteristics due to advanced caching mechanisms, context switching overhead, and shared resource contention [36]. Modern computing architectures demonstrate diminishing returns in energy consumption as additional cores are utilized, creating sublinear relationships between computational resources and power consumption [183, 184, 75, 185, 37, 64]. This complexity necessitates sophisticated modeling approaches that can capture the nuanced energy characteristics of heterogeneous computing environments. Machine learning approaches have shown promise for developing adaptive energy models that can automatically adjust to different hardware platforms and workload characteristics. These approaches leverage historical execution data to build predictive models for energy consumption that can inform energy-aware scheduling decisions with improved accuracy compared to traditional analytical models.

Modern computing architectures exhibit non-linear energy consumption characteristics due to several factors, including advanced caching mechanisms, context switching overhead, and shared file system operations [36]. Rather than following a linear scaling pattern, energy consumption in multi-core processors typically demonstrates diminishing returns as additional cores are utilized. This phenomenon occurs because activating successive cores results in progressively smaller increases in total system energy consumption, creating a sublinear relationship between computational resources and power draw [183, 184, 75, 185, 37, 64].

2.5.2 Energy-Aware Scheduling Algorithms and Schedulers

There is increasing interest in sustainable and energy efficient computing and recent work focuses on approaches for energy awareness for scheduling tasks on multi-core machines [186] and also identical parallel machines [187]. For larger computation tasks, approaches for energy-aware modeling and workload prediction hope to optimize data centers [188]. For more fine-grained control, there are approaches to optimize virtual machine placement based on energy-efficiency [112]. These demonstrate that there is a desire to analyze and optimize computation for energy in distributed computation environments. For scientific workflow execution in the cloud, energy-aware job management [189] and energy-aware resource allocation [42] focus on trying to take into account energy considerations as well as performance in cloud environments. The focus of this paper is on High-Performance Computing (HPC), and in particular compute clusters.

HPC focuses on achieving the best performance for computationally intensive tasks [92]. Due to this, massive data centers have been established to supply the required computational power to tackle the ever-increasing processing demand. These provide high computation power but also consume a lot of energy and cost a lot to maintain. Energy sustainability is emerging as an issue that needs addressing, and hence there is a need to focus on computing systems that have an energy budget or are energy efficient [94]. To support this, it is important to be able to measure the energy impact of computation, which can be achieved at both hardware and software levels.

An Energy-Efficient Task Offloading (EETO) policy has been developed to schedule and offload real-time IoT applications [190]. The policy makes use of the Lyapunov optimization technique to minimize the queuing of tasks and achieves an energy consumption reduction of about 23.79% as compared to the current methods. Similarly, a dynamic offloading and resource scheduling policy is developed to reduce energy consumption and shorten application completion time [191]. This policy dynamically optimizes the CPU clock frequency and the wireless transmission power to achieve a reduction in the energy-efficiency cost (EEC) of the workflow.

A novel framework has been introduced to provide a comparative analysis of energy-time data of a parallel computation [92]. Scheduling algorithms have also been introduced to meet time deadlines of a computation while minimizing the energy consumption [192, 44]. An energy-efficient scheduling policy for cloud-based distributed computing has been proposed and evaluated [193]. The policies are divided based on the allocation of virtual machines in the cloud, and their performances are compared with the state-of-the-art energy-efficient scheduling policy EnReal [42]. They concluded that their policies perform significantly better than EnReal and achieved a 70% energy reduction. Another study introduced two new task scheduling algorithms called Enhancing Heterogeneous Earliest Finish Time (EHEFT) and Enhancing Critical Path on a Processor ECPOP for shutting down inefficient processors and effectively rescheduling the tasks [194]. Both studies concluded that their scheduling algorithms are effective at minimizing energy consumption at a reasonable expense of execution time.

Scheduling algorithms have also been introduced to meet time deadlines of computation while minimizing the energy consumption [192, 44]. A scheduler based on a polynomial time algorithm is proposed to provide real-time dynamic resource allocation to achieve reasonable solutions in a particular time [195]. Chebyshev scalarization function is used to develop an energy-aware multi-objective reinforcement learning (EnMORL) algorithm to reduce the makespan and energy consumption of a workflow [155]. A task scheduling algorithm that takes into consideration the current energy consumption of the containers aims to handle requests in real-time and schedule the requests in an energy-balanced manner is developed [122]. Scheduling algorithms have also been introduced to meet time deadlines of a computation while minimizing the energy consumption [192, 44]. A scheduler based on a polynomial-time algorithm is proposed to provide real-time dynamic resource allocation to obtain reasonable solutions in a particular time [195]. Chebyshev scalarization function is used to develop an energy-aware multi-objective

reinforcement learning (EnMORL) algorithm to reduce the makespan and energy consumption of a workflow [155]. The Chebyshev scalarization function has been employed to develop an energy-aware multi-objective reinforcement learning (EnMORL) algorithm, aiming to reduce both makespan and energy consumption of workflows [155].

Two novel algorithms called SPSS-EB (Static Provisioning-Static Scheduling under Energy and Budget Constraints) and SPSS-ED (Static Provisioning-Static Scheduling under Energy and Deadline Constraints) were introduced for effective scheduling of resources and tasks to reduce energy consumption in cloud-based distributed computing systems [196]. New task schedulers focusing on the inter-dependency of functions were introduced to achieve similar results [197]. They achieved a 22.7% reduction in energy at no cost to the make span of the algorithm. A bi-objective optimization problem is identified, and a reformulated algorithm is introduced, aiming to reduce the make span and energy consumption of the Multi Objective Heterogeneous Earliest Finish Time (MOHEFT) scheduling algorithm [45]. They achieved an 85% reduction in energy in some cases while achieving a 3.3% reduction in the make span of the workflows by using realistic energy consumption and performance models for task execution.

A scheduler was developed that minimizes data transfer requirements and task interdependencies in edge computing, achieving a 22.7% reduction in computational costs while maintaining comparable performance levels [191, 103]. Another scheduling heuristic was created to reduce energy consumption in cloud computations executed on Virtual Machines and external systems [140]. An energy-aware task scheduling approach has been developed that dynamically adjusts the voltage and frequency of edge compute resources to minimize computational energy consumption [198].

2.5.3 Scheduling Techniques for Energy-Aware Execution

Recent research in energy-aware scheduling has developed sophisticated approaches that integrate energy optimization with traditional performance objectives. Table 2.4 summarizes key research developments in energy-aware scheduling, highlighting the diversity of approaches and techniques employed.

Task/resource consolidation in scheduling involves grouping similar tasks or resources to improve efficiency. This approach reduces setup times, minimizes resource transitions, and lowers operational costs. [40, 134, 67, 87] makes use of task consolidation to reduce the number of parallel tasks to be executed or group smaller tasks together to avoid the overhead expense of implementing them individually. [199, 41] consolidates different resources (such as virtual machines) to meet the requirements for a particular task.

Scheduling Approach	Task/Resource Consolidation	Dynamic Voltage Frequency Scaling	Regression Analysis	Adaptive Scheduling	User Preference	Multi-User Awareness	Multi Objective	Machine Learning Mechanisms
Srikantaiah et al. [40]	✓							
EASVMC [199]	✓						✓	
MAMFO [41]	✓						✓	
Yassa et al. [46]		✓					✓	
EDL [134]	✓	✓						
PAAS [200]		✓						
Liu et al. [201]							✓	✓
MODPSO [202]		✓					✓	✓
Choudhary et al. [67]	✓	✓		✓	✓			
Ghosh et al. [87]	✓			✓				
Yang et al. [132]			✓					
MAPE-K [133]			✓					✓
Zhang et al. [58]						✓		
ELSH [86]				✓		✓		
Shang [203]					✓		✓	✓

Table 2.4: Review of Energy-Aware Scheduling Techniques and their Characteristics

Dynamic Voltage and Frequency Scaling (DVFS) is a power management technique that dynamically adjusts a processor's voltage and operating frequency based on computational demands [109]. DVFS can significantly decrease power consumption while maintaining necessary performance levels by reducing voltage and frequency during periods of lower processing requirements. Recent research by [46, 134, 200] has developed advanced DVFS-enabled scheduling algorithms that optimize energy efficiency with minimal performance trade-offs. Under-volting the processor is a power reduction provided to a processor to reduce its frequency and minimize energy consumption at a penalty to its processing time. A scheduling algorithm that makes use of this concept is introduced to reduce the energy consumption of scientific workflows [44].

Regression analysis is a technique that uses historical data to model relationships between dependent and independent variables. This technique identifies patterns in existing datasets to quantify how changes in computation correlate with outcomes such as energy consumption or performance. Researchers have demonstrated its effectiveness by developing regression-based models trained on historical data to generate evidence-based predictions of computational performance and energy consumption with high accuracy [132, 133, 37]. Adaptive scheduling is a dynamic resource allocation approach that continuously adjusts task execution priorities and resource assignments based on real-time system conditions and performance feedback. Research conducted by [67, 87, 86, 37] showed significant improvement in the computation performance or energy consumption when an adaptive scheduler is used to monitor workload characteristics, resource availability, and system performance metrics.

Most of the existing research has predominantly focused on single-user computing environments, focusing on improving performance or reducing energy consumption for the computation. This approach has limited practical application, as modern computing infrastructure typically serves multiple users while executing numerous computations simultaneously. This contention introduces complexities. While [67] developed a technique that respects user preferences during scheduling, other researchers like [58, 86] have developed contention-aware scheduling algorithms designed explicitly for multi-user computing environments, addressing the challenges of resource competition and optimization across concurrent workloads.

Multi-objective energy-aware workflow scheduling techniques address the challenge of optimizing multiple competing objectives simultaneously. Various methods, including evolutionary algorithms, heuristic approaches, and machine learning techniques, are employed to navigate this complex optimization space. Multi-Objective algorithms have been developed that focus on different aspects of computations such as energy consumption, makespan, and resource utilization [199, 41, 46, 201, 202, 73]. Evolutionary and Machine Learning techniques have also been proposed to navigate through complex scheduling situations [201, 202, 133, 203].

2.6 Related Work in Experimental Cluster Computing

Experimental research in cluster computing provides the foundations and insights that inform the development of energy-aware scientific workflow scheduling systems. This research area combines the design, implementation, and evaluation of computing systems with emphasis on understanding system behavior, performance characteristics, and optimization opportunities through controlled experimentation.

2.6.1 Experimental Cluster Computing Studies

Data mining algorithms are essential tools to extract information from the increasing number of large data sets, also called Big Data [204]. They are high-power and performance-demanding algorithms. Executing two of these algorithms (Apriori and K-means) on SBC clusters and daily-use computers showed that Single Board Computer (SBC) clusters outperformed while proving to be the most cost-effective [78, 94, 205, 204]. They compared the results from SBC clusters with those of a high-performance computing (HPC) platform. They concluded that even though the SBC cluster performs lower than the HPC platform, they are very effective at energy consumption and value for money. SBC clusters have also been concluded as an effective alternative for mobile Hadoop clusters and robust computing performances [206, 101, 96]. Comparative study of performances of clusters with different numbers of nodes showed that the 10-node SBC cluster was superior in performance to a single computer and the 5-node SBC cluster by 20% and 80%, respectively.

The majority of modern-day computation has been offloaded to huge remote data centers [93, 101]. To tackle issues such as power efficiency and thermal output, a novel architecture, PiStack, was developed. It implements novel ideas to reduce cluttering by powering nodes through the cluster case and saving power by introducing heartbeat functionalities for each node [93]. Similar to PiStack, novel alternatives for 1U rack in big data centers involved stacking RPi's to achieve maximum accessibility and easier replacement [101]. They also prove that SBC clusters can be a feasible approach to solving Big Data issues of storing and retrieving data.

Cloud computing is an example of edge computing in which a large number of computations are performed remotely on the data centres through some form of message passing [206]. Edge computing means having compute resources near the data sources [93]. In the modern world, there is less and less space available for the installation of large computing hardware. Hence, an increasing number of data centers and installations are being built in remote locations. There are minimal power resources in remote locations, and thus power efficiency in terms of GFLOPS/W is an important consideration [93]. Essential factors such as network latency, overheads, power, and energy consumption of edge and cloud computing have been compared in different SBC clusters [93, 206].

Numerous studies have shown that single board computers (SBCs) are more energy efficient than desktop computers, and this paved a pathway towards studies about energy consumption in different SBC clusters [204]. Terms like power/ energy consumption (GFLOPS/W) and value for money (GFLOPS/$) are used frequently to depict the performance and cost of the cluster. Using SBC clusters to perform cloud simulations and provide virtualization of resources has been studied [72, 207]. Cloud infrastructure has been simulated by iCanCloud [89] and CloudSim [88] in the past. Same functionality has been achieved using SBC clusters in Glasgow Raspberry PiCloud [207]. PiCloud simulates every layer of the cloud infrastructure, ranging from resource virtualization to network behavior [207].

Another application that benefits from parallelism is image processing. Computation time is a very important factor during image processing, and clusters have been proven to reduce the computation time significantly. OpenCV exploits parallelism to process real-time images faster. SBC clusters executing image processing libraries significantly outperformed single computers in frame processing of a live stream video [208]. Scikit Image library was used to evaluate the accuracy of the cluster by executing two parallel algorithms – Watershed and Edge detection on several images [209]. OpenMP is an application programming interface that supports shared-memory multiprocessing. It has been used in clusters to perform image processing faster and to provide a comparative study of different image processing libraries [210, 211].

2.6.2 Container-Based Energy-Aware Execution

As Docker has been gaining popularity in both industry and research, multiple attempts at optimizing Docker workloads have been conducted. These optimizations are in both the performance and the energy consumption of the Docker containers. Docker containers consume a lot of energy if not properly configured or managed. Even though the primary contributor towards the energy consumption of a Docker container is its CPU load, [212] identified that there are other components such as the strain on the host Operating System that can contribute towards the increased energy usage of a container.

A Workload-aware Energy Efficient Container (WEEC) brokering system is introduced for Docker containers to reduce their energy consumption in Docker-based cloud data centers [213]. A Docker-based energy management system (DEMS) architecture is developed as an improvement to the traditional energy management system (TEMS). It aims to fix the issues of slow deployment and low flexibility [214]. DEMS reduces the number of web releases of a container by 3 times and the workload by 2 times. An approach called brownout is proposed to dynamically activate or reactivate optional containers in data centers [121]. This approach was able to reduce about 10% - 40% energy consumption of the micro-services hosted by the data center. Similarly, an Energy-Aware scaling algorithm was developed to dynamically load balance the requests [215, 216]. The algorithm is able to spawn new containers during heavy loads and kill existing containers to save energy based on certain thresholds [215, 217].

Energy consumption increase due to Docker containerization was compared between different Docker workloads. [118] concluded that running bare metal computations for I/O overheads is better than using Docker for execution. A comparison between the energy consumption of different virtualization technologies, such as Virtual Machine, Docker, and Kubernetes, has been conducted [218, 219]. They noted significant energy consumption differences between the virtualized technologies. The energy consumption of a single web app container has been captured in response to the scaling and balancing of the load [79]. The majority of the energy-aware work has been conducted in relation to a single Docker workload and/ or Docker workloads deployed in the cloud. The current research in the field of energy-aware Docker Computing is generic and tries to reduce or monitor the energy consumption of Docker as a whole. They do not profile or analyze the energy footprint of typical Docker workloads.

2.6.3 Optimization Techniques for Workflow Scheduling

Cloud computing environments usually optimize their computations to maximize the performance of the resources. A mixed-integer programming (MIP) optimization, along with heuristics, has been proposed to enhance a cloud system's performance while meeting the deadline constraints of users [70]. Schedule exploration and optimization are performed using an optimization framework in tensor computation environments [220]. This framework uses various heuristic techniques and machine learning approaches to generate hardware-specific scheduling configurations.

Heuristics is a well-known method for finding an optimal solution for a problem in polynomial time complexity [84]. They work on the philosophy of trial and error for discovering the optimal solution. Heuristics do not guarantee the most optimal solution, but they do a "good enough" job considering the other overheads of computing the optimal solution [84]. A static list-based task scheduling algorithm called Heterogeneous Earliest Finish Time (HEFT) was developed for task scheduling [22]. HEFT is greedy and often results in a local optimal solution. Another list-based scheduling algorithm, called Predict Priority Task Scheduling (PPTS), has been developed that

can look ahead not only in the processor selection phase but also in the task prioritization phase by using a predict cost matrix (PCM) [221].

A mathematical framework has been proposed that utilizes Genetic Algorithms (GAs) to minimize the energy consumption of computation by considering key factors such as idle periods, computational startup latency, and power cycling durations [222, 74]. The developed mathematical model takes into account different machine attributes, including idle time, launch time for computation, and power-on/off cycles, to achieve near-optimal solutions for the lowest energy consumption [222]. Additionally, an energy loss optimization scheduling technique using multi-objective fuzzy algorithms has been presented, targeting energy costs and scheduling time in IoT environments [223]. This approach focuses on single-target energy loss problems by identifying device idle periods and coordinating job allocation to maximize resource usage and reduce overall energy consumption.

Mixed integer programming models utilizing Genetic Algorithms (GA) have been implemented for scheduling solutions that minimize deadline misses in cloud computing [70]. Hybrid optimization frameworks for tensor computation on heterogeneous systems employ machine learning and heuristic methods to develop hardware-specific schedules [220]. A hybrid meta-heuristic approach combining Whale Optimization Algorithm (WOA) with Evolutionary Algorithm (EA) has been designed to enhance energy efficiency in IoT networks by considering workloads, operating temperatures, and residual energy levels [74, 224]. Server energy conservation has been addressed through heuristic algorithms that consolidate virtual clusters onto physical servers while maintaining SLA compliance [113]. Swarm intelligence approaches include enhanced Whale Optimization Algorithms for cloud resource scheduling [225], Ant Colony Optimization (ACO) for minimizing makespan and reducing energy costs [226, 227], and Ant Mating Optimization (AMO) for balancing task completion time and energy consumption in fog computing [228]. Additionally, improved Particle Swarm Optimization (PSO) using Longest Job to Fastest Processor (LJFP) and Minimum Completion Time (MCT) heuristics has been proposed to optimize makespan, performance, and energy consumption in cloud infrastructure [69].

Various metaheuristic approaches have been developed to address multi-objective energy optimization problems in distributed computing environments. Genetic algorithms and evolutionary approaches effectively handle these challenges by maintaining populations of solutions that represent different trade-offs between competing objectives, automatically exploring energy-performance trade-offs without requiring detailed a priori knowledge of optimal solutions [162, 222]. Particle Swarm Optimization (PSO) provides bio-inspired approaches that have demonstrated effectiveness for energy-aware scheduling problems, with recent research developing enhanced PSO variants that incorporate domain-specific heuristics such as Longest Job to Fastest Processor (LJFP) and Minimum Completion Time (MCT) techniques for improved initialization and performance evaluation across makespan, performance, and energy consumption metrics in cloud infrastructure [164, 69]. Ant Colony Optimization and other swarm intelligence

approaches offer alternative metaheuristic techniques that achieve good solutions with reasonable computational overhead by incorporating problem-specific knowledge through pheromone trail updates and heuristic guidance mechanisms [226, 227]. Additionally, nature-inspired algorithms including Whale Optimization Algorithm, Grasshopper Algorithm, and Cuckoo Search have demonstrated effectiveness for energy optimization problems by offering different exploration and exploitation characteristics that may be advantageous for specific types of energy optimization challenges [225, 163, 165].

Machine learning techniques provide adaptive approaches for energy optimization in complex computing systems where traditional optimization methods may be constrained by incomplete system models or computational complexity. Supervised learning approaches develop predictive models for energy consumption using historical data, enabling accurate energy-aware scheduling decisions through neural networks that model complex scheduling-energy relationships and decision trees that provide interpretable guidance for optimization strategies [171, 77]. Reinforcement learning techniques continuously improve energy optimization through observed system behavior, with Q-learning and model-free approaches developing adaptive scheduling policies that learn optimal energy-performance trade-offs without requiring detailed system models [155]. Unsupervised learning methods complement these approaches by identifying behavioral patterns that inform optimization strategies, including clustering techniques for workload grouping and anomaly detection for identifying unusual energy consumption patterns, collectively enabling systems to automatically adapt to changing operational conditions.

2.7 Research Gaps and Opportunities

The review of existing literature reveals several significant gaps and open issues that limit the effectiveness of current approaches to energy-aware scientific workflow scheduling. These gaps represent essential research opportunities that this thesis seeks to address through the development of novel methodologies, algorithms, and systems. Understanding these limitations is crucial for positioning the contributions of this research within the broader scientific computing landscape.

2.7.1 Energy Modeling and Measurement Limitations

Current approaches to energy modeling and measurement exhibit several significant limitations that restrict their effectiveness for energy-aware workflow scheduling. Most existing energy models rely on simplified assumptions about hardware behavior and fail to capture the complex interactions between application characteristics, system utilization patterns, and actual energy consumption. The lack of standardized energy measurement methodologies makes it challenging to compare different approaches or validate energy optimization strategies across diverse computing environments. Many measurement approaches introduce significant overhead or provide insufficient granularity for real-time scheduling decisions.

Key deficiencies in this area include:

- No definitive framework exists to evaluate workflows based on energy consumption, with existing frameworks focusing primarily on performance and efficiency metrics
- Energy savings are often calculated theoretically without presenting the real energy usage of computations
- Limited frameworks exist for cost-performance analysis of clusters, preventing an understanding of energy aspects and consumption patterns

2.7.2 Scalability and Practical Deployment Challenges

Many existing energy-aware scheduling approaches face significant scalability limitations that restrict their applicability to large-scale scientific workflows and computing systems. Mathematical optimization approaches often exhibit exponential computational complexity that makes them impractical for workflows with thousands of tasks or computing systems with thousands of processors. The integration of energy-aware scheduling with existing workflow management systems and resource schedulers remains challenging, with most research focusing on algorithmic development without adequate consideration of practical deployment requirements.

Additional scalability concerns include:

- Limited frameworks for evaluating, analyzing, and benchmarking clusters with different configurations, with most benchmarks testing only single configurations
- Workflows used in related research are typically applied as-is without tailoring for specific cluster configurations, missing opportunities for resource-based optimization
- Over-reliance on simulation rather than real-world implementation leads to solutions that fail to address practical deployment issues

2.7.3 Limited Adaptability and Dynamic Response

Current energy-aware scheduling approaches often make static decisions based on initial workflow and system characterization, with limited ability to adapt to changing conditions during execution [85, 45]. Real-world workflow execution involves dynamic conditions, including varying system load, changing resource availability, and unpredictable task execution behavior that can significantly impact both performance and energy consumption [55, 86]. The lack of adaptive mechanisms limits the effectiveness of energy-aware scheduling in practical deployment scenarios where system conditions may differ significantly from initial assumptions [87, 61].

2.7.4 Insufficient User Interface and Multi-User Considerations

The complexity of energy-aware scheduling creates significant usability challenges that limit adoption by the broader scientific computing community. Most existing approaches require detailed knowledge of energy modeling, optimization techniques, and system characteristics that may not be available to domain scientists. The lack of intuitive interfaces for specifying energy-performance preferences and the absence of automated decision-making capabilities create barriers to the practical deployment of energy-aware scheduling systems.

Critical user-centric gaps include:

- User preferences regarding performance-energy tradeoffs are not considered during scheduling decisions
- Energy efficiency and multi-user preference awareness require greater focus, as most solutions target single-user scenarios while overlooking the complexity of real-world computing infrastructure, where multiple users share resources
- Insufficient attention to usability prevents broader adoption by domain scientists who lack specialized knowledge in energy optimization

2.7.5 Lack of Unified Framework

Modern scientific workflows execute across diverse cluster configurations, each offering unique performance-energy tradeoffs. When combined with multi-user environments featuring competing priorities and preferences, scheduling decisions become increasingly complex. While research has advanced in specific domains, a unified approach that addresses these multifaceted challenges remains absent. Current research faces several notable challenges that highlight the need for comprehensive solutions:

- Workflow-specific designs that are not generalized, preventing broader applicability
- Theoretical approaches to energy reduction that do not translate effectively to real-world scenarios
- Testing predominantly in simulated environments that fail to capture the complexity of actual compute systems
- Absence of consideration for optimizations at workflow, job, and cluster levels simultaneously

These identified gaps highlight the critical need for a generic scheduling framework that exploits different policies across workflow, job, and cluster levels. A comprehensive scheduler system that can respect user preferences and navigate the complexities of workflow execution in multi-user environments could enable significant advances in this domain. Such a system should integrate energy awareness with performance optimization while maintaining practical applicability and user accessibility.

2.8 Thesis Contributions to Address Identified Gaps

This thesis directly addresses the identified gaps in current energy-aware scientific workflow scheduling research through the development of comprehensive solutions that advance both theoretical understanding and practical capabilities. The research contributions are specifically designed to overcome the limitations of existing approaches while providing tangible benefits for the scientific computing community.

2.8.1 Comprehensive Energy Monitoring and Modeling Framework

In this thesis, a comprehensive energy monitoring framework that provides standardized, accurate, and low-overhead energy measurement capabilities across diverse computing platforms is developed, addressing the critical gaps in energy modeling and measurement. This contribution directly tackles the following identified limitations:

- **Standardized Energy Measurement Methodology:** The framework establishes a unified approach to energy measurement that enables consistent evaluation and comparison of different energy-aware scheduling strategies across heterogeneous computing environments. This addresses the current lack of standardized methodologies that hampers cross-platform validation and comparison of energy optimization approaches.
- **Multi-Granularity Energy Modeling:** Unlike existing simplified models, the framework captures complex interactions between application characteristics, system utilization patterns, and actual energy consumption at multiple levels—from individual task execution to system-wide consumption patterns. This comprehensive modeling approach overcomes the limitations of current methods that rely on oversimplified hardware assumptions.
- **Low-Overhead Real-Time Monitoring:** The framework provides detailed energy consumption information with minimal measurement overhead, enabling real-time scheduling decisions without compromising system performance. This directly addresses the current limitation where measurement approaches either introduce significant overhead or provide insufficient granularity for practical scheduling applications.

2.8.2 Scalable Hybrid Optimization Architecture

The scalability and practical deployment challenges are addressed through the development of novel hybrid optimization-based scheduling algorithms that efficiently handle large-scale workflows while providing near-optimal solutions. This contribution includes:

- **Scalable Algorithm Design:** The research develops hybrid approaches that combine the optimality guarantees of mathematical optimization with the computational efficiency of heuristic methods, enabling effective scheduling of workflows containing thousands of tasks on computing systems with thousands of processors. This directly addresses the exponential computational complexity that renders current mathematical optimization approaches impractical for large-scale systems.

- **Practical Integration Framework:** A comprehensive integration framework provides seamless deployment capabilities that work with existing workflow management systems and resource schedulers, addressing the current gap between algorithmic research and practical deployment requirements.

- **Dynamic Cluster Configuration Optimization:** The framework dynamically optimizes cluster configurations based on workflow requirements and energy constraints, enabling intelligent resource management that can power down underutilized nodes and consolidate workloads to minimize energy waste.

2.8.3 Intelligent Adaptive Scheduling Mechanisms

This thesis addresses the limited adaptability of current approaches through the development of sophisticated adaptive scheduling mechanisms that dynamically adjust energy-aware scheduling decisions based on real-time system conditions and observed execution behavior. Key adaptive capabilities include:

- **Real-Time System Monitoring and Response:** The adaptive algorithms continuously monitor system performance, resource utilization, and energy consumption patterns, using this information to refine scheduling decisions and improve both energy efficiency and overall system performance during execution.

- **Dynamic Resource Consolidation:** The framework intelligently identifies underutilized resources during execution and implements dynamic consolidation strategies. For example, when multiple nodes are operating at partial capacity, the system can migrate workloads to fewer nodes and power down unused resources, resulting in significant energy savings.

- **Condition-Aware Optimization:** The adaptive capability enables the scheduling system to maintain effectiveness even when actual execution conditions differ significantly from initial assumptions, addressing a critical limitation of current static scheduling approaches.

2.8.4 User-Centric Interface and Multi-User Preference Management

The usability challenges and multi-user environment complexities are addressed through the development of an integrated system that abstracts scheduling complexity while providing intuitive preference specification mechanisms:

- **Automated Decision-Making with User Preferences:** The system incorporates user preferences regarding performance-energy tradeoffs into scheduling decisions, enabling scientists to benefit from energy optimization without requiring expertise in scheduling algorithms or energy modeling.
- **Multi-User Environment Support:** Unlike existing single-user focused solutions, the framework explicitly addresses the complexity of real-world computing infrastructure where multiple users share resources with competing priorities and preferences.

2.8.5 Impact of the Contributions on Field of Energy-Aware Computing

The combined contributions of this thesis create a comprehensive solution that addresses the identified gaps through a unified, practical, and user-friendly framework. The integrated system provides:

- A standardized platform for energy-aware workflow scheduling that can be applied across diverse computing environments and workflow types
- Scalable algorithms that maintain effectiveness from small-scale research computing to large-scale high-performance computing facilities
- Adaptive capabilities that ensure robust performance under varying system conditions and workload characteristics
- User-centric design that enables broad adoption by the scientific computing community without requiring specialized expertise
- Real-world validation and practical deployment capabilities that bridge the gap between research innovation and operational implementation

These contributions collectively advance the field of energy-aware scientific workflow scheduling from fragmented, theoretical approaches to a comprehensive, practical solution that can significantly impact energy efficiency in scientific computing environments. The next chapter presents the detailed system architecture and implementation strategies that realize these contributions.

2.9 Summary

This literature review has provided a comprehensive examination of the research landscape relevant to energy-aware scientific workflow scheduling, establishing the theoretical foundations and identifying the research gaps that motivate this thesis. The review began with essential background knowledge covering the hardware and software components that comprise modern scientific computing environments, followed by a detailed examination of scientific workflow execution systems and methodologies.

The analysis of related work revealed significant research activity in multiple areas relevant to energy-aware workflow scheduling, including experimental cluster computing, energy-aware workflow execution, scheduling techniques, and optimization methods for energy reduction. However, the review also identified several critical gaps in current research, including limitations in energy modeling and measurement, scalability challenges, insufficient adaptability, and usability barriers that restrict practical deployment.

The positioning of this thesis within the research landscape demonstrates how the proposed contributions directly address these gaps through a comprehensive solution that advances both theoretical understanding and practical capabilities. The development of an energy monitoring framework, scalable optimization algorithms, and adaptive scheduling mechanisms represents significant advances over current state-of-the-art approaches.

The reviews and analyses in this chapter laid the necessary groundwork for the forthcoming chapters. Firstly, by having reviewed the effects of hardware and software components on the performance, a framework that will cover all these parameters and evaluate them accordingly can be developed. Secondly, reviewing and analyzing the work of past researchers allows a definite set of requirements to be outlined for addressing the research question and open gaps. The literature also allows a comprehensive evaluation of our proposed system. Finally, the literature provides us with the basic understanding of designing, implementing, and evaluating the proposed system for a particular architecture system.

Chapter 3

Monitoring the Energy Consumption of Scientific Workloads

The contents of the chapter have been published in the following articles:

1. Warade, M., Schneider, J.-G., & Lee, K. (2022). Measuring the energy and performance of scientific workflows on low-power clusters. *Electronics*, 11(11), 1801. https://doi.org/10.3390/electronics11111801
2. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2023). Monitoring the energy consumption of Docker containers. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1703–1710). IEEE. https://doi.org/10.1109/COMPSAC57700.2023.00263

3.1 Overview

Scientific computation has evolved to be structured as a series of interconnected tasks within sophisticated workflows [4]. These workflows are executed by specialized workflow engines that manage data dependencies, task orchestration, and execution reporting [11, 12, 13]. Workflow engines provide valuable abstractions that allow scientists to focus on computational optimization without infrastructure concerns, but this results in scientists being unaware of the environmental impact of their computations. The deployment landscape has also been revolutionized by containerization technologies, particularly Docker, which has become the standard for application deployment and execution [229].

The widespread adoption of Docker containers in scientific computing environments has introduced additional complexity to energy consumption patterns. Docker containers provide

lightweight, deterministic, and manageable isolated environments that enable agile computing resources [116]. According to industry reports, organizations are running an average of 154 individual Docker containers on a single host, representing a 50% increase from previous years [230].

The surge in usage of scientific workflows and containerization technologies creates a compelling case for the development of a reliable energy monitoring framework. Scientific workflows have a complex structure that has many dependencies, complex file management, and computation characteristics that make the trade-offs between performance and energy consumption less clear [14]. Similarly, the growth of Docker deployments has significant implications for energy consumption, as containers can be very efficient in terms of resource utilization but may also consume substantial energy if not properly managed [71]. Despite Docker becoming one of the most used modern deployment technologies, there is very little information on the energy consumption of Docker containers and Docker workloads that can help in developing or making informed decisions about how to optimize Docker deployments [231, 232].

There is a need for a standardized way to measure, analyze, and improve the energy consumption of scientific and containerized computation [92]. Reliable monitoring of the energy consumption of computation can allow for a greater understanding of the behavior and factors affecting the energy consumption of those computations. Due to this, the development of a reliable framework to monitor the energy consumption represents a critical gap in current scientific computing infrastructure.

This chapter investigates the fundamental challenge of energy consumption monitoring and identifies the factors affecting energy consumption in scientific computing environments. This will enable the development of a reliable framework for monitoring energy consumption in a computing environment. The trade-offs between the energy consumption and performance of different computational workloads are also investigated.

The research presents a comprehensive framework for reliable energy monitoring that enables detailed analysis of energy consumption patterns across different workflow types, sizes, and cluster configurations. The framework is evaluated using different scientific workflows and containerized workloads to confirm its reliability and accuracy. The framework serves as the foundation for subsequent energy-aware scheduling strategies presented in the later chapters of this thesis.

The key outcomes of this chapter are as follows:

1. A methodical approach for accurately measuring energy consumption in scientific workflows, addressing the reliability issues present in existing theoretical solutions.
2. The design and implementation of a reliable approach for energy monitoring of scientific and containerized computation.

3. Providing in-depth analysis of different factors that affect the energy consumption of scientific and containerized computation.

4. Motivating energy-aware scheduling as a solution for the energy-efficient execution of different computations on compute clusters.

5. Establishing an empirical foundation necessary for developing energy-aware schedulers for computations.

The remainder of this chapter is organized as follows. Section 3.2 presents the theoretical foundation, design, and implementation for the energy monitoring framework, including the architectural design and container-based monitoring approach. Section 3.3 describes the experimental hardware and software setup upon which the evaluation of the framework is conducted. Sections 3.4 and 3.5 present an in-depth analysis of energy consumption and performance metrics across varying complexity of Montage and Bioinformatic workflow types and cluster configurations, respectively. Section 3.6 focuses specifically on fingerprinting the energy consumption characteristics of Docker containers of different technologies.Finally, Section 3.7 summarizes the key outcomes and discusses the findings of this chapter.

3.2 A Framework for Reliable Energy Monitoring

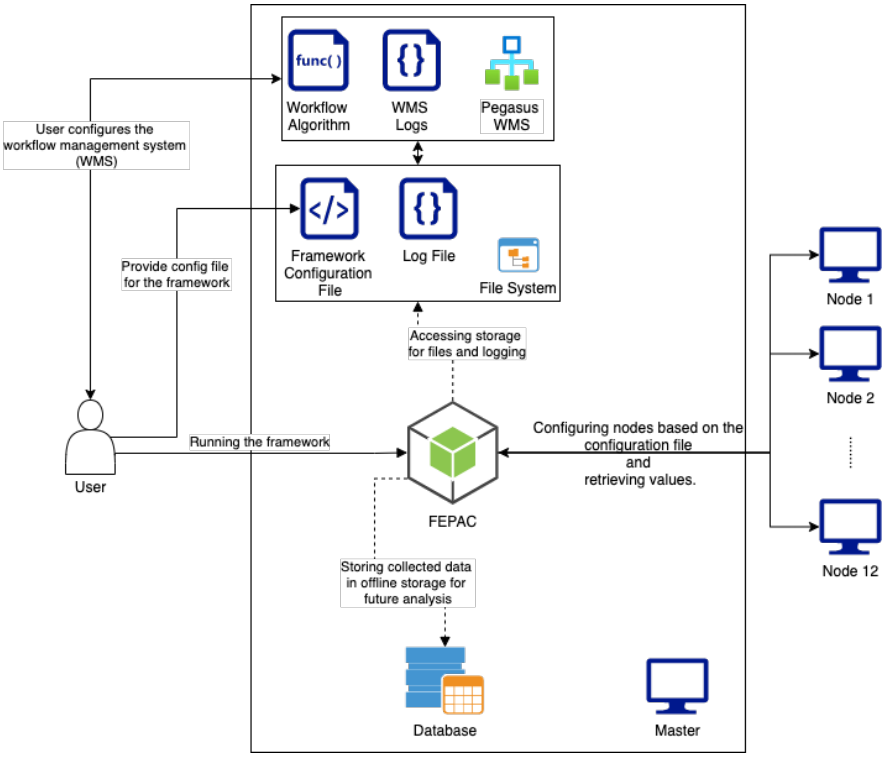


Figure 3.1: Proposed Architecture Diagram for a Reliable Energy Monitoring Framework

The experimental setup consists of multiple computing nodes connected through a high-speed network infrastructure, with a dedicated x86 Linux PC serving as the central management and monitoring coordinator [15]. This management node communicates with the compute cluster nodes, energy monitoring hardware, and provides centralized data collection and analysis capabilities for the entire system [94].

Figure 3.1 illustrates the comprehensive software architecture implemented for this study. The x86 Linux PC hosts two primary software systems: the FEPAC (Framework for Evaluating Parallel Algorithms on Cluster Architectures) energy monitoring framework [92] and a workflow management system (Pegasus) [11]. These systems work in coordination to provide both energy monitoring capabilities and scientific workflow execution management [64].

FEPAC was previously developed by the authors as a generic solution for monitoring energy usage of parallel computations on cluster systems [92]. The framework provides standardized interfaces for collecting energy consumption data from various hardware monitoring sources and presents this information in a format suitable for computational decision-making [152]. In contrast to its original implementation, the framework has been significantly expanded in this work to integrate with existing energy monitoring infrastructure, including sensors embedded in networking devices, external power monitoring equipment, and system-level hardware performance counters [65]. This expansion enhances the framework's functionality by providing software-accessible energy data that workflow management systems can utilize to make informed scheduling and resource allocation decisions [67].

The Pegasus workflow management system serves as the computational orchestration layer, responsible for parsing workflow descriptions, scheduling tasks across available computing resources, and managing data dependencies between workflow components [21]. Pegasus provides sophisticated scheduling algorithms that can incorporate external constraints and optimization objectives, making it suitable for integration with energy-aware scheduling approaches [23]. The system maintains detailed execution logs that record the precise timing and resource allocation for each workflow task, enabling accurate correlation with energy consumption measurements [233].

Energy data collected from the various monitoring sources is systematically extracted, processed, and stored in a MySQL relational database [92]. This database serves as the central repository for all energy consumption information and provides structured access to historical energy usage patterns, enabling both real-time monitoring and post-execution analysis [64]. The workflow management system can query this database to retrieve current and historical energy consumption data, which can then be incorporated into scheduling decisions and resource allocation strategies.

The integration between the energy monitoring framework and the workflow management system is achieved through timestamp synchronization and cross-correlation of execution logs [37]. Detailed execution logs generated by the workflow management system provide precise information

about when specific workflow tasks are executing and on which computing nodes they are allocated. These logs include task start and completion times, resource assignments, and intermediate data transfer operations. The energy monitoring framework continuously records power consumption measurements with corresponding timestamps, synchronized to the master node's system clock to ensure temporal accuracy [94]. By cross-linking the energy measurements stored in the database with the execution logs using synchronized timestamps, the system can accurately attribute energy consumption to specific workflow tasks and computing resources, enabling detailed analysis of the energy efficiency of different workflow execution strategies [73].

3.3 Experimental Setup

The previous sections outlined the proposal and implementation of the framework for monitoring and analyzing the energy consumption of different computations. For this study, two distinct experimental setups were implemented to evaluate the proper functioning of the proposed solution across different computational paradigms. The framework is evaluated with scientific workflows and containerized computations using different hardware configurations and monitoring approaches. In this section, the experimental setup, including the hardware, software, and integration of the framework into the existing execution infrastructure, has been provided.

3.3.1 Scientific Workflow Execution on Raspberry Pi Cluster Setup

Computing Hardware Infrastructure

The Raspberry Pi Foundation (`https://www.raspberrypi.org/about/`, accessed on 6 May 2022) is a UK-based charity responsible for the manufacturing and development of Raspberry Pi boards [207]. During the study period, the Raspberry Pi Model 4B was the primary board available, though the Raspberry Pi 5 has since been released as the latest model [93]. The Raspberry Pi 4B uses a Broadcom BCM2711, quad-core ARM Cortex-A72 processor with a reported peak performance of 13.5 GFLOPS (`http://web.eece.maine.edu/~vweaver/group/green_machines.html`, accessed on 6 May 2022), and includes Gigabit Ethernet connectivity, USB, USB-C, and Micro HDMI ports [94]. Users can choose from 2 GB, 4 GB or 8 GB of RAM configurations—for these experiments, the 2 GB variant was selected to represent resource-constrained computing environments.

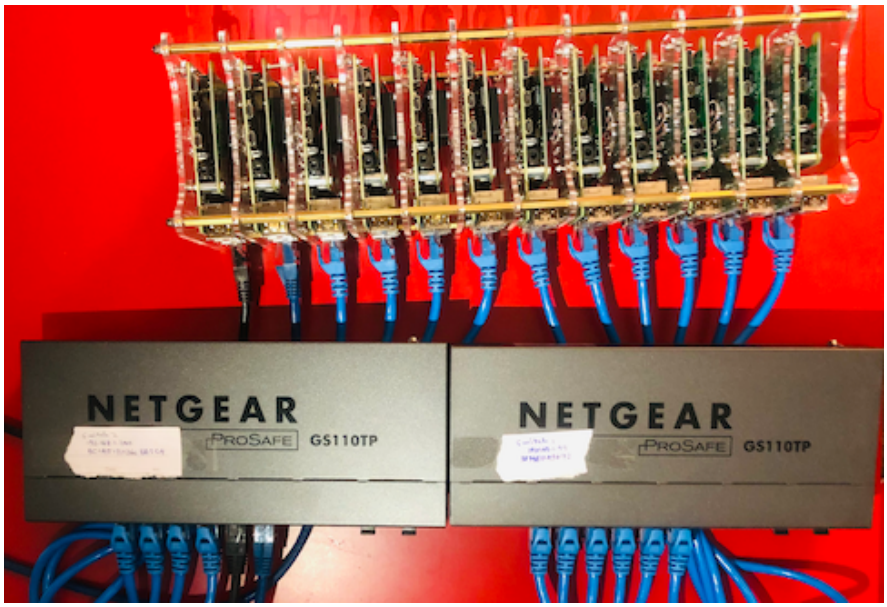


Figure 3.2: Raspberry Pi Cluster Setup for Scientific Workflow Execution

A cluster of 12 Raspberry Pi 4B nodes was built for scientific workflow evaluation [234]. A standard x86 Linux-based PC (8 core with Linux Mint OS (`https://linuxmint.com/`, accessed on 6 May 2022)) acted as a master node for workflow management and coordination [15]. The computing nodes and the master node were connected through 2 Netgear GS110TP switches. The nodes are powered through Power over Ethernet (PoE), which enables monitoring of their energy usage through the switches' internal sensors (see Figure 3.2) [64].

For enhanced system management and experimental control, network booting of a lightweight OS (DietPi) was implemented for each node in the cluster rather than traditional SD card booting [235]. Each client requests boot instructions from a DHCP server (the Master node), which assigns IP addresses and provides boot instructions over the network. The boot files and root file system are stored on the server, with clients accessing files via Network File System Protocol (NFS) [15]. A cluster of small board computers serves as a proxy for industry-scale clusters, offering greater flexibility with configurations, minimal operating system overhead, and fine-grained access to runtime metrics such as energy usage data on a per-node basis [234, 93]. Energy usage patterns can be extrapolated and generalized for other cluster architectures [236, 207].

Software Infrastructure

For this study, two different scientific workflows - Montage (see Section 2.4.6) and Bioinformatics (see Section 2.4.6) have been used to evaluate the functioning of the proposed framework. The workflows are varied in their complexities to compare how different factors affect the energy consumption of the computation. Pegasus workflow engine, a Workflow Management System (WMS), is used to manage and execute the scientific workflows (see Section 2.4.2). The cluster

Figure 3.3: Computing Node Setup for Containerized Computation

Figure 3.4: Smart Energy monitoring plugs with ESP8266

management software chosen for executing scientific workflows is HTCondor [128] or Condor, which is a batch job scheduler and resource management system for high-throughput computing on distributed resources (see Section 2.2.2). Monitoring the energy consumption for the execution of these workflows on the RPi 4B cluster is done through a modified version of FEPAC [92]. FEPAC can get the energy consumption data from the internal sensor from the network switch that is powering the RPi nodes (see Figure 3.2).

3.3.2 Containerized Computation Setup on Intel NUC

Computing Hardware Infrastructure

The second experimental setup focuses on containerized computation evaluation using an Intel-based x86 NUC as the primary host machine. The computing hardware setup for evaluating containerized computation is shown in Figure 3.3. This setup simulates real-world containerized deployment scenarios commonly found in modern data centers and edge computing environments [102]. The host system runs Debian-based Linux Mint 20 (`https://linuxmint.`

com/, accessed on 4 December 2022) as the operating system [115]. The Intel NUC configuration includes an 11th Generation Intel(R) Core(TM) i7-1165G7 processor running at 2.80GHz with 4 physical cores and 8 threads, representing typical hardware used for containerized workloads. This configuration provides sufficient computational resources for evaluating container energy consumption patterns while maintaining compatibility with standard Docker deployment practices.

The host machine is powered using the main power plug via a smart energy monitoring plug (see Figure 3.4). The smart plug has an integrated ESP8266 [237] chip that allows the smart plug to connect to the local network and enable the collection of energy data using application programming interface (API) calls. Internally, the smart plugs collect energy consumption using the HLW8032 energy meter sensor. The smart plug is running the latest version of ESPHome Firmware. The firmware has been modified to measure and deliver the energy data every 500 ms. This data is collected at a set interval and stored in the database. There are different endpoints that can be queried to get different measurements from the smart plug. The API endpoint `smart_plug_v2_current` provides us with the current in Ampere. Similarly, the endpoints `v2_power`, `v2_energy`, `v2_voltage` provide the consumption data such as the power (in W), Energy (in kWh), and Voltage (in V), respectively. These endpoints return the data in a JSON format, which is then parsed and stored in the database.

Software Infrastructure

For the containerized computations, the Docker daemon version 19.03.8, build afacb8b7f0 [115] is used Section 2.2.2. The Docker daemon uses the default configuration. The versions of the Docker images used in this study are the most recent stable versions from the DockerHub repository [117]. In terms of energy consumption, FEPAC [92] was modified to collect energy data from the Smart plugs using API calls. FEPAC identifies the smart plug using its IP address when connected to the local network. Multiple API endpoints provide comprehensive energy metrics, which are collected and stored in a database every 500 ms. The data in the database is synced with the computation on the host machine via the timestamp field. The synced data is then analyzed, and necessary conclusions are made.

3.4 Astronomy Workflow Energy Evaluation

In this section, an in-depth experimental evaluation of the performance and energy consumption of a scientific workflow in the domain of Astronomy on a low-power cluster is presented. The experiments vary the workflow size and cluster size to investigate how different factors affect the performance and energy consumption, respectively. The experiments in this section are based on the experimental setup from Section 3.3.

3.4.1 Workflow Description

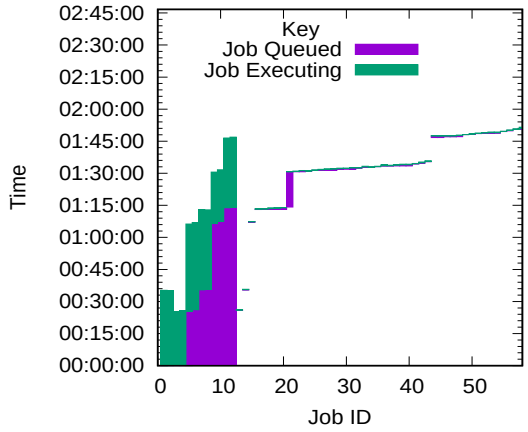
The scientific workflow used for the experimental evaluation in this section is the Montage workflow [29, 1]. Figure 2.7 illustrates the high-level DAG representation for the Montage workflow. Note that Figure 2.7 only shows the computation jobs that execute on cluster nodes. To manage the workflow, Pegasus creates many other data transfer and logging jobs that execute before and after each computation job. This evaluation focuses on the main execution jobs, as the energy impact on the other jobs is minimal and can be mostly removed with caching.

3.4.2 Montage Computation Characteristics

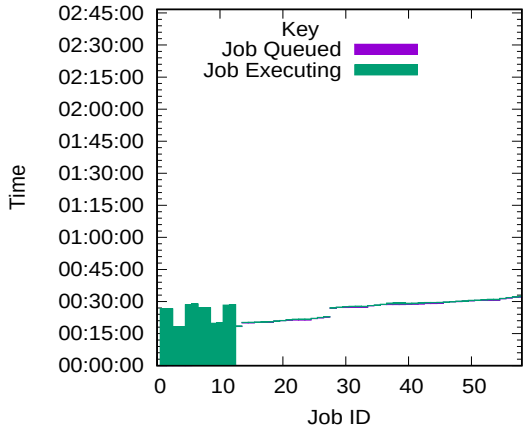
Montage has specific, unique characteristics to its structure. For example, `mProject`, which are computationally more intensive than others. Depending on the size of the workflow being executed, there are more of these particular jobs. The combination of the size of the workflow and the number of cluster nodes has a direct impact on the job queue time.

To illustrate this, six experiments have been performed by varying the size of the workflow and the size of the cluster. The workflow complexity is varied from 0.5 to 1.0 and then 1.5 degrees (see Table 2.2). The cluster size was varied between 1 node and 6 nodes. Each node had 4 hardware threads (for a maximum of 24 hardware threads with 6 nodes). Figures 3.5–3.7 illustrate the execution of Montage 0.5, 1.0 and 1.5 degree workflows on 1 node and 6 node low-power clusters. The cluster was setup and data was collected based on the experimental setup detailed in Section 3.3. The Y-axis in these three figures denotes the execution time (hours:minutes:seconds) of the corresponding workflows. The graphs illustrate two metrics for each Montage job. *Job Queued* is the time from the job being ready to execute (*i.e.* its dependencies in the DAG being met) to the time it starts executing. *Job Queued* measures the delay or queue time due to a temporary lack of resources. *Job Executing* measures the time the job starts executing until it completes the task and is removed from the cluster.

Figure 3.5 illustrates that a 0.5 degree workflow has some queuing of the `mProject` jobs on 1 cluster node (*cf.* Figure 3.5 a). This is due to the workflow having 12 parallel-capable `mProject` jobs, but the cluster only has 4 threads to execute them. Consequently, the remaining `mProject` jobs have to wait until the first one is complete. The remainder of the workflow proceeds as expected. Executing the same 0.5 degree workflow on a 6 node cluster results in no queuing of parallel-capable jobs as there are 24 threads available compared to 12 `mProject` jobs (*cf.* Figure 3.5 b).

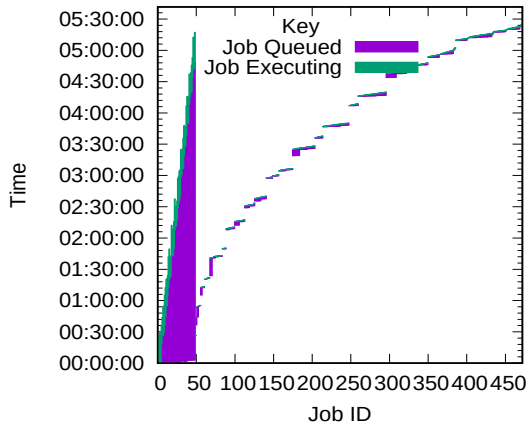


(a) 0.5 degree Montage execution on a small cluster

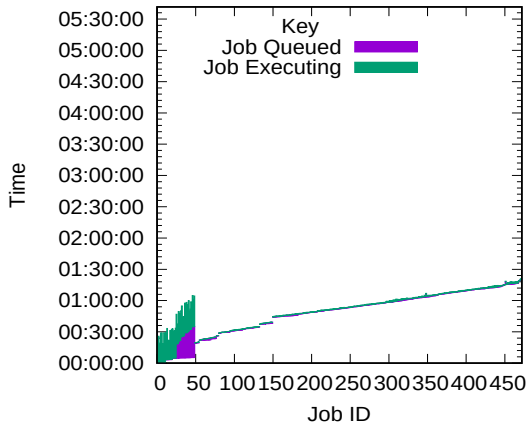


(b) 0.5 degree Montage execution on a large cluster

Figure 3.5: Execution of a Montage 0.5 degree workflow on a 1 node vs. 6 node cluster.

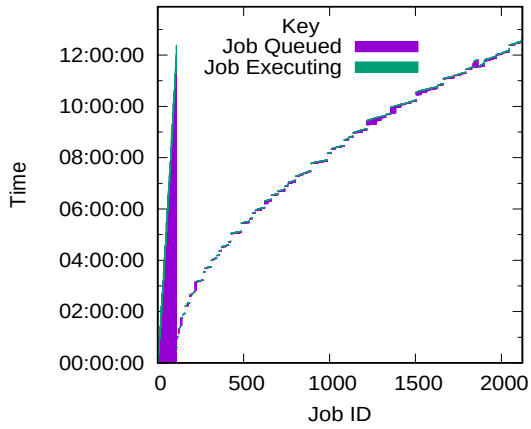


(a) 1.0 degree Montage execution on a small cluster

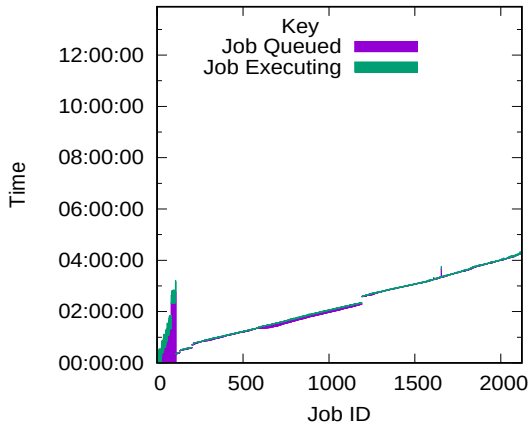


(b) 1.0 degree Montage execution on a large cluster

Figure 3.6: Execution of a Montage 1.0 degree workflow on a 1 node vs. 6 node cluster.



(a) 1.5 degree Montage on a small cluster



(b) 1.5 degree Montage on a large cluster

Figure 3.7: Execution of a Montage 1.5 degree workflow on a 1 node vs. 6 node cluster.

This is echoed in Figure 3.6 for the 1.0 degree workflow which shows a large amount of queuing (*cf.* Figure 3.6 a) when there are a lot less cluster nodes/available threads than jobs. This is a lot more pronounced compared to the 0.5 workflow due to there being 48 `mProject` jobs vs. 4 threads. For the 1.0 degree workflow, there is even queuing in the 6 node cluster as there are 24 cluster threads vs. 48 `mProject` jobs. This pattern is even more pronounced as the workflow size is increased to 1.5 degrees (*cf.* Figure 3.7). When there is only a single cluster node, the majority of `mProject` jobs are queued for a long time; this is due to there being 108 jobs and only 4 cluster threads. When the number of cluster nodes is increased to 6 there is less but still substantial queuing.

3.4.3 Montage 1.0 Degree Workflow Execution Results

Montage workflow presents an interesting and relatively complex workload for analyzing the computation and energy consumption characteristics of workflows on clusters. In this section, the results of experiments for a medium-sized workflow, Montage 1.0 degrees, with varying cluster sizes of 1 to 12 nodes are presented.

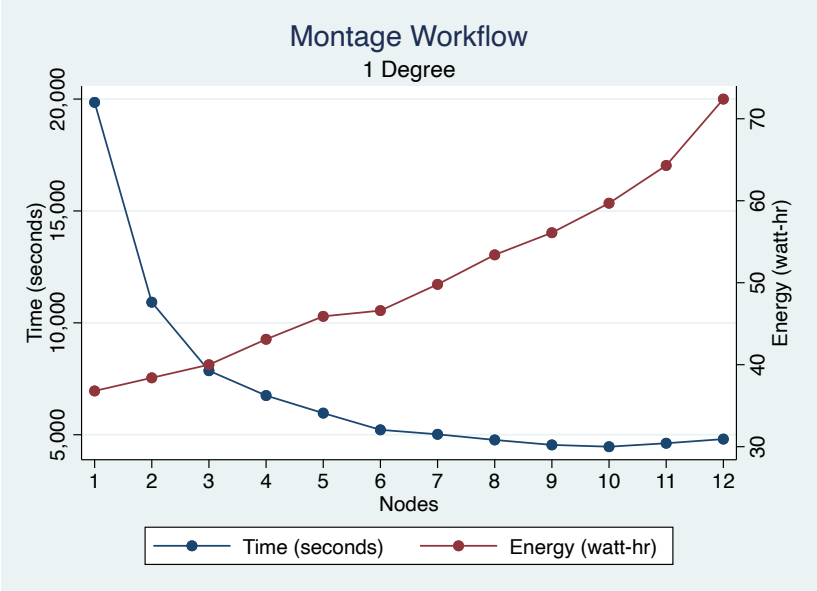


Figure 3.8: Montage 1.0 Degree Workflow Execution on Varying Cluster Sizes

Figure 3.8 illustrates both the execution time and energy consumption of a Montage 1.0 degree workflow on varying sizes of clusters. The left Y-axis is time in seconds from the first workflow job being queued to the last job completing. Activities from the master node are not included in this evaluation. The X-axis indicates the number of cluster Nodes from 1 to 12; note that each node has four hardware threads, so the number of available threads varies from 4 to 48 in increments of 4.

The results shown in Figure 3.8 illustrate that the execution time decreases with more nodes being added. Increasing the cluster size from 1 node to 2 nodes reduces the execution time from 19,544 s (approximately 5 h and 25 min) to 10,675 s (approximately 3 h), hence resulting in a speed-up of 1.83. However, increasing the cluster to 3 and 4 nodes only reduces the execution time to 7622 and 6384 s, respectively. After 5 nodes (execution time of 5702 s), a little reduction in execution time is achieved, and the curve flattens out. The maximum speed-up compared to 1 node is 4.81 achieved with 11 nodes, hence falling considerably short of a linear speed-up.

The reason why the speed-up flattens out is two-fold. First, the management overhead to trigger job executions and transfer data to/from the nodes increases with the number of nodes available to execute the computation. Second, the reduction in execution time largely depends on parallel execution of the computationally intensive `mProject` jobs. Whereas the median execution time of a single `mProject` job is around 1500 s (irrespective of the number of nodes used), the other seven job types have a much smaller execution times, with medians of around 65 s (for `mBackground`) and 15 s (for `mDiffFit`) for the two job types with a non-constant number of instances. The most significant contributing factor to speed-up for the Montage 1.0 degree workflow lies in the parallel execution of `mProject` jobs. It is seen that far less gain can be achieved by executing the remaining jobs on an increased number of nodes.

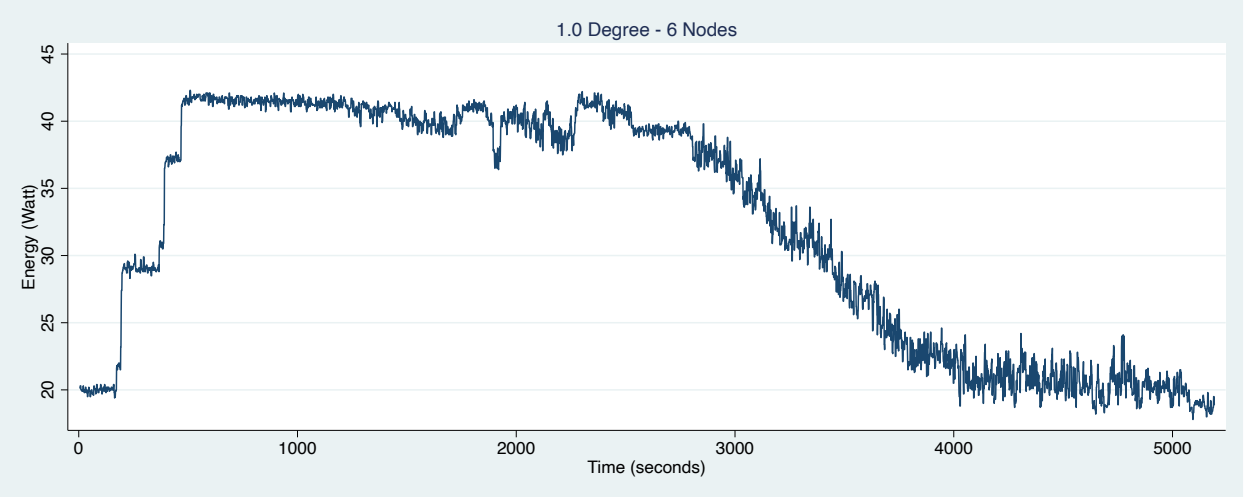
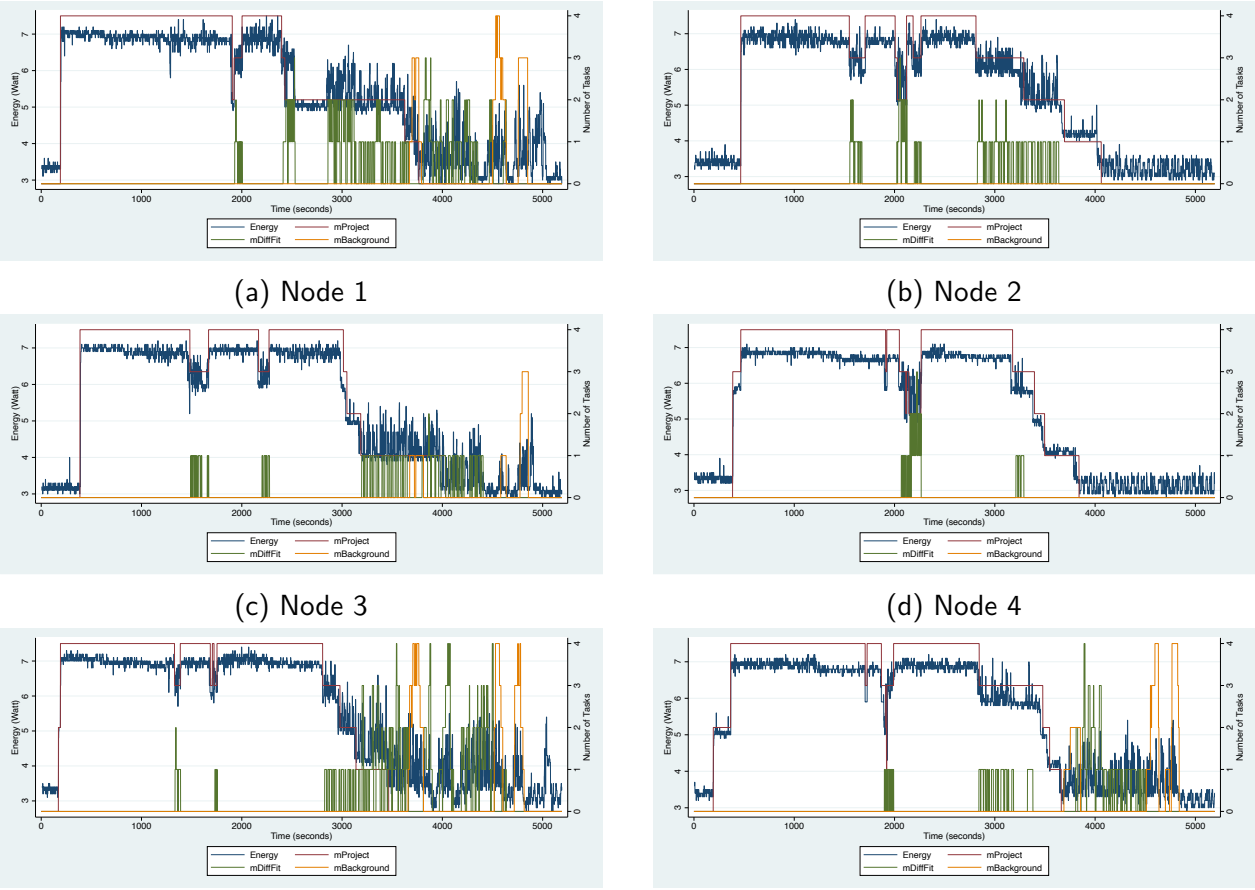


Figure 3.9: Energy consumption of a Montage 1.0 degree workflow on a 6 node cluster.

When looking at total energy consumption, it increases steadily in a mostly linear way as the number of nodes increases. A single node uses a total of around 35 Watt-hours, and each additional node adds around 2–3 extra Watt-hours. It should be noted that this is dependent on the length of the execution time, which is reducing with each node being added, as illustrated on the left Y-axis. This demonstrates that the main difference in total energy consumption is the overhead for each node. These results indicate that increasing the number of nodes reduces the total time taken, but also comes at the cost of cluster node overhead, at least for a Montage 1.0 degree workflow.

To further investigate the characteristics of the Montage 1.0 degree workflow execution, Figure 3.9 illustrates the total energy consumption of the workflow on a 6-node cluster. For each point in time (X-axis), the current energy consumption for all 6 nodes is summed up and presented on the graph. It illustrates that the energy consumption varies significantly over the lifetime of a single workflow's execution. The initial 'stepped' delay is due to the time it takes to obtain the data in the correct location before computation can start. The cluster is being maxed out between 500 and 2800 s by mainly executing `mProject` jobs. From this point onward, many job dependencies are being completed and less-CPU-intensive jobs are executed, hence there is a reduction in the overall energy consumption. A statistical analysis has shown that there is a strong correlation between the total number of `mProject` jobs being executed and the total energy consumption (Pearson's correlation coefficient $r = 0.995$ with $p < 0.01$), but only very weak correlations between energy consumption and the execution of the other workflow jobs.



(a) Node 1 (b) Node 2

(c) Node 3 (d) Node 4

(e) Node 5 (f) Node 6

Figure 3.10: Energy and Job Execution analysis of 1 degree Montage Workflow across six nodes

A further analysis shows that the cluster nodes are not all executing the same workload. Consider Figure 3.10 that illustrates the energy consumption and jobs split for individual cluster nodes for a Montage 1.0 degree workflow on a 6-node cluster. For each of the sub-graphs, the left-hand Y-axis is the energy consumption at a point in time (represented by the blue 'energy' line). The right-hand Y-axis is the number of jobs of the three types of Montage jobs

(mProject, mDiffFit, and mBackground) at specific points in time. It shows that throughout the workflow, the pattern is generally the same, with the larger energy consumption being correlated to CPU-intensive jobs. The latter parts of the workflow execution are primarily dedicated to data aggregation, which is distinctly less CPU-intensive and, therefore, uses less energy.

Figure 3.10 further illustrates that the execution of mProject jobs (the 'red' lines in the sub-graphs) is "interrupted" by mDiffFit jobs (the 'green' lines in the sub-graphs). This is because the scheduling of jobs follows a *depth-first* strategy that prioritises jobs "lower" in the workflow DAG. Consequently, if mProject and mDiffFit jobs are queued, priority is given to the mDiffFit jobs and they are executed before any queued mProject jobs. This effect is evident for nodes 2 and 4.

3.4.4 Montage 0.5 Degree Workflow Execution Results

To investigate the energy cost of much smaller workloads on varying-sized clusters, a series of experiments with a Montage 0.5 degree workflow was performed. As given in Table 2.2, the complexity of the Montage Workflow has far fewer mProject, mDiffFit, and mBackground jobs that need to be executed, and, consequently, a much smaller execution time is expected.

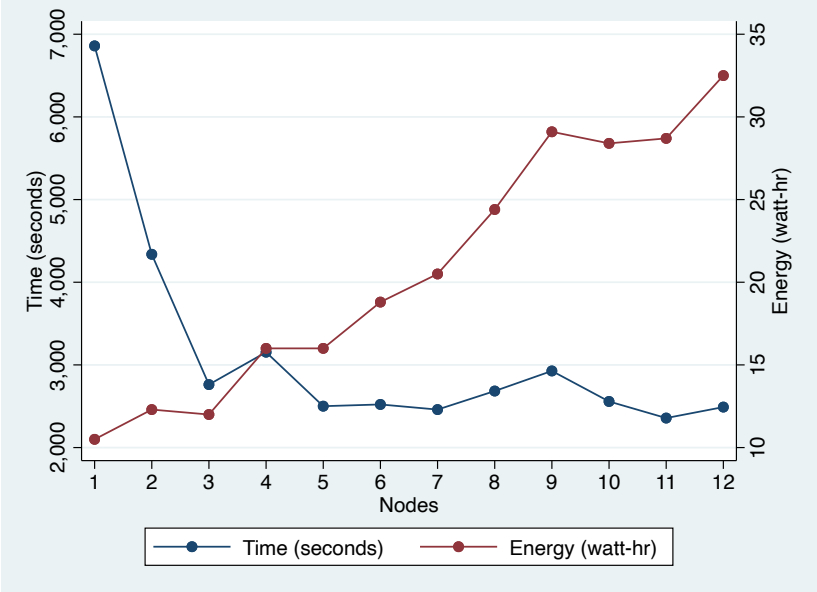


Figure 3.11: Montage 0.5 Degree Workflow Execution on Varying Cluster Size

Figure 3.11 illustrates the execution time vs. energy consumption result of our experiments with a Montage 0.5 degree workflow. Similar to the 1.0 degree workflow, a reduced execution time for 2 nodes (4187 s) and 3 nodes (2614 s) compared to 1 node (6696 s), resulting in a speed-up of 1.60 and 2.56, respectively, is observed. This is mainly due to the 12 computationally-intensive mProject jobs being executed on 8 available threads (for 2 nodes) and 12 threads (for 3 nodes).

However, from 4 nodes onward, no further reduction in the execution time is observed. It remains mostly constant. A closer inspection of the data showed that due to job dependencies, jobs are only ever run on the *same 3 nodes* and any further available nodes remain inactive during the duration of the workflow execution.

When looking at total energy consumption, it increases steadily in a mostly linear way when more nodes are added (similar to the 1.0 degree workflow). The main difference is that, as discussed above, only 3 nodes are used for workflow execution—the additional nodes run "idle", but contribute to the overall energy use. The workflow is *too small* for a cluster of more than 3 nodes and any additional nodes just increase the overall energy used but do not reduce the execution time.

3.4.5 Montage 1.5 Degree Workflow Execution Results

To investigate the cluster performance for a much larger workflow, a series of experiments with a Montage 1.5-degree workflow were performed. Figure 3.12 illustrates the execution time vs. energy consumption of a 1.5 degree montage workflow on varying sizes of clusters.

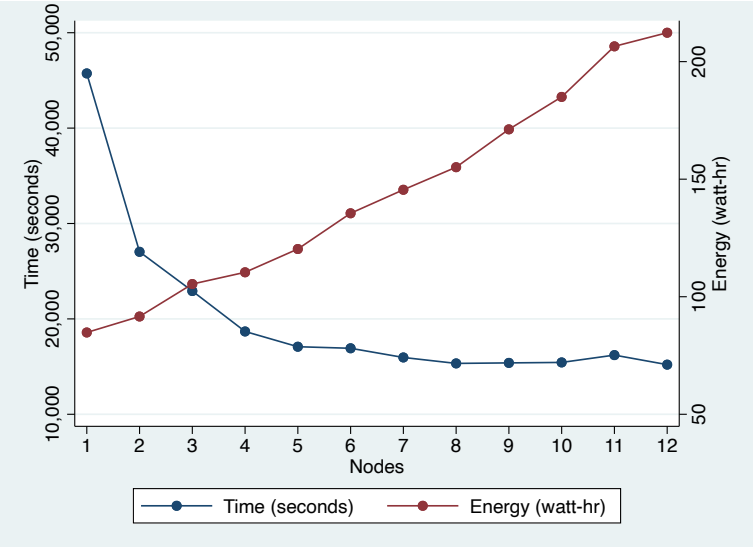


Figure 3.12: Montage 1.5 Degree Workflow Execution on Varying Cluster Size

A very similar pattern as shown in Figure 3.8 can be observed: a speed-up in execution time from 1 node (45,548 s - 12 h and 39 min) to 6 nodes (15,868 s - 4h and 24 min), followed by a flattening of the execution time curve with only marginal improvements. The maximal speed-up observed (for 12 nodes) is 3.34 compared to the execution time for 1 node. As there are 108 computationally-intensive but parallelisable `mProject` jobs to execute, a much better speed-up is anticipated than what is observed. Let's look into the reasons for this behavior in more detail.

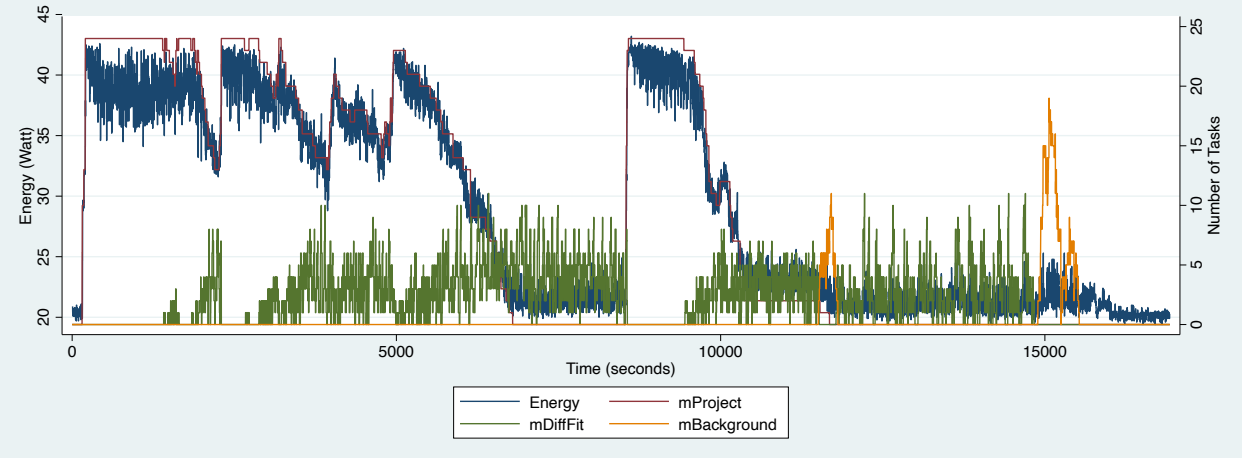


Figure 3.13: Execution of a Montage 1.5 degree workflow on a 6 node cluster.

Figure 3.13 illustrates the energy consumption and jobs split for a 1.5 degree workflow on a 6 node cluster. In contrast to Figure 3.10 where the energy consumption and job divided for each of the 6 nodes was illustrated, Figure 3.13 presents a summation across all nodes. The depth-first execution strategy adopted by the workflow engine can be observed through the interleaving of `mProject` and `mDiffFit` jobs (starting at 1496 s). The higher priority of `mDiffFit` jobs results in a complete pause of `mProject` jobs for several minutes (between 6793 and 8530 s), where only queued `mDiffFit` jobs are executed. Once they are completed, the execution of the remaining 34 `mProject` jobs resumes. Montage workflows for 0.5 degree and 1.0 degree did not result in an execution behavior where `mProject` jobs were paused across all nodes and for such an extended period of time, respectively.

All `mDiffFit` jobs depend on the completion of 2 specific `mProject` jobs only. Hence at 6793 s, many `mDiffFit` jobs are ready for execution. However, during this period, a median of 4 `mDiffFit` jobs are executed (a maximum of 10) but there are always many more queued jobs that, based on this interpretation, should be executable. The execution time for all `mDiffFit` jobs stays stable around 10,000 s, irrespective of the number of available nodes. There is less time for `mDiffFit` jobs to interleave with `mProject` (the times both types of jobs are being executed together reduces steadily the more nodes are added) and the gain of a faster execution of all `mProject` jobs is offset by the reduction in overlapping executions—the times when only `mDiffFit` jobs are executed increase as more nodes are added. As these two job types combined contribute the most towards the overall execution time, this explains the lack of significant reduction in execution time from 6 nodes onwards.

Activities	1 Node		3 Nodes		6 Nodes		9 Nodes		12 Nodes	
Total (seconds)	45,584		22,038		15,868		13,916		13,654	
mProject > 0	44,193	96.9%	20,097	91.2%	9,787	61.7%	6,586	47.3%	5,600	41.0%
mProject > 0 alone	33,791	74.1%	10,362	47.0%	3,643	23.0%	1,584	11.4%	1,918	14.0%
mDiffFit > 0	9,774	21.4%	10,176	46.2%	10,217	64.4%	10,153	73.0%	9,014	66.0%
mDiffFit > 0 alone	420	0.9%	724	3.3%	3,852	24.3%	4,787	34.4%	5,044	36.9%
mProject > 0 & mDiffFit > 0	9,354	20.5%	9,449	42.9%	5,970	37.6%	5,002	35.9%	3,682	27.0%
mProject > 0 or mDiffFit > 0	44,613	97.9%	20,824	94.5%	14,034	88.4%	11,737	84.3%	10,932	80.1%
mBackground > 0	1,031	2.3%	970	4.4%	933	5.9%	892	6.4%	860	6.3%
mBackground > 0 alone	292	0.6%	565	2.6%	688	4.3%	786	5.6%	781	5.7%

Table 3.1: Execution Pattern Analysis of Montage 1.5 degree workflow

Table 3.1 summarizes execution patterns of the three job types with a varying number of instances (`mProject`, `mDiffFit` and `mBackground`). More specifically, the table shows for how many time intervals some of these jobs are running—this gives an idea of how long it takes, for example, to run all 108 `mProject` jobs—as well as the percentage thereof of the duration of the entire workflow. From 3 nodes onwards, an almost linear speed-up for the execution time of all `mProject` jobs is observed. Hence, the workflow engine does exploit the increasing number of available compute resources quite effectively for `mProject`. A steady decline in the execution times for all `mBackground` jobs was observed but not to the same extent as for `mProject`. Therefore, the experiments for a Montage 1.5 degree workflow showed that only `mProject` jobs were effectively parallelised when more nodes were added. There was a marginal speed-up for `mBackground` but a mostly constant execution time for `mDiffFit`, respectively.

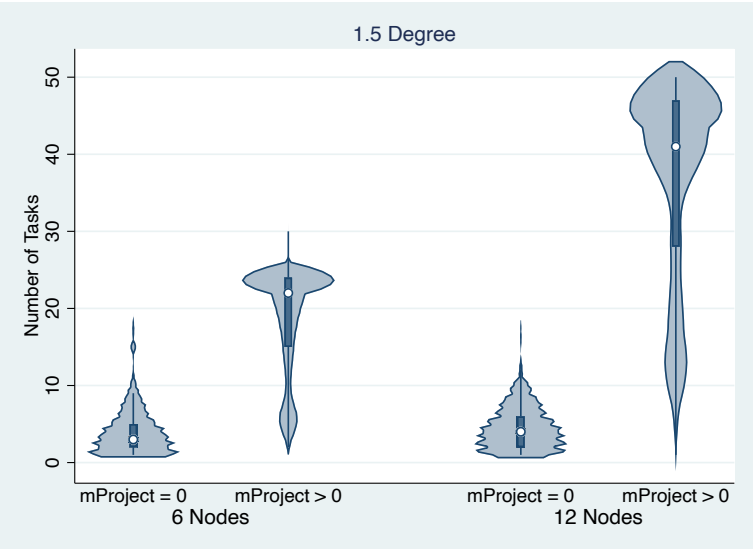


Figure 3.14: Analysis of mProject Job Execution for 6 nodes vs. 12 nodes

This is further illustrated in Figure 3.14. The figure shows the number of active jobs (or tasks) when at least one `mProject` job is running (*i.e.*, `mProject` > 0) compared to the number of active jobs when no `mProject` job is running (*i.e.*, `mProject` $= 0$), for both 6 (maximum of 24 threads) and 12 nodes (maximum of 48 threads), respectively, in the form of a Violin plot. Figure 3.14 clearly shows that the workflow engine gets close to using up all available compute threads when `mProject` jobs are being executed (Median of 22 and 41 for 6 and 12 nodes, respectively) but fails to do so when no `mProject` jobs are running (Median of 3 and 4, respectively). As the execution time for all `mProject` jobs gradually decreases with the addition of more nodes, the cluster becomes increasingly underutilized.

Apart from `mProject`, none of the Montage jobs effectively uses the available resources. Fewer nodes are active outside `mProject` execution. For a both a 6 and 12 nodes cluster, 3 nodes would suffice about 70% of the time with 4–5 nodes required for the remaining 30%. This explains the almost linear increase in the total energy consumption (*cf.* Figure 3.12)—additional nodes are used extensively whilst `mProject` jobs are running, but are either used scarcely or not at all otherwise. They still contribute to the overall energy consumption even when running "idle"—an energy-aware scheduler could either switch these nodes off or make them available for other users.

3.4.6 Discussion

An experimental evaluation of the Montage workflow execution on a small board compute cluster was presented in this section. Analysis of data produced through these experiments has unveiled some interesting results, which are discussed in the remainder of this section.

Impact of Cluster Size

The presented experiments have shown that, as expected, increasing the cluster size has a significant impact on the computation time for workflows with many parallelisable jobs. Increasing the size of the cluster from 1 node (4 threads) to 2 nodes (8 threads) will result in an almost halving of the computation time (ie linear speed-up). This pattern generally holds for all workloads and extends to 3 and 4 cluster nodes. These gains due to increasing cluster size do not continue past around 5–6 cluster nodes (20–24 threads) for the chosen workflows, though. Increasing the number of cluster nodes beyond this has a negligible impact on the overall computation time of the workflow. This is the case even with large workflows with a large number of parallelisable jobs. This pattern can generally be attributed to several factors, including increasing dependencies between workflow jobs, the overhead of coordinating more cluster nodes, and the increased data transfer needed. It can be concluded that, for any workflow that requires some level of coordination and/or synchronization among jobs, there will be a point where adding further compute nodes will only result in a marginal reduction in computation time.

Impact of Cluster Configuration

The experience of configuring and analyzing various cluster configurations has shown that many factors affect computation performance and energy. Configuring a cluster to be efficient requires considerable effort. In this paper, the experiments used the default configuration for Condor, and an effort was made to minimize the data transfer needed for jobs to execute. As all nodes were configured using NFS, minimizing transfers had a significant impact on the overall performance. Experiments have shown that when considering energy, the overhead of running nodes has the most significant impact for larger cluster sizes, as there are diminishing returns with regard to run-time performance. From these experiments, it can be concluded that for most energy efficiency, it is best to either use a node to the maximum or have this node not present at all. Consequently, this may require further work in adjusting workflow scheduling to make them more "energy-aware."

Impact of Workflow Structure

The results presented in this paper show that the most significant impact on the workflow execution time is the structure and size of a workflow. Increasing the size of the workflow will increase the workload and the computational requirements. However, increasing the workflow size also increases the number of jobs that integrate and, therefore, depend on the results from previous jobs. Overall, when profiling jobs, the most important are computationally intensive jobs as these not only require the most computation, but also delay dependent jobs the most.

Comparing Performance vs. Energy

The focus of this paper is to analyze the impact on performance and energy of varying sizes of clusters and workflows. The expectation was that there would be a trade-off between performance and energy consumption. Although this is true for small clusters, this does not continue for larger clusters. For larger cluster sizes, regardless of workflow size, increasing the number of nodes does not continue to improve the performance of workflows beyond a threshold. As illustrated in Table 3.1, adding further nodes to our 12-node cluster will not result in an improvement of the overall run-time performance for a Montage 1.5 degree workflow. It is stipulated that a similar pattern will emerge if the complexity of the Montage workflow is increased to 2.0 degrees and beyond.

3.5 Bioinformatics Workflow Energy Evaluation

In this section, an experimental evaluation of the performance and energy consumption of a scientific workflow in the domain of Bioinformatics on a low-power cluster is presented. The experiments vary the workflow size and cluster size to investigate how different factors affect the performance and energy consumption, respectively. The experiments in this section use the experimental setup as described in Section 3.3.

3.5.1 Workflow Description

The Bioinformatics workflow used for the experimental evaluation in this section is based on the data collected by the 1000 Genomes Project [174, 2]. Figure 2.8 illustrates a directed acyclic graph (DAG) of the Bioinformatics workflow being evaluated in this section. The DAG has three levels of jobs which have dependencies from prior levels. For example, the `individuals` jobs have no dependencies, but prevent the dependent `individual_merge` jobs from executing until they have finished. Similarly, a `frequency` job can only be executed once all dependent `sifting` jobs are completed. Similar to Section 3.4, Figure 2.8 only shows the computation jobs that execute on cluster nodes and not the jobs that are running on the master node.

3.5.2 Workflow Characteristics

The Bioinformatics workflow has many characteristics which are specific to the workflow. In particular, there are two different input variables that can be controlled by the user—the size of workflow (the data to be computed) and the number of parallel jobs. Depending on the user's need and the resources available there can be more or less of these jobs. The combination of the number of jobs and the size of workflow directly relates to the queuing of the jobs, the execution time, and the energy consumption of the workflow.

The computing nodes (RPis) used in this experiment have 2 GB of RAM available on them (see Section 3.3). The workflow loads the whole data-set into memory. This meant that the workflow execution was limited by the memory capacity of the nodes. After removing the memory used by the OS and other necessary processes, around 1.6 GB of memory is available for computation. This translates to less than 400 MB of RAM for each thread in the computing nodes. This was the highest limit of data that the workflow could process. After analysis of the data, it was found that 1 MB of data was equal to 100 lines of individual data in the file. After testing a few different sizes of workload, the sizes of data to be used in experiments were finalized to 10 k for small workload, 20 k for medium workload and 30 k for large workload.

3.5.3 Workflow Execution Results on a Single Node

The results of varying the number of parallelisable jobs on a single node cluster for a large workflow (30k data) are presented in this section. This is mainly performed to showcase the optimal number of parallel jobs to be executed on a single node when comparing the time of execution with the energy consumption of the workflow.

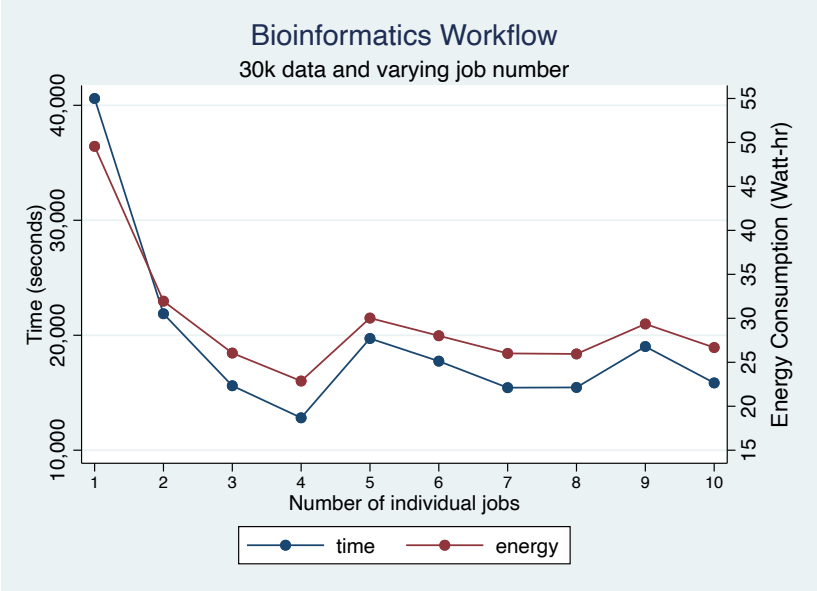


Figure 3.15: Bioinformatics Workflow Analysis of Varying Number of Jobs on a Single Node

Figure 3.15 illustrates both the execution time and energy consumption of a large workload Bioinformatics workflow on varying numbers of parallel jobs. The left Y-axis denotes time in seconds and the right Y-axis indicates the energy consumption used by the node for the duration of the execution. It is important to note that the energy data does not take into account the energy of the master node. The X-axis indicates the number of parallel jobs being queued on the node. The results shown in Figure 3.15 illustrate that the execution time decreases with the number of parallel jobs until all 4 threads are being used. The time decreases from 40,595 s (approximately 11 h and 16 min) to 12,821 s (approximately 3 h and 33 min), hence resulting in a speedup of 3.16. As it can be seen, the energy consumption closely relates to the number of jobs being executed. After four jobs, the time and the energy consumption deteriorates from 12,821 s (approximately 3 h and 33 min) to 19,721 s (approximately 5 h and 28 min) which is a 53% increase. Similar results can be found for energy consumption as well.

The major reason for this observation can be that the compute node only has 4 hardware threads for parallel computation and any jobs more than four leads to queuing and higher overheads of scheduling jobs, transferring files and merging. The optimal performance of the nodes in a cluster can be achieved when the number of jobs to be scheduled on any node equals the number of hardware threads available [92]. This is also illustrated in Figure 3.15 and concludes that to

achieve the sweet spot between energy consumption and execution time, the computing power of a single node needs to be completely utilised before scheduling jobs on any other nodes.

3.5.4 Workflow Execution Results for a Small Workload

To investigate the energy consumption of a small workload (10k data computation) on varying size of cluster, a series of experiments with varying data and cluster sizes were performed using the Bioinformatics workflow. As seen from Table 2.3, the number of jobs do not change as compared to any other workload but the amount of computation by each job increases when the workload increases. In this section, the execution results of the Bioinformatics workflow with a small sized workload (10 k data with 50 individual jobs) being executed on 1 to 12 nodes is presented.

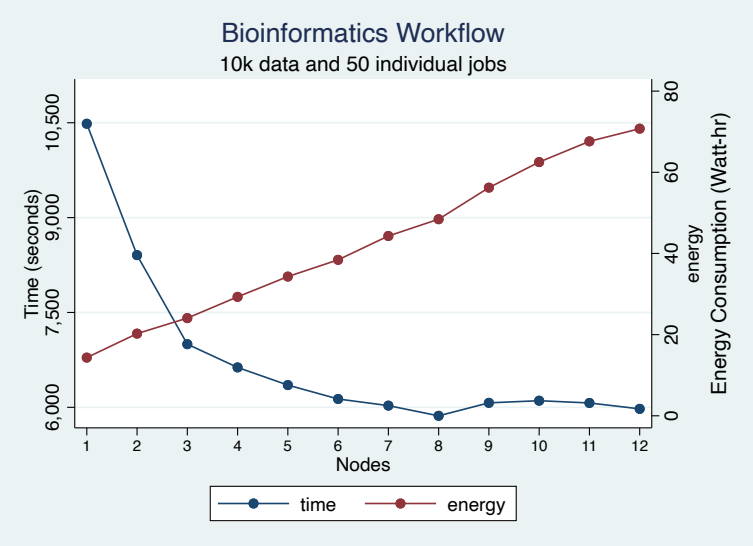


Figure 3.16: Bioinformatics Workflow Analysis -Time vs. Energy Consumption for 10k data

Figure 3.16 illustrates both the execution time and energy consumption of a Bioinformatics workflow with 10 k data workload on varying size of cluster. The left Y-axis is the execution time in seconds. The right Y-axis is the energy consumption of the nodes in Watt-hours. The X-axis indicates the cluster configuration, *i.e.*, how many nodes were used to compute. Moreover, no jobs on the master node were considered as they were mainly folder creation and file transfers. It can be seen that the execution time decreases with more nodes being used for computation. This is expected as more computation power should result in faster workflow execution. The lowest execution time in the experiment was recorded for 8 nodes as 5866 s (approximately 1 h and 37 min) which was 44.05% lower than that of 1 node (10,484 s; 2 h and 54 min).

A steady increase in energy consumption was observed similar to Section 3.4. A single node used around 14.34 Watt-hours during the computation and every additional node increased the energy consumption by around 4.5 Watt-hours. As the energy consumption takes into account

the execution time of the workflow, the difference between the energy consumption between two nodes is due to the increased overhead of computing on additional nodes. The results illustrate that faster execution of workflow by increasing the number of nodes comes at a cost of increased energy consumption. This led to further investigation on finding the sweet spot between execution time and energy consumption in the following sections.

The increase in energy consumption is almost linear and further investigation into the data shows that the reason for the increase is totally different than that found in Section 3.4. The linear increase is a result of just 1 thread being used and the majority of the nodes being idle during the execution of `individual_merge` job. Further analysis showed that, for 1 node execution, the `individual_merge` job was being executed for 4032 s (approximately 1 h and 7 min) while the total execution time was 10,484 s (2 h and 54 min) for the workflow. This meant that for almost 38.45% of the execution time, the majority of the cluster was idle and consuming energy without contributing to the computation. The results become more prominent as we increase the number of nodes. The total execution time for a 12 node cluster is 5976 s (approximately 1 h and 39 min) and `individual_merge` job monopolises 4132 s (approximately 1 h and 8 min) for itself. This results in the majority of the cluster being idle for 69.15% of the time.

3.5.5 Workflow Execution Results for a Medium Workload

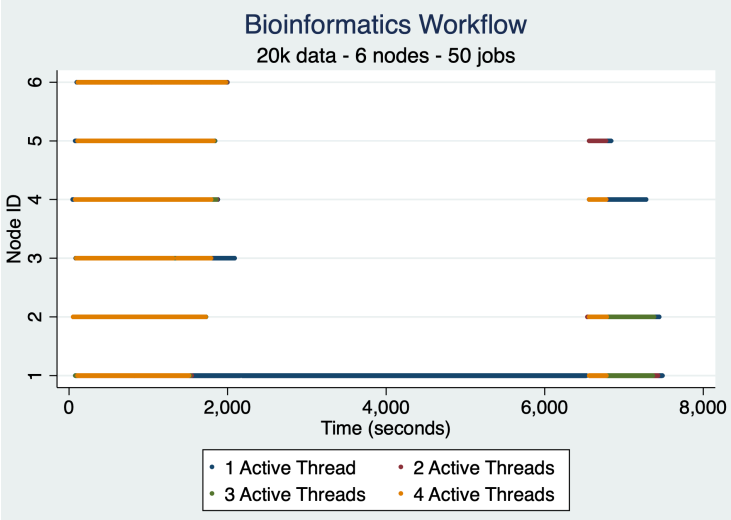


Figure 3.17: Number of Active Threads Analysis for 20k Bioinformatics Workflow on 6 Nodes

An investigation of a medium size workload on the Bioinformatics workflow was conducted using 20 k data points. As given in Table 2.3, the number of jobs stays the same for this set of experiments but the work performed by each job increases by 100% as compared to the small workload. Similar to Figure 3.16, a reduction in the execution time of the workflow was observed when more nodes were added. The execution time gradually decreases till 12 nodes. The maximum difference of

55% increase in time between the 1 node execution (14,924 s—approximately 4 h and 8 min) and the 12 nodes execution (6645 s—approximately 1 h and 50 min) was 8279 s (approximately 2 h and 17 min), resulting in a speed up of approximately 2.25. When analysing the total energy consumption, similar patterns as shown in Figure 3.16 were seen in this experiment. Energy consumption increases by 267.9% for 12 node execution when compared with 1 node execution. This increase is observed as the reduction in the execution time does not compensate for the increase in the averaged energy consumption of the cluster when a new node is added.

Figure 3.17 presents the number of active threads for 6 nodes during the execution of a medium sized Bioinformatics workflow with 50 jobs. The different colors represent the number of active threads on a particular node. As illustrated in Figures 2.8 and 3.17, `individual_merge` job acts as a bottleneck for the workflow and the workflow cannot proceed further unless the job has been completed. This job utilises just 1 thread of 1 node out of all the computing power available. Due to this, during the execution of this job, most of the nodes are idle. They still contribute to the total energy consumption of the cluster but do not help in the workflow execution. This results in perceived increase in energy consumption for the whole cluster where in case only 1 node is being partly used. This leads to the investigation into how to leverage this situation to reduction in the energy consumption of the cluster.

3.5.6 Workflow Execution Results for a Large Workload

To further investigate the performance of the Bioinformatics workflow, a series of experiments were performed with a large workload of 30 k data points. As the number of jobs was the same, this means the computation for each job increased by 200% as compared to the smaller workload. A steady decline in the execution time is observed between 1 node (19,013 s - approximately 5 h and 16 min) and 7 node (7740 s - 2 h and 9 min). A speed-up of 2.45 or a 59.3% decrease in the execution time was achieved from these experiments. After node 7, the execution time stabilises and only 5–6% variation between the execution times is observed. The energy consumption of the experiments followed the same reasoning as Figure 3.16. The nodes are idle for the majority of the time and this leads to the idle energy consumption values tainting the actual energy consumption of the cluster during the computation.

3.5.7 Discussion

An experimental evaluation of a Bioinformatics workflow has been presented in the previous sections. The data obtained from these experiments demonstrated several key findings regarding the workflow and how it is executed. These results are discussed in the remainder of this section.

Impact of Cluster Size and Configuration

The results presented show that the cluster size and configuration significantly impact the execution time of the workflow. Generally, increasing the size of the cluster from 1 node (4 threads) to 2 nodes (8 threads) will result in an almost halving of the computation time but due to the particular characteristics and the bottlenecks of the jobs in this workflow, a non-linear speed-up was observed for the different workload executions. For example, increasing from 1 node to 2 nodes resulted in a 19% and 29% decrease in execution time for 10 k and 20 k data workload, respectively. Similarly, a 35% decrease is observed for 30 k data workload.

The gains obtained are marginal and not prominent past 5–6 cluster nodes (20–24 threads) for the chosen workflow. This can be due to a large number of factors, including increased overheads, dependencies, and data transfer. As for the energy efficiency of the cluster, efforts were undertaken to reduce the data transfer and dependencies of the jobs without affecting the underlying workflow or condor configurations. These proved to be unfruitful as the gains were quickly diminished by the bottlenecks inherent to the workflow.

It can be concluded that for any workflow to maximise the execution time and energy efficiency, it is best to utilise all the available computing power of a single node before scheduling jobs on other nodes. A workflow with inter-dependent jobs will eventually come to a point when adding more computation power will result in a very small increase in performance. Moreover, for any workflow with a bottleneck, further work is required to make it "energy aware". This can include altering the scheduling of the jobs, changing the default dependencies, and changing the cluster configuration.

Impact of Workflow Structure

The workflow structure has a huge impact on all the aspects of its execution. A poorly implemented workflow can be very inefficient leading to long execution times and wasted resources. In this paper, the Bioinformatics workflow has a constant number of jobs for any workload. This meant that increasing the workload led to a direct increase in the computation required. A spike in the energy usage of the cluster is observed when a computational job is being executed. The results presented in this paper show that there is a huge bottleneck during the execution of a single job in the Bioinformatics workflow. This leads to a delay in the dependent jobs which results in a waste of energy and computing resources. This bottleneck can be removed by energy-aware execution and scheduling of the workflow. This will require changes to the workflow and the underlying scheduler. These changes and their gains are beyond the scope of this paper and will be researched in future.

Comparing Performance vs. Energy

The experiments performed in this section present the impact of varying sizes of cluster and workflow on the performance and energy consumption of the workflow. The expectation was that there would be a gain in performance of the workflow at an expense of the energy consumption. This was observed to be true but an in-depth analysis of the workflow showed that there is a way to optimize the workflow to gain more energy savings or execution time. For larger cluster sizes, no considerable performance gain was observed for the increase in energy expenditure. It can be stipulated that similar results will be observed on the execution of the Bioinformatics workflow with different workloads.

3.5.8 Results Compared to the Literature

In this section, the approach taken is to analyze the energy usage of the workflow as a whole. This is in contrast to previous work which focused on the individual jobs and the factors affecting their execution [55]. Results were consistent between both approaches with both identifying that execution can be optimized for energy without a huge reduction in performance by identifying and reducing the overheads and dependencies between the jobs.

The workflows chosen for this evaluation reflect two distinct areas of scientific computing and have different computational characteristics. Montage is I/O intensive and Bioinformatics is CPU intensive [238]. There is not much overlap between these domains and hence the results obtained can be considered relatively generic. Similar approaches have been taken where two different workflows with different characteristics are analyzed to provide conceptual results that can be applied generically to any workflow executions [51, 199]. The experiments are performed based on varying the size of the workflow and the workload. This is performed mainly to analyzes the effect of execution time and energy consumption of varying complexities. Previous studies have focused on executing three workflows on a single cluster configuration [239, 240]. Their results are similar to the ones obtained in this study. This study expands on these previous studies by varying the configuration of the cluster and the workflows. This provides a comprehensive set of results which can help generalize the concepts to multiple workflows. In this section, a detailed analysis of the computation/energy characteristics alongside analyzing the detailed structure of the workflow is performed. Prior work has focused on the structure of workflows along with the inter-dependencies of the jobs to find the trade-off between make span and cost [54]. Analyzing the workflow as a whole along with its structure has allowed for understanding the workflow in depth and similar factors affecting the performance and energy of the workflow.

3.5.9 Energy-Aware Workflow Execution

The results from previous experiments motivate a need for *energy-aware execution* of workflows on compute clusters to result in better trade-offs between run-time performance and energy consumption. In this section, the cause is further supported through detailed analysis of the workflows and proposing policies which can act as a rule base for development of a novel energy-aware workflow execution software/scheduler. It is shown that the policies, when implemented, can help in reducing the energy footprint of the workflows by theoretically implementing them in the experiments.

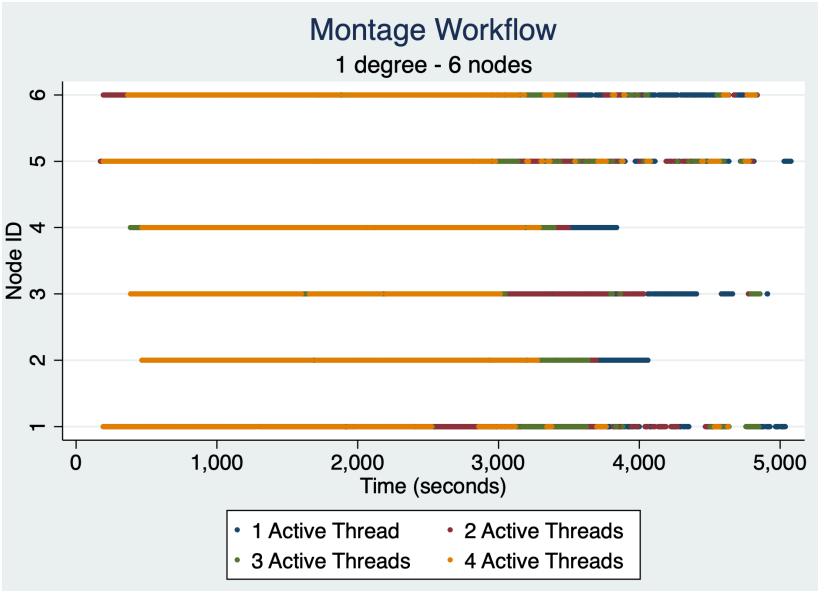


Figure 3.18: Number of Active Threads Analysis for Montage 1.0 degree Workflow on 6 Nodes

Workflow Analysis

To complement what was concluded in Sections 3.4 and 3.5, the number of *active threads* per node during the execution of the workflows is investigated as a measure of how much the resources of the cluster are being utilized (*cf.* Figures 3.17 and 3.18). For both figures, usage of all 4 threads is indicated by 'orange' dot and usage of 3 threads is indicated by 'green' dot. 'red ' and 'blue' colored dots are used to indicate usage of two and one threads, respectively. The y-axis denotes the individual node and the x-axis the timestamp at any given instant. The analysis only considers the jobs that were executed on the nodes and not the local jobs, such as file/folder creation and file transfers.

For the Montage workflow, the 1 degree workload on a 6 node cluster was analyzed. During the first part of workflow execution where mostly `mProject` jobs are executed, each node uses all 4 available threads. However, in the second part of the execution (*i.e.*, *post* `mProject`), neither 4 nor 3 threads were being used completely, but most nodes have only two or even one

active threads. For the Bioinformatics workflow, a workload of 20 k data was provided to the workflow to be executed on 6 nodes. As expected, during the execution of `individuals` jobs, all threads from all nodes were being used and the number of jobs exceeds the number of available threads. However, during the second part of execution, only 1 thread was being utilized. This job `individual_merge` takes up the majority of the workflow execution time in which most of the nodes are idle. During the third part of workflow execution, 4 nodes are being utilized with a mix of 3 and 2 threads each.

The execution behavior of these two workflows is not yet fully understood as there are instances when the jobs could be queued better to save more energy or time. For example, during the second part of the Montage workflow, the 17 jobs to be executed could have been scheduled on just 4–5 nodes and 1 node could have been turned off to save energy. Fine grain analysis of the job dependencies, and scheduler settings is required to further understand and explain a particular execution of workflow. In conclusion, using a set of rules (or "policies") to govern the execution of jobs and the configuration of cluster can help improve the energy consumption of the workflow. These policies can vary in a wide range from scheduling jobs on particular nodes to changing the configuration of the cluster based on the energy budget of the workflow, respectively.

Use Case Analysis of Energy-Aware Policies

To investigate the actual impact of the different policies, the execution data of one use case of each workflow is analyzed. These policies are explained further in this section. The analysis of the data can be seen in Figures 3.19 and 3.20).

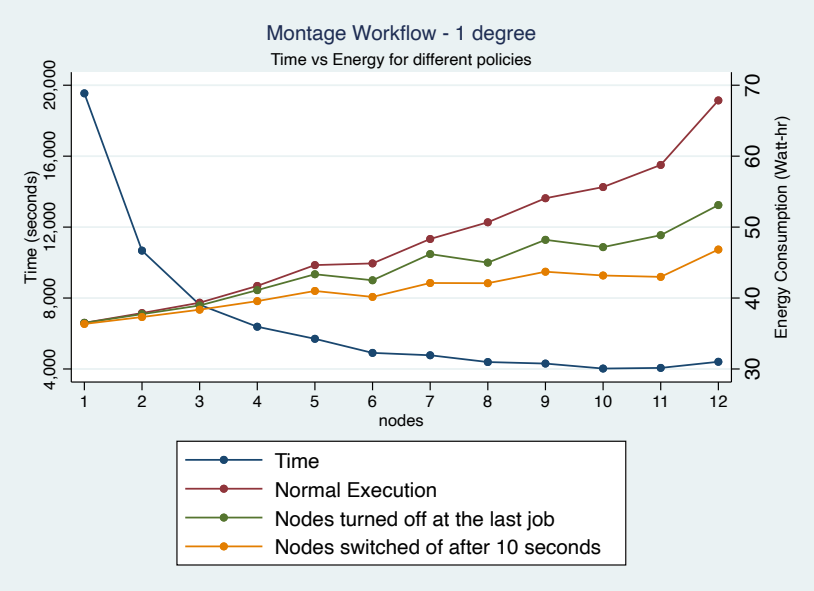


Figure 3.19: Time vs. Energy Analysis for Montage Workflow as per Different Policies

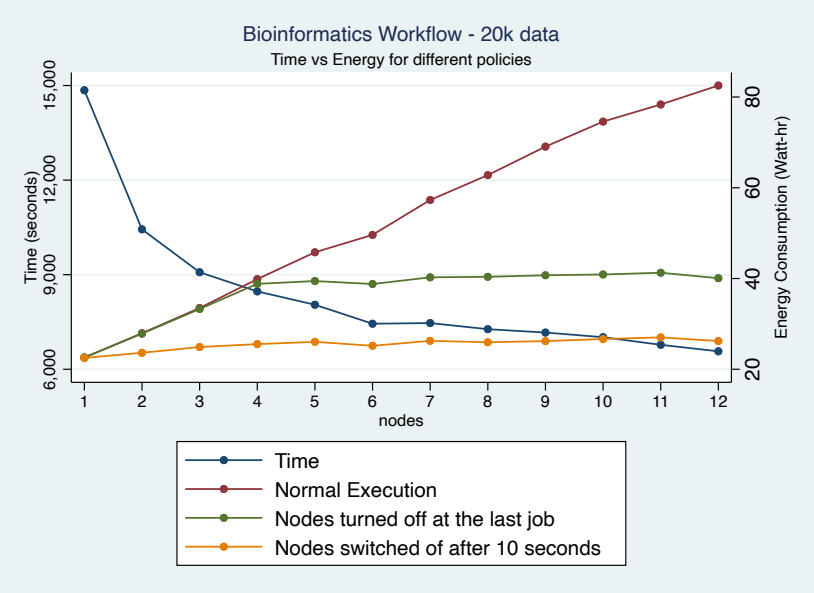


Figure 3.20: Time vs. Energy Analysis for Bioinformatics Workflow as per Different Policies

The workload being used to compare between the policies for Montage and Bioinformatics workflows are 1.0 degree and 20 k data, respectively. For both figures, the x-axis denotes the number of nodes. The left and right y-axis denote the execution time of the workflow and energy consumption for the particular policy, respectively. The "maroon" colored line depicts the energy consumption when the workflow is executed normally as their authors intended it to be. There is no changes made to the workflows. The "green" line denotes the execution of workflow in which the nodes are turned off at the end of their last job. This includes the idle time in between the first job and the last job on each node. Finally, the "yellow" color line shows the energy consumption when the policy to turn nodes off after 10 s of inactivity is applied. This gives us a lower bound of energy consumption as we do not incorporate the impact of re-starting nodes (if required). This analysis only considers the jobs which were executed on the nodes and not the jobs on the master node such as file/folder creation and file transfers.

In the case of Montage, the experiments have shown that the "optimal" number of nodes is not a constant and may vary during the execution of a workflow (as is shown Figure 3.18). An intelligent scheduler could exploit this and use fewer nodes for parts of a workflow execution without impacting overall run-time performance. This can reduce the energy consumption as shown in Figure 3.19. For a Montage 1.0 degree workflow on 6 nodes, for example, if the impact of nodes 2 and 4 were excluded once they become inactive (*cf.* Figure 3.18), a reduction of approximately 10% in overall energy consumption would be achieved. For a Montage 1.5 degree workflow on 12 nodes, a similar exclusion of inactive nodes would result in an approximate 25% reduction. Figure 3.19 shows the same and can help in identifying the optimal cluster configuration and execution time. The maximum reduction in energy consumption that was seen during the experiments was 30% for 12 nodes.

The Bioinformatics workflow has a single process that runs on its own for a considerable amount of time with all other nodes being inactive. Some of these nodes will be needed again in the third part of the execution. This property of the workflow can be exploited and can lead to significant savings in energy consumption as illustrated in Figure 3.17. For 12 nodes, for example, a 68% energy saving is achieved when the nodes are turned off after 10 s of inactivity.

The observations show that for any size of workflow, there is a point of cluster size which is optimal from both a performance and an energy consumption perspective. Moreover, the scheduling of jobs can be further optimized to provide a boost in the energy savings. The traditional workflows are not fully optimized to save energy or time. These observations motivate development of an *energy-aware architecture/program* that will make use of policies and smart scheduling to result in better trade-offs between run-time performance and energy consumption. This will also result in improved energy costs of a workflow while preserving performance.

3.6 Energy Consumption Analysis of Docker Workloads

Section 3.3 presented the experimental setup for the experiments conducted throughout this study. While the primary focus of this research centers on scientific workflow energy consumption, it is essential to establish the foundation and validate the energy monitoring framework using more controlled and well-understood computational workloads before applying it to the complex, multi-task nature of scientific workflows [92, 64]. Docker containers provide an ideal testing environment for this validation due to their deterministic resource usage patterns, standardized execution environments, and widespread adoption in both research and production computing environments [115, 118].

In this section, a comprehensive set of workloads and experiments were conducted on different web server and database Docker containers to test their performance characteristics and measure their impact on energy consumption [241]. The section also serves to confirm the accuracy and reliability of the energy monitoring device (smart energy-monitoring power plugs) being used throughout this study [237]. The reliability and accuracy of the developed monitoring system is systematically tested and validated through controlled experiments with known computational loads [64]. All the experiments conducted in this section utilize the hardware and software setup from Experiment Setup 2 as described in Section 3.3.2 and illustrated in Figure 3.3.

3.6.1 Measuring the impact of CPU load on the energy consumption

An initial experiment to measure the change in energy consumption of the host machine with respect to varying workloads was conducted to confirm the proper and accurate functioning of the framework proposed in Section 3.2. Energy consumption data were collected every 500 ms and the load on the host machine was gradually increased and decreased by maxing out the threads using the command '$yes > /dev/null$'. Every time this command is executed, a thread on the host machine is maxed out that outputs the string 'yes' continuously.

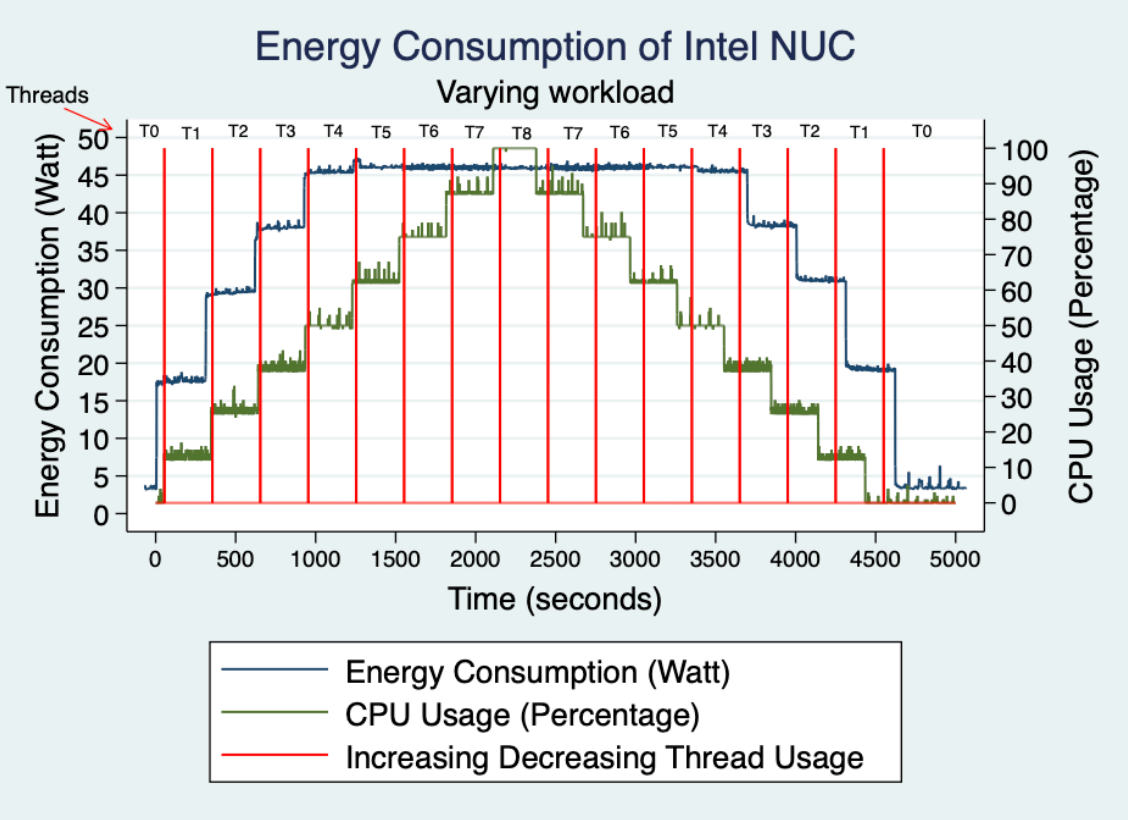


Figure 3.21: Energy consumption of Docker Container under Stress

The execution data of the experiment is shown in Figure 3.21. The orange lines denote the number of threads being maxed out (also denoted by T0 - T8 at the top). The X-axis indicates the execution time (in seconds). The left Y-axis denotes the instantaneous energy consumption (in W) of the host machine at that particular execution time. The right Y-axis shows the CPU usage (in percentage).

Based on the data from Figure 3.21, the energy consumption of the host machine increases gradually until the 4 threads are maxed out on the host machine. After that, even though the CPU usage is increasing as more threads are maxed out, the energy consumption is stable. This is expected as the host machine is quad-core (i.e. 4 cores and 8 threads). This means that 4 CPUs can perform 8 tasks parallelly. The base energy consumption of the host machine is around 3.75

W and the maximum it consumes is around 47 W when the load is maxed out (see Figure 3.3). The increase in energy can be seen before the increase in CPU usage with a delay of around 2 sec. This experiment successfully shows that smart plugs (see Figure 3.4) can be used to accurately measure the energy consumption of any device connected to it. Also, this provides evidence that it is recommended to use all 8 threads on the host computer as compared to just using 4 threads.

3.6.2 Energy Overhead of Docker

Standard execution of any workload on Docker containers includes starting the Docker daemon and the service in the background and then building and/or running the containers to perform a certain task. This experiment aims to identify the initial overhead of starting the Docker daemon and service in the background. The data from this experiment showcases the initial impact of Docker on the energy without running any specific workload on the computer. Based on the data from this experiment, if the initial energy overhead of Docker is too high then it might be considered during the calculation of any future experiments.

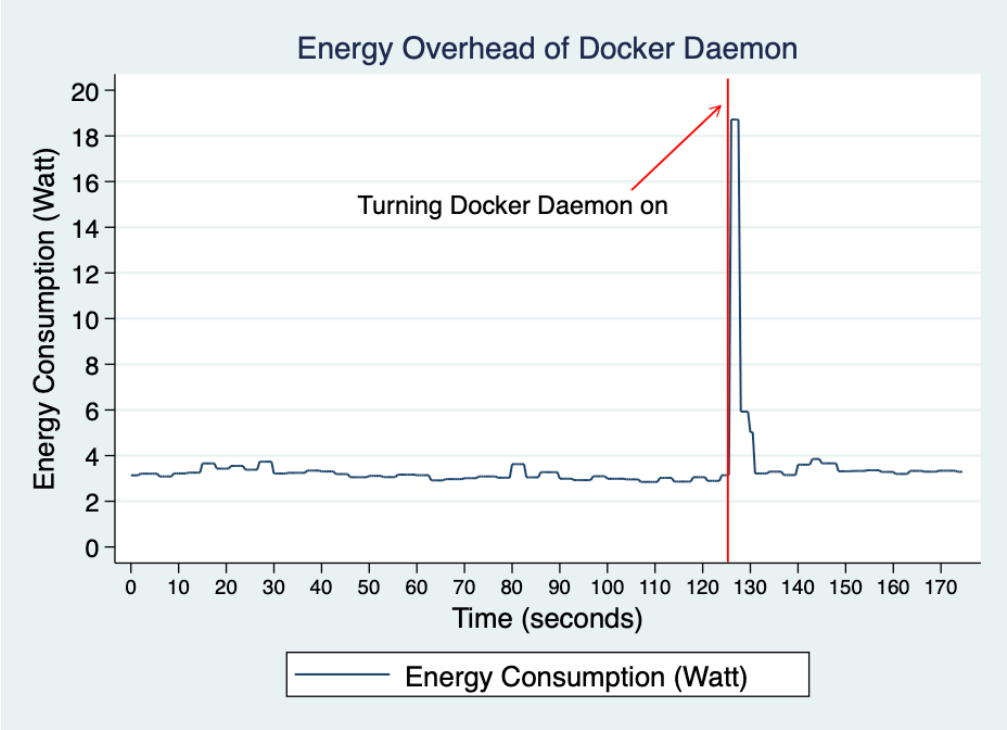


Figure 3.22: Energy Consumption of Starting Docker Daemon.

Figure 3.22 illustrates the energy consumption of the host machine before and after turning the Docker daemon on. The X-axis indicates the execution time (in seconds) and the Y-axis denotes the instantaneous energy consumption of the host machine (in W). The energy consumption of the host machine was collected for a few minutes before and after turning the Docker daemon on. It can be seen that the base energy consumption of the host machine is around 3.75 W. As soon

as the Docker daemon is turned on (denoted by the Orange line), the energy consumption of the host machine increased to around 18.25 W for a few seconds and then it came back to the mean average energy consumption. The difference in the idle energy consumption of the host machine before and after the Docker daemon is minimal. This is expected as once the Docker daemon is turned on, it does not actually use any resource of the host until a container is activated. Due to the minimal and short-lived increase in energy consumption of the computer when turning the Docker daemon on, this energy is not considered for any future experiments.

3.6.3 Measuring Energy Consumption of Web Server Container

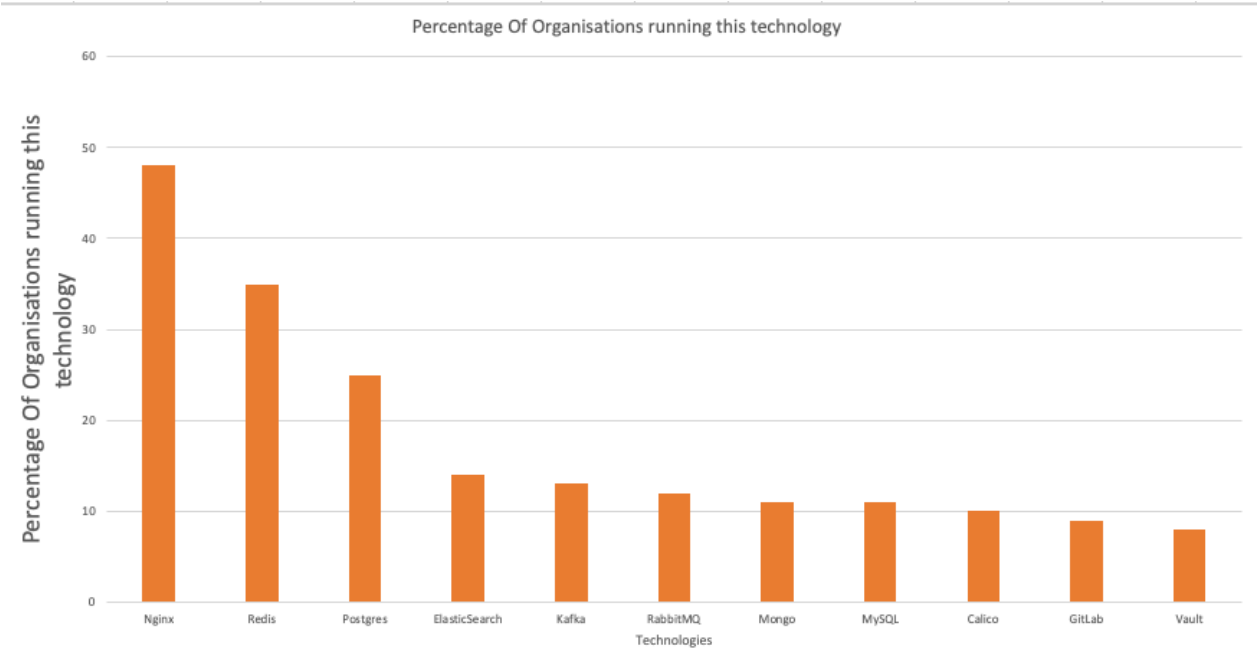


Figure 3.23: Percentage of Organizations using different Docker Technologies

Figure 3.23 illustrates a study that grouped organizations together with the Docker containers that they use for their deployments. It showed that a few of the top most used technologies running using Docker are web servers and database servers [242]. Out of the top 10 technologies deployed using Docker, 5 are database-related and 3 are web hosting-related technologies [242].

In this section, two web-based Docker containers (Nginx and Apache) are compared. Several different experiments were conducted to measure the performance and energy impact of these containers. A simple HTML web page of size 615 bytes is being served using these web technologies. The web servers are stress tested. The stress test includes sending a massive amount of requests to the web servers and measuring the latency of serving the web page or request. The energy consumption of the system is constantly measured to identify the energy impact these Docker containers have on the system.

Web technology fingerprinting and stress test

Fingerprinting web technologies include measuring the energy consumption of starting the containers. This is done mainly to check the overhead of the container. The containers are then subjected to a standard stress test using ***wrk2*** benchmarking tool [243]. For this study, the benchmarking tool is requesting 500,000 requests to the web server every second and keeping 500 concurrent connections alive.

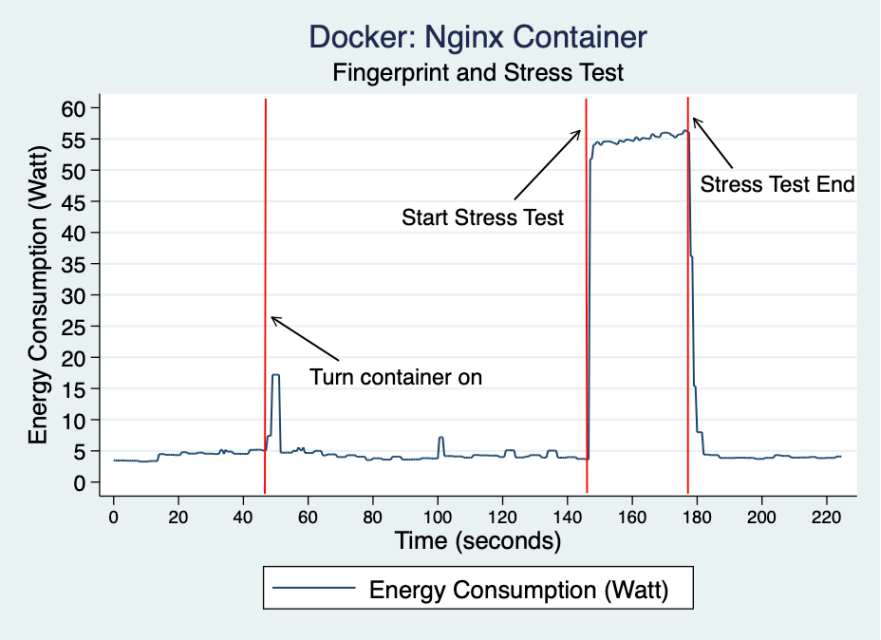


Figure 3.24: Energy Usage of Nginx Container during Startup and Under Load.

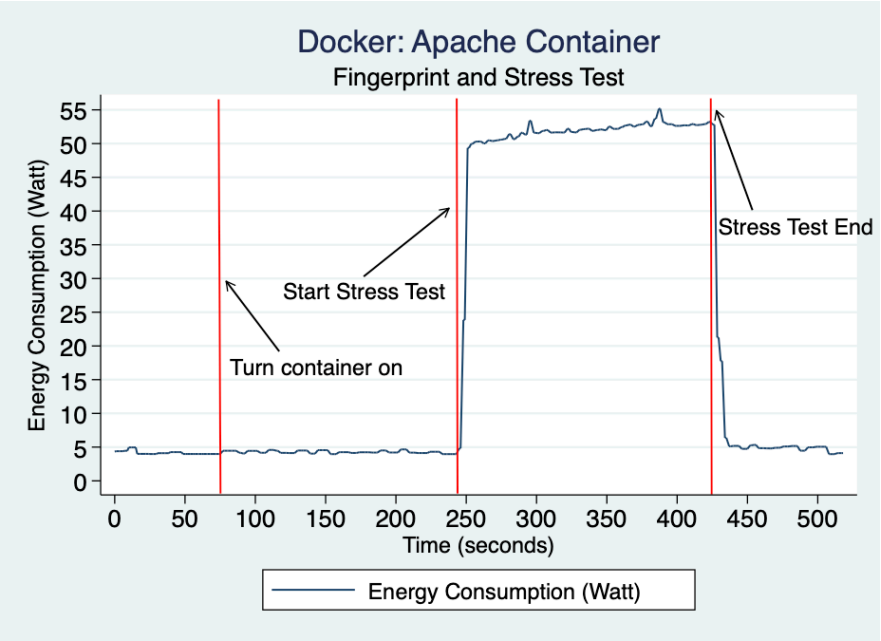


Figure 3.25: Energy Usage of Apache Container during Startup and Under Load.

Figures 3.24 and 3.25 illustrate the energy consumption of the host machine during the fingerprinting and stress test of the Docker web technologies. The X-axis indicates the execution time (in sec) and the Y-axis denotes the instantaneous energy consumption of the host machine (in W). The energy consumption of the host machine was collected for a few minutes before and after the experiment. The time when the Docker containers were started and the stress test was conducted is marked in Figures 3.24 and 3.25 (denoted by the Orange line). A clear spike in the energy consumption of the host machine can be seen which is distinctly greater than the base energy consumption. Comparing the energy consumption spike during the initialization of the containers, it can be seen that the startup energy cost of the Nginx container is greater than that of the Apache container.

Comparing the energy consumption during the stress test, it can be seen that both containers consume around the same amount of energy during the peak load. The energy consumption range of both containers is between 50 to 56 W. The Nginx server served a maximum of 114,717 requests per second and Apache served around 38,636 requests per second. Thus, for the same amount of energy consumption and the same amount of stress test, the Nginx container served more requests than the Apache server. Nginx container can be considered more energy efficient as it consistently and without any latency served more requests made by the benchmarking tool in contrast to Apache. It is important to note that being able to answer more requests is dependent on many factors such as the inherent application and what it was optimized for. In this experiment, the vanilla implementation of the containers without any changes or optimizations is compared.

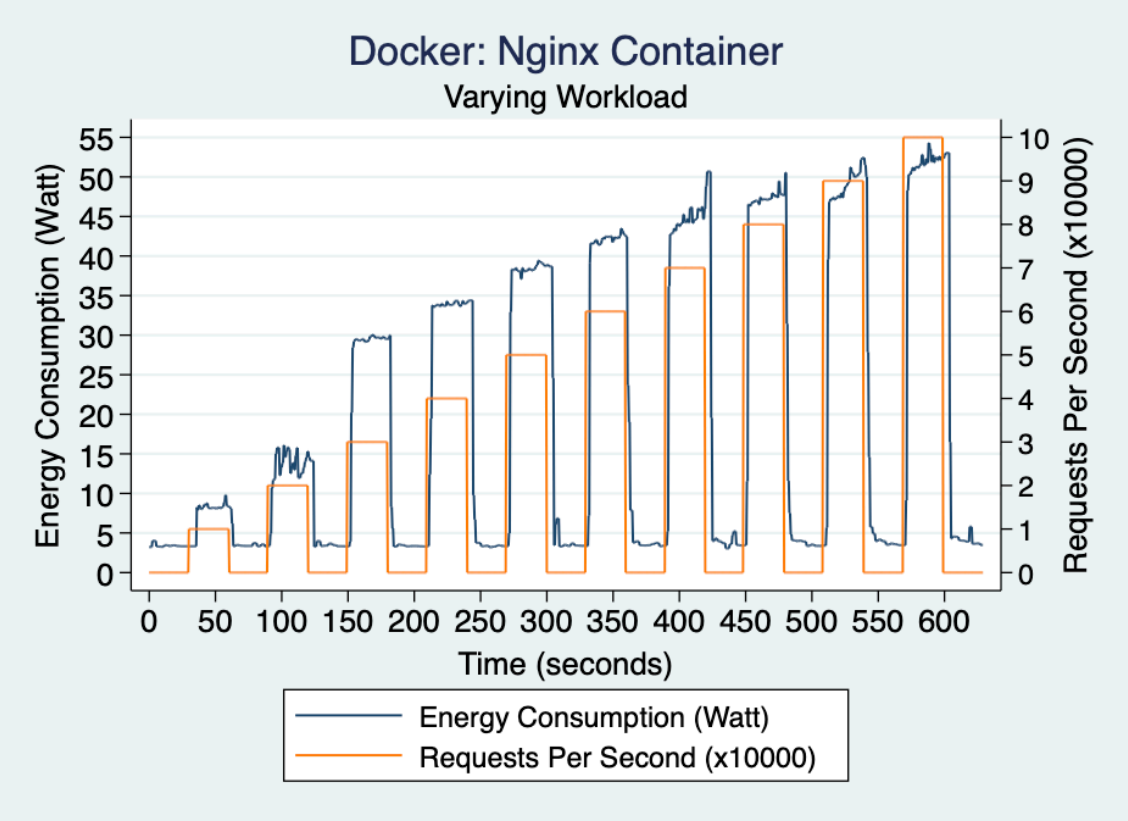


Figure 3.26: Nginx Docker Container's Energy Usage with Changing Workloads

Impact of Varying workloads on Web Docker Containers

In this experiment, the same benchmarking tool is used to stress test the servers using varying workloads. The only change in the configuration is the number of requests requested per second from the server. The number of requests gradually increase from 10,000 requests per second to 100,000 requests per second. The data from this experiment will help co-relate energy with the performance of the Docker containers. This is done to check the impact of the workload on the energy consumption of the Docker web server.

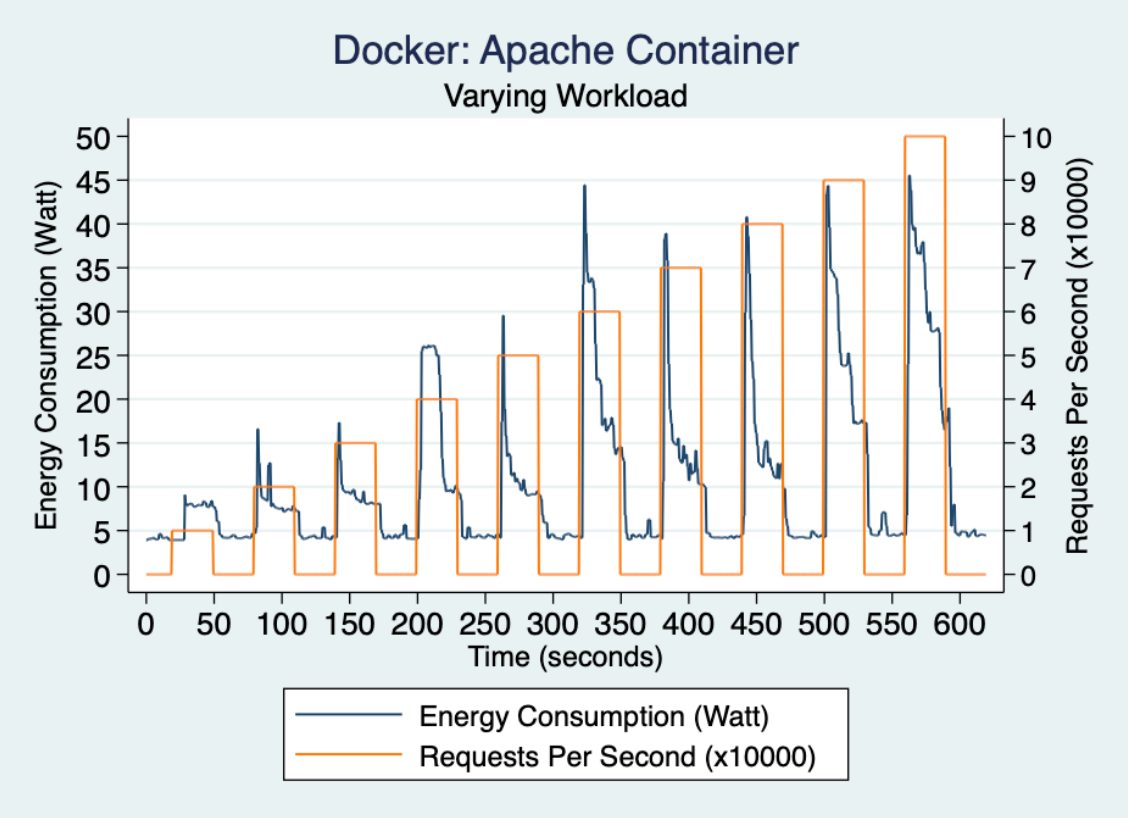


Figure 3.27: Apache Docker Container's Energy Usage with Changing Workloads

Figures 3.26 and 3.27 illustrate the energy consumption of the host machine during the experiment. It should be noted that the benchmarking tool requests more than what the web server can process just so as to stress it to its maximum. The X-axis indicates the execution time (in sec) and the left Y-axis denotes the instantaneous energy consumption of the host machine (in W). The right Y-axis denotes the requests-per-second requested by the benchmarking tool.

Figures 3.26 and 3.27 present distinct differences between the energy consumption of the Nginx and Apache Docker containers under varying workloads. The Nginx container has consistent behavior in terms of energy consumption and is consistently serving the requests made by the benchmarking tool. A clear increase in energy consumption can be seen during each benchmark experiment. Also, the peak energy consumption gradually increases as the workload increases.

In Apache containers, many of the requests were of high latency and some even failed. This behavior is confirmed by the energy consumption of the Docker container during the benchmark test. The energy consumption during each test spiked instantaneously and then gradually decreased during the benchmark indicating that the container was not under stress for the whole duration of the experiment. Due to this, multiple requests were failing or not being served consistently. A more in-depth understanding of the Apache web server and its internal load balancing is required to understand why this behavior was observed.

Energy consumption comparison: Nginx vs Apache

Inconsistent behavior in terms of energy consumption and performance was observed between the Nginx and Apache Docker containers in the previous section. Apache server drops more requests than Nginx and due to that the energy utilization data is skewed. A fair comparison between the energy consumption of the containers/ servers can be done by considering the energy consumption of the containers when their successful performance (successful completion of requests) is compared. Stress tests similar to the previous Section were conducted. The stress tests provided the number of successful requests being served by the server and the number of failed/ dropped requests. This number of successful requests served by both servers was compared for varying loads of stress.

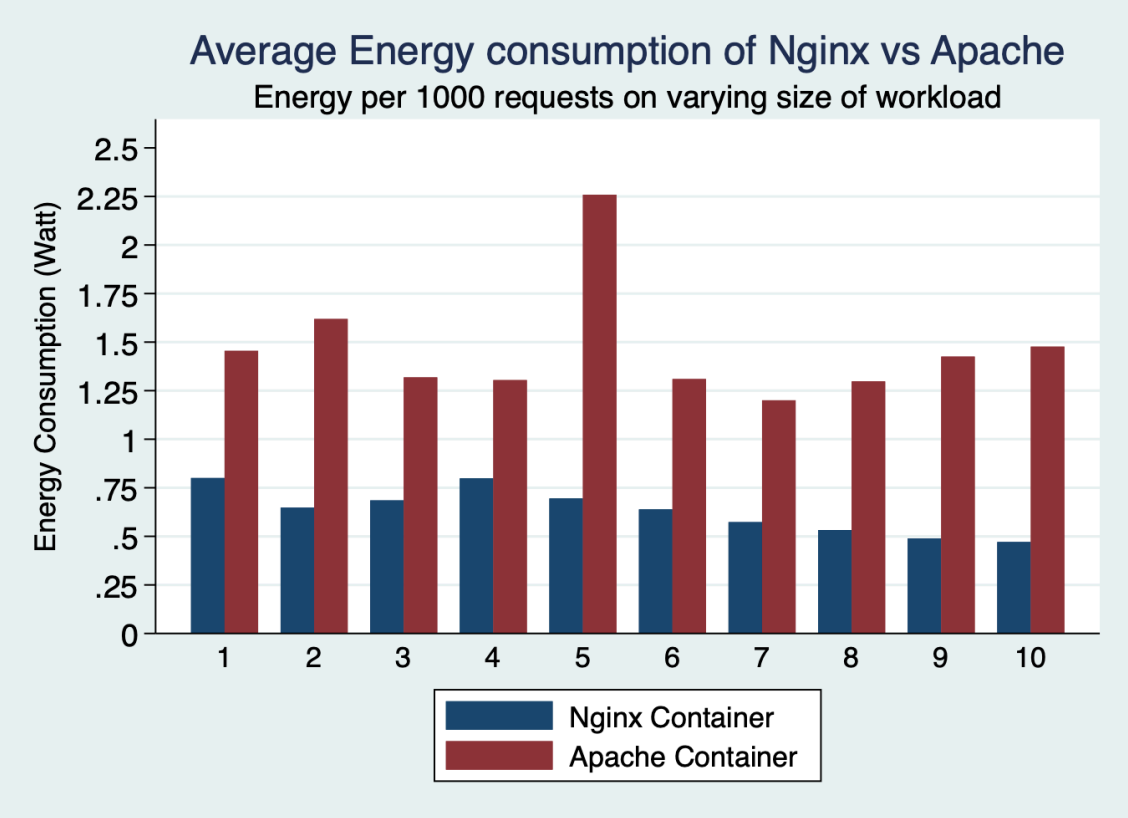


Figure 3.28: Average Energy Consumption of Nginx vs Apache Docker Containers

Figure 3.28 illustrate the energy consumption of the host machine during varying workloads as compared with every 1000 successful requests. As seen from Figure 3.28, the Nginx Docker container is more energy efficient as compared to Apache for different workloads. Nginx has also outperformed the Apache web server in terms of memory, latency, and CPU load due to its inherent architecture [241, 244]. A study showed that Nginx is around 2.5 times faster than Apache web server [245]. Similar, results can be seen in terms of the energy consumption of the Docker containers. Apache Docker containers consume around 2 to 3 times more energy than Nginx to successfully serve 1000 requests.

3.6.4 Measuring Energy Consumption of Database Docker Container

In the previous section, popular Docker-based web technologies have been compared with each other in terms of their performance and energy consumption. The second most used Docker technology after Web server is Databases. In this section, two widely used Database Docker containers (Mongo DB and Postgres) have been compared for their overhead, and a stress test has been performed with a focus on energy consumption. The stress test is conducted using the ***POC Driver*** [246] for Mongo DB and ***pgbench*** [247] for the Postgres.

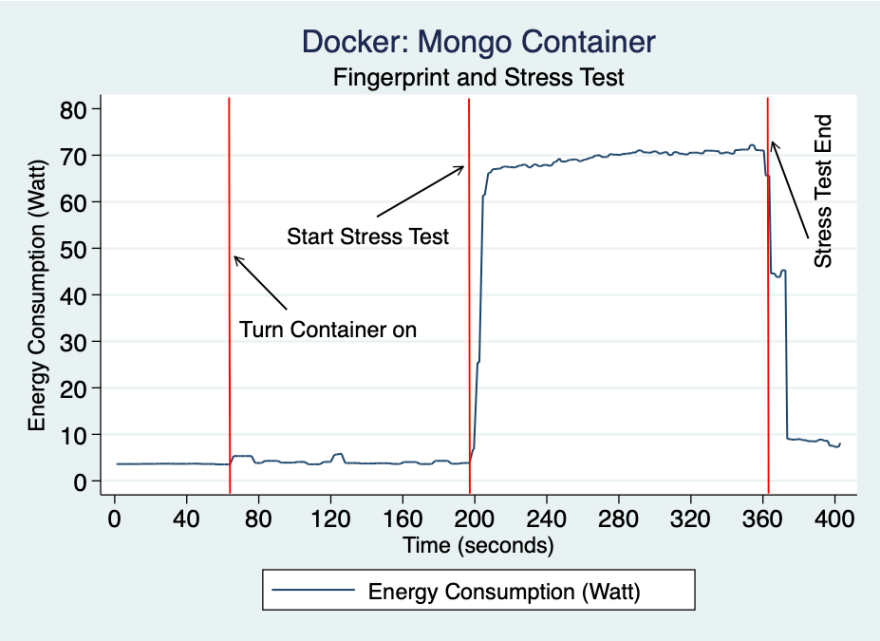


Figure 3.29: Energy Consumption Analysis of Mongo DB Docker Container

Figures 3.29 and 3.30 illustrate the starting and stress testing of these Docker containers. The time when the containers were turned on and the stress test started and ended has been marked in the Figures. The X-axis denotes the time in seconds, and the Y-axis represents the instantaneous energy consumption of the host machine at that given time. There is no clear impact on energy for both containers being turned on. The slight increase in energy consumption during that time

could not be associated with the start of the containers due to a wide range of internal processes by the host machine.

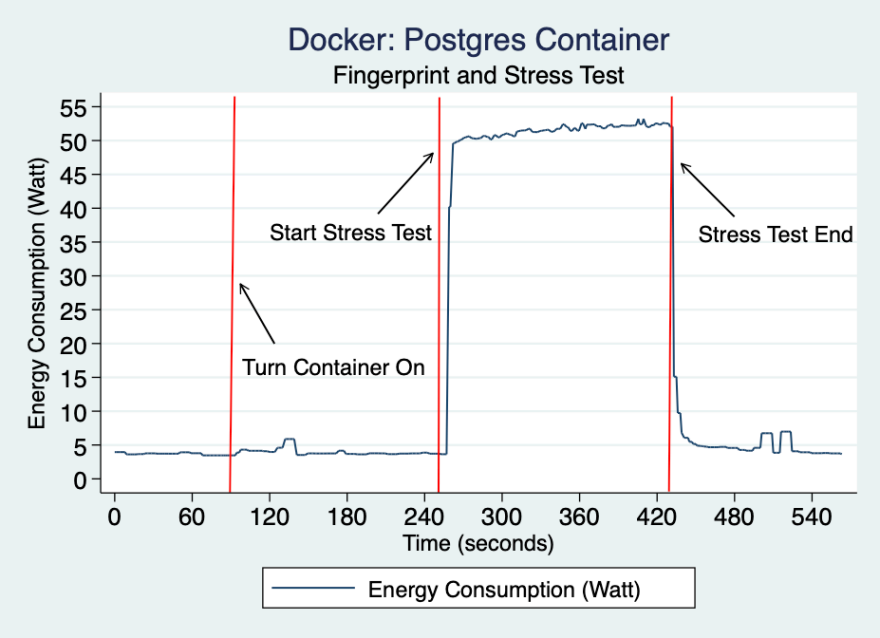


Figure 3.30: Energy Consumption Analysis of Postgres DB Docker Container

Even though no energy overhead of starting Mongo DB and Postgres containers can be identified, these containers could be performing some other background processes. A notable difference can be seen in the energy consumption during the stress test of the two containers. The peak energy consumption of the Mongo DB is between 65 W and 73 W and for Postgres, it is between 50 W and 53 W. Mongo DB is consuming a lot more energy for the same amount of performance as compared to Postgres.

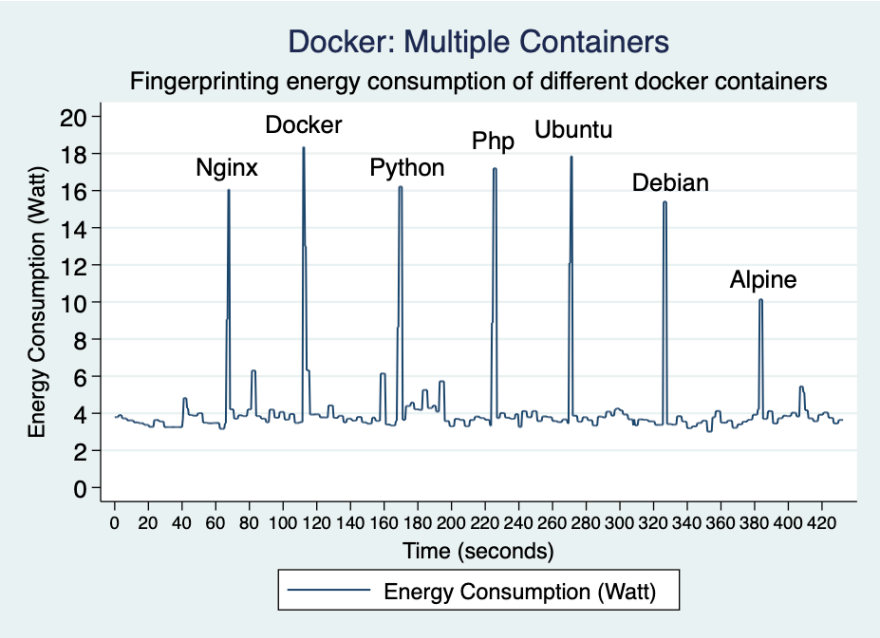


Figure 3.31: Fingerprinting Energy Consumption of Starting Multiple Docker Containers

3.6.5 Measuring Energy Overheads of Multiple Containers

Starting a Docker container saw an increase in the energy consumption of the host computer (see Figure 3.24). In this section, multiple different Docker containers were initialized and the energy consumption was measured to investigate the overheads of different Docker containers based on their functionality. The containers were turned off and a delay was added to ensure the containers do not affect each other's consumption data.

Figure 3.31 illustrates the energy consumption of different Docker containers being started one after the other. The type of Docker container is provided on the top of each corresponding spike in energy consumption. The X-axis denotes the time in seconds and the Y-axis represents the instantaneous energy consumption of the host machine at that particular time. As seen from Figure 3.31, differences in the peak energy consumption can be clearly seen between a few groups of Docker containers. Nginx container and Python container have similar energy overhead starting at around 16 W. The vanilla Docker container (Hello World - second from left) was found to be consuming the most amount of energy which was around 18 W. PHP and Ubuntu containers consumed around the same amount of energy between 17 W and 18 W.

The Debian and Alpine containers consumed less energy to start as compared to all other containers. The Debian container used around 15 W of energy and the Alpine container used around 10 W (the lowest among the group). This fingerprinting of different Docker containers can have multiple security implications as it shows that the energy consumption of just starting the containers can be used to uniquely identify the type of container being started.

3.6.6 Analyzing Factors that Affect the Energy Consumption

To further investigate the differences between the energy consumption of different Docker containers and the factors that might be affecting the energy consumption, a series of experiments were performed with a focus on getting the internal statistics of the computer and finding the root cause of the increase in energy consumption. The three main factors that are being considered are the CPU, Memory, and Input-Output calls (I/O Utilization). This will help co-relate the CPU, Memory, and I/O utilization with the energy consumption of the host computer.

Figures 3.32 and 3.33 illustrate the energy consumption of web (Apache) and database-based (Mongo DB) Docker containers being started and stress tested. The graphs also include system information such as CPU, Memory, and I/O Utilisation. The point when the containers are started and the stress test is conducted is marked on the graphs. The X-axis denotes the time in seconds. The left Y-axis denotes the instantaneous energy consumption in watts of the host computer at any given instant and the right Y-axis presents the following three attributes as a percentage - the CPU, Memory, and I/O utilization.

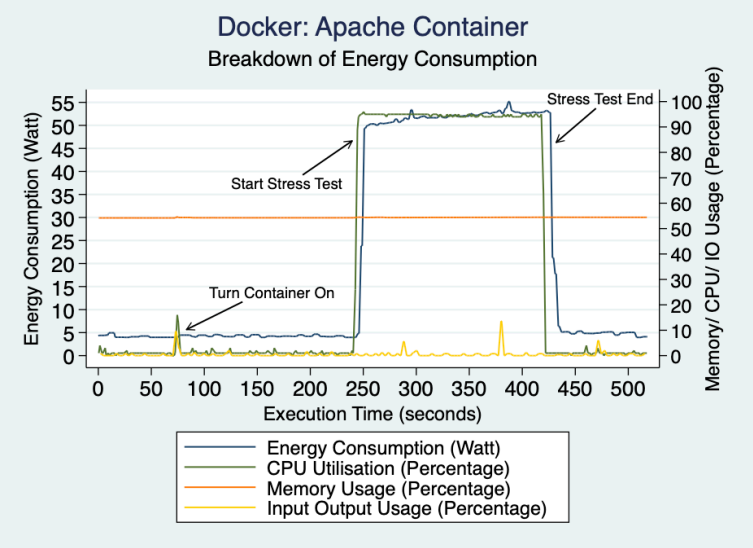


Figure 3.32: Energy Usage Breakdown of Apache Container during Stress Test

Figure 3.32 illustrates the energy consumption of the Apache container as compared to the CPU, Memory, and I/O utilization of the host computer. In contrast to Figure 3.25, a spike in energy consumption of the Apache container is not seen when the container is started. An increase in CPU and I/O utilization can be seen at the same time. The memory utilization remains constant during the whole execution of the stress test and container starting up. During the stress test, a corresponding increase in CPU utilization can be seen with less to no comparable increase in I/O utilization. The energy consumption increases as the CPU utilization increases and it ranges between 50 to 55 W. As there is no change in the memory or I/O utilization, the major factor for the energy consumption of the computer can be attributed to its CPU utilization.

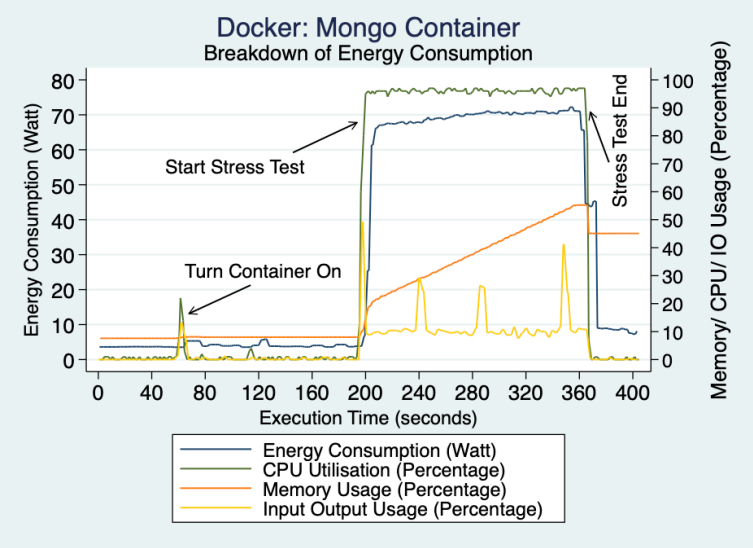


Figure 3.33: Energy Usage Breakdown of Mongo DB Container during Stress Test

Figure 3.33 illustrates the energy consumption of the Mongo DB container as compared to the CPU, Memory, and I/O utilization of the host computer. Mongo DB is a database technology

and its major functionality includes storing, reading, and updating data into memory and disk. The stress test being conducted tries to insert 100,000 records into the database which in turn gets stored on the disk. As expected, the CPU utilization is near 100% with all the processing of memory and reading and writing to the disk. As CPU utilization increases, a corresponding spike in energy consumption is also seen. Database technologies are I/O (input-output) intensive technology. As expected, the memory utilization gradually increases as we request the container to insert more records into the disk. These records are first stored in the memory and then a batch of these records is written to the disk altogether. Due to this, constant spikes in the I/O calls to the disk are seen at regular intervals.

Based on the data from Figures 3.32 and 3.33, the energy consumption of the host computer is majorly dependent on the CPU usage of the computation or workload. The memory and the I/O utilization on their own marginally increase energy consumption. A comparable increase in energy can be seen when all three attributes - CPU, Memory, and I/O are being utilized. The maximum energy consumption range of the Apache container was around 50 - 55 W and that of the Mongo DB container was around 68 - 73 W. The CPU utilization is almost near 100% in both cases but in the latter case, we have a huge utilization of Memory and the I/O calls to the disk. There are several factors that can affect the energy consumption of computers. The type of processor, the amount of RAM installed, the type of cooling system used, and the type of Operating System can all have an impact on the amount of energy a computer consumes. The experiments presented in this section are a step towards transparency on factors that affect the energy consumption of computers and Docker.

3.7 Summary

The chapter presented a comprehensive framework for reliable energy monitoring in scientific computing environments. The framework addresses the critical gap in current infrastructure by providing methodical approaches for accurately measuring energy consumption across different computational workloads. A proof-of-concept implementation was developed and deployed on a low-power cluster infrastructure, demonstrating the practical feasibility of the proposed monitoring approach. The framework was evaluated against multiple scientific workflows and containerized workloads to validate its reliability and accuracy in diverse computing scenarios.

The experimental analysis clearly shows that there are significant opportunities for energy optimization in scientific computing. The results demonstrate that while dedicating more resources to workflow execution can result in substantial performance gains, there is always a point where using additional resources does not provide proportional energy and performance benefits. The framework successfully identified energy consumption patterns and revealed that strategic node management, such as shutting off cluster nodes during periods of inactivity,

can substantially reduce overall energy consumption without compromising computational performance. The analysis further revealed that different computational workloads exhibit distinct energy consumption characteristics, highlighting the need for workload-specific optimization strategies.

The containerized workload analysis provided essential insights that bridge the gap between controlled experimental validation and real-world scientific workflow execution. Container experiments demonstrated the framework's ability to accurately capture energy consumption patterns for deterministic workloads, establishing confidence in the monitoring infrastructure's precision and reliability. These controlled experiments enabled systematic validation of the energy monitoring approach before applying it to the more complex, multi-task nature of scientific workflows. The findings establish a strong empirical foundation for the development of energy-aware scheduling systems. The monitoring framework provides the necessary insights into energy consumption patterns that can inform intelligent scheduling decisions. The trade-offs between computational response time and energy consumption were clearly quantified, demonstrating that optimal cluster configurations vary depending on workload characteristics and computational requirements. These insights are essential for developing sophisticated energy-aware schedulers that can dynamically adjust resource allocation based on real-time energy consumption data.

In conclusion, the chapter demonstrates that energy-aware frameworks are not only necessary but can genuinely help in improving computational costs while motivating research in more sustainable execution strategies. The reliable energy monitoring capability established in this chapter serves as the foundation for subsequent energy-aware scheduling approaches, enabling the development of more environmentally conscious and cost-effective scientific computing systems. The work presented establishes a clear path toward more sustainable high-performance computing by providing the tools and insights necessary for energy-conscious computational decision-making.

Chapter 4

Energy-Aware Scientific Workflow Scheduling and Execution

The contents of the chapter have been published in the following articles:

1. Warade, M., Schneider, J.-G., & Lee, K. (2022). Towards energy-aware scheduling of scientific workflows. In *2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)* (pp. 93–98). IEEE. https://doi.org/10.1109/GECOST55694.2022.10010634

2. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2022). Energy aware adaptive scheduling of workflows. In *2022 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)* (pp. 562–570). IEEE. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00078

3. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2024). Energy and Scientific Workflows: Smart Scheduling and Execution. *Journal of Information Science and Engineering*, 40, 957–977. https://doi.org/10.6688/JISE.202409_40(5).0002

4.1 Overview

Current approaches to scientific workflow execution prioritize computational performance. It is assumed that providing more resources to a computation inherently lead to better results [22]. However, with growing concerns about global energy consumption and rising operational costs, there is an urgent need to understand and optimize the energy implications of high-performance

computations [62, 35]. To achieve energy-aware scientific workflow execution, it is important to understand the complex relationships between computational workload characteristics, system resource utilization, and actual energy consumption [68]. Furthermore, practical energy-aware scheduling must integrate seamlessly with existing workflow management infrastructure while providing real-time adaptation to changing computational conditions [21, 37].

This chapter addresses the critical gap between energy awareness and practical scheduling implementations by developing adaptive, energy-aware scheduling framework for scientific workflows. The work presented here takes one step ahead from energy monitoring and analysis to active energy reduction through intelligent scheduling decisions. The chapter demonstrates how the insights gained from energy profiling from Chapter 3 can be translated into actionable scheduling policies that reduce energy consumption while maintaining or improving computational performance. In addition to that, the proposed scheduler system supports static and adaptive scheduling by building upon the Monitor-Analyze-Plan-Execute (MAPE) model from Autonomic Computing [248]. The framework provides a foundation for developing energy-conscious workflow management systems that can dynamically balance performance requirements with energy efficiency objectives.

The remainder of this chapter is organized to provide a comprehensive examination of energy-aware scheduling from theoretical foundations through practical implementation. Section 4.2 presents the theoretical foundations and design principles for energy-aware scheduling, including detailed requirements analysis, challenge identification, and architectural frameworks for both static and adaptive scheduling approaches. Section 4.3 describes the comprehensive evaluation environment, including hardware configurations, software frameworks, and the scientific workflows used for validation studies. Section 4.4 provides detailed experimental results and analysis, demonstrating the effectiveness of the proposed static scheduling approaches across multiple scenarios and workflow types. Section 4.5 presents the experimental evaluation of the proposed adaptive scheduler with varying policies for execution of scientific workflows. Finally, Section 4.6 summarizes the key findings and outcomes, discusses limitations and future research directions, and contextualizes the work within the broader landscape of energy-aware scientific computing.

4.2 A framework for Energy-Aware Scheduling of Scientific Workflows

The proposal for energy-aware scheduling of scientific workflows emerges from the recognition that existing workflow management systems prioritize performance improvements without adequate consideration of energy consumption [22, 35]. Scientific workflows, due to their inherent modular structure and explicit task dependencies, present unique opportunities for reducing energy

consumption during execution [14, 4]. The dependency relationships between workflow tasks can be strategically exploited to optimize resource allocation and minimize energy usage while maintaining computational efficiency [37, 67].

In this section, a comprehensive scheduling framework that exploits these opportunities to reduce the energy consumption of workflow computations at minimal cost to performance is presented. The framework is designed as a two-part process comprising of both static and adaptive scheduling approaches. The static scheduling framework provides a foundational and controlled approach to intelligent workflow task scheduling, offering predictable energy optimization through pre-computed scheduling decisions. The adaptive scheduler builds upon the static foundation by incorporating dynamic system monitoring and real-time decision-making capabilities, adding sophisticated complexity to the scheduling process that can respond to changing computational conditions and energy consumption patterns.

4.2.1 Static Energy-Aware Scheduling Framework

The initial framework development focuses on static energy-aware scheduling, where scheduling decisions are made prior to workflow execution based on historical data and workflow characteristics. This approach provides several advantages such as predictability, simplicity and established baseline for comparing the effectiveness of more sophisticated adaptive approaches [45, 22, 44]. The static scheduling framework is able to achieve its desired goals of reducing energy consumption by different techniques which can be categorized based on what level of execution these techniques affect. The framework operates through two distinct control levels: cluster-level optimizations that focus on hardware resource management and infrastructure configuration, and job-level optimizations that target individual workflow task characteristics and dependencies [152].

Cluster-Level Optimizations

Cluster level Optimization refers to the all the ways in which the proposed scheduler can exploit the execution cluster and hardware systems to achieve its goals. These techniques are performed at hardware level and before the computation is executed to have the computing hardware ready to accept and execute computation in an efficient manner. At the cluster level, the scheduler analyzes the overall resource requirements of the submitted workflow and makes strategic decisions about resource allocation:

- **Dynamic Node Management**: The scheduler analyzes the workflow and decides the maximum number of nodes the workflow might use based on the maximum number of parallel jobs.

- **Resource Configuration**: Controlling or scaling of CPU frequency, memory allocation, and network settings based on workflow computational patterns. This helps in making the optimal use of the resources.
- **Load Balancing**: Strategic distribution of computational load across heterogeneous resources to maximize energy efficiency.

These strategic decisions made by the scheduler need to be implemented in a real-world execution environment. These decisions can be implemented using the following techniques:

1. **Switching off unused nodes**: The scheduler analyzes the workflow and decides the maximum number of nodes the workflow might use based on the maximum number of parallel jobs. The nodes that are known to be idle during the whole execution of the workflow, i.e. surplus computing resources, can be shut down to save energy. In certain cases, when the node is initially used and then idle for a long time, it can be dynamically turned off to save energy.
2. **Dynamic node activation**: During times of peak computation, if the available resources are insufficient, then the scheduler can dynamically power a node on. This is only done when the performance increase compensates for the increase in energy consumption.
3. **Thread optimization**: According to the complexity of the jobs in a workflow, the scheduler can specify the number of threads to use on certain nodes depending on the memory requirements. Computationally expensive tasks can benefit from more CPU and memory usage.
4. **CPU frequency optimization**: Overvolting is performed to increase the energy consumption of the CPU to increase the computing power of the CPU. This affects the energy consumption and performance of a node substantially. Computationally intensive tasks can benefit from overvolting as long as the performance increase is greater than the energy consumption. Similarly, undervolting is performed when less computationally intensive tasks, such as data transfer, unzipping, zipping, etc, are to be executed.
5. **Network settings optimization**: There are many different cluster setups used for high-performance computing. They vary from one huge supercomputer to multiple small distributed computer networks across the globe. The scheduler can exploit the network settings further to improve the energy consumption of the whole cluster. For example, during the network booting of nodes, changing the NFS block size can affect how fast the changes are synced. For distributed computing systems, different settings such as connection medium, Ethernet links, and switch configurations can be altered to optimize the cluster for the inter-communication of nodes.

6. **Background process management**: The scheduler can turn off all non-essential processes and components on the node that consume energy and do not contribute to the computation. Background processes such as system logs and redundant processes from closed apps can consume a lot of energy and CPU memory usage. Components such as empty USB ports, LEDs, HDMI, and AUX can be turned off to save the energy consumption of a node.

Job-Level Optimizations

Job level Optimizations refers to the techniques at the individual workflow or job level that can be used to achieve the goals set by the scheduler system. The scheduler also considers the overall workflow structure and dependencies to optimize execution patterns. In this, individual workflow tasks are analyzed and scheduled based on their specific computational characteristics:

- **Task-Resource Matching**: Intelligent assignment of tasks to computing resources based on performance-per-watt metrics and task computational requirements. Matching of task to resource plays a vital role in the end performance and the energy consumption of executing the task. Smartly scheduling the the task will enable the framework to control all the aspects of the execution.
- **Priority-Based Scheduling**: Identification and prioritization of bottleneck tasks to minimize overall workflow execution time and energy consumption. Bottlenecks result in non-maximal usage of the computing hardware which results in high energy usage at no progress to the computation.
- **Resource Requirement Optimization**: Fine-tuning of individual task resource requirements based on historical execution data. Fine-tuning the task requirements will make sure that only the required amount of resources are being used and the resources are being used to their maximum.
- **Dependency Analysis**: Dependency analysis comprises of understanding task dependencies and identifying opportunities for improved scheduling. The independent nature of these tasks enable the scheduler system to have individual control over the scheduling of the tasks as long as their dependencies are completed.
- **Critical Path Optimization**: Focus on reducing energy consumption along the workflow's critical execution path. Critical path in a computation refers to the execution of the tasks that enable release of maximum number of jobs. Simply, the task that is a dependency for the most number of jobs is scheduled and executed first so that all the child jobs are then available to be scheduled.
- **Inter-Task Communication Optimization**: Minimization of data transfer overhead between dependent tasks. Data transfer consume a huge amount of the computation time

during which the compute system is not available to execute further computation. This results in higher energy consumption. Smart scheduling jobs such that the data transfers between the jobs are minimized can result in substantial energy and execution time savings.

The scheduler implements these job-level optimizations through several practical mechanisms that target specific workflow characteristics and execution patterns. These mechanisms can include characteristics such as pre-processing, data, network usage, CPU load and post-processing. These job-level optimization can be implemented using the following techniques:

1. **Bottleneck job prioritization**: The job priority determines which job gets queued before the others. If multiple jobs have the same priorities then all the jobs get queued and are executed as the resources become available. Normally, all same-level jobs in a workflow are given the same priority, and this is an issue as one job could be a bottleneck, and releasing it early can be beneficial for the overall execution of the workflow. The scheduler understands the DAG of the workflow and identifies the bottlenecks. It can then assign higher priorities to the jobs that may benefit from executing early.

2. **Node targeting**: In a hybrid cluster environment with a different mixture of OSs, processor architectures, and memory allocations, it is beneficial to target jobs to specific nodes based on their complexity. Data-intensive jobs can benefit from non-distributed cluster setups. Similarly, distributed nodes with huge memory and CPU power can be used to schedule CPU and memory-intensive jobs. The scheduler makes use of different aspects of the tasks and can find the optimal scheduling technique to reduce the overall energy consumption of the computation.

3. **Job size prioritization**: CPU-intensive jobs are often paired with high memory requirements. This might not always be the case, and the scheduler can identify these discrepancies. It can then edit the requirements of the jobs to execute them specifically on nodes that can execute the jobs effectively and with less energy usage.

4. **Local binary utilization**: Many workflows require installation of workflow specific binaries. The scheduler can analyze the workflow and decide to locally install the binaries on the nodes beforehand, thus saving communication overhead.

5. **Scheduling algorithm selection**: Depending on the complexity of the workflow and bottlenecks, different scheduling algorithms such as round-robin and grasshopper can be used for effective scheduling.

6. **Multiple workflow execution**: Multiple workflows can be executed in tandem on a resource pool. The scheduler can identify the independent and bottleneck jobs in multiple workflows and execute them first for smooth flow.

The static energy-aware scheduling framework provides a foundational approach to workflow energy optimization through systematic application of cluster-level and job-level techniques. By implementing these optimization strategies prior to execution, the framework establishes predictable energy reduction patterns that serve as a baseline for more sophisticated adaptive approaches. The techniques presented here demonstrate that significant energy savings can be achieved through careful analysis of workflow characteristics and strategic resource allocation decisions, even without runtime adaptation capabilities [42, 37].

4.2.2 Adaptive Energy-Aware Scheduling Framework

Building upon the static scheduling foundation, the adaptive framework introduces runtime adaptation capabilities that respond to dynamic execution conditions. This approach leverages the Monitor-Analyze-Plan-Execute (MAPE) model from autonomic computing [158] to create an adaptive scheduling system. Adaptations to the execution of the workflow or the cluster can be done during run-time with the help of new knowledge obtained from the current behavior of the computation [23]. This new knowledge can be used to schedule jobs according to a specific goal [249, 33]. The performance of long running scientific workflows stands to benefit from adapting to changes in their environment. Autonomic Computing provides methodologies for managing run-time adaptations in managed systems. A common understood way of achieving this is to use the MAPE model. In this section, the different considerations and requirements of the MAPE model's subsystems are identified and described. The scheduler system contains different subsystems that correspond to the different functionality of the MAPE model. The following describes the monitoring, adapting, planning and execution for workflow adaptations (see Section 2.4.4).

Monitor Component

The monitoring subsystem continuously collects execution data across multiple dimensions which will be used by other subsystems to make future scheduling decisions. All the computing information collected by this subsystem is presented below:

M.1 **Total compute resource** This is a measure of all the computing resources in a compute cluster. This can include both the free and used resources.

M.2 **Available computing resource** This is a measure of all the available computing resources that can be used for executing a task.

M.3 **Progress of the workflow.** This is a measure of how the workflow has progressed. Details of different tasks that have been completed and the tasks that are yet to be completed.

M.4 **Progress of a service.** This is a measure of how the individual task has progressed. It also monitors the input, output and low level computation of the task.

M.5 **Completion of a service.** This monitors the completion rate of tasks. This will focus on individual tasks and help determine the factors affecting their completion rate.

M.6 **Data Usage of a service.** This is a measure of the amount of data or bandwidth that a task is using.

M.7 **Load on an existing computing resource.** This is measure of how a compute resources is being utilized. The amount of resource being used and how much is available for execution of new tasks.

M.8 **Memory usage of an existing computing resource.** This is a measure of the amount of memory being used by a task on the compute resource.

M.9 **Load on the computing network.** This is a measure of the network resources being used.

Monitoring component inside the *Autonomic Manager* collects this data with the help of *sensors* and system log files. Monitoring the available resources and computations can help in determining the appropriate pair of task-resource pair to maximize the throughput of the computing resource. Likewise, monitoring the current state of the computation can help understand the overall factors affecting the performance and can be used to improve other workflow executions.

Analyze Component

Analysis component of the *Autonomic Computing*, as the name suggests, analyzes the data from the *Monitoring* component based on certain rules and specifications and presents the data to the next component in *Autonomic Computing*. The analysis engine processes monitoring data to identify optimization opportunities. A list of possible analysis criteria for the study is provided below:

A.1 **Bottleneck detection**: Bottleneck occurs when a number of different tasks need to be completed to progress to the next step in computation. This may be detected by using M.3, M.4, M.5 and M.9 data.

A.2 **Resource allocation imbalance**: This is where the allocation of resources is not homogeneous. Some tasks may be executed on new resources even though the old resources can execute them. This may be detectable using M.1, M.2, M.7 and M.8.

A.3 **Missing system goals**: The current progress of the workflow and the progress of each task being executed can help in estimating whether the system goal can be achieved or not. For this, data from M.3 and M.4 can be used to compare with the original schedule.

A.4 **Free capacity analysis**: This is to measure the amount of resource that is underutilized for extended period of time. This may be detectable by M.2, M.7 and M.8 data.

A.5 **New task availability**: This determines the next task in the workflow to be executed. This may be done by analyzing the M.3 data.

A.6 **New resource availability**: This determines if there is any additional resource that is made available for the computing. This may be detected by M.1 and M.2 data.

A.7 **Node utilization analysis**: It may be possible to analyze the individual node utilization by analyzing the data from M.6, M.7, M.8 and M.9 data.

Plan Component

Planning component in the *Autonomic Manager* decides on the necessary adaptations to be conducted to achieve the system goal. Based on analysis results, the planning subsystem develops specific adaptation strategies. These adaptations may led to the removal of one or more problems (e.g., A.1- A.3) in the computation.

P.1 **Bottleneck removal**: To prioritize and remove the bottleneck at the start. The bottleneck task can be completed on priority basis by completing its dependencies first. This is in response to the problems detected by A.1 and A.3.

P.2 **Task rescheduling**: To reschedule tasks that failed. Any failed task needs to be analyzed and rescheduled on different resource or configuration based on the reason for its failure. This is in response to problem A.2, A.3 and A.5.

P.3 **Parallel resource increase**: To improve performance by increasing the parallel computing resources. Based on the type of computation and number of parallel independent jobs, it may be possible to complete the computation faster. This is in response to A.3, A.4, A.5 and A.6.

P.4 **Smart task scheduling**: To make a task complete faster by smartly scheduling a task to specific nodes. Different resources might specialize in different types of executions (e.g. GPU for image processing). This is in response to A.2, A.5, A.6 and A.7.

P.5 **Free resource assignment**: To assign free resources to new tasks. The available resources need to be smartly allocated to tasks to maximize utilization and reduce overheads. This is in response to problems/ opportunities detected by A.2, A.4 and A.7.

Execute Component

The execution subsystem implements planned adaptations while maintaining workflow integrity. This is mainly done by invoking effectors which can include multiple steps such as suspending the workflow, making changes to environment variables, upcoming tasks, cluster resources or hardware components and resuming the workflow.

4.2.3 Requirements Analysis for Energy-Aware Schedulers

The high-level proposal for a static and adaptive energy-aware scheduler system has been provided in the previous section. The development of effective energy-aware scheduler must address several critical requirements that ensure both functionality and practical applicability. The primary aim of the scheduler is to reduce the energy consumption of workflow executions and to achieve this, the following requirements have been identified that the scheduler systems must meet to achieve their goals:

R1 Energy Consumption Reduction: The scheduler shall attempt to reduce the energy used for executing a job, workflow or set of workflows, respectively.

R2 Output Integrity: The proposed scheduler shall not vary the output of the workflow, ensuring scientific reproducibility and accuracy.

R3 Configurable Thresholds: The scheduler shall have a set of pre-defined resource usage thresholds per job, workflow or set of workflows that can be configured based on user requirements.

R4 User Override Capabilities: The proposed scheduler shall allow a user to override predefined thresholds when domain-specific knowledge suggests alternative approaches.

R5 Fault Tolerance: The proposed scheduler shall be fault tolerant and handle workflow failure gracefully without compromising overall cluster stability.

R6 Comprehensive Logging: All data generated by the scheduler shall be logged for debugging, analysis, and continuous improvement.

R7 Data Export Capabilities: The proposed scheduler shall allow export of data for external analysis and reporting.

R8 Adaptive scheduling: The proposed adaptive scheduler shall make use of a feedback loop to collect the current execution data, analyze it and determine the optimal adaptations to achieve the goal set by the user.

R9 Platform independent: The proposed scheduler shall be compatible across different platforms.

4.2.4 Challenge Analysis and Mitigation Strategies

The development of energy-aware schedulers faces several significant challenges that must be addressed through careful design and implementation. The following challenges were identified during the literature review:

1. **Workflow management complexity**: Workflow Management Systems perform a number of tasks such as executing, logging, pausing, resuming workflows. These are independent tasks that handle different aspects of workflow management. The scheduler needs to be able to understand the workings and execution of workflows and take decisions based on all factors.

   Mitigation Strategy: Development of modular scheduler architectures that interface with existing workflow management systems through well-defined APIs, minimizing the need for system modifications.

2. **Multiple Workflow Management Systems (WMS) availability**: Different workflow management systems like Apache Airavata [250], Kepler [13], Apache Airflow [136], and Pegasus [251] have been developed with different use cases and features.

   Mitigation Strategy: Creation of abstraction layers that provide common interfaces across different systems, enabling scheduler deployment across diverse computational environments.

3. **Different parallel message passing interfaces**: Depending on the workflow, different message-passing interfaces such as mpich [252], MPI [127], and mpi4py [253] are used for inter-node communication.

   Mitigation Strategy: Implementation of interface-agnostic optimization strategies that focus on resource allocation rather than communication protocol specifics.

4. **Collection of accurate energy data**: Energy consumption data collected from sensors are not very precise due to sensor error margins and accessibility issues.

   Mitigation Strategy: Implementation of statistical approaches that account for measurement uncertainty and development of lightweight monitoring systems that minimize overhead.

5. **Multiple workflows with different parameters**: Workflows with varying parameters and structures make it hard to develop a generic scheduler that can take into consideration all parameters.

Mitigation Strategy: Development of adaptive learning mechanisms that can characterize new workflows based on execution patterns and historical data.

6. **Resource contention**: Different workflows can be scheduled on a pool of resources, leading to resource contention that can affect the energy cost of a workflow. Attempting to reduce the energy of a workflow in such a scenario is a complex task depending on the number of external and internal factors of the workflow and the computing resources.

   Mitigation Strategy: Development of global optimization algorithms that consider multi-workflow interactions and implement fair resource allocation policies.

4.2.5 Proposed Energy-Aware Scheduler System Architecture

Energy-aware scheduling of scientific workflows requires a comprehensive architectural approach that can systematically analyze workflow characteristics and dynamically configure computing resources for optimal energy efficiency. In this section, a generic high level design for an energy aware scheduler is presented. The proposed scheduler system architecture takes into account all the attributes of the workflow and accordingly configures the cluster and schedules the workflow in an energy efficient manner. The scheduler requires minimal domain understanding and can understand and schedule workflows based totally on jobs, inter-dependencies and complexity. The design follows and aims to meet all the user requirements outlined in Section 4.2.3.

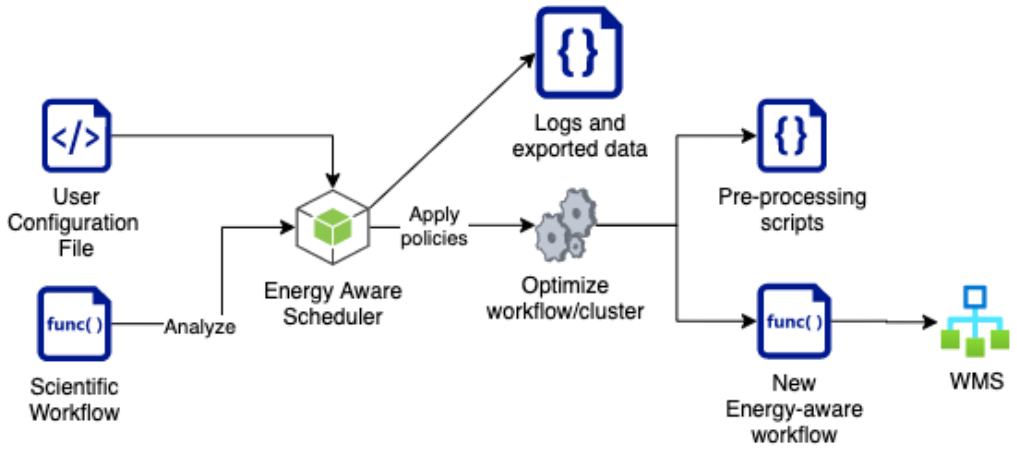


Figure 4.1: High-Level Architecture for Energy-Aware Scheduler

The high-level working of the scheduler can be described as shown in Figure 4.1. The design works independently to generate new energy-aware workflows and configure clusters according to a set of policies. This new workflow can then be executed using existing Workflow Management Systems (WMS). The design requires minimal installation for setup and operation.

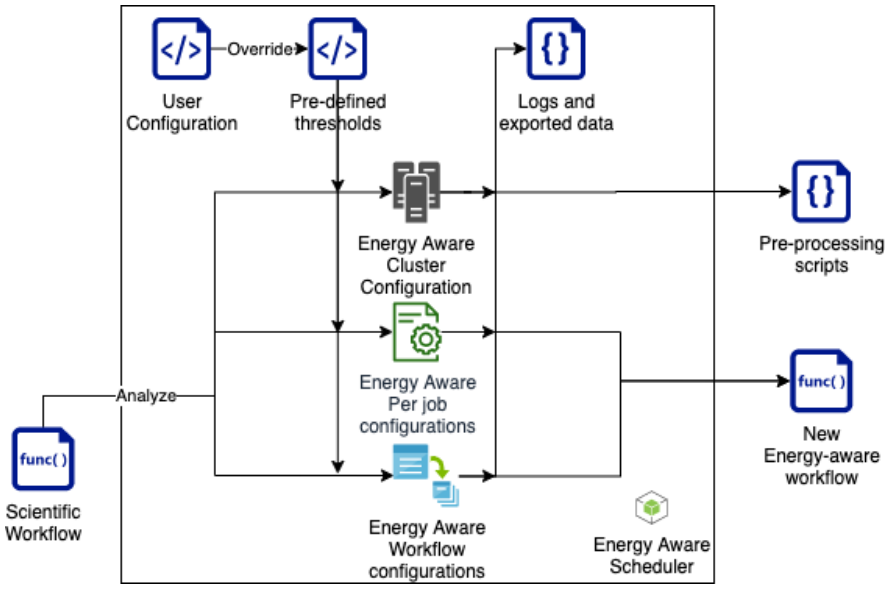


Figure 4.2: Detailed Breakdown of the Energy-Aware Scheduler Components

In response to the requirements set out in Section 4.2.3, the core working of the scheduler can be categorized into three main components as shown in Figure 4.2. The scheduler analyzes the workflow, its execution, dependencies, the computing environment, and can configure the workflow and cluster to maximize cluster utilization and reduce energy consumption. The scheduler analyzes the workflow and develops an energy efficient solution using this three part process. A set of predefined thresholds are provided to the scheduler which relate to the user's needs and workflow characteristics. A user configuration file can be defined which overrides the default thresholds. These functionalities are in response to R3 and R4. The scheduler is developed in a platform-independent universal programming language and is capable of generating log files that can be exported for further analysis, addressing requirements R7 and R9.

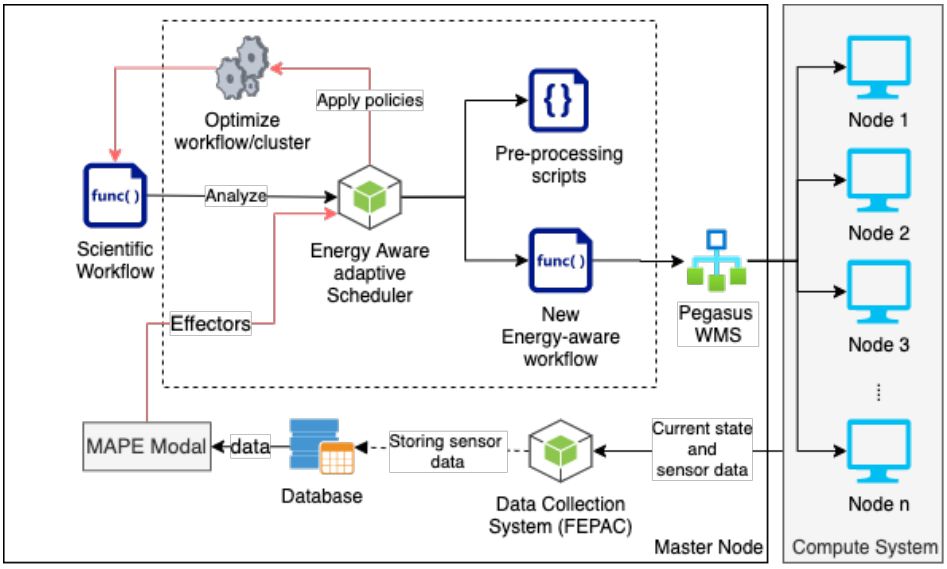


Figure 4.3: Energy-Aware Adaptive Scheduler Architecture based on MAPE model

The scheduler does not change the original workflow or the data used for computation. Consequently, the scheduler does not vary the output of the workflow during multiple executions of the same workflow as outlined in requirement R2. The scheduler generates a new energy-efficient workflow that is executed using a WMS. This workflow is tested for breaks or faults before submitting it for execution, addressing requirement R5.

The high level generic design of the adaptive scheduler is presented in Figure 4.3. The design is guided by the requirements set out in the previous section. The scheduler tracks the current state of the execution with the help of sensors or logs and dynamically optimizes the workflow to achieve the goal set out by the user. FEPAC, a framework developed for collecting and analyzing the execution data [92], is used to create a constant feedback loop between the workflow execution and the adaptations conducted by the scheduler by using the MAPE model (shown by red lines).

The proposed scheduler, by default, constantly analyzes the current state of the execution for any exploits that can be used to improve the performance or the energy consumption of the workflow. This can be done through number of policies that are pre-defined and known to achieve reduction in energy consumption R1 and R8. The proposed scheduler only affects the scheduling of the jobs and does not change the underlying data or computation of the workflow R2.

A set of predefined goals and thresholds are provided to the adaptive schedule. A user specified configuration file can overwrite these default goals and thresholds. This aligns to the requirement R3 and R4. These goals or thresholds can be a result of budget or compute resources constraints and workflow characteristics. The scheduler does not aim to replace any existing scheduler that is executing the workflow but works in tandem with the scheduler by editing the constraints to the scheduler to achieve the user's goal. The proposed scheduler tests the workflow for any faults before submitting it to the scheduler. The default scheduler can handle any breaks gracefully if occurred R5.

The proposed scheduler logs each step of the MAPE model (monitoring data, analysis output, planning decisions, execution triggers) that is used to perform adaptations. This helps in better understanding which policy was used by the scheduler and how it was implemented in case of debugging R6. Along with this, the scheduler can allow export of the logged data so that it can be analyzed by any third party tools R7.

4.2.6 Possible Energy-Focused Scheduling Adaptations

The MAPE approach to decomposing adaptations enables a range of adaptations to be supported. The aim of these adaptations is to have an impact on the energy consumption of the executing workflow, ideally leading to a reduction. The following are possible adaptations which can be supported with an energy-focused adaptive workflow scheduler:

- Change the allocation of workflow jobs to different physical processors to change the energy profile of the job
- Change the number of jobs allocated to each physical node to change the overall load pattern across nodes
- Switching on or off physical nodes to change the overall available computation capacity or the energy consumption of the cluster
- Change the allocation proportion to different types of nodes dependent on the performance per Watt for particular jobs
- React to changes in the usage of nodes which will have an impact on the availability of physical nodes
- Change the priority of execution of specific workflow jobs to force jobs to be allocated to resources sooner
- Reschedule workflow jobs if physical cluster nodes are using more than expected energy

This comprehensive proposal establishes the theoretical and methodological foundation for developing practical energy-aware scheduling solutions that can significantly reduce the environmental impact of scientific computing while maintaining the performance levels required for cutting-edge research.

4.3 Experimental Setup

The experimental evaluation of the proposed energy-aware scheduling frameworks requires a comprehensive testing environment that accurately represents real-world scientific computing scenarios while providing precise measurement capabilities for both performance and energy consumption metrics. In this section, the hardware infrastructure, software environments, representative scientific workflows, and measurement methodologies that collectively enable rigorous assessment of scheduling effectiveness are presented.

4.3.1 Hardware Infrastructure Configuration

The experimental infrastructure consists of a purpose-built heterogeneous cluster designed to reflect the diversity commonly found in modern research computing environments. The cluster architecture enables evaluation of scheduling policies across different computational capabilities

and energy profiles, providing insights into the effectiveness of energy-aware scheduling across varied hardware configurations.

Similar to Figure 3.2, the experimental cluster comprises ten single-board computers organized into two distinct hardware categories to create a heterogeneous computing environment. Five **High-Performance Nodes (Node IDs 1-5)** were Raspberry Pi 4B units with 2GB RAM, Broadcom BCM2711 Quad-core ARM Cortex-A72 processors and provides enhanced computational capabilities for intensive tasks. The remaining five **Standard-Performance Nodes (Node IDs 6-10)** were Raspberry Pi 3B+ units equipped with 1GB RAM, Broadcom BCM2837 Quad-core ARM Cortex-A53 processors and were used for standard computational capabilities for general processing and to provide heterogeneity to the cluster system. This configuration provides a representative heterogeneous environment where scheduling decisions can demonstrate significant impact on both performance and energy consumption.

All cluster nodes are interconnected through two Netgear GS110TP managed switches that provide both network connectivity and power distribution via Power over Ethernet (PoE) technology. The network infrastructure offered several advantages such as integrated power monitoring, centralized management of the cluster and possibility of scalability. The cluster is managed by an Intel NUC with quad-core processor (8 threads) running Linux Mint OS (`https://linuxmint.com/`), serving as the master node for workflow submission, monitoring, and control. The master node operates independently of the compute cluster to avoid interference with energy measurements.

Rationale for Single-Board Computer Selection

Single board computers have always been known for their energy efficient nature [92, 64, 254]. For this study, a 10 node heterogeneous single board cluster was developed. This was done to better understand the effect of different compute resources have on computation. This also help to evaluate the proof of concept scheduler by introducing different compute resources. The choice of single-board computers for the experimental cluster is motivated by several factors that enhance the relevance and accuracy of the evaluation:

- **Energy Efficiency Focus**: Single-board computers are inherently energy-efficient, making them ideal platforms for demonstrating energy optimization techniques that may be more difficult to observe in high-power systems [92, 64, 254]
- **Cost-Effective Scaling**: The relatively low cost of single-board computers enables construction of multi-node clusters with limited budget constraints, facilitating realistic multi-node experiments

- **Heterogeneity Representation**: The availability of different single-board computer models with varying performance characteristics enables creation of heterogeneous environments representative of production clusters
- **Measurement Precision**: The lower absolute power consumption of single-board computers improves the signal-to-noise ratio of energy measurements, enabling more precise evaluation of optimization effectiveness

4.3.2 Software Environment Configuration

The experimental environment utilizes the Pegasus Workflow Management System [11] as the primary platform for workflow execution and management. Pegasus offers several advantages for energy aware scheduling research as outlined in Section 2.4.2. Pegasus makes use of HTCondor [128], a batch job scheduler and resource management system, to submit and execute jobs on the cluster (see Section 2.2.2). The 1.5 degrees variant of the Montage Workflow (see Section 2.4.6) and Bioinformatics workflow with 10k complexity (see Section 2.4.6) was executed on the cluster of 10 nodes. As the Raspberry Pi's (RPis) have quad code processors, HtCondor considers each core as an independent execution resource/ thread. Considering that the cluster is made up of 10 Rpis, there are total of 40 threads that can execute jobs in parallel.

4.4 Experimental Evaluation of Static Scheduler

The evaluation of energy-aware scheduling frameworks requires comprehensive assessment across multiple dimensions including energy efficiency, performance impact, and practical applicability. The evaluation methodology compares proposed scheduling approaches against standard workflow execution to quantify both benefits and trade-offs associated with energy-aware optimization. Multiple experimental scenarios are conducted to validate the effectiveness of different scheduling policies and demonstrate the practical viability of energy-aware scheduling in real-world scientific computing environments. The jobs that are being executed on the cluster nodes are being considered in the analysis of the data. Master node specific jobs such as file/folder creation or file transfers are not considered.

4.4.1 Standard Execution of Bioinformatics Workflow

Standard execution of the Bioinformatics Workflow includes creating the workflow using Pegasus and submitting it to the cluster. There are no extra configurations associated with executing the workflow. All the default and standard configuration options are used. This experiment denotes

the normal workflow execution that any scientist performs using Pegasus. The data from this experiment is considered a baseline for any future comparisons between the scheduler's different policies.

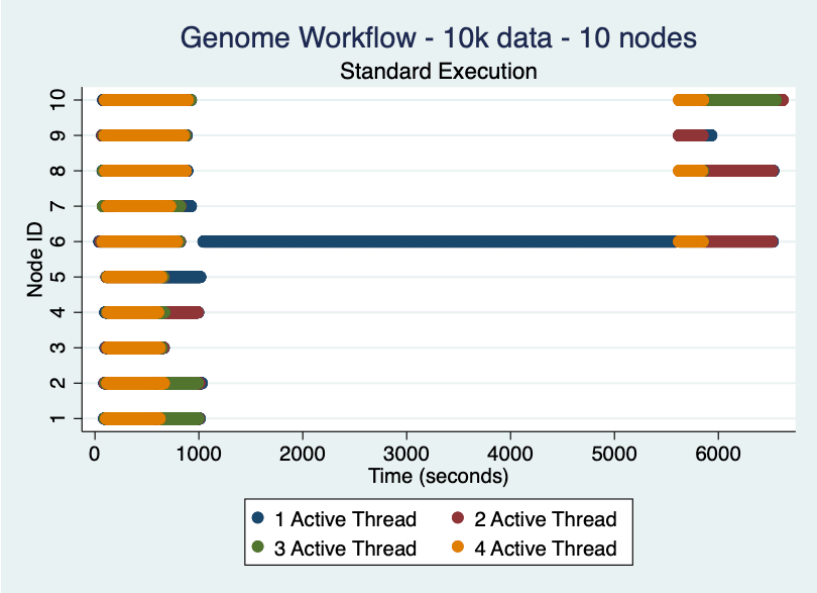


Figure 4.4: Active Threads Analysis for Standard Execution of Bioinformatics Workflow

The execution data of the experiment is shown below. Figure 4.4 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant, and the Y-axis indicates the node ID. Different usage of nodes is denoted by different colored dots. The analysis only considers the jobs that are executed on the cluster nodes.

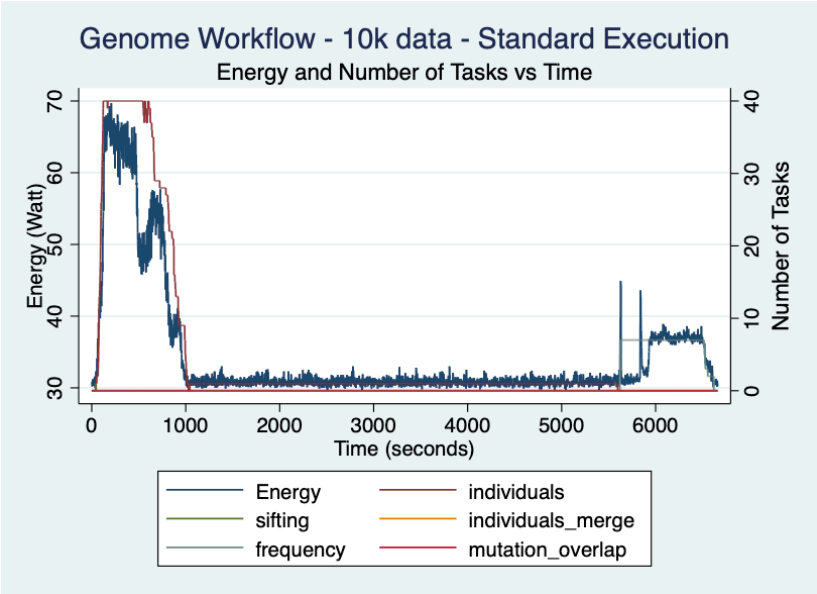


Figure 4.5: Job Analysis for Standard Execution of Bioinformatics Workflow

Figure 4.5 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster, and the right Y-axis denotes the number of jobs for each job type being executed on the cluster. Different colors denote different job types.

The standard execution of the Bioinformatics workflow on the heterogeneous cluster of 5 RPi 4B and 5 RPi 3B+ took 6,658 seconds (approximately 1 hr and 51 min). The cluster uses the maximum energy during the execution of the *individuals* job. As can be seen from Figures 4.4 and 4.5, the majority of the cluster is idle during the execution of *individuals_merge* job. This is expected behavior as the *indiviual_merge* job is a bottleneck in the Bioinformatics workflow (see Figure 2.8). The execution of the workflow has to stop completely and cannot proceed without the completion of this job. The *individuals* job is compute-intensive, and a drop in energy consumption can occur as soon as the job is complete and other jobs are executed. Considering that this experiment denotes the standard execution of a workflow, the idle nodes still consume their base energy during the execution. The standard execution of the Bioinformatics workflows consumed around 63.92 Watt-hr of energy.

It is important to note that the selection of the node to execute the bottleneck job is dependent on the internal scheduling policies of the HTCondor [128]. HTCondor schedules jobs based on the requirements of the job and the available resources. In this instance, all the nodes can execute the job, but only node 6 (see Figure 4.4) was available at that instant, which resulted in HTCondor scheduling the job on those nodes. The execution time and energy consumption data of each individual job are used to develop policies that will help the scheduler in achieving the reduction in energy consumption and improvement of performance in the computation. For example, *individuals* job on the RPi 4B took around 507 seconds (approximately 8 min and 27 sec) and RPi 3B+ took around 749 seconds (approximately 12 min and 29 sec). Similar data were collected for all the different jobs and used to develop policies for the scheduler accordingly. The data found that RPi 4B (Nodes 1-5) performs better than RPi 3B+ (Nodes 6-10) in most computations.

4.4.2 Bioinformatics Workflow: Scheduling Single Set of Jobs Policy

The previous section presented the standard execution data of the Bioinformatics workflow. A proof of concept static scheduler based on the conceptual architecture presented in Section 4.2.1 was developed, and the Bioinformatics workflow is scheduled and executed according to the policies set by the scheduler. The main aim of the scheduler is to achieve a reduction in the energy consumption of the workflow at no or minimal cost to the performance. A policy to utilize the whole cluster to execute the compute-intensive jobs and then schedule the remaining jobs on nodes that have better performance is being evaluated in this section.

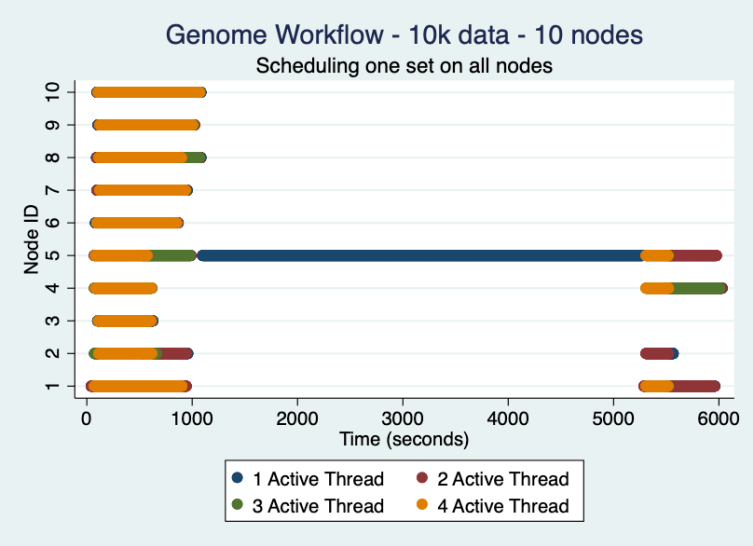


Figure 4.6: Bioinformatics Workflow Execution Analysis for Scheduling a Single Set of Jobs

Figure 4.6 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant, and the Y-axis indicates the node ID. Different colored dots denote different usage of nodes.

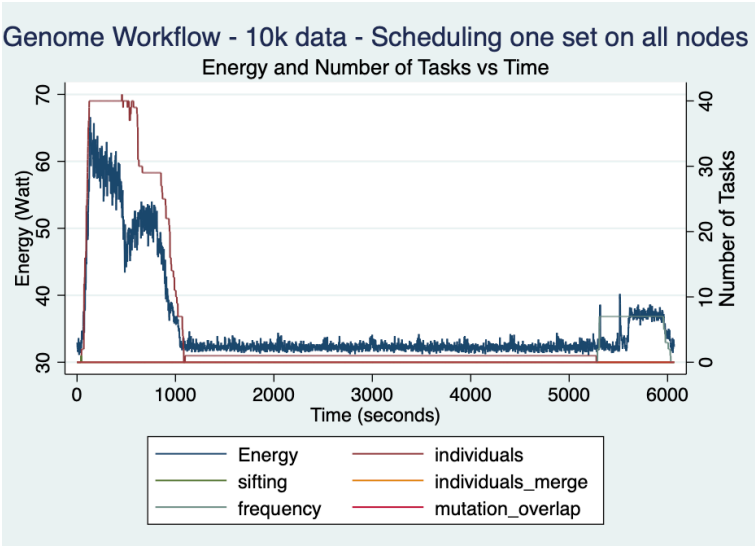


Figure 4.7: Job Analysis for Scheduling a Single Set of Jobs for Bioinformatics Workflow

Figure 4.7 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster, and the right Y-axis denotes the total number of jobs being executed on the cluster. Different colors represent different jobs.

The scheduler used the following policy for the execution - after the initial execution of compute-intensive jobs on all the nodes, all the remaining jobs are scheduled only on the nodes with higher performance. This can be confirmed from Figure 4.6 that illustrates the scheduled tasks on the nodes. A similar observation to that of the standard execution can be seen while scheduling the bottleneck job (scheduled on node 5). The total execution time of the workflow was 6,067 seconds (approximately 1 hr and 41 min), an improvement of around 8.86% in the performance of the workflow as compared to that of standard execution. The energy consumption of the execution is found to be 38.25 Watt-hr (40.26% reduction in energy consumption as compared to that of standard execution). As presented in Section 4.2.1, the scheduler powers the idle nodes off.

4.4.3 Bioinformatics Workflow: Scheduling All Jobs Policy

The previous experiment showed that the RPi 4B is very energy efficient as compared to the RPi 3B+. To investigate this further, the scheduler was used to execute the same workflow again with a different policy. The policy was to schedule the jobs only on the RPi 4B and to consider the RPi 3B+ as powered off and not in use.

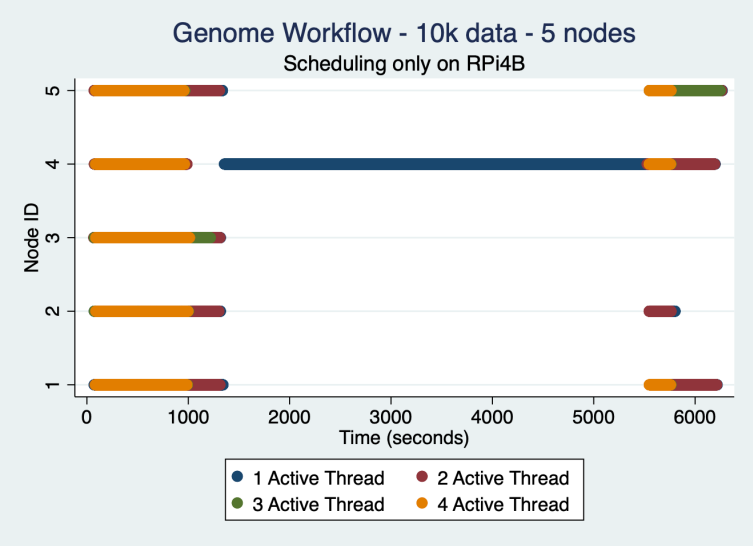


Figure 4.8: Active Threads Analysis for Bioinformatics Workflow Execution and Scheduling on Energy-Efficient Nodes

The execution data of the experiment is shown below. Figure 4.8 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant, and the Y-axis indicates the node ID. Different usage of nodes is denoted by different colored dots. The analysis only considers the job that is executed on the cluster nodes.

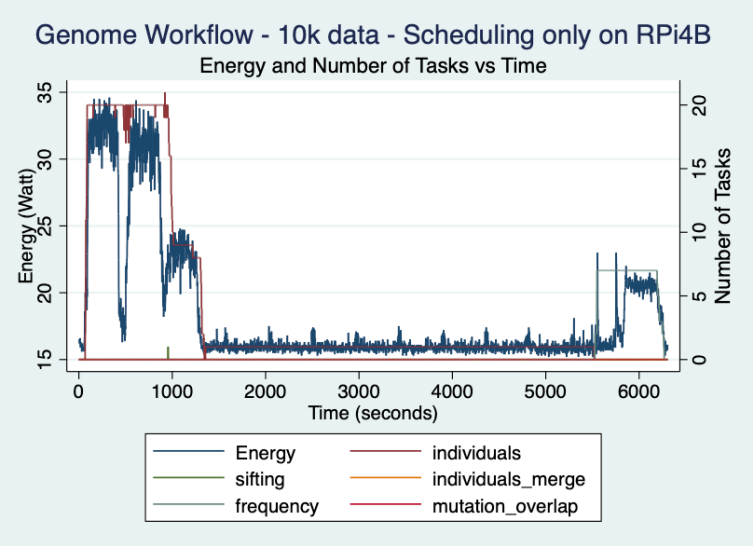


Figure 4.9: Number of tasks and Energy Consumption Analysis for Bioinformatics Workflow Execution and Scheduling on Energy-Efficient Nodes

Figure 4.9 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster and the right Y-axis denotes the total number of jobs being executed on the cluster. Different jobs are denoted by different colors.

As the number of nodes that are being used is only 5, the maximum number of jobs that can be parallelly executed is 20. This can be confirmed from Figure 4.9. A similar observation to that of the standard execution can be seen while scheduling the bottleneck job (scheduled on node 4). The maximum number of the *individuals* jobs that are being executed parallelly is 20. Due to this, the maximum energy consumption of the whole cluster is also around 35 Watts as compared to 65 Watts in the previous experiments. The total execution time of the workflow was 6,305 seconds (approximately 1 hr and 45 min). The execution time is 5.29% less than that of the standard execution but 3.92% higher as compared to the previous policy.

The total energy consumption of the computation in this policy was around 32.23 Watt-hr. This is a decrease of around 49.97% as compared to the standard execution (see Section 4.4.1). Comparing this with the execution data from the previous policy, this policy achieved a reduction of 15.74% in energy consumption. In terms of the execution time, the policy to schedule all the jobs on RPi 4B performs poorly than the policy to schedule compute-intensive jobs on all nodes and then schedule all remaining jobs on high-performing nodes. When comparing energy consumption, the current policy performs better than the standard execution and the execution using the previous policy. The debatable question here can be if the reduction of energy consumption is worth the increase in execution time.

This comprehensive evaluation demonstrates that energy-aware scheduling can achieve significant energy efficiency improvements while maintaining or enhancing performance in scientific workflow execution. The results validate both the theoretical frameworks and practical implementation approaches proposed in this chapter, establishing a foundation for broader adoption of energy-aware scheduling in scientific computing environments.

4.5 Experimental Evaluation of the Adaptive Scheduler

The previous section confirmed the workings of the static scheduler developed in this chapter. In this section, the adaptive scheduler is evaluated against the Montage workflow to compare the effectiveness of the adaptive functionality against the static scheduling approach.

The standard execution of the workflow (see below) is compared with the execution of the adaptive scheduler and the performance and energy consumption of the computation is collected and analyzed. Two major metrics were used to compare the standard and adaptive scheduler's execution – amount of resources being used (illustrated by Figures 4.10, 4.12 and 4.14), and total number of jobs being executed on the cluster (illustrated by Figures 4.11, 4.13 and 4.15).

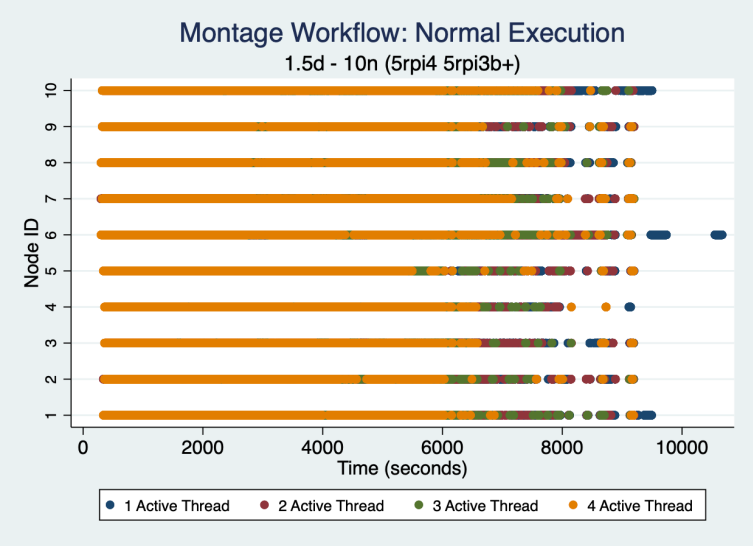


Figure 4.10: Active Threads Analysis for Standard Execution of Montage Workflow

4.5.1 Standard Execution of Montage Workflow

Standard execution of the Montage Workflow includes creating the workflow using Pegasus and submitting it to the cluster. There are no extra configurations associated with executing the workflow. All the default and standard configuration options are used. This experiment denotes the normal workflow execution that any scientist performs using Pegasus. The data from this

experiment is considered as a baseline for any future comparisons between the different policies of the adaptive scheduler.

The execution data of the experiment is shown below. Figure 4.10 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant and the Y-axis indicates the node ID. Different usage of nodes are denoted by different colored dots. The analysis only considers the jobs that are executed on the cluster nodes.

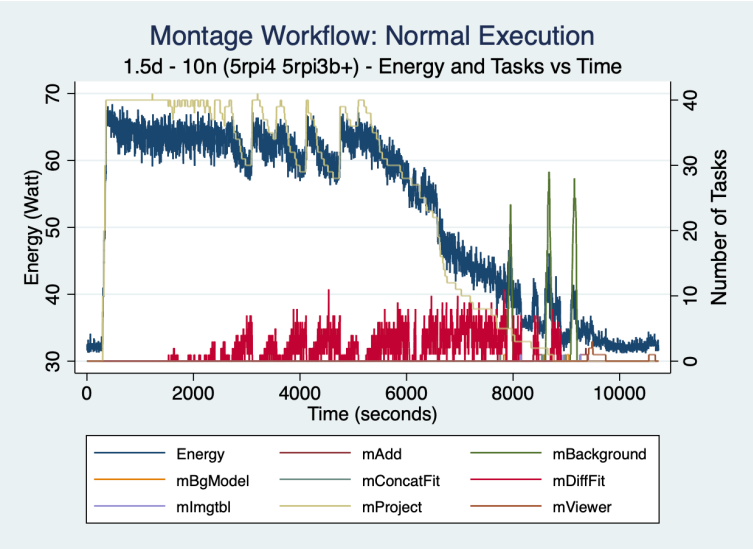


Figure 4.11: Job Analysis for Standard Execution of Montage Workflow

Figure 4.11 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster and the right Y-axis denotes the number of jobs for each job type being executed on the cluster. Different job types are denoted by different colors.

The standard execution of the Montage workflow on the heterogeneous cluster took 10,730 seconds (approximately 2 h and 59 min). For majority of the execution, the cluster resources were being fully utilized (denoted by orange color in Figure 4.10). As seen from Figure 4.11, the cluster uses maximum energy during the execution of mProject jobs. This is expected as the mProject job is highly compute intensive and maximizes the resource computing power during execution. As the mProject jobs are completed and other jobs start to execute, a drop in the energy consumption of the cluster is observed that shows that the other jobs are comparatively less compute intensive. During the total execution of the workflow, the computation consumed around 154.07 Watt-hr of energy.

4.5.2 Montage Workflow: Adapting after 1st set of jobs are completed

A proof of concept adaptive scheduler based on the design in Section 4.2.2 was developed, and the Montage workflow is executed using it. The Adaptive scheduler is configured to achieve a reduction in energy consumption of the computation. The scheduler waits for the first set of jobs from different nodes to complete and schedules the remaining jobs based on the data analysis of the executed jobs.

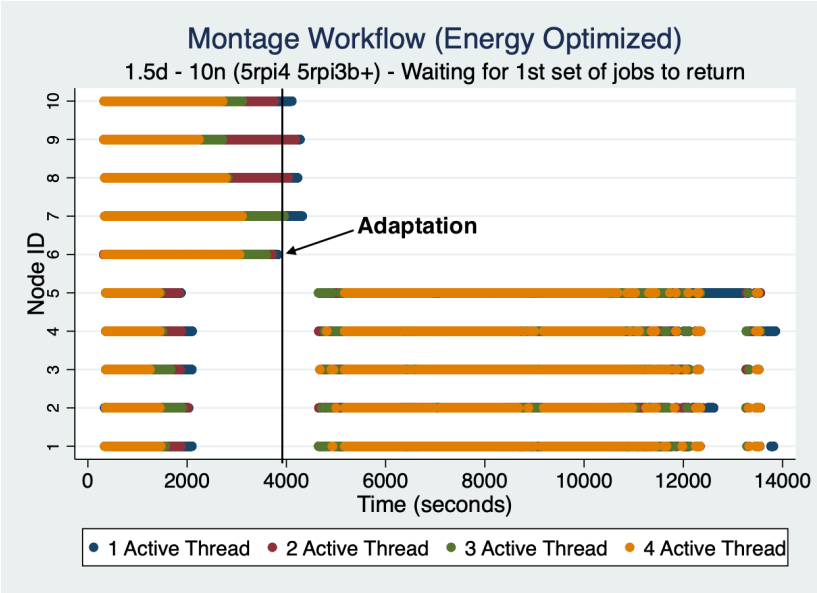


Figure 4.12: Active Threads Analysis for Adaptive Scheduler Execution of Montage Workflow - Waiting for completion of first job on different nodes

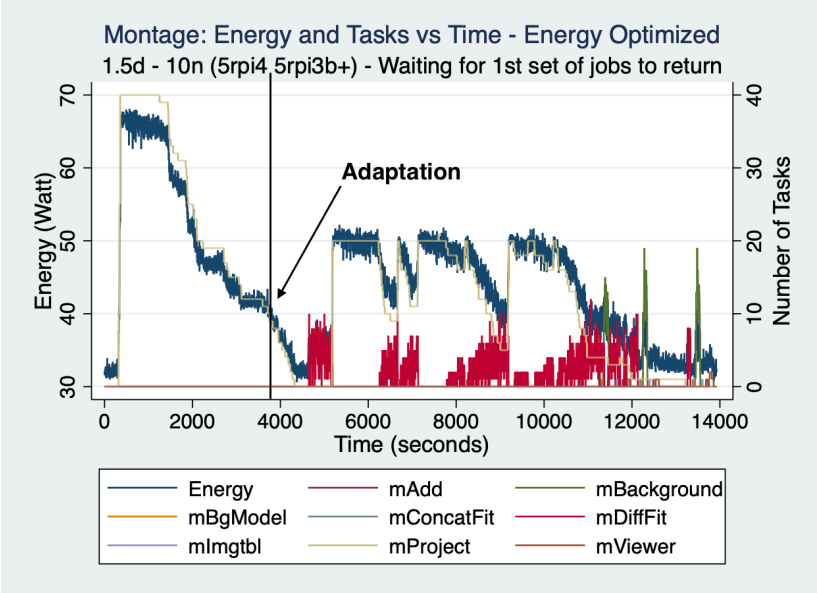


Figure 4.13: Job and Energy Consumption Analysis for Adaptive Scheduler Execution of Montage Workflow - Waiting for completion of first job on different nodes

The execution data of the experiment is shown below. Figure 4.12 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant and the Y-axis indicates the node ID. Different usage of nodes are denoted by different colored dots. The analysis only considers the jobs that are executed on the cluster nodes.

Figure 4.13 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster and the right Y-axis denotes the total number jobs being executed on the cluster.

As seen in Figure 4.12, the scheduler refrains from scheduling any new job on the cluster until the first set of jobs are completed. This is the policy that was set at the beginning of the experiment and this helps the scheduler understand the execution power of all nodes and take decisions accordingly. The total execution time of the workflow was 13,636 seconds (approximately 3 h and 47 min). The time does not include the time taken by the scheduler to analyze, adapt the workflow and reschedule the jobs.

The Rpi 4B finishes their first job faster (around 26 min and 20 s) than Rpi3B+ that took around 1 h to complete. The scheduler receives the execution data around that time and reschedules the jobs according to the policies set out in Section 4.2.2. In this particular experiment, the scheduler decided that the Rpi 4B, in terms of energy consumption per job, was better at executing jobs than the Rpi 3B+. Thus, all the rescheduled jobs were only executed on Rpi 4B and the Rpi 3B+ were powered off. This lead to the total energy consumption of the 129.66 Watt-hr for the computation. The energy consumption of idle Rpi4B after they complete their jobs are also considered as they have not been powered off during that time.

4.5.3 Dynamic Adaptation Analysis of Montage Workflow

In the previous experiment, it can be seen that the Rpi 4B were idle during the execution of the jobs on Rpi 3B+. To further investigate the energy consumption of the workflow when the nodes are being fully utilized, the following experiment was conducted.

In this experiment, the adaptive scheduler is configured to achieve reduction in energy consumption of the workflow by analyzing the first job being completed on different configuration nodes. But until the first job is completed on nodes that are different, the execution of the workflow is progressed in the standard way. Once the first job is completed on the different node, the scheduler will analyze, adapt and reschedule all the non executing jobs in the workflow. This will make sure that the new jobs are scheduled and executed according to the policies set out in Section 4.2.2.

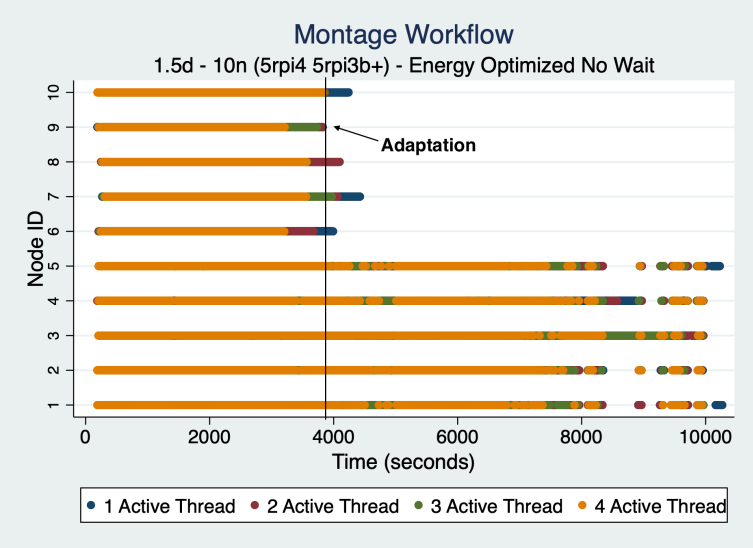


Figure 4.14: Active Threads Analysis for Adaptive Scheduler Execution of Montage Workflow - Normal scheduling until the completion of the first job on different nodes

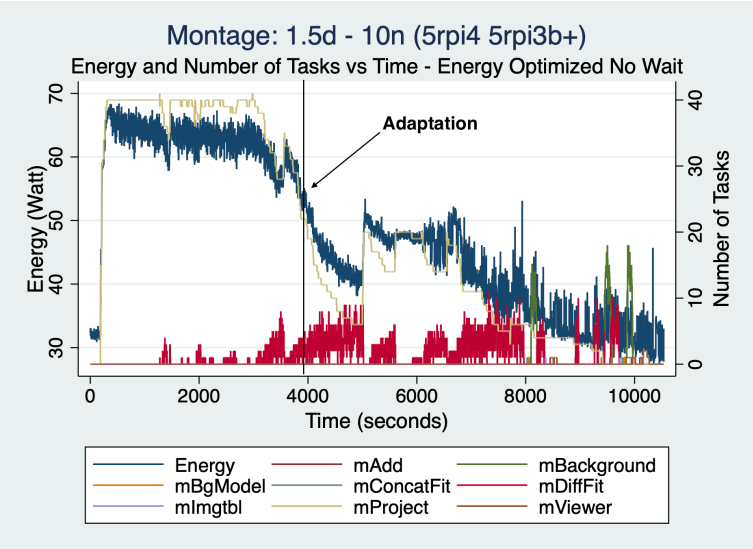


Figure 4.15: Job and Energy Consumption Analysis for Adaptive Scheduler Execution of Montage Workflow - Normal scheduling until first set of jobs complete

The execution data of the experiment is shown below. Figure 4.14 illustrates the number of active threads on each node during the execution of the workflow. The X-axis indicates the execution time (in seconds) of the workflow at any given instant and the Y-axis indicates the node ID. Different usage of nodes are denoted by different colored dots. The analysis only considers the job that are executed on the cluster nodes. Figure 4.15 illustrates the breakdown of the jobs that are being executed on the cluster and the energy consumption of the cluster. The X-axis indicates the execution time (in seconds) of the workflow at any given instant. The left Y-axis is the instantaneous energy consumption of the cluster and the right Y-axis denotes the total number jobs being executed on the cluster. As seen in Figure 4.14, the scheduler schedules the jobs normally until the first job is completed on a different node. This means that when the first

job on Rpi 3B+ was completed, the Rpi 4B were executing their second set of jobs. This can be confirmed from the figure that shows that Rpi 4B was always fully utilized. The total execution time of the workflow was 10,531 seconds (approximately 2 h and 56 min). The time does not include the time taken by the scheduler to analyze, adapt the workflow, and reschedule the jobs.

The scheduler then makes the necessary adaptations to the remaining non-executing jobs of the workflow according to its analysis. The execution time of each job on Rpi 4B and Rpi 3B+ was the same as the previous experiment. Similar to the last experiment, the scheduler decided to schedule any new jobs on the Raspberry Pi 4B only, as it was a better candidate to execute jobs in terms of energy efficiency. The Rpi 3B+ was considered powered off after the execution of its first set of jobs. The total energy consumption of this experiment was 115.82 Watt-hr.

The results illustrated in Figure 4.10 might suggest that the cluster resources are being fully utilized and that this is the optimal execution solution to the workflow. This is proven wrong by the adaptive scheduler that finds a more optimal scheduling of jobs that leads to reduced energy consumption of the workflow. The standard execution of the workflow took 10,730 seconds with an energy consumption of 154.07 Watt-hr. The first adaptation policy resulted in execution time being 13,636 seconds and consumed 129.66 Watt-hr of energy. The execution time increased around 27% when executed using the adaptive scheduler, but a reduction of 15.84% is achieved in energy consumption. This is expected as the scheduler was configured to favor more energy-efficient nodes rather than performance.

Similar results can be found when comparing the standard solution with a different policy of the adaptive scheduler. The second policy execution of the workflow resulted in an execution time of 10,531 seconds and an energy consumption of 115.82 Watt-hr. Coincidentally, this policy resulted in an increase in performance as well as a decrease in energy consumption of the workflow execution. A decrease of 1.85% was obtained on the execution time, and a 24.82% reduction was achieved in the energy consumption. Hence, the third policy resulted in a reduction of *both*, execution time and energy consumption compared to the standard execution.

The performance and energy consumption data obtained from the three experiments conducted align and further solidify the need for an energy-aware adaptive scheduler. The evaluation of the proof-of-concept adaptive scheduler shows that it can help in reducing the energy consumption of the workflow. Section 4.2 describes the generic conceptual design of an Energy-Aware Scheduler based on the requirements set out in Section 4.2.3. The scheduler introduces a solution to reduce the energy consumption of the computation by tackling its three main parts – whole cluster, individual jobs, and the workflow dependencies [36]. This three-point solution focuses on the significant aspects of workflow execution that can affect energy consumption and aid the scheduler in the decision-making process. To confirm the functioning of the proposed scheduler, a proof-of-concept scheduler is developed and evaluated using a scientific workflow in this section.

4.6 Summary

This chapter has presented a comprehensive framework for energy-aware scheduling of scientific workflows, demonstrating significant advances in sustainable high-performance computing through intelligent resource allocation and adaptive optimization strategies. The research contributions span theoretical framework development, practical implementation methodologies, and empirical validation of energy optimization effectiveness across diverse computational scenarios.

This chapter has successfully demonstrated that energy-aware scheduling of scientific workflows can achieve significant energy efficiency improvements while maintaining or enhancing computational performance. The theoretical frameworks, implementation methodologies, and experimental validation presented here establish a solid foundation for advancing sustainable practices in scientific computing.

The combination of static and adaptive scheduling approaches provides a comprehensive toolkit for addressing different optimization scenarios and user requirements. The static scheduling framework offers predictable optimization with minimal implementation complexity, while the adaptive MAPE-based approach enables sophisticated runtime optimization that can respond to dynamic execution conditions.

In the next chapter, a more advanced energy-aware scheduler will be implemented and evaluated. The focus will be on workflow-specific adaptations that understand the characteristics of particular workflows as they are submitted. More advanced approaches include the ability to schedule multiple different types of workflow simultaneously on a set of shared resources. This further work aims to improve energy-efficient workflow execution in more advanced and realistic scenarios.

Chapter 5

EMWOS: Energy Monitoring and Workflow Optimization Scheduler System

The contents of the chapter have been published in the following articles:

1. Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. (2023). Optimizing workflow execution for energy consumption and performance. In *2023 IEEE/ACM 7th International Workshop on Green and Sustainable Software (GREENS)* (pp. 24–29). IEEE. `https://doi.org/10.1109/GREENS59328.2023.00011`

Contents of this chapter have been submitted for publication in the following venues:

- Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. A Multi-User Energy-Aware Scheduler for Scientific Workflows. *IEEE Transactions on Emerging Topics in Computing*.
- Warade, M., Lee, K., Ranaweera, C., & Schneider, J.-G. Optimization of scientific workflows for energy consumption and performance. *IEEE/ACM Automated Software Engineering (ASE)*.

5.1 Overview

Scheduling scientific workflows presents unique challenges due to their inherently complex nature, intricate task dependencies, heterogeneous resource requirements, and varying execution patterns [37]. These workflows often exhibit complex structures with diverse computational characteristics, making the development of universally effective scheduling strategies a challenging endeavor. Modern computing infrastructure is predominantly heterogeneous, wherein specialized

hardware components demonstrate superior performance for different types of computations. Such components differ significantly in terms of processing capabilities, energy efficiency, and communication costs. The dynamic nature of execution environments introduces substantial uncertainty that must be addressed through robust and adaptive scheduling approaches. Optimization techniques provide a systematic methodology for addressing this multifaceted scheduling problem by mathematically modeling the trade-offs between conflicting objectives, such as minimizing makespan, energy consumption, and resource utilization [227].

When optimizing for energy consumption and performance in workflow computation, it becomes essential to consider the various elements that can influence both the energy consumption and performance characteristics of the computation [255]. These elements include the underlying hardware and software infrastructure, the algorithms and data structures used, and the overall system architecture. Additionally, the type of workloads processed can have significant impact on both cost and performance metrics of the computation. This comprehensive understanding has been explored extensively during the development and evaluation of the reliable energy monitoring system presented in Chapter 3.

Chapter 4 presented a scheduling framework for intelligent and adaptive execution of scientific workflows. The research demonstrated that achieving substantial reductions in energy consumption for computational tasks without compromising performance is indeed feasible. Building upon these foundational insights, this chapter introduces a novel **Energy Monitoring and Workflow Optimization Scheduler System (EMWOS)** that addresses the pressing need for an energy-aware scheduler system capable of optimizing workflow execution in multi-user computing environments.

This chapter adopts a systematic approach toward the development of EMWOS. Initially, a proposal for an optimization-enabled framework is presented, providing the foundational architecture for the system. Subsequently, the framework is extended to incorporate support for multi-user environments. At this developmental stage, EMWOS performs scheduling decisions based on pre-defined policies and historical knowledge of execution outcomes. However, this approach becomes less feasible as computing environments grow increasingly complex. Consequently, the scheduling decisions in EMWOS are ultimately managed through optimization algorithms that minimize user intervention requirements. This evolution results in a self-sufficient system capable of optimizing scientific workflow execution to achieve goals established by users or system administrators within multi-user execution environments.

The proposed scheduling approach is evaluated through multiple test scenarios, comprising of both single and multiple workflow executions with varied user preferences on a cluster designed to replicate real-world computing environments. The results confirm that the system successfully manages energy consumption while maintaining performance levels that align with diverse user preferences. This research significantly advances the field of energy-aware workflow scheduling

by providing a practical framework for data centers and high-performance computing (HPC) environments to effectively manage energy consumption while meeting diverse user performance requirements.

The remainder of this chapter is organized as follows. Section 5.2 presents the mathematical modeling of different computational components necessary for developing an optimization framework. Section 5.3 introduces the proposed Energy Monitoring and Workflow Optimization Scheduler System (EMWOS). Section 5.4 details the iterative implementation of the EMWOS system and outlines the specific functions of each subsystem. Initially, EMWOS is implemented to provide support for multi-user computing environments, subsequently enhanced with optimization algorithms for intelligent scheduling of computations. Section 5.5 presents the experimental infrastructure used to evaluate the proposed EMWOS system. Section 5.6 presents the evaluation of EMWOS for its multi-user computing environment support. Section 5.7 provides comprehensive testing and evaluation of EMWOS with implemented optimization algorithms. Finally, Section 5.8 summarizes the chapter findings and outlines current achievements and future research directions.

5.2 Mathematical Modeling and Problem Formulation

Having established the need for an energy-aware scheduler system in multi-user environments, the mathematical foundations necessary for developing the EMWOS optimization framework are presented in this section. The complexity of scheduling scientific workflows in heterogeneous computing environments requires complex mathematical modeling to capture the intricate relationships between workflow characteristics, resource capabilities, and optimization objectives. This section develops formal representations of workflows, resources, and multi-user constraints, resulting in a comprehensive multi-objective optimization problem formulation.

Different estimation models have been developed to bridge the gap between theoretical formulation and practical implementation, focusing on the execution time and energy consumption based on various runtime factors. These models are inspired by existing industry standard models [63, 66]. These models enable more accurate scheduling decisions in real-world heterogeneous environments. The mathematical models presented here form the theoretical backbone for the EMWOS system design and implementation discussed in subsequent sections.

5.2.1 Workflow Representation and Modeling

A Scientific workflow (see Figure 5.1) is modelled as a directed acyclic graph (DAG) G, where $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$ representing the set of computational tasks and $E \subseteq V \times V$ represents the dependencies between these tasks. Each task $v_i \in V$ denotes a specific independent

computational task within the scientific workflow. A directed edge $e_{ij} \in E$ from task v_i to task v_j indicates that task v_j can only commence execution after the completion of task v_i, establishing precedence constraints within the workflow. In this instance, the task v_i is characterized as the parent of the task v_j.

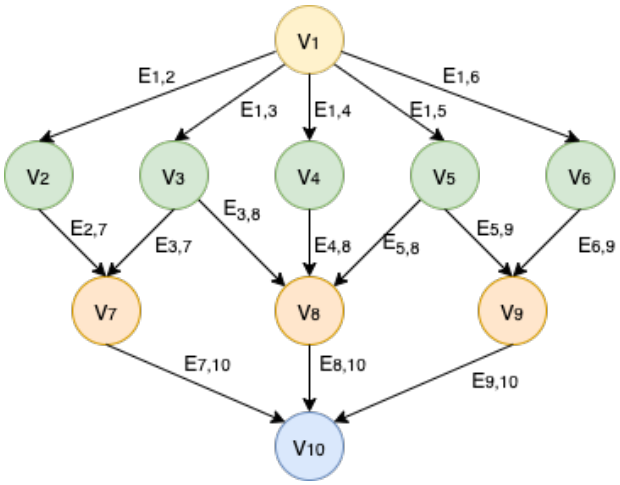


Figure 5.1: Scientific Workflow Representation of Ten Tasks

The dependency relationship between tasks is formally defined as:

$$D_{ij} = \begin{cases} 1, & \text{if task } t_i \text{ is parent of } t_j \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

The DAG structure enables precise modeling of task dependencies and parallel execution opportunities, facilitating efficient resource allocation and scheduling decisions.

5.2.2 Power Consumption Model

Many factors affect the power consumption of a computation. The ideal scenario would be to monitor the energy consumption using a dedicated monitoring infrastructure. Still, it is not always possible, and power modeling is widely accepted as an industry standard to calculate the energy consumption of a computation [200]. An industry standard model is also developed here based on similar research [129] in power prediction for workflow execution. The power consumption $P(t)$ at any time t is modeled as a linear function of CPU utilization:

$$P(t) = (P_{\text{max}} - P_{\text{idle}}) \cdot u(t) \cdot \frac{1}{n} \tag{5.2}$$

Where:

- P_{max}: Maximum power consumption at peak utilization
- P_{idle}: Idle power consumption
- $u(t)$: CPU utilization at time t
- n: Number of processor cores being utilized

The total energy consumption E for a task is calculated as:

$$E = r \cdot P_{\text{idle}} + \int_0^r P(t)dt \tag{5.3}$$

Where r represents the task's execution time. This model accounts for both static and dynamic power consumption components, enabling accurate energy usage prediction and optimization.

5.2.3 Resource Modeling

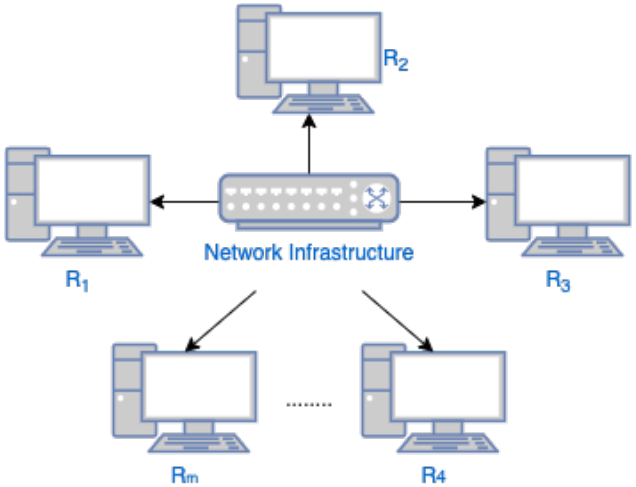


Figure 5.2: Compute Cluster Modeling for Resource Allocation

The computing cluster consists of interconnected resources managed by a master node. The resource components are defined as presented in Figure 5.2. Figure 5.2 shows a scaled-down version of a compute cluster with a central switch or networking infrastructure connected with four compute nodes and a master node.

Each compute node R_i is characterized by the following notations:

R_m : Master node responsible for workflow management
R_i : Compute nodes where $i \in \{1, 2, ..., n\}$
M_{R_i} : Memory capacity of resource i
S_{R_i} : Storage capacity of resource i
C_{R_i} : Computing capacity of resource i
$M_{R_{\text{max}}}$: Maximum available memory across all resources
$S_{R_{\text{max}}}$: Maximum available storage across all resources

The master node R_m orchestrates workflow execution by (i) accepting and parsing workflow definitions, (ii) tracking task dependencies, (iii) monitoring resource utilization, and (iv) coordinating task assignments. The processing capability C_{R_i} directly influences task execution time and is a crucial factor in scheduling decisions that balance performance and energy consumption.

The following parameters characterize each resource r_j:

- P_r: Raw Processing power of resource r (measured in MIPS - Millions of Instructions Per Second)
- $P_{base}(r)$: Base power consumption of resource r (measured in watts)

Resources exhibit heterogeneity in terms of computational capabilities and energy efficiency. This heterogeneity must be considered when making scheduling decisions, as different tasks may exhibit varying performance and energy consumption profiles across different resources.

5.2.4 Formalizing the Scheduling of Scientific Workflows

A scheduler maps jobs to available computing resources while maintaining execution requirements and dependencies.

A scheduling algorithm adapted to scientific workflow execution is presented in Algorithm 1. A valid schedule must ensure task dependencies from the workflow $D = (T, V)$ are maintained during execution. For any task, the $ft(t_i)$ is the finish time of parent task t_i and $st(t_j)$ is the start time of child task t_j.:

$$ft(t_i) < st(t_j) \tag{5.4}$$

Algorithm 1 Dependency-Aware Workflow Scheduler

Require: Workflow DAG $D = (T, V)$, Resources R_i
Ensure: Valid Schedule S
$S \leftarrow \emptyset$
$Q \leftarrow$ Initialize queue with entry tasks
$C \leftarrow$ Track completion times
while $Q \neq \emptyset$ **do**
    $t \leftarrow$ Get next task from Q
    if all parents of t in C **then**
        $est \leftarrow \max\{ft(p) : p \text{ is parent of } t\}$
        Find R_i available after est
        $S \leftarrow S \cup \{(t, R_i, est)\}$
        $C \leftarrow C \cup \{(t, est + \text{duration}(t))\}$
        Add ready children to Q
    end if
end while
return S

5.2.5 Modelling Estimated Completion Time (ECT)

The ECT model predicts the execution time that a task j will take to execute on a resource r.

$$\text{ECT}(j, r, S, D, N) = T_{\text{transfer}}(j, D, N) + T_{\text{compute}}(j, r, S) \tag{5.5}$$

Where:

- j is the job to be executed.
- r is the computational resource.
- S is the current state of the resource, such as being assigned, busy, allocated, and job completing.
- D is the data source information.
- N is the network information.

T_{transfer} is the time it takes to transfer all the required input files for the job j to execute and is calculated as:

$$T_{\text{transfer}}(j, D, N) = \frac{B_j}{B_N} \cdot (1 + L_{source}) \tag{5.6}$$

Where:

- B_j is the data size of job j (in MB).

- B_N is the network bandwidth (in MB/s).
- L_{source} is the CPU load percentage of the data source (in decimal).

The computation time T_{compute} is calculated as:

$$T_{\text{compute}}(j, r, S) = \frac{I_j}{P_r} \cdot (1 + L_r) \tag{5.7}$$

Where:

- I_j is the number of CPU instructions required by job j.
- P_r is the MIPS (Millions of Instructions Per Second) performance of resource r.
- L_r is the CPU load percentage of resource r (in decimal).

This formulation captures data transfer time and computation time, considering network conditions and current CPU loads to estimate job completion time efficiently.

5.2.6 Modelling Estimated Energy Consumption (EEC)

The EEC model predicts the energy consumption that a job will consume while executing on a resource.

$$\text{EEC}(j, r, S) = P_{\text{base}}(r) \cdot T_{\text{compute}}(j, r, S) \cdot L_E(S) \tag{5.8}$$

Where:

- j is the job to be executed.
- r is the computational resource.
- S is the current state of the resource, such as being assigned, busy, allocated, and job completing.
- $P_{\text{base}}(r)$ is the base power consumption of resource r (in watts).
- T_{compute} is the computation time from the ECT function (in seconds).
- $L_E(S)$ is the energy load factor.

The energy load factor $L_E(S)$ is defined as:

$$L_E(S) = 1 + \frac{L_r}{\gamma} \tag{5.9}$$

Where L_r is the CPU load percentage of resource r, and γ is a system-dependent scaling parameter that determines the rate at which energy consumption increases with CPU load. This formulation captures the superlinear relationship between CPU load and energy consumption, where energy usage increases more rapidly than execution time under high system loads [65].

This formulation focuses solely on computational energy consumption, providing a clean estimate based on resource power characteristics and current system load.

5.2.7 Optimization Problem Formulation

The optimization problem aims to minimize both makespan and energy consumption:

$$\text{Minimize} \quad \alpha \cdot C + \beta \cdot E \tag{5.10}$$

where:

1. $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$ are weighting parameters that balance performance and energy objectives.

2. The makespan (C) of the workflow, calculated as the sum of the execution time of all the tasks:

$$C = \sum_{i \in V} C_i \tag{5.11}$$

   where C_i represents the completion time of task v_i. C_i is calculated as below.

$$C_i = S_{i,j} + ECT_{i,j} \quad \forall i \in V, \forall j \in R \tag{5.12}$$

   The $S_{i,j}$ and $ECT_{i,j}$ represent the start time and the Estimated Completion Time of task v_i on resource r_j, respectively.

3. The total energy consumed (E) during workflow execution, calculated as:

$$E = \sum_{j \in R} \sum_{i \in V} x_{i,j} \cdot EEC_{i,j} \quad \forall i \in V, \forall j \in R \tag{5.13}$$

where $x_{i,j} \in \{0, 1\}$ is a binary decision variable that indicates whether task v_i is assigned to resource r_j and $EEC_{i,j}$ represents the Estimated Energy Consumption of task v_i on resource r_j.

5.2.8 Multi-User Environment Constraints

The search for the solution to the proposed optimization problem is subject to constraints provided below:

1. **Assignment Constraints**: Each task must be assigned to exactly one resource.

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad \forall v_i \in V \tag{5.14}$$

2. **Precedence Constraints**: A task can only start after all its predecessors have completed.

$$S_k \geq C_i, \quad \text{if } i \text{ precedes } k \quad \forall i, k \in V \tag{5.15}$$

3. **Resource Capacity Constraint:** Tasks on the same resource must execute sequentially.

$$C_i \leq S_j \text{ OR } C_j \leq S_i \quad \forall v_i, v_j \in V, \forall r_k \in R \tag{5.16}$$

$$\text{where } x_{i,k} = x_{j,k} = 1, i \neq j \tag{5.17}$$

5.2.9 Sets, Parameters and Decision Variables

The mathematical formulation of the energy-aware scheduling problem requires a precise definition of the computational entities, system characteristics, and optimization variables that capture the essential aspects of workflow execution [22, 45]. The sets, parameters, and decision variables used in the problem formulation are explained in this section. Sets represent the collections of entities in the problem domain, parameters capture the known characteristics and constraints of the system, and decision variables represent the scheduling choices to be optimized [50]. These definitions establish the foundation for the subsequent mathematical formulations of execution time estimation, energy consumption modeling, and the hybrid optimization framework.

The mathematical model represents scientific workflows as directed acyclic graphs (DAGs) where computational tasks are connected through data dependencies, and the scheduling problem involves optimal assignment of these tasks to heterogeneous computing resources while minimizing both execution time and energy consumption [14, 37].

Sets:

The fundamental entities in the energy-aware scheduling problem are organized into three primary sets that define the workflow structure and available computing infrastructure:

- $V = \{v_1, v_2, \ldots, v_n\}$: Set of computational tasks in the workflow
- $R = \{r_1, r_2, \ldots, r_m\}$: Set of heterogeneous computing resources
- $E \subseteq V \times V$: Set of directed edges representing task dependencies

Parameters:

The system parameters capture both the computational characteristics of workflow tasks and the performance capabilities of computing resources. These parameters are essential for accurate modeling of execution time and energy consumption:

- I_j: Number of CPU instructions required by the j^{th} task in V, where $j \in V$
- P_r: Processing power of resource r (in MIPS - Millions of Instructions Per Second), where $r \in R$
- $P_{\text{base}}(r)$: Base power consumption of resource r (in watts), where $r \in R$
- B_j: Data size of task v_j (in MB), where $j \in V$
- B_N: Network bandwidth (in MB/s)
- L_r: CPU load percentage of resource r (in decimal), where $r \in R$
- L_{source}: CPU load percentage of the data source (in decimal)
- $ECT_{i,j}$: Estimated completion time of task v_i on resource r_j, where $i \in V$ and $j \in R$
- $EEC_{i,j}$: Estimated energy consumption of task v_i on resource r_j, where $i \in V$ and $j \in R$
- $\alpha, \beta \in [0, 1]$: Weighting parameters for makespan and energy objectives, where $\alpha + \beta = 1$
- γ: System-dependent scaling parameter for energy-load relationship, where $\gamma > 0$

Decision Variables:

The decision variables represent the scheduling choices that the optimization algorithm must determine to achieve optimal workflow execution. These variables define the complete schedule and resource allocation strategy:

- $x_{i,j} \in \{0, 1\}$: Binary variable indicating whether task v_i is assigned to resource r_j
- $S_{i,j}$: Start time of task v_i on resource r_j
- C_i: Completion time of task v_i

5.3 Energy Monitoring and Workflow Optimization Scheduler System (EMWOS)

Building upon the mathematical foundations established in the previous section, this section introduces the core architecture and design principles of the Energy Monitoring and Workflow Optimization Scheduler System (EMWOS). The system represents a significant advancement from the basic scheduling frameworks discussed in earlier chapters by incorporating real-time energy monitoring, multi-user management, and intelligent optimization capabilities. This section details the system's modular architecture, examining how each component contributes to achieving energy-efficient workflow scheduling while maintaining fairness and performance in multi-user environments. The architectural design presented here provides the blueprint for the implementation phases discussed in the following section.

Algorithm 2 Energy-Aware Priority-Based Scheduling (EAPBS)

Require: Workflows $W = \{w_1, ..., w_n\}$, User preferences $P = \{p_1, ..., p_m\}$, Resources $R = \{r_1, ..., r_k\}$
Ensure: Schedule S meeting preferences, energy constraints and DAG dependencies

$S \leftarrow \emptyset$; $Q \leftarrow \emptyset$ ▷ Initialize schedule and priority queue
for each $w_i \in W$ **do**
    Identify critical path CP_i and calculate base priority BP_i for w_i
    for each task $t \in w_i$ **do**
        $priority_t \leftarrow \alpha \cdot BP_i + \beta \cdot$ CalculateWeight$(t, P) - \gamma \cdot$ EstimateEnergy(t, R)
        $Q.enqueue(t, priority_t)$
    end for
end for
while $Q \neq \emptyset$ **do**
    $t \leftarrow Q.dequeue()$
    if AllDependenciesMet(t) **then**
        $r_{best} \leftarrow$ FindOptimalResource$(t,$ GetAvailableResources$(R))$
        $S \leftarrow S \cup \{(t, r_{best},$ FindEarliestSlot$(r_{best}))\}$
        UpdateResourceState(r_{best})
    else
        $Q.enqueue(t,$ UpdatePriority$(priority_t))$
    end if
end while
return S

5.3.1 Energy-Aware Priority-Based Scheduling Algorithm

A novel Energy-Aware Priority-Based Scheduling Algorithm (EAPBS) has been developed to form a part of the core scheduler of the proposed scheduler system. This algorithm is aware of the energy consumption of the computing system and determines the job scheduling order based on it. The algorithm respects the user preference and prioritizes the scheduling of jobs depending on multiple factors such as the weightage of the jobs, critical path priority, overall performance, and contention of resources. Algorithm 2 provides the pseudo-code for the EAPBS algorithm.

5.3.2 Core Scheduling Policies

The core scheduling policies that govern workflow scheduling serve a distinct purpose within the scheduler system, catering to diverse operational needs and objectives. By adhering to these policies, the system ensures efficient energy monitoring and workflow optimal scheduling, ultimately enhancing productivity and sustainability.

- **Performance High**: The "Performance High" policy prioritizes system efficiency and computational throughput. The scheduler maximizes resource utilization to expedite workflow execution, allocating resources based on workload demands. It suits scenarios where completion time is the primary concern, such as deadline-driven scientific computations or time-critical data processing. The policy prioritizes high-performance computing and processing, enabling the system to achieve peak operational capacity.
- **Balanced**: The "Balanced" policy implements equitable resource distribution across workflows, ensuring fair allocation in multi-user environments. It prevents resource monopolization by maintaining balanced utilization across all active workflows, making it particularly effective in shared computing environments where multiple users compete for resources.
- **Sustainable High**: The "Sustainable High" policy aims for minimal energy consumption while maintaining acceptable performance levels. The scheduler employs algorithms and techniques derived from previous research [36] to minimize energy usage without significantly compromising execution time, making it suitable for users prioritizing sustainability or operating under power constraints.

5.3.3 Policy Implementation Rules

The development of effective energy-aware scheduling policies requires systematic analysis of existing workflow scheduling approaches and their adaptation to incorporate energy

considerations [35, 67]. The core scheduling policies presented here are derived from previous research and established principles in high-performance computing, energy-efficient system design, and multi-objective optimization research [22, 36, 64, 68, 201]. Each policy category addresses specific optimization objectives while incorporating lessons learned from both theoretical scheduling research and practical deployment experiences in scientific computing environments [23, 37]. The core scheduling policies use specific rules governing resource allocation and task scheduling.

Performance High Scheduling Policy Rules

The Performance High policy draws from established high-performance computing scheduling principles that have been extensively validated in production HPC environments [17, 31]:

1. Prioritize tasks on the critical path.
2. Implement task clustering for reduced communication overhead.
3. Maximize parallel execution of jobs.
4. Schedule the jobs first on high-compute power resources.

Energy High Scheduling Policy Rules

The Energy High policy incorporates principles from energy-efficient computing research and green computing initiatives that focus on minimizing power consumption while maintaining computational effectiveness [16, 40]:

1. Prioritize scheduling on active resources to minimize new node power-up overhead.
2. Maximize resource utilization before activating additional nodes.
3. Consider power state transitions in scheduling decisions.

Balanced Scheduling Policy Rules

The Balanced policy uses principles from fair scheduling research and multi-objective optimization studies that address the need to balance competing objectives while ensuring equitable resource access [33, 45]:

1. Maintain equitable resource distribution across workflows.

2. Implement fair-share scheduling for multiple users. All resources should be equally burdened, and the computation should progress equally.

3. Balance the workloads across available resources to prevent hotspots.

4. Schedule compute-intensive jobs with less intensive jobs on a resource instead of over-stressing a resource.

The scheduler system implements these rules and the scheduling algorithm to ensure it respects user preferences and energy constraints. The relative weight of each rule can be adjusted based on the active policy and specific deployment requirements through parameterized policy configuration that allows fine-tuning based on workload characteristics and system constraints.

Algorithm 3 Proposed Hybrid Optimization Algorithm

Input:
Workflow $G = (V, E)$
Resources R
Weights α, β
Scheduling Time Budget: T
Maximum time allocated for MIP: T_M
Minimum time allocation for Heuristics: T_H = (T - T_M)
Output: Schedule $S = \{(v_i, r_j, t_{start})\}$
Phase 1: MIP Optimization
— Set termination constraints —
$max\_mip\_time\_limit \leftarrow \min(T \times 0.7, T_M)$
$improvement\_threshold \leftarrow 0.01$ // 1% improvement needed
$stagnation\_limit \leftarrow 3$ // iterations without improvement
— Objective function formulation —
Formulate MIP with objective: Minimize $\alpha \cdot C + \beta \cdot E$
Subject to constraints
— Solve MIP to obtain the mip optimized schedule S^*_{mip} —

$$S^*_{mip} \leftarrow mip.optimize(G, R, \alpha, \beta, mip\_time\_limit, improvement\_threshold, stagnation\_limit)$$

Phase 2: Heuristic Enhancement
$remaining\_time \leftarrow T - elapsed\_time$
if $remaining\_time > T_H$ **then**
$S \leftarrow heuristic.solve(S^*_{mip}, G, R, \alpha, \beta, remaining\_time)$
else
$S \leftarrow S^*_{mip}$
end if
return $S = \{(v_i, r_j, t_{start}) : \forall v_i \in V\}$

5.3.4 Hybrid Optimization Algorithm

The framework employs a sequential optimization approach that maximizes solution quality while respecting computational constraints. This approach has been implemented as a part of the Hybrid Optimization Algorithm.

The pseudocode for the proposed hybrid optimization framework is presented in Algorithm 3. The algorithm enables users to specify the maximum time budget for computing the optimal solution, providing flexibility in balancing solution quality with computational constraints. This approach efficiently allocates computational effort between exact optimization and heuristic refinement based on solution progress and available time resources. The algorithm generates a schedule represented as a set of tuples, where each tuple contains the job identifier, assigned resource, and the start time of the job execution on the designated resource.

5.3.5 Two-Phase Optimization Strategy

The framework employs a sequential two-phase approach that capitalizes on the complementary strengths of Mixed-Integer Programming (MIP) and heuristic optimization methods. The first phase utilizes MIP to establish a strong foundation solution by systematically exploring the solution space and identifying globally optimal or near-optimal schedules. The second phase employs heuristics to refine and enhance the MIP solution, focusing on local improvements that may be computationally tedious for exact methods within the remaining time budget.

This sequential approach is motivated by the observation that MIP solvers excel at finding high-quality solutions early in the optimization process but often exhibit diminishing returns as they approach optimality [159, 256]. Conversely, heuristic methods can quickly identify local improvements and explore solution neighborhoods that the systematic search patterns of exact methods may overlook [257, 167]. By combining these approaches sequentially, the framework ensures that the computational budget is utilized most effectively [171, 82].

The time allocation strategy between phases is designed to be both adaptive and conservative. The framework allocates a maximum of 70% of the total time budget to the MIP phase, ensuring that sufficient time remains for heuristic enhancement regardless of MIP performance. This allocation strikes a balance between allowing the MIP solver enough time to find high-quality solutions and ensuring that the heuristic phase has ample time to contribute meaningful improvements and has been empirically validated to provide an optimal balance between solution quality and computational efficiency [73]..

5.3.6 Hybrid Optimization System Architecture

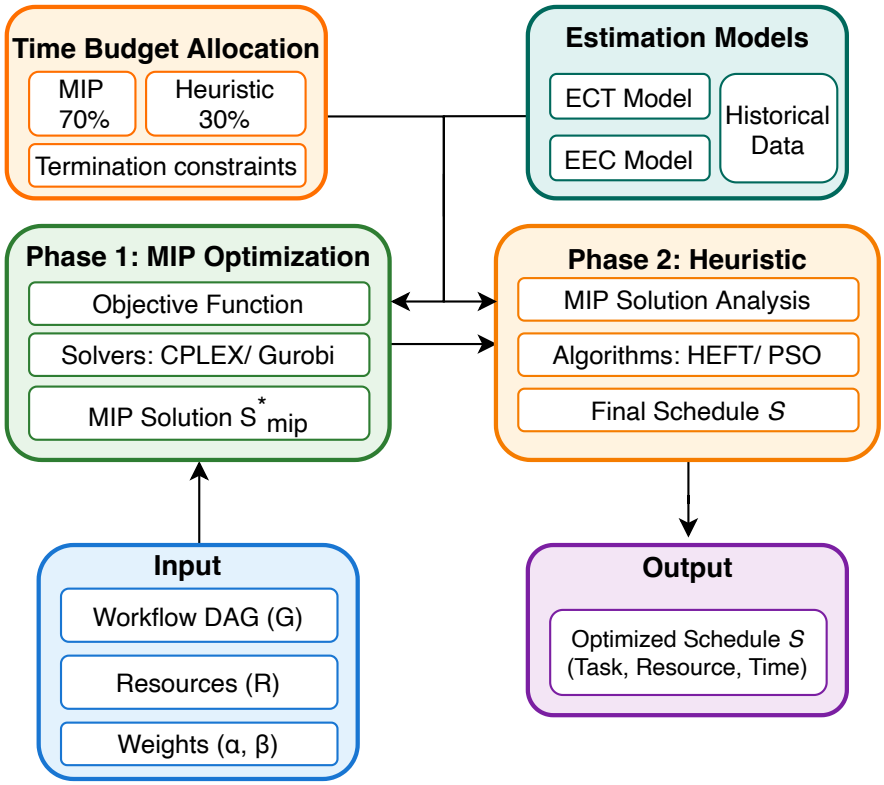


Figure 5.3: Hybrid Optimization Framework Architecture

The hybrid scheduling framework represents the two-stage optimization system that systematically addresses the computational complexity inherent in energy-aware workflow scheduling while maintaining solution quality and practical applicability [45, 47]. The framework accepts workflow DAGs and resource specifications as input. It produces optimized schedules through sequential solution refinement, combining the mathematical rigor of Mixed Integer Programming (MIP) with the computational efficiency of heuristic optimization methods [50, 159]. The framework architecture, illustrated in Figure 5.3, employs MIP optimization as the primary solution method, followed by heuristic enhancement to achieve near-optimal solutions within practical time constraints.

Framework Input Processing and Configuration

The framework begins with processing the input that establishes the optimization parameters and constraints necessary for effective energy-aware scheduling. The input processing stage accepts three critical components that define the complete scheduling problem:

The **Workflow DAG (G)** serves as the primary structural input, representing the scientific workflow as a directed acyclic graph where nodes correspond to computational tasks and edges

represent data dependencies [8]. The framework performs extensive analysis of the DAG structure, identifying critical paths, parallelization opportunities, and communication bottlenecks that influence both performance and energy consumption. This analysis includes calculation of task priorities based on dependency depth, estimation of parallelism levels, and identification of resource-intensive operations that require special scheduling consideration.

The **Resources (R)** specification provides detailed information about the available computing infrastructure, including processor capabilities, memory configurations, network connectivity, and energy consumption characteristics [31]. The framework processes this information to create performance and energy models for each resource, enabling accurate prediction of task execution times and energy consumption across the heterogeneous computing environment. Resource characterization includes static properties such as processing power and memory capacity, as well as dynamic factors such as current utilization levels and power state information.

The **Weights (α, β)** parameters allow users to specify their preferences for the trade-off between minimizing execution time and reducing energy consumption. These weights are processed through the framework's objective function configuration system, which ensures that the optimization process appropriately balances performance and energy objectives according to user requirements and system constraints.

Time Budget Allocation and Optimization Strategy

The hybrid framework utilizes an intelligent time budget allocation mechanism to dynamically distribute computational resources between the MIP optimization phase and the heuristic refinement phase. The framework allocates 70% of the available optimization time to MIP-based exact optimization and 30% to heuristic-based solution enhancement. This distribution has been empirically validated to provide an optimal balance between solution quality and computational efficiency [73].

The time budget allocation incorporates **termination constraints** that prevent excessive computation time while ensuring meaningful optimization progress [82]. These constraints include maximum wall-clock time limits, convergence criteria based on objective function improvement rates, and solution quality thresholds that trigger early termination when acceptable solutions are found. The framework continuously monitors optimization progress and can dynamically adjust the time allocation between phases based on problem complexity and solution convergence characteristics.

Estimation Models and Predictive Analytics

The framework incorporates sophisticated **Estimation Models** that provide accurate predictions of task execution times and energy consumption across different resource configurations [64, 68]. These models are continuously updated using **Historical Data** collected from previous workflow executions, enabling the framework to improve its predictive accuracy over time through machine learning techniques and statistical analysis.

The **ECT (Estimated Completion Time) Model** utilizes task computational requirements, resource processing capabilities, and current system load to predict execution times for each task-resource combination. The model incorporates factors such as CPU instruction counts, memory access patterns, and I/O requirements to provide accurate time estimates that account for the heterogeneous nature of modern computing systems.

The **EEC (Estimated Energy Consumption) Model** combines resource power characteristics, task computational intensity, and system utilization patterns to predict energy consumption for different scheduling decisions. This model considers both static power consumption (baseline resource energy usage) and dynamic power consumption (additional energy required for computational work), enabling comprehensive energy-aware optimization.

Phase 1: Mixed Integer Programming Optimization

The first phase of the hybrid framework employs exact optimization techniques using Mixed Integer Programming to find mathematically optimal or near-optimal solutions to the energy-aware scheduling problem [50, 160]. This phase formulates the scheduling problem as a mathematical optimization model with clearly defined objective functions, decision variables, and constraints. The **Objective Function** implements multi-objective optimization that simultaneously minimizes workflow makespan and total energy consumption according to the user-specified weight parameters [48]. The objective function is carefully formulated to ensure that both performance and energy objectives are appropriately scaled and balanced, preventing one objective from dominating the optimization process.

The framework employs professional-grade **Solvers** including CPLEX and Gurobi, which represent state-of-the-art commercial optimization engines capable of handling large-scale mixed integer programming problems [256]. These solvers utilize advanced branch-and-bound algorithms, cutting plane methods, and heuristic techniques to explore the solution space and identify optimal task-resource assignments efficiently. The MIP optimization produces an initial solution S^*_{mip} that represents the best achievable schedule within the allocated time budget and computational constraints. This solution serves as both a high-quality baseline and the starting point for subsequent heuristic refinement. The MIP phase maintains complete solution feasibility

throughout the optimization process, ensuring that all dependency constraints and resource limitations are strictly observed.

Phase 2: Heuristic Enhancement and Solution Refinement

The second phase of the hybrid framework applies heuristic optimization techniques to further improve the MIP solution through targeted local search and intelligent solution space exploration [169, 257]. This phase recognizes that while MIP optimization provides mathematically rigorous solutions, practical time constraints may prevent full convergence to global optima, creating opportunities for heuristic improvement.

The **MIP Solution Analysis** component performs a detailed examination of the initial MIP solution to identify potential areas for improvement. This analysis includes bottleneck identification, resource utilization assessment, and energy consumption pattern evaluation. The framework identifies specific aspects of the schedule that may benefit from heuristic optimization, such as task reordering opportunities, resource reallocation possibilities, and load balancing improvements.

The framework implements multiple **Algorithms** including HEFT (Heterogeneous Earliest Finish Time) and PSO (Particle Swarm Optimization) to explore different regions of the solution space and identify superior scheduling configurations [22, 164]. HEFT provides efficient task prioritization and resource selection based on computational requirements and resource capabilities. At the same time, PSO enables global search techniques that can escape local optima and discover innovative scheduling strategies. The heuristic phase produces the **Final Schedule S** that represents the best achievable solution combining mathematical optimization rigor with heuristic innovation. This schedule undergoes final validation to ensure feasibility and constraint satisfaction before being presented as the framework output.

Framework State Management and Solution Quality Assurance

The framework maintains a comprehensive **optimization state** throughout both stages, enabling seamless transition between exact and approximate optimization methods while preserving solution quality and feasibility. This state management system tracks solution evolution, maintains constraint satisfaction, and ensures that heuristic modifications do not violate fundamental scheduling requirements. The state management system includes solution version control that allows the framework to revert to previous solutions if heuristic modifications result in degraded performance. Quality assurance mechanisms continuously monitor objective function values, constraint satisfaction, and solution feasibility throughout the optimization process.

Output Generation and Schedule Specification

The framework produces a comprehensive **Optimized Schedule S** that specifies the complete task assignment and timing information necessary for workflow execution [179]. The output includes detailed **Task, Resource, and Time** specifications that define exactly when each workflow task should be executed, on which computing resource, and with what configuration parameters.

The final schedule output includes execution timing information that respects all workflow dependencies, resource allocation details that optimize both performance and energy consumption, and configuration parameters that specify optimal resource settings for each scheduled task. This comprehensive output enables direct implementation by workflow management systems while providing detailed information for performance analysis and energy monitoring.

The hybrid optimization framework thus provides a robust, scalable, and practical solution to the energy-aware workflow scheduling problem, combining the mathematical rigor necessary for optimal resource utilization with the computational efficiency required for real-world deployment scenarios.

5.3.7 Scheduler System Architecture

The proposed scheduling system has been implemented through a distributed architecture comprising of four specialized daemons: Controller, Monitoring, Allocator, and Executor. Each daemon has been designed with a specific responsibility that works in coordination to achieve the system's objectives. This modular approach enables robust workflow management while maintaining system flexibility and scalability.

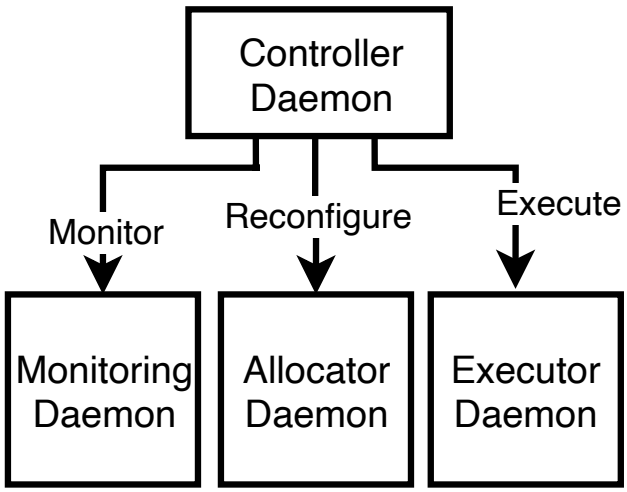


Figure 5.4: Controller Daemon Architecture

Controller Daemon

The Controller Daemon functions as the central orchestrator of the scheduling system. This component has been developed to coordinate the activities of other daemons by:

- Managing communication between system components.
- Initiating workflow execution processes.
- Understanding the requirements of the execution.
- Invoking other Daemons with necessary parameters or needed information.
- Maintaining system state and ensuring operational coherence.

Figure 5.4 provides the architecture of the controller daemon. The controller is the first daemon that is invoked and has access to all the computation and workflow requirements. It is responsible for executing the EAPBS algorithm, along with the help of other daemons, to achieve the system's goals.

Monitoring Daemon

The Monitoring Daemon has been implemented to provide comprehensive energy consumption tracking through specialized power monitoring hardware. The daemon:

- Collects real-time energy consumption data from compute nodes.
- Maintains a centralized database of power usage metrics.
- Analyze the energy data to provide valuable insights into how the system's goals are being achieved.
- Provide the energy data to other daemons to help improve the scheduling decisions.

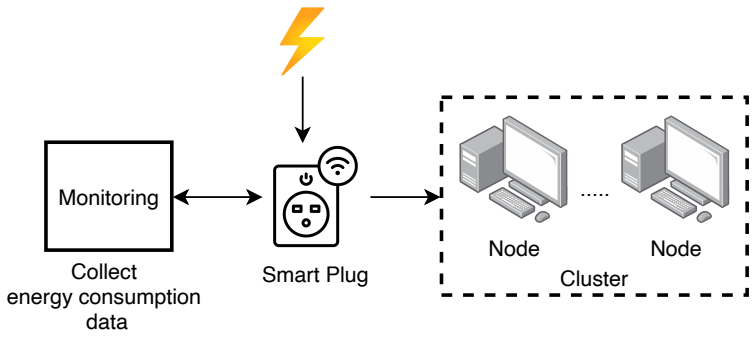


Figure 5.5: Monitoring Daemon Architecture

Figure 5.5 provides the architecture of the monitoring daemon. Many specialized energy monitoring hardware systems are available that provide an interface for the data to be collected in an accurate and streamlined manner. This enables the collection of the **real energy consumption** data instead of predicting or calculating the energy consumption of the computation.

Resource Allocator Daemon

The Resource Allocator Daemon represents a critical component of the energy-aware scheduling framework, designed to dynamically configure compute nodes based on workflow requirements and system optimization objectives. This daemon serves as the bridge between high-level scheduling decisions and low-level resource management, ensuring that computing resources are optimally configured to support both performance requirements and energy efficiency goals. The daemon operates continuously throughout workflow execution, monitoring system state and adjusting resource allocations in response to changing computational demands and energy consumption patterns.

The Resource Allocator Daemon is responsible for intelligent resource configuration that adapts to workload characteristics and system constraints. Computationally expensive tasks can benefit from increased CPU core allocation and expanded memory access, while I/O-intensive operations may require enhanced storage bandwidth and network connectivity. The daemon analyzes task requirements in real-time and implements appropriate resource configurations that maximize computational efficiency while minimizing energy consumption.

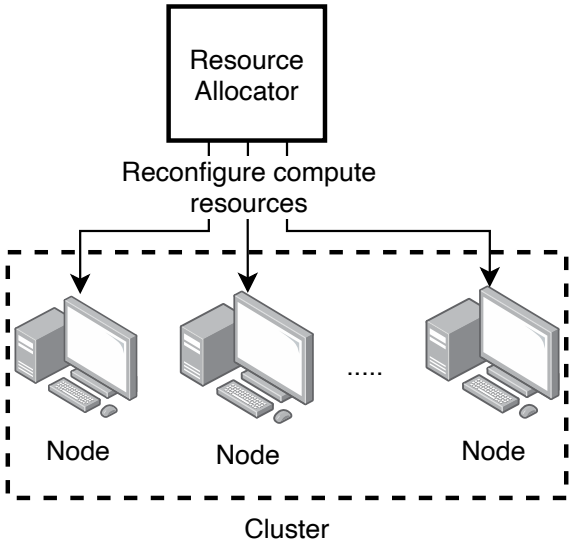


Figure 5.6: Resource Allocator Daemon Architecture

The daemon's operation extends beyond simple resource allocation to include comprehensive system optimization that considers thermal management, power consumption patterns, and long-

term resource availability. By intelligently managing resource configurations, the daemon can significantly impact overall workflow performance and energy efficiency, making it a crucial component for achieving sustainable high-performance computing.

Figure 5.6 illustrates the comprehensive architecture of the Resource Allocator Daemon. The block diagram highlights the sophisticated interaction between cluster-wide monitoring, individual node management, and energy-aware optimization components. The architecture demonstrates how compute nodes are intelligently configured to support different resource allocations for various jobs while maintaining system-wide coordination and optimization objectives.

Algorithm 4 Resource Allocation Algorithm

```
Retrieve current cluster state and node information
Extract node specifications (CPUs, memory, network capacity)
Determine optimal configuration based on workflow requirements
for each target node in cluster do
    Read current node configuration
    Update configuration parameters:
        • Parallel execution slots
        • Memory allocation
        • CPU core assignment
        • Storage allocation
        • Network bandwidth limits
    Apply configuration changes and restart the scheduler on the node
    Verify node state after reconfiguration
end for
```

The resource management algorithm presented in Algorithm 4 provides the detailed procedural framework that governs daemon operation, ensuring consistent and optimal resource allocation across diverse workflow execution scenarios. The algorithm incorporates comprehensive validation and monitoring capabilities that enable continuous optimization and adaptation to changing system conditions, making it suitable for deployment in production high-performance computing environments where reliability and efficiency are paramount concerns.

Executor Daemon

The Executor Daemon serves as the operational core of the energy-aware scheduling framework. It is responsible for implementing the optimized scheduling decisions into actual workflow execution while maintaining continuous oversight of system performance and energy consumption. This daemon represents the final execution layer that bridges the gap between theoretical scheduling optimization and practical workflow deployment, ensuring that computed schedules are implemented effectively while adapting to real-time system conditions and execution dynamics.

The daemon's responsibilities extend beyond simple task execution to include comprehensive workflow life-cycle management that encompasses pre-execution validation, runtime monitoring, performance optimization, and post-execution analysis. The system is designed to handle the dynamic nature of scientific workflow execution, where computational requirements may vary significantly between tasks and system conditions can change rapidly due to resource contention, hardware failures, or network connectivity issues.

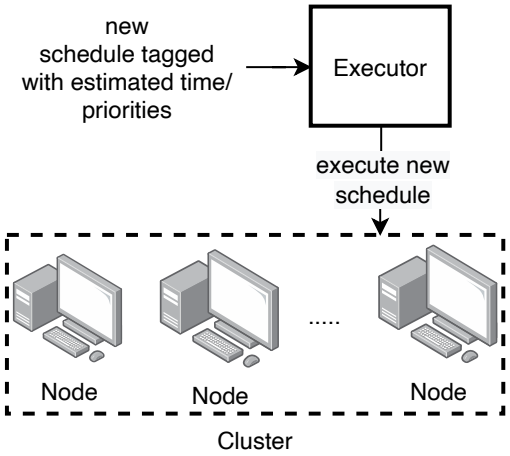


Figure 5.7: Executor Daemon Architecture

Algorithm 5 Scheduling Algorithm

```
A job is ready to be scheduled
Add the job to the queue
If a resource becomes available:
for each job in the queue do
    Identify the requirements of the job in the queue
    Identify the resource configurations available
    Match the requirements vs the available resource configurations
    if the node can execute the job depending on the requirements then
        Schedule the job on the node
        Remove job from the queue
        Mark resource as occupied
    else
        Move on to the next job in the queue
    end if
end for
if no jobs could be scheduled then
    Wait for next resource availability notification
end if
```

The architectural design illustrated in Figure 5.7 demonstrates the integration between schedule implementation, resource management, execution monitoring, and energy optimization components that comprise the Executor Daemon. The architecture emphasizes the daemon's role as the central execution coordinator that translates high-level scheduling decisions into practical

workflow execution while maintaining continuous optimization of both performance and energy efficiency.

The new schedule development process results from the Energy-Aware Priority-Based Scheduling (EAPBS) algorithm implementation that considers both computational requirements and energy efficiency objectives in scheduling decisions. The daemon executes jobs on allocated resources as calculated by the optimization algorithm and configured by the Resource Allocator Daemon, ensuring that the energy-aware scheduling framework achieves its dual objectives of computational performance and energy efficiency. The scheduling algorithm presented in Algorithm 5 provides the detailed operational framework that governs daemon execution management, incorporating energy-aware decision making at every stage of the scheduling process. The algorithm balances immediate execution opportunities with long-term energy optimization objectives, enabling sustainable high-performance workflow execution that meets both computational requirements and environmental responsibility goals.

5.3.8 Overarching EMWOS System Workflow

The scheduling system operates through a systematic workflow that coordinates the actions of all components. The following outlines the overall steps followed by the proposed system to schedule the computation and achieve the goals set out by the user:

1. Initial workflow submission and priority specification.
2. Activation of energy monitoring systems.
3. Generation of the optimal schedule.
4. Implementing Resource configuration.
5. Workflow modification for resource-specific execution.
6. Schedule execution and monitoring.

This orchestrated process ensures efficient workflow execution while maintaining energy awareness and adherence to user-specified priorities. The system has been designed to be extensible, allowing additional components and strategies to be integrated as requirements evolve.

5.4 Implementation and System Development

With the EMWOS architecture defined in the previous section, this section details the practical implementation and development methodology used to realize the system. The implementation follows a phased approach, beginning with basic multi-user support and progressively incorporating advanced optimization capabilities. This incremental development strategy allows for systematic validation of each component while maintaining system stability and functionality. The section examines the technical challenges encountered during development, the solutions implemented, and the integration strategies employed to ensure compatibility with existing workflow management systems. This implementation foundation is essential for the experimental evaluation presented in subsequent sections.

5.4.1 Phase 1: Basic Multi-User Support

This is the first part of the implementation where the Multi-User functionality is added to the scheduler system. This is built upon the theoretical approach presented in Section 5.3. The framework uses core scheduling policies outlined in Section 5.3.2 for its scheduling decisions.

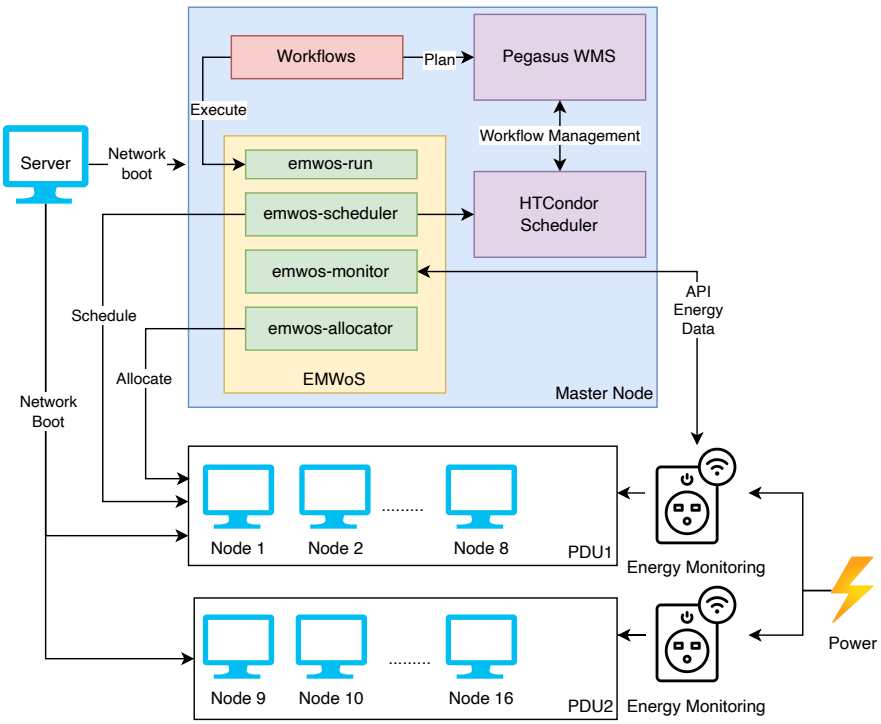


Figure 5.8: Schematic diagram of the experimental setup integrating the EMWOS System

Figure 5.8 illustrates the high-level experimental setup. The proposed system is implemented as an Energy Monitoring and Workflow Scheduler (EMWoS) system. This is the first system developed that adds the multi-user functionality. The experimental setup consists of multiple

interconnected compute nodes working together to execute the computation. The master node hosts the EMWoS system along with other essential components needed for executing scientific workflows. Each compute node has its own memory and power source. The EMWoS system is implemented as a sum of four distinct daemons that work together to achieve the system's goals as shown in Section 5.3.7. The different functionalities of the daemons are provided below:

emwos-run

The *emwos-run* Controller Daemon is the central orchestration mechanism in the EMWoS framework. It is in charge of spawning and monitoring other daemons. The workflow is first provided to the controller daemon. The dependencies and the available resources are identified, and then the daemon uses the EAPBS algorithm (refer to Algorithm 2) to find optimal resource allocation. The controller daemon is responsible for maintaining the state of execution and implementing recovery strategies when failures occur.

emwos-monitor

The *emwos-monitor* daemon provides essential monitoring functionality for the EMWoS system, continuously collecting computation execution data for processing by other daemons. Energy data is gathered through custom firmware on the monitoring infrastructure via multiple queryable endpoints that measure amperage, wattage consumption, Kilowatt-hour usage, and voltage, all returned in JSON format for parsing and storage.

Execution status is monitored through native operating system utilities and HTCondor tools. The *df* command tracks file system space, while *free -m* monitors available node memory. HTCondor-specific utilities include *condor_q* for execution statistics and *condor_status* for compute node allocation information, allowing the system to make execution adjustments based on comprehensive monitoring data.

emwos-allocator

The *emwos-allocator* daemon is responsible for allocating different resources of a compute node for particular jobs and getting the node ready for execution. A compute node consists of resources not necessarily used by a particular job. When the daemon identifies that the node has an excess of resources available for any other available job, it can isolate the resources and make them available for the job to execute. This ensures there is no wastage of compute resources and the cluster is fully utilized by providing dedicated compute resource space for each job.

```
mehul@master: condor_status -constraint Machine==\"alpha\"
Name          OpSys      Arch   State     Activity     LoadAv Mem   ActvtyTime

slot1@alpha   LINUX      X86_64 Unclaimed Benchmarking  0.000 1952  0+00:00:00
slot2@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot3@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot4@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot5@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot6@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot7@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00
slot8@alpha   LINUX      X86_64 Unclaimed Idle          0.000 1952  0+00:00:00

               Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

  X86_64/LINUX     8     0       0         8       0          0        0      0

         Total     8     0       0         8       0          0        0      0
mehul@master: ./emwos-allocator -node alpha -config "{'slots': 3}"
Node [alpha] has been configured. Restarting now.
[alpha] condor node restarted.
mehul@master: condor_status -constraint Machine==\"alpha\"
Name          OpSys      Arch   State     Activity     LoadAv Mem   ActvtyTime

slot1@alpha   LINUX      X86_64 Unclaimed Benchmarking  0.000 5207  0+00:00:00
slot2@alpha   LINUX      X86_64 Unclaimed Idle          0.000 5207  0+00:00:00
slot3@alpha   LINUX      X86_64 Unclaimed Idle          0.000 5207  0+00:00:00

               Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

  X86_64/LINUX     3     0       0         3       0          0        0      0

         Total     3     0       0         3       0          0        0      0
```

Figure 5.9: Resource Allocator in action

Figure 5.9 shows the *emwos-allocator* daemon in action. The resource *alpha* has a total of 8 cores (each of them allocated to a slot) and 16GB of memory (distributed equally among the slots) available for computation. Each of these slots is available for a job execution, but if a job requires more memory or more cores, then *emwos-allocator* can configure the nodes based on the requirement. After configuration is complete, the same node *alpha* now has 3 slots available with 5.2GB of memory available for each slot for the execution of jobs. This segregation of resources enables the computation to be completed faster as the jobs do not compete for resources. This daemon uses the Algorithm 4 to achieve its functionality.

emwos-scheduler

The *emwos-scheduler* is the final daemon that is responsible for the execution of the jobs. The schedule is generated using the EAPBS algorithm (Refer to Algorithm 2) as a part of the controller daemon. The scheduler works with the meta-scheduler HTCondor to check dependencies and execute the available jobs. This daemon uses the Algorithm 5 to achieve its functionality.

Figure 5.10 presents how the EMWoS integrates into the existing workflow execution system. This ensures that the functionality of the existing execution system is not affected. The integration enhances the current system's capabilities by providing the user with extra configuration and scheduling options.

In this section, EMWoS's architecture with its four core components for energy-aware workflow scheduling are presented. Despite its comprehensive design, EMWoS has several limitations: it requires compatible infrastructure for energy monitoring, depends on stable network connectivity,

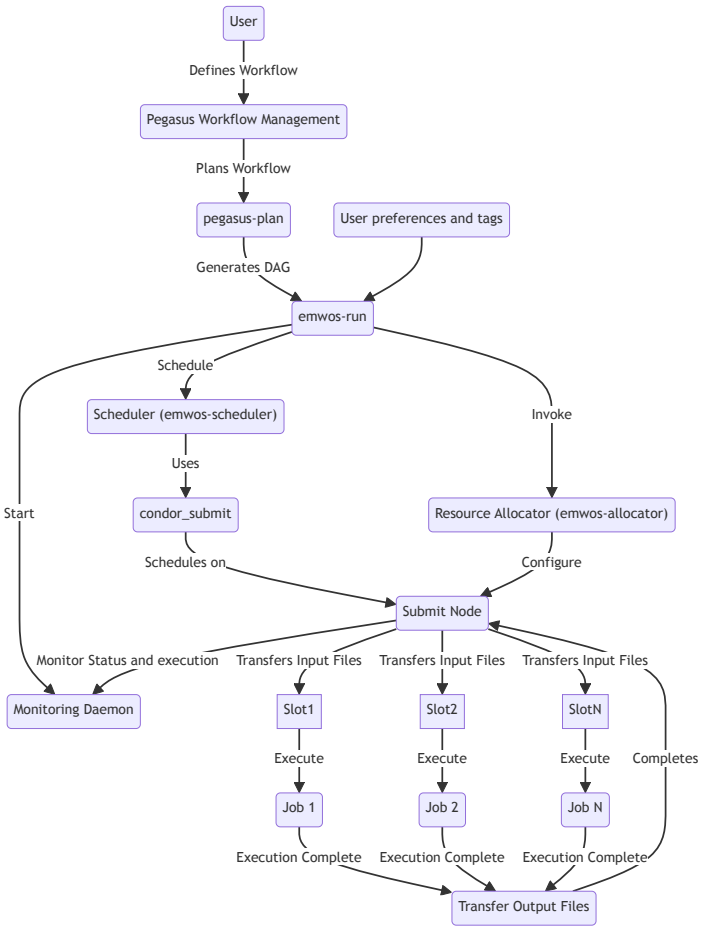


Figure 5.10: Integration of EMWoS System with Existing Workflow Execution Systems

needs access to a centralized database for energy data analysis, and is currently limited to HTCondor clusters. The system's effectiveness may also vary across fluctuating workloads and heterogeneous hardware environments. The following sections present experiments to evaluate EMWoS's effectiveness and highlight practical applications of energy-aware scheduling in real-world scientific computing environments.

5.4.2 Phase 2: Optimization Algorithm Integration

In the second part, the proposed framework is improved to provide support for the optimization of workflows. The framework in Part 1 conducted energy-aware scheduling of workflows based on predefined rules outlined in Section 5.3.2. This is not feasible for complex compute scenarios. The implementation of the optimization algorithms and support helps automate the process of scheduling decisions for the framework.

The hybrid optimization approach has been developed as part of a scheduler system to validate its effectiveness. Figure 5.11 illustrates the high-level setup of the hybrid optimization approach.

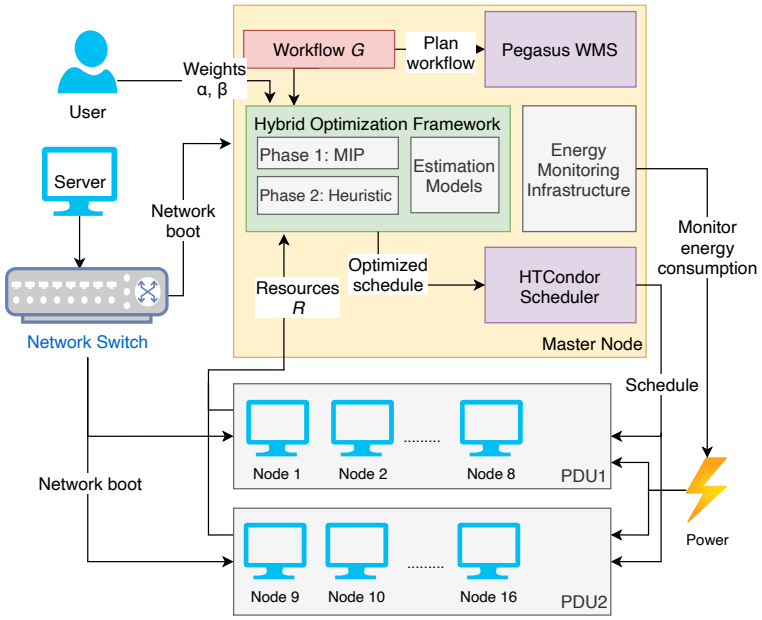


Figure 5.11: Schematic Diagram of the Hybrid Optimization Framework

The proposed framework is implemented as a part of a scheduler system that integrates seamlessly within any existing high-compute infrastructure setup. Multiple interconnected nodes work together to execute the computation. A master node, responsible for scheduling and managing the resources, hosts the scheduler system that houses the proposed optimization framework. The optimization framework is built using the Python programming language and uses multiple optimization libraries to find the optimal solution. A separate energy monitoring infrastructure tracks the live energy consumption of each resource and stores the data for further analysis.

5.5 Experimental Setup and Evaluation Environment

Following the implementation details presented in the previous section, this section establishes the comprehensive experimental infrastructure necessary for evaluating the EMWOS system. The evaluation methodology requires a carefully designed experimental environment that can accurately represent real-world multi-user computing scenarios while providing precise measurement capabilities for both energy consumption and performance metrics. This section describes the hardware infrastructure, software configuration, and benchmark workflows selected to thoroughly assess the system's capabilities. The experimental framework established here forms the foundation for the detailed evaluations conducted in the subsequent sections, ensuring that results are both rigorous and representative of practical deployment scenarios.

5.5.1 Compute Cluster Setup

The experimental setup implements a scaled-down compute cluster that effectively mirrors production environments. The experimental setup included a cluster consisting of 16 compute nodes, a dedicated master node, and a server node, each playing a distinct role in the distributed computing infrastructure (see Figure 5.11).

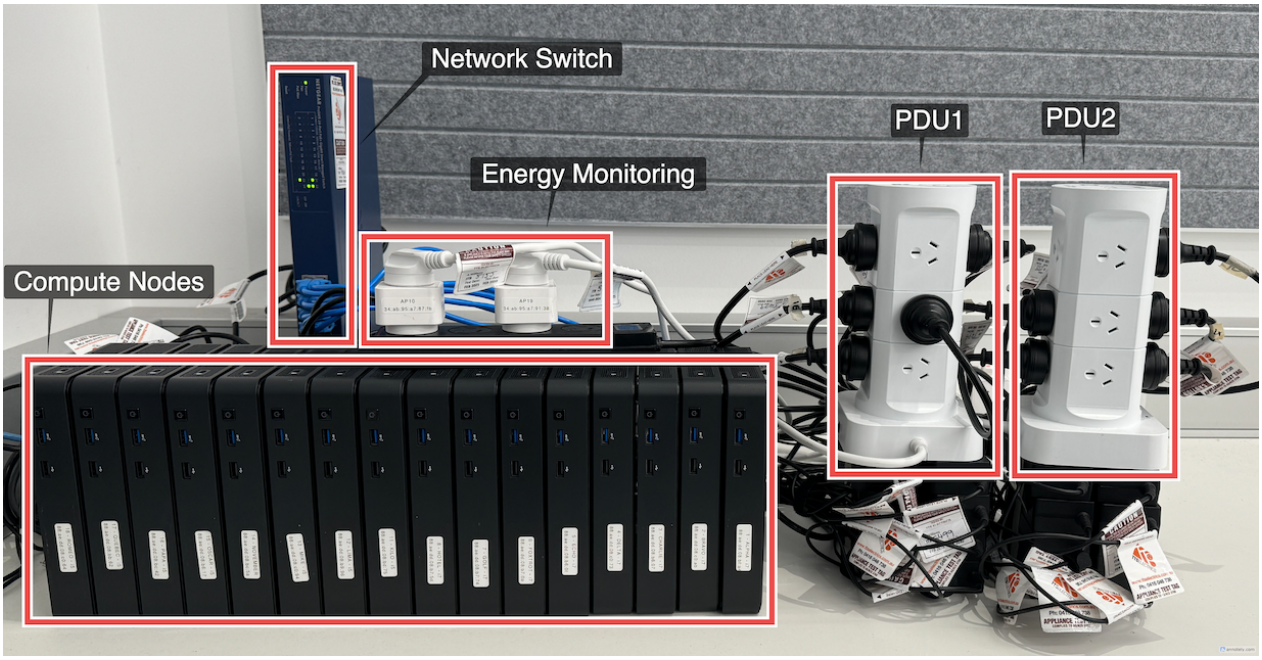


Figure 5.12: Annotated Image of the Experimental Compute Cluster Setup

Figure 5.12 showcases the cluster of 16 compute nodes assembled for this study. The server node is the central infrastructure hub, hosting the shared filesystem via the Network File System (NFS) and the base filesystem for the entire cluster. The compute nodes, comprising eight i7 compute elements (1165G7 generation processor) with 16GB RAM and eight i5 compute elements (1145G7 generation processor) with 8GB RAM, operate using a network-booted Debian-based Linux Mint 20 (`https://linuxmint.com/`, accessed on 1 March 2025) operating system.

The master node coordinates computational tasks across this network-booted infrastructure, efficiently distributing workloads while accounting for the heterogeneous nature of the compute resources. The combination of network-booted operating systems and centralized storage creates an efficient, maintainable infrastructure that closely mirrors enterprise-grade compute clusters while providing a controlled environment for research and experimentation.

5.5.2 Smart Energy Monitoring Infrastructure

Energy consumption data is collected exclusively from compute nodes performing power-intensive calculations. Two power distribution units (PDUs) supply power to groups of 8 nodes each.

Energy monitoring hardware connects to these PDUs, ensuring accurate readings despite minor fluctuations (see Figure 5.11). ESP8266 microcontrollers enable precise measurements, which are accessible through an application programming interface (API). These monitoring tools utilize custom-built ESPHome firmware that collects data via the integrated HLW8032 sensors at 500ms intervals.

5.5.3 Cluster Management System

A cluster management system is used to set up and configure a cluster easily. The cluster management system used for this study is HTCondor (Condor) [138]. Condor offers easy setup and configurable functionalities for a wide range of use cases. Condor is used to create an interconnected cluster of resources that can accept and execute different computations in parallel. Condor has an integrated Directed Acyclic Graph Manager (DAGMan) that schedules jobs using user's preferences and different techniques [138]. The DAGMan schedules jobs by matching their requirements with those of the available resources. DAGMan is responsible for managing the task dependencies and ensuring that jobs are only submitted for execution after all their dependencies are complete and a compute resource is available to execute the job.

5.5.4 Workflow Management System

Now that the cluster is set up and ready to accept and execute computations, a Workflow Management System (WMS) manages and executes the computations on the cluster. For this study, the Pegasus workflow engine is used [21]. Different types of computations can be executed on a cluster. Workflows are increasingly used to manage and perform complex computations as they can be broken down into independent tasks that can be executed individually. Pegasus can generate a complete workflow from its metadata description or an abstract XML-based *dax*. This description and *dax* usually defines the workflow's tasks and data that can be developed into a concrete workflow which can be executed on a cluster system. Pegasus comes with standard scheduling strategies but can accept a custom schedule based on different scheduling algorithms [21]. The schedule generated by the optimization framework will be provided to Pegasus for execution on the cluster.

5.5.5 Workflows used for Evaluation

Two distinct workflow types have been used in this study to assess the proposed hybrid optimization framework across diverse computational scenarios and validate its effectiveness under both controlled and realistic conditions.

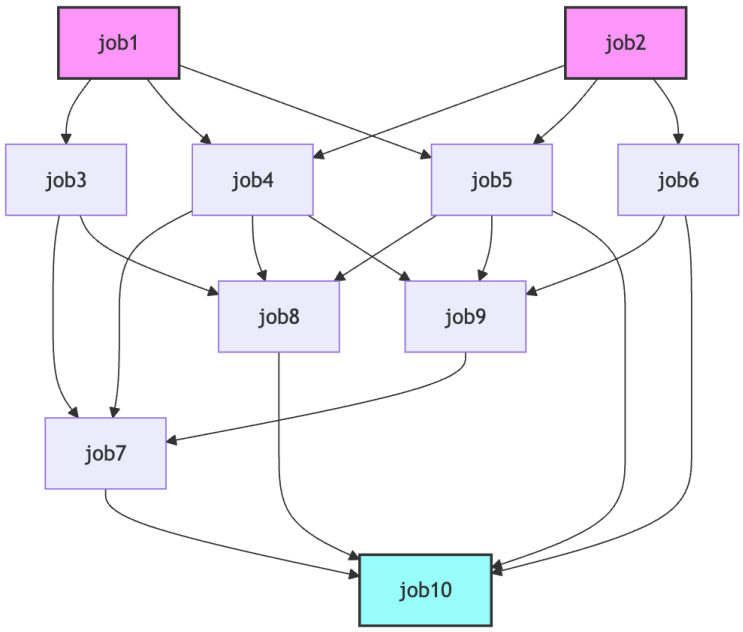


Figure 5.13: Synthetic Workflow Structure and Setup

Synthetic workflows are generated to provide controlled experimental conditions, enabling the systematic evaluation of the framework's performance characteristics. These workflows maintain a consistent structural pattern while allowing parametric variation in workflow size, enabling assessment of scalability and algorithmic behavior without introducing unintended computational complexities. The synthetic workflow generator creates directed acyclic graphs with configurable numbers of tasks while preserving dependency structures that reflect typical scientific workflow patterns [90]. Figure 5.13 illustrates a synthetic workflow created for this study. Each task includes details about its complexity and computation.

The Montage workflow is a real-world scientific application for evaluating the practical implications and performance of the proposed framework under authentic computational scenarios. Montage is an astronomical image processing toolkit developed by NASA/IPAC [1] that creates accurate astronomical image mosaics by combining multiple FITS sky images while preserving astrometric calibration and positional fidelity. Characterized as I/O-bound, Montage's computational requirements can be customized based on sky coverage degrees and color channels [1]. The workflow architecture follows a directed acyclic graph structure with nine interdependent processing stages, as illustrated in Figure 2.7.

This approach facilitates controlled experimentation across different workflow scales, ranging from small instances with tens of tasks to large-scale workflows containing hundreds of computational elements. The synthetic workflows incorporate realistic execution time estimates and communication costs derived from the job information database, ensuring that generated workloads maintain computational characteristics representative of scientific applications while providing the flexibility necessary for algorithmic evaluation.

5.6 Evaluating Support for Multi-User Environments

Having established the experimental infrastructure in the previous section, the first phase of EMWOS evaluation, focusing specifically on the system's capability to manage multi-user environments effectively, is presented here. This evaluation phase examines the fundamental scheduling capabilities of EMWOS without the advanced optimization algorithms, providing essential baseline performance metrics and validating the core multi-user functionality. The analysis encompasses fairness in resource allocation, quality of service maintenance, and system behavior under varying user loads and workflow complexities. The results from this evaluation establish the foundation for understanding how the optimization enhancements improve upon these baseline capabilities.

5.6.1 Single-User Baseline Performance

The simplest form of computation that any scientist performs using Pegasus is a Single workflow execution. The experiment includes creating a workflow using Pegasus and submitting it to the EMWoS system with different scheduling policies. The workflow is generated using default configuration options. The data from this experiment will help confirm the functioning and the effectiveness of the EMWoS system. The EMWoS system has varied control over the execution as given below:

1. Fine-grained control: The scheduler has individual job-level control within the workflow.
2. Coarse-grained control: The scheduler manages batches of 10 jobs simultaneously.

A single montage workflow with 1.5 degree complexity and tagged with specific policy and control preferences was executed. Two primary metrics were measured:

1. Execution Time: measured in seconds from workflow submission to completion.
2. Energy Consumption: measured in Watt-hours, calculated based on power draw over the execution period.

To ensure accuracy, each policy configuration was executed 10 times, and the results were averaged out for comparison between each other. Background processes were minimized to reduce interference, and controls were implemented to manage system load across runs. The experiment results are summarized in Figure 5.14, which displays both the execution time and energy consumption.

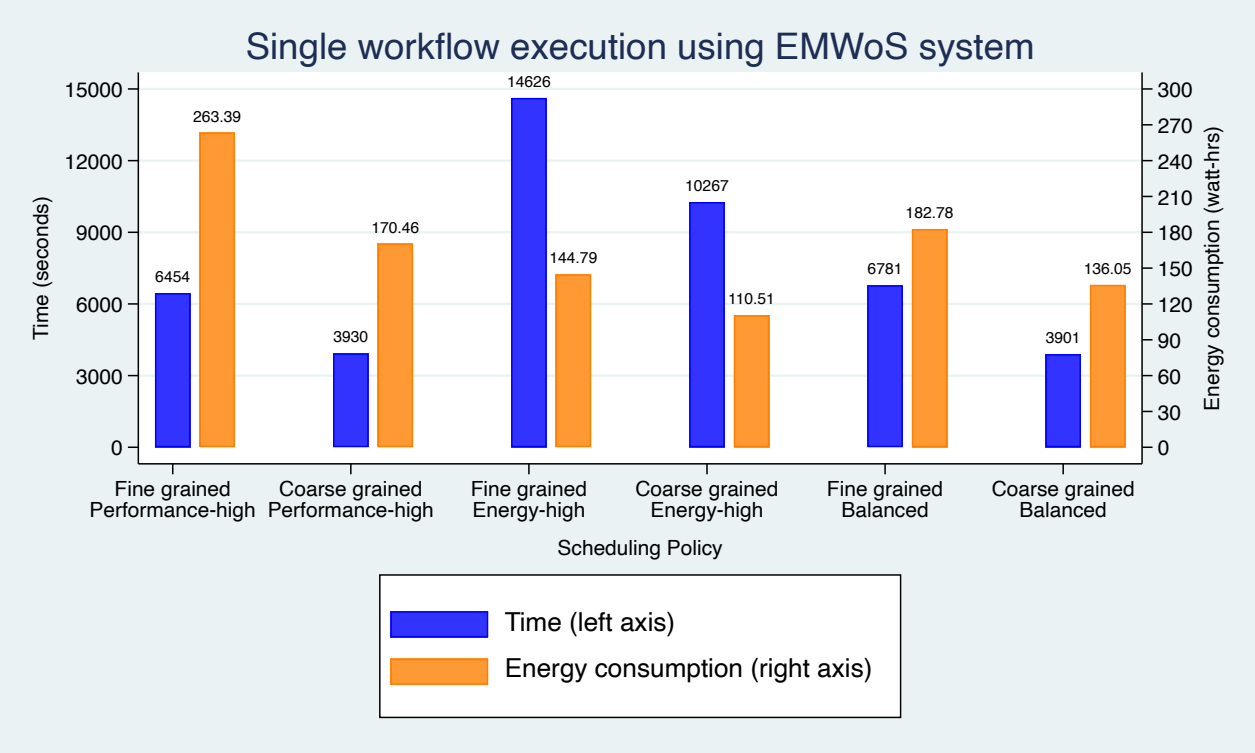


Figure 5.14: Single Workflow Execution Analysis using the EMWoS System

Performance Analysis The performance-high policy utilized the complete cluster for execution. The policy achieved an execution time of 1 h 5 min with coarse-grained control and 1 h 47 min with fine-grained control execution. The balanced policy utilized half of the cluster for execution. It achieved a moderate execution time of 1 h 53 min with fine-grained control and 1 h 5 min with coarse-grained control over execution. The energy-high policy had the worst execution as it utilized only one node. The workflow executed in 4 h 3 min with fine-grained control and 2 h 51 min with coarse-grained control.

The results indicate that having a coarse-grained control over the execution of the jobs always results in improved performance. This unexpected efficiency of coarse-grained control indicates that grouping jobs may reduce scheduling overhead and lead to more stable resource utilization. The poor performance of fine-grained control suggests that fine-grained granularity can introduce inefficiencies through frequent job migrations and scheduling decisions. These results indicate that utilizing more computational resources generally leads to faster execution times. Still, the relationship is not always linear, as a balanced policy performed better than a performance-high policy when coarse-grained control was provided during the execution.

Energy Efficiency Analysis Similar to the execution time, the energy consumption of the computation is always better when the system has coarse-grained control over the execution. The coarse-grained energy-high execution consumed the least amount of energy at 110.51 Watt-hours. This confirms the effectiveness of the system in minimizing power consumption. Interestingly, the

energy consumption between the energy-high fine-grained and balanced coarse-grained executions differs marginally. This can be mainly attributed to the execution times of these experiments. Using fewer resources results in 4 times more execution time, resulting in no improvement in the energy consumption. Similar results can be seen between balanced fine-grained execution and the performance-high coarse-grained execution.

These findings reveal that energy consumption depends on the computation's performance and resource allocation. Energy consumption can be high even if a limited number of resources are being used due to compensation for lower performance. The fine-grained policy's unexpectedly high energy consumption suggests that frequent job migrations or scheduling decisions may increase power usage. The coarse-grained policy's efficiency indicates that stable resource allocation and reduced scheduling overhead can significantly improve energy efficiency.

System Effectiveness The results demonstrate that the scheduler effectively implemented each policy. The performance-high policy effectively improved the execution time, while the energy-high policy minimized power consumption. The balanced policy successfully achieved a compromise between the two extremes.

The fine-grained and coarse-grained policies showed interesting dynamics. While fine-grained control was expected to provide better performance, the results suggest that the overhead of managing individual jobs may outweigh the benefits in some scenarios. Coarse-grained control was comparably more efficient in all scenarios, possibly due to reduced scheduling overhead.

The results validate the scheduler's ability to implement diverse policies according to workflow preferences. These findings highlight an important insight: the optimal level of control granularity must balance scheduling flexibility with operational overhead. The experiment demonstrates clear trade-offs between execution time and energy consumption, confirming the importance of energy-aware scheduling in modern computing environments.

5.6.2 Multi-User Scheduling Analysis for Standard Execution

This section evaluates the performance of the EMWoS system in multi-user environments where multiple workflows compete for shared cluster resources. This scenario closely resembles operational conditions in modern data centers, where multiple users simultaneously submit computational tasks.

The evaluation uses five montage workflows with 1.5-degree complexity (Workflows 1 - 5) submitted to the compute cluster simultaneously. The EMWoS system schedules and executes the workflows. The primary metrics measured were the workflow performance (tracked through system logs) and energy consumption (calculated through the compute load and total energy

consumption). In a multi-user environment, a compute node may simultaneously execute jobs from multiple workflows

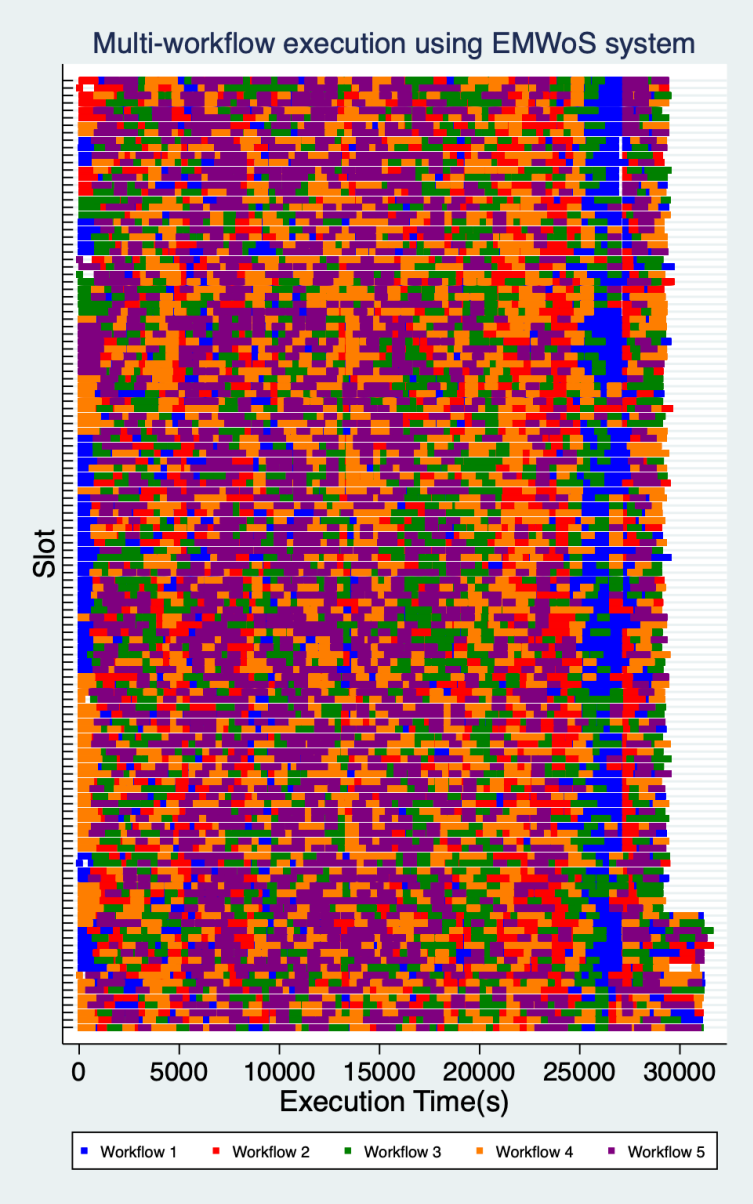


Figure 5.15: Multi-Workflow Execution Analysis using the EMWoS System

Figure 5.15 presents the job scheduling sequence of a standard execution of multiple workflows on the cluster. The X-axis presents the execution time in seconds. The Y-axis presents the 128 slots (16 nodes with 8 slots each) that are used to execute the workflows. Jobs from different workflows are color-coded (blue, red, green, orange, and purple), and the graph denotes when the job started and ended executing on the particular slot.

The execution of multiple workflows on the cluster shows no specific execution pattern as presented in Figure 5.15. All five workflows execute jobs concurrently with no distinct boundaries or pattern. All the workflows compete for resources and progress together, irrespective of their submission time. All workflows completed execution around the same time - 31,515 seconds (8 h

45 min). This is expected as the default scheduler does not prioritize any workflow over another and aims to provide equal computing power for all the computations. The energy consumed by each workflow is derived from the execution time and the allocated resources. Each workflow consumed around 260 Watt-hours of energy.

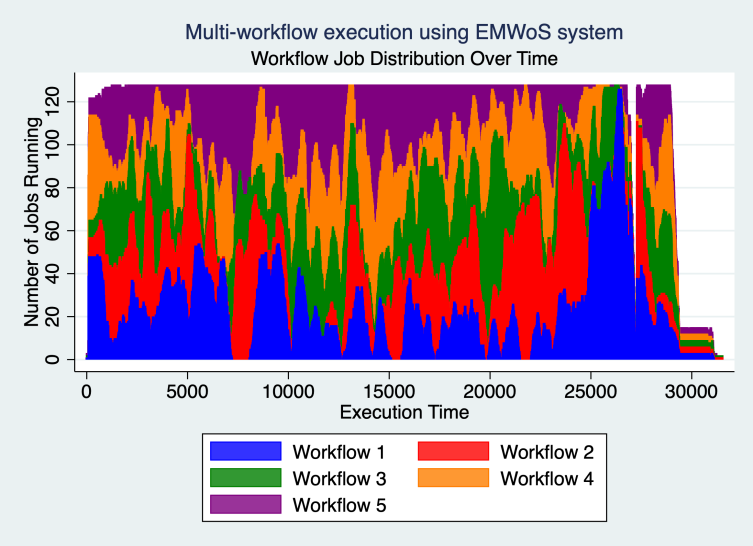


Figure 5.16: Job Distribution of Multi-Workflow Execution using EMWoS

Figure 5.16 presents the workflow job distribution during the multi-workflow execution. The X-axis denotes the execution time in seconds, and the Y-axis the number of jobs executing at any time. This figure confirms that all the jobs from all five workflows were being executed simultaneously. During the majority of the execution, the maximum number of jobs being executed is 128, which suggests that the compute cluster is being utilized to its maximum capacity.

The results validate the scheduler's ability to execute multiple workflows simultaneously. The standard execution, without any scheduling policy, means that scheduling is based on job arrival time or resource availability. These findings highlight an important insight: standard execution is agnostic to user preferences, workflow priorities, or system-level objectives such as energy efficiency. Significant improvements can be achieved when the scheduler system is aware of the user's or system's goals and employs scheduling policies to achieve those goals.

5.6.3 Multi-User Scheduling Analysis for Performance-Aware Execution

The EMWoS system is analyzed in more complex scenarios, such as managing multiple concurrent workflows with distinct individual preference specifications. The system is aware of the execution's performance and uses scheduling policies to maximize it. Workflow-level preferences allow users to specify their desired execution characteristics for their submitted workflows independently.

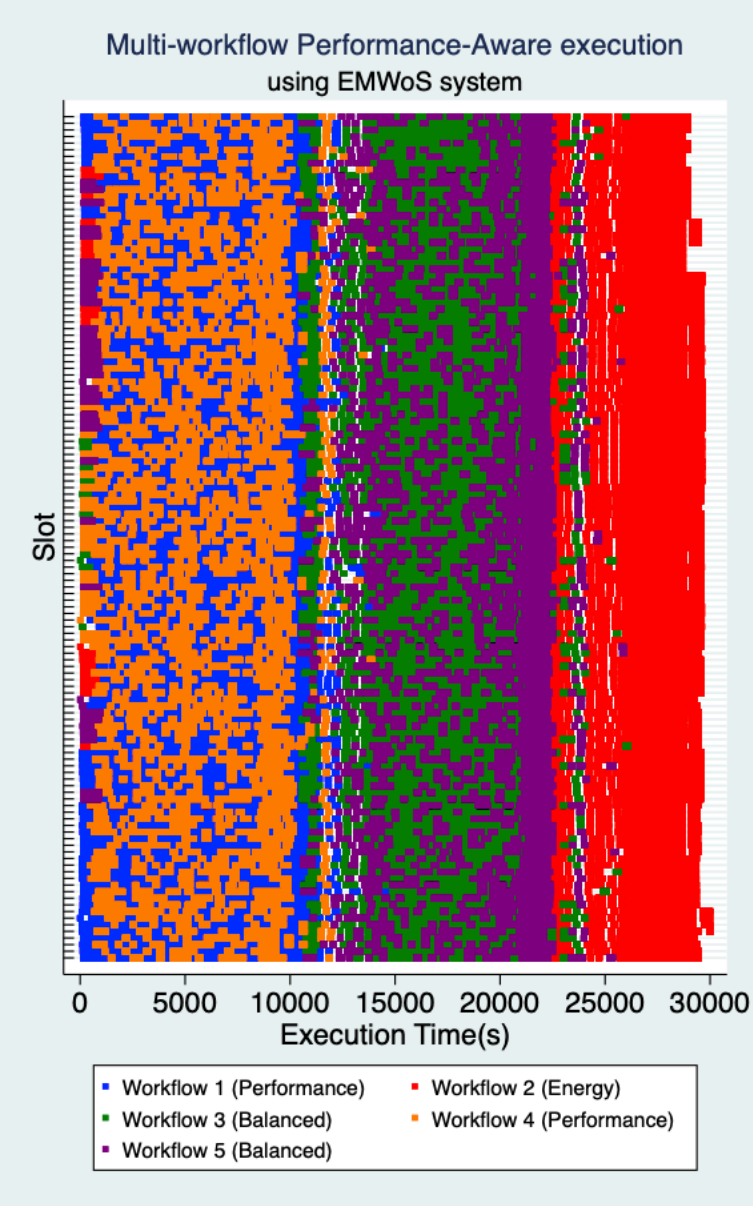


Figure 5.17: Multi-workflow Performance-Aware Execution using EMWoS

This experiment significantly increases scheduling complexity, as the system must now manage potential conflicting user preferences in a shared computing environment. Similar to the previous section, five workflows were submitted simultaneously to the cluster to be executed, but different preference specifications were assigned to each workflow. The preference options include performance, balanced, and energy. Performance preference indicates that the user prefers their workflow to be prioritized and executed in a minimal amount of time. Balanced preference tries to balance between energy consumption and performance. Energy preference indicates that performance is not the main goal of the user. The scheduler system aims to respect the preferences set by the users.

The evaluation metrics included the energy consumption and individual workflow execution times. The key assessment criteria for this experiment focused on the system's ability to respect

individual workflow preferences while maintaining reasonable overall system performance and energy efficiency. This experiment provides critical insights into the proposed system's flexibility and effectiveness.

Figure 5.17 shows the allocation of five workflows across a 128-slot computational cluster over time. Workflows 1 and 4 are performance-tagged and represented by blue and orange colors, respectively. Workflows 3 and 5 are balanced-tagged and represented by green and purple colors, respectively. Workflow 2, represented by red, is tagged with energy preference. The x-axis represents execution time in seconds, while the y-axis represents the computational slots. Figure 5.17 reveals a highly structured execution pattern, different from the randomized allocation seen in the standard execution approach (Figure 5.15). Performance-prioritized workflows were allocated resources first and executed to near completion. The balanced-policy workflows followed in the execution sequence, receiving substantial resource allocation only after the performance-critical workflows were allocated entirely. Finally, the energy-prioritized workflow was scheduled last, receiving most of its resource allocation only after the higher-priority workflows were processed.

The performance-aware approach resulted in significant improvement of the performance for the high-priority workflows. Workflows 1 and 4 (tagged for performance) completed execution faster (4 h 4 min) than they did in the standard execution scenario (8 h 45 min), with completion times reduced by approximately 52% for both workflows. This performance improvement is a direct result of scheduling these workflows first and utilizing the maximum computing power to complete these workflows, as opposed to the resource contention that occurred in the standard execution. Workflows with balanced priorities (3 and 5) experienced moderate performance improvement, resulting in 16.6% speedup (approximately 7 h 12 min for execution of the workflows). Workflow tagged with energy-preference (Workflow 2) was considered the lowest priority for performance and was scheduled last. This resulted in the execution time for the workflow to be 30,333 seconds (approximately 8 h 25 min), resulting in a speedup of 2.52%.

Notable improvements in energy efficiency were also achieved. Performance-prioritized workflows (1 and 4) maintained relatively efficient energy consumption at 255.27 Watt-hours and 248.90 Watt-hours, representing reductions of 1.8% and 4.3%, respectively. Workflows 3 and 5 (tagged as balanced) demonstrated the most significant energy savings, consuming approximately 220.35 Watt-hours and 214.08 Watt-hours, respectively, compared to the standard execution's 260 Watt-hours, representing reductions of 15.2% and 17.7%, respectively. The energy-preferred Workflow 2 achieved a 4.2% reduction in energy consumption (248.99 Watt-hours) as compared to standard execution. The results demonstrate that the system could adhere to the user preferences while simultaneously reducing energy consumption across all workflow categories.

Figure 5.18 presents the workflow job distribution during the multi-workflow execution. The X-axis denotes the execution time in seconds, and the Y-axis the number of jobs executing at

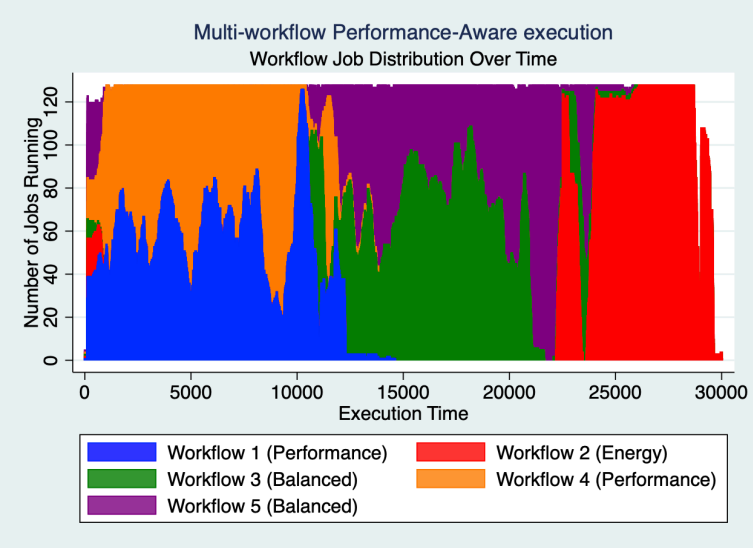


Figure 5.18: Job Distribution of Multi-Workflow Performance-Aware Execution using EMWoS

any time. The figures confirm the findings that the performance-aware execution demonstrates a sequential progression aligned with the specified priorities. The EMWoS system also maintained high resource utilization, with the compute cluster operating at or near its maximum capacity.

The results validate that performance-aware scheduling effectively addresses one of the major limitations of standard execution by incorporating user preferences into the resource allocation decision-making process. This approach enables the system to better align computational resources with workflow priorities, resulting in improved user satisfaction and more efficient resource utilization.

5.6.4 Multi-User Scheduling Analysis for Energy-Aware Execution

The aim of EMWoS is to intelligently schedule workflows to minimize energy consumption while respecting workflow preferences and maintaining acceptable performance. This section evaluates the system's energy-aware capabilities in high-performance computing environments. Using five concurrent workflows with different preference specifications (performance, balanced, and energy), the experiment measures execution time and energy consumption to assess how effectively EMWoS reduces energy use while respecting performance requirements.

Figure 5.19 illustrates the allocation of five workflows across the computational cluster over time in the energy-aware execution scenario. A significantly different execution pattern can be seen from the standard and performance-aware approaches. All the system's scheduling decisions were made based on previous experiments (see Figure 5.14) and predefined rules (see Section 5.3.3) based on existing research. Performance-prioritized workflows were allocated all the available resources for execution, similar to performance-aware execution. This resulted in similar performance for those workflows, around 13,700 sec (3 h 48 min). This is done so that

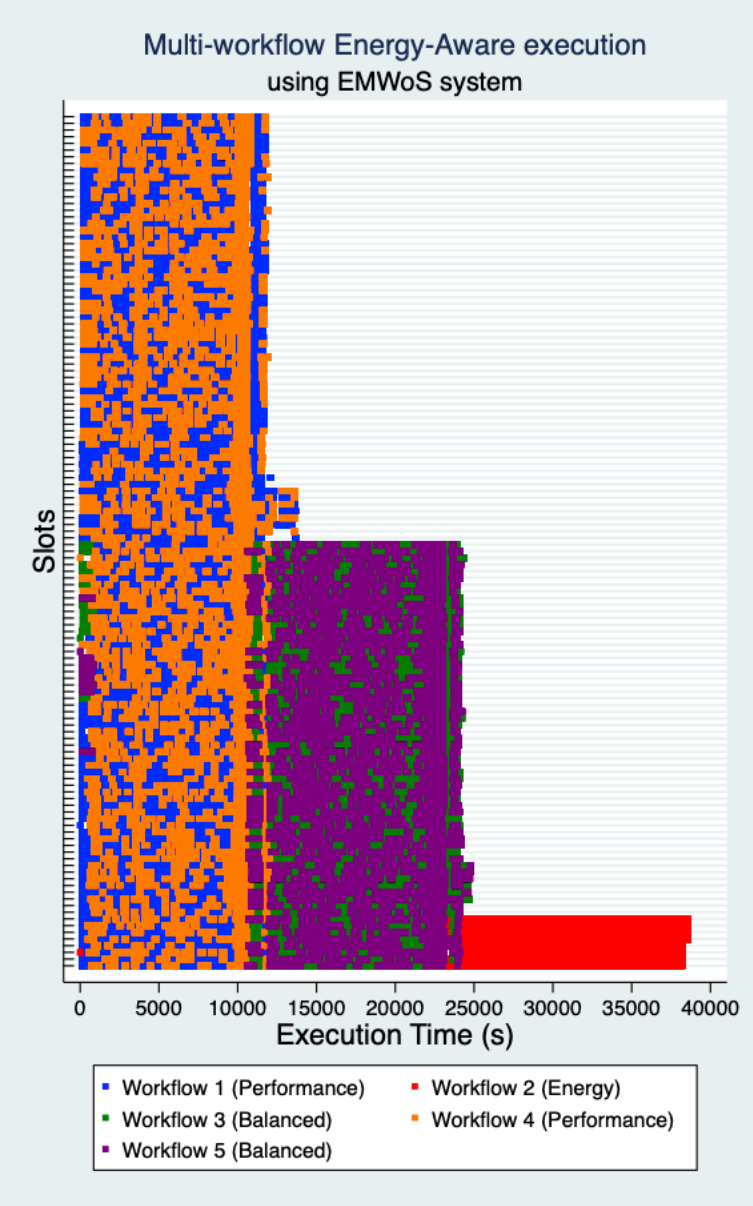


Figure 5.19: Multi-Workflow Energy-Aware Execution using EMWoS

the performance goals for those workflows are not missed. The balanced-tagged workflows were scheduled to receive a balanced allocation of half the resources, which resulted in an execution time of around 24,800 sec (6 h 53 min). Finally, the energy-efficient workflow was scheduled last, receiving a minimal allocation of just one node, which resulted in a high execution time of 38,600 sec (10 h 43 min). This is a substantial 27.6% increase in execution time as compared to the performance-aware execution. This is done to reduce the workflow's energy footprint and achieve the lowest energy consumption.

The energy-aware approach demonstrated significant energy efficiency gains, particularly for workflows with energy and balanced priorities. The energy-preferred Workflow 2 achieved the most improvement, consuming only 154.18 Watt-hours compared to the standard execution's 260 Watt-hours, a substantial 40.7% reduction and far exceeding the 4.2% reduction seen in

the performance-aware approach. Even though the workflow took longer to execute, the energy consumption of a single node compensates for the increase in execution time. The overheads of scheduling on multiple resources far exceed the execution time improvement that it provides. Balanced workflows (3 and 5) also showed notable improvements, consuming approximately 192.25 Watt-hours and 193.00 Watt-hours. This is a reduction of 26.1% and 25.8% as compared to standard execution. As expected, performance-tagged workflows (1 and 4) maintained similar energy consumption patterns across both approaches, with values of around 250.18 Watt-hours, similar to the 255.27 Watt-hours observed in the performance-aware scenario. This consistency reflects their similar execution patterns regardless of the scheduling approach used. It highlights how targeted energy-aware scheduling can dramatically reduce workflow consumption where energy efficiency is prioritized.

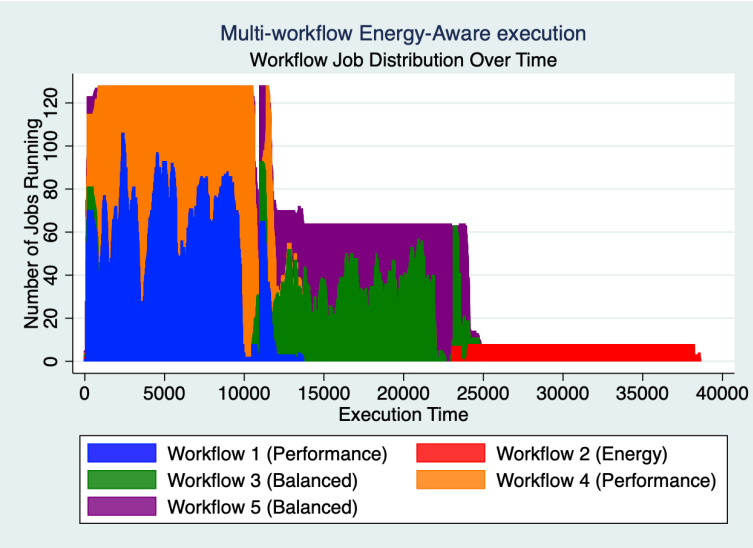


Figure 5.20: Job Distribution of Multi-Workflow Energy-Aware Execution using EMWoS

Figure 5.20 presents the workflow job distribution during the energy-aware execution. The X-axis denotes the execution time in seconds, and the Y-axis represents the number of jobs executing at any given time. Unlike the performance-aware execution, the energy-aware approach shows a more controlled resource allocation pattern.

This tiered resource allocation pattern (all resources for performance, half for balanced, and minimal for energy preferences) allows the EMWoS system to maintain lower average power consumption while still respecting user-specified priorities. By smartly limiting the maximum number of resources available to each workflow category and distributing computational load over time, the system avoids sustained periods of maximum power draw across all available resources, achieving energy savings compared to both standard and performance-aware execution approaches.

The results of this experiment demonstrate that the EMWoS system successfully balances the competing objectives of performance and energy efficiency. The execution time of a few workflows is longer due to reduced resource allocation, but the system compensates for this by

achieving significant energy savings through strategic resource allocation and workload distribution over time. The next section analyzes the findings from all experiments to provide a comprehensive assessment of the EMWoS system's capabilities and contributions to the field of scientific workflow management.

5.6.5 Discussion

Figure 5.21 illustrates how EMWoS effectively tailors workflow execution based on user preferences across different scheduling policies. In the standard execution (green bars), all workflows experience similar execution times around 31,000 seconds, regardless of their priority, indicating resource contention and a lack of preference-based scheduling.

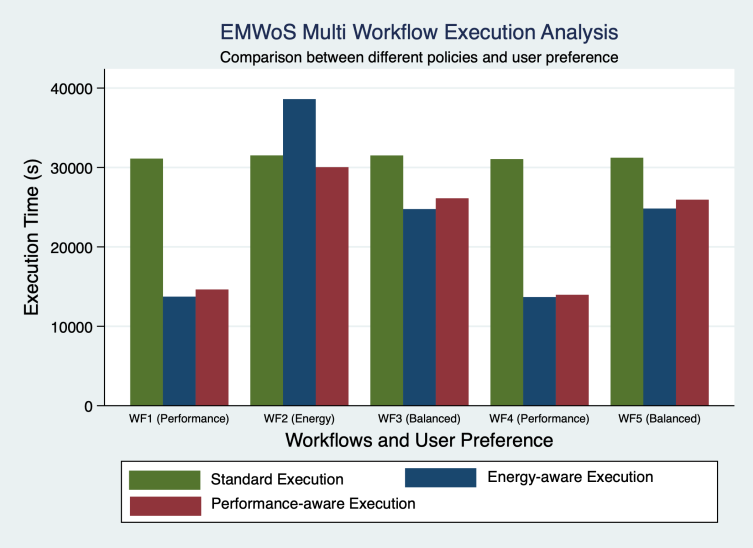


Figure 5.21: Performance Analysis of Multi-Workflow Execution using Different Policies

The performance-aware execution (maroon bars) demonstrates significant improvements for performance-tagged workflows (WF1 and WF4), reducing their execution times by approximately 52% compared to standard execution. This is achieved by prioritizing these workflows and allocating maximum available resources to them. Balanced workflows (WF3 and WF5) show moderate improvements with a 16.6% speedup, while the energy-preferred workflow (WF2) receives the lowest priority and shows minimal performance gains of around 3.9%.

The energy-aware policy (blue bars) maintains a similar execution time for performance-tagged workflows (WF1 and WF4) and balanced-preference workflows (WF3 and WF5) compared to the performance-aware execution. However, due to minimal resource allocation, the execution time for the energy-tagged workflow (WF2) increased significantly. There was an increase of 27.6% in the execution time for the WF2, but substantial energy savings of 40.7%. This shows that energy consumption can be improved even though the execution time for a computation increases.

This comparison clearly demonstrates how EMWoS successfully implements differentiated scheduling policies that honor user preferences. It prioritizes performance while making tradeoffs to achieve energy efficiency goals.

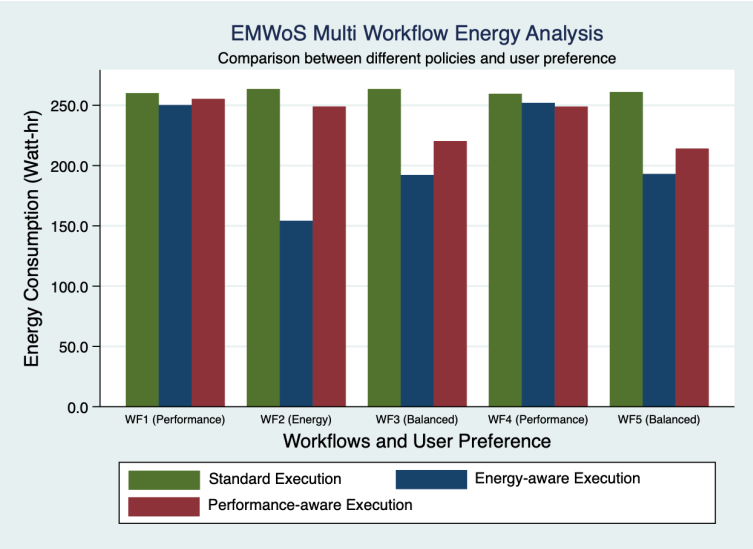


Figure 5.22: Energy Consumption Analysis of Multi-Workflow Execution using Different Policies

Figure 5.22 presents a clear visualization of energy consumption patterns across different scheduling policies in EMWoS. In the standard execution (green bars), all workflows consume approximately 260 Watt-hours regardless of their priority The performance-aware policy (maroon bars) achieves modest energy savings across all workflows while prioritizing execution speed for performance-tagged workflows. Most notably, balanced workflows (WF3 and WF5) show a 15-17% reduction in energy consumption, while performance-tagged workflows (WF1 and WF4) maintain near-standard energy consumption, reflecting their prioritization of speed over efficiency.

The energy-aware policy (blue bars) demonstrates the system's most significant energy improvements, particularly for workflows tagged with energy preference. The energy-tagged workflow (WF2) shows a remarkable 40.7% reduction in energy consumption compared to standard execution. Similarly, balanced workflows (WF3 and WF5) achieved substantial energy savings of approximately 26%, while performance-tagged workflows maintain energy consumption patterns similar to the performance-aware approach, ensuring performance goals are not compromised.

5.7 Evaluating the Optimization Algorithm Performance

Building upon the multi-user evaluation results from the previous section, a comprehensive assessment of EMWOS with fully integrated optimization algorithms is presented here. This evaluation represents the development of the system, demonstrating how the optimization engine enhances the basic multi-user capabilities to achieve superior energy efficiency while

maintaining performance and fairness objectives. The analysis examines the effectiveness of various optimization strategies, quantifies the energy-performance trade-offs, and assesses system scalability under realistic computational loads.

5.7.1 Synthetic Workflow Execution Analysis

The experimental evaluation focuses on synthetic workflow execution scenarios with 300 jobs and 128 resources to analyze the performance characteristics of various optimization approaches. The proposed Hybrid Optimization Algorithm, shown in Algorithm 3, implements a time budget allocation strategy where 70% of the available computation time is allocated to the Mixed Integer Programming (MIP) optimization phase, while the remaining 30% is reserved for heuristic enhancement. The goal of this experiment is to justify the need for a two-phase optimization strategy and how the proposed approach can achieve a near-optimal solution in a short time. The algorithm implementation, detailed in Algorithm 3, demonstrates the two-phase approach where Phase 1 utilizes MIP optimization with termination constraints to obtain an initial solution, followed by Phase 2, which applies heuristic algorithms to enhance the solution quality within the remaining time budget.

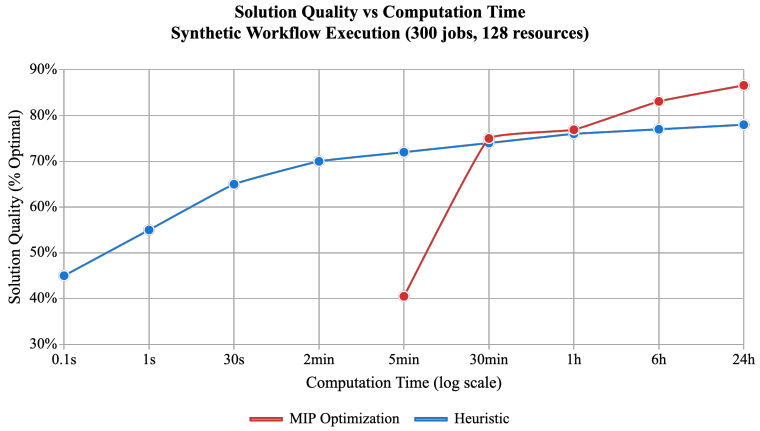


Figure 5.23: Solution Quality vs Computation Time for Different Optimization Approaches

Comparison between MIP optimization and heuristic approaches in terms of solution quality achieved over varying computation times is presented in Figure 5.23. The experimental results demonstrate that heuristic algorithms provide rapid convergence, achieving approximately 55% solution quality within 1 second and reaching 78% quality within 24 hours. In contrast, MIP optimization exhibits slower initial progress, achieving only 40% solution quality at 5 minutes but demonstrating superior long-term performance, reaching 86% solution quality at 24 hours. Notably, MIP optimization shows a sharp improvement between 5 minutes and 30 minutes, indicating the effectiveness of allowing sufficient time for mathematical programming techniques to explore the solution space.

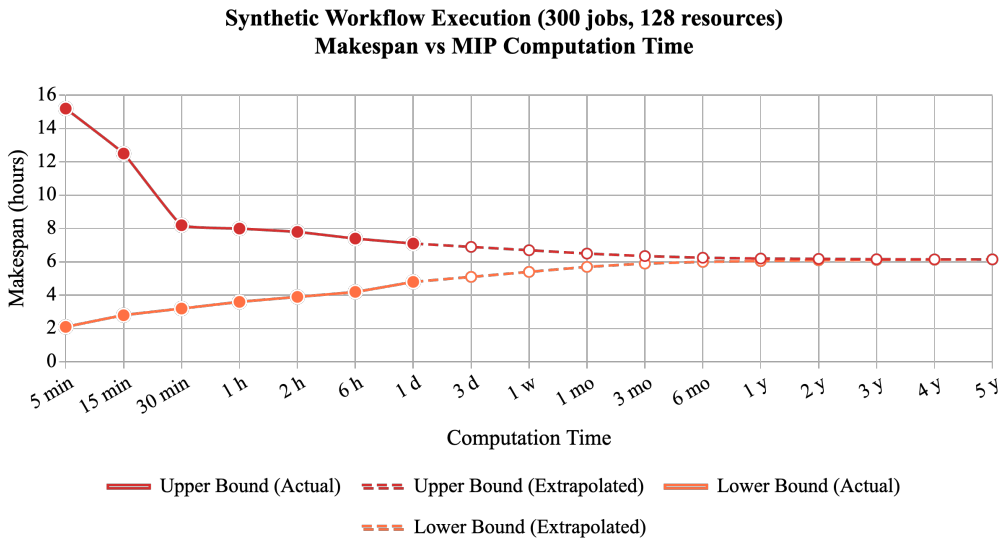


Figure 5.24: Makespan vs MIP Computation Time Analysis

The makespan analysis, presented in Figure 5.24, provides deeper insights into the long-term behavior of MIP optimization and establishes both upper and lower bounds for solution quality expectations. The results show that the upper bound (actual performance) starts at 15.2 hours for a 5-minute computation time and rapidly decreases to approximately 8 hours within the first hour of computation. Beyond this point, improvements become marginal, with the makespan theoretically stabilizing around 6.2 hours after extended computation periods. The lower bound analysis indicates the theoretical best achievable performance, starting at 2.1 hours and gradually increasing to approximately 6 hours, suggesting diminishing returns for extended MIP computation.

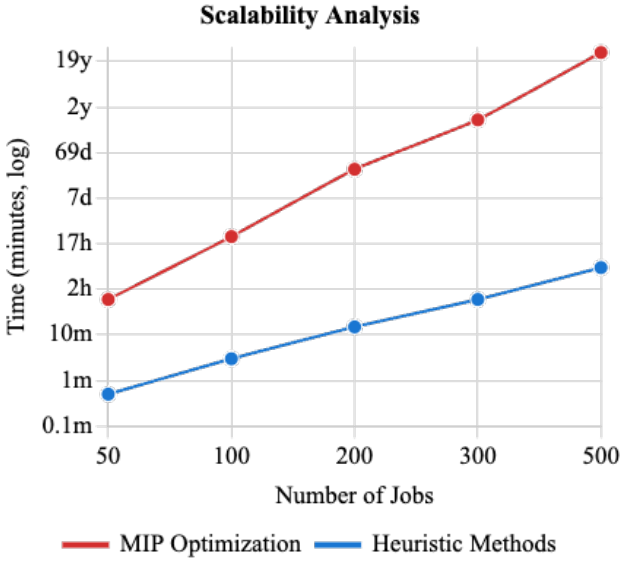


Figure 5.25: Theoretical Total Computation Time required vs Number of Jobs

The scalability characteristics of different optimization approaches are examined in Figure 5.25, which presents computation time requirements as a function of problem size (number of jobs)

for both MIP optimization and heuristic methods. The results reveal different scalability behaviors between the two approaches. Heuristic methods demonstrate excellent scalability, with computation time growing from 0.6 minutes for 50 jobs to around 3 minutes for 500 jobs, representing a relatively linear growth pattern. This behavior makes heuristics particularly suitable for large-scale problems where rapid solutions are required.

In contrast, MIP optimization exhibits exponential growth in computation time as problem size increases. The computation time escalates from 1 hour for 50 jobs to over 19 years for 500 jobs, clearly demonstrating the NP-hard nature of the optimization problem. This exponential growth pattern highlights the practical limitations of pure MIP approaches for large-scale scheduling problems. The scalability analysis reveals a critical insight: while MIP optimization provides superior solution quality for smaller problem instances, its computational requirements become an overhead as problem size increases. At 200 jobs, MIP requires approximately 30 days of computation time, whereas heuristic methods complete the same problem in under 20 minutes. This substantial difference underscores the importance of hybrid approaches that can leverage MIP's optimization capabilities for smaller subproblems while relying on heuristics for overall scalability.

5.7.2 Montage Workflow Analysis for different scheduling techniques

To validate the effectiveness of the proposed hybrid optimization framework in realistic scenarios, experiments were conducted using the Montage workflow, a well-established astronomy application from the Pegasus workflow gallery (see Figure 2.7). The experimental setup employed a 1.5-degree Montage workflow, executed on 128 heterogeneous resources (See Figure 5.12), providing a testbed for evaluating scheduling performance through real-world execution rather than simulation.

The experimental comparison comprises five distinct scheduling approaches: the default scheduling technique for htcondor, the Heterogeneous Earliest Finish Time (HEFT) algorithm, and three configurations of the proposed hybrid optimization method with varying objective function weights. The hybrid approach configurations explore different trade-offs between makespan minimization and energy consumption reduction through variations in the $alpha$ and $beta$ parameters, where $alpha$ is the weight for minimizing makespan and $beta$ is the weight for reducing energy consumption.

A comparison of makespan and energy consumption performance across all evaluated scheduling techniques is presented in Figure 5.26. The results demonstrate the multi-objective optimization capabilities of the proposed hybrid framework and its ability to achieve different trade-offs based on parameter configuration.

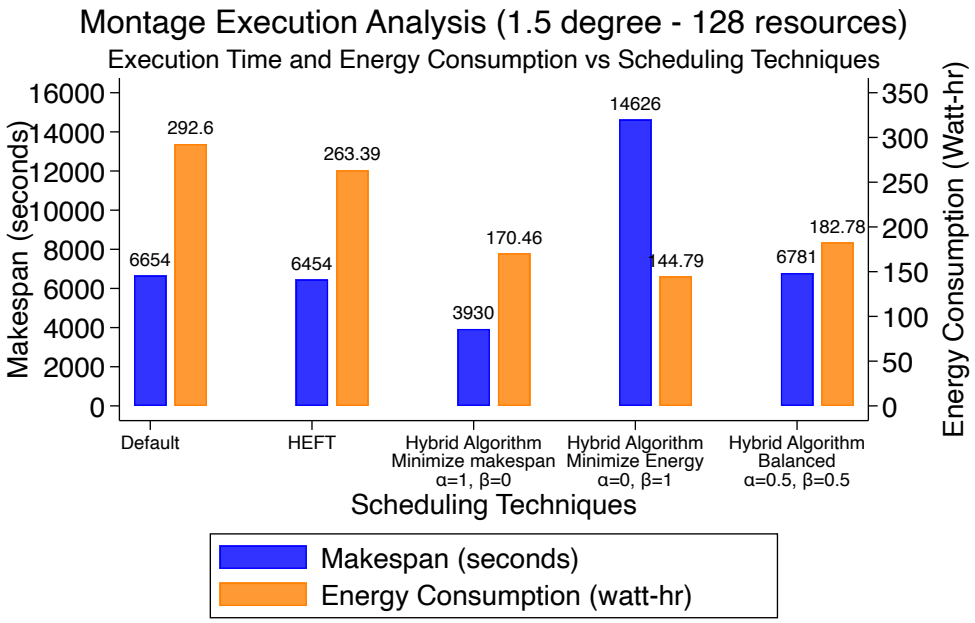


Figure 5.26: Scheduling Technique Analysis for Montage Workflow Execution

Default Scheduling Technique

This experiment utilized the default First-Come, First-Served (FCFS) scheduling technique of HTCondor. This establishes the baseline for subsequent experiments and provides a reference point for comparing results from various scheduling approaches. The default scheduling technique achieved a makespan of 6,654 seconds (approximately 1 h 51 min) with an energy consumption of 292.6 watt-hours. It should be noted that the energy consumption measurements are based on real energy data collected from the energy monitoring infrastructure implemented in the system (see Figure 5.11). There are no notable scheduling patterns observed in this experiment, as the jobs were scheduled once their dependencies were complete and resources were available for execution.

Heterogeneous Earliest Finish Time (HEFT)

The HEFT algorithm is a well-established scheduling heuristic that schedules a set of dependent tasks onto a network of heterogeneous workers while considering communication overhead. The HEFT algorithm demonstrated marginal improvements compared to the default scheduling approach, achieving a makespan of 6,454 seconds (approximately 1 h 47 min) and energy consumption of 263.39 watt-hours. This represents a 3% reduction in execution time and a 10% improvement in energy efficiency compared to the default approach.

Makespan Minimization (α=1, β=0)

The proposed hybrid optimization framework configurations demonstrate the effectiveness of the hybrid optimization approach. The makespan-focused configuration (α=1, β=0) achieved optimal execution time performance with a makespan of 3,930 seconds (approximately 1 h 5 min), representing a 41% improvement over the default scheduling and a 39% improvement over HEFT. Notably, this configuration also achieved significant energy savings of 170.46 watt-hours, demonstrating that optimizing for makespan does not necessarily compromise energy efficiency.

Further analysis revealed that the hybrid optimization algorithm successfully identified an optimal strategy for co-scheduling compute-intensive and non-intensive tasks on the same resources, resulting in enhanced performance without resource overutilization. Additionally, tasks on the critical path and closely related tasks are strategically scheduled together on the same resource, resulting in substantial savings in overhead costs, such as data transfers.

Energy Consumption Minimization (α=0, β=1)

The energy-focused configuration (α=0, β=1) prioritized energy consumption minimization, achieving the lowest energy consumption of 144.79 watt-hours, representing a 51% reduction compared to the default scheduling and a 45% reduction compared to HEFT. This energy optimization came at the expense of increased execution time, with a makespan of 14,626 seconds (approximately 4 h 4 min), highlighting the fundamental trade-off between performance and energy efficiency. This approx. 120% increase in makespan resulted in a corresponding 51% reduction in energy consumption.

Analysis of the scheduling patterns revealed that the hybrid optimization algorithm exclusively scheduled all jobs on a single resource, resulting in an extended makespan. However, this was offset by significant savings in overhead costs that would have been incurred by distributing jobs across multiple resources. The algorithm effectively determined that scheduling across multiple resources would yield lower improvements in energy consumption relative to the corresponding reduction in makespan. This behavior demonstrates that the algorithm genuinely prioritizes energy consumption minimization without considering makespan constraints for this configuration of weightages.

Balanced Scheduling Approach (α=0.5, β=0.5)

The balanced configuration (α=0.5, β=0.5) provides a practical compromise between the two optimization extremes, achieving a makespan of 6,781 seconds (approximately 1 h 53 min) and energy consumption of 182.78 watt-hours. This configuration demonstrates the framework's

capability to achieve reasonable performance across both objectives simultaneously, maintaining a makespan comparable to the default approach while achieving a 38% reduction in energy consumption. This result demonstrates that intelligent scheduling can achieve substantial energy cost savings without compromising computational makespan.

The comparison with established algorithms such as HEFT demonstrates the superior performance of the hybrid optimization approach across all evaluated metrics. Even the balanced configuration outperforms HEFT in both makespan and energy consumption, validating the effectiveness of the proposed framework for multi-objective workflow scheduling.

5.8 Summary

A proposal for an Energy Monitoring and Workflow Optimization Scheduler System (EMWOS) that represents a significant advancement in sustainable high-performance computing has been presented in this chapter. The systematic development of EMWOS demonstrates how rigorous theoretical foundations in mathematical modeling, multi-objective optimization, and energy-aware computing can be effectively translated into practical solutions that address real-world challenges in modern scientific computing environments. The experimental evaluations have conclusively demonstrated that EMWOS successfully balances energy efficiency with performance requirements while respecting diverse user preferences and operational constraints. The dual-phase approach of the hybrid optimization framework enables the system to leverage the mathematical rigor of Mixed Integer Programming while incorporating the flexibility and adaptability of heuristic methods, resulting in scheduling decisions that are both theoretically sound and practically implementable.

The EMWOS architecture demonstrates innovation in its integration of energy monitoring, dynamic scheduling, and adaptive resource management within a unified framework that addresses the complete workflow execution lifecycle. The system's three-tier daemon architecture—comprising the Monitor Daemon, Resource Allocator Daemon, and Executor Daemon—provides comprehensive workflow life-cycle management that spans from initial energy profiling through final execution optimization. The Monitor Daemon provides real-time energy consumption and system state assessment. This enables continuous optimization based on actual operational conditions rather than theoretical estimates. The Resource Allocator Daemon introduces dynamic resource configuration capabilities that can optimize node-level settings. This includes CPU frequency scaling, memory allocation patterns, and power management configurations based on workflow characteristics and energy efficiency objectives. The Executor Daemon provides workflow execution management that implements energy-aware scheduling decisions at no cost to performance.

The evaluations presented in this chapter have demonstrated the effectiveness and practical

viability of the EMWOS system across multiple dimensions of assessment. The research outcomes extend beyond immediate technical achievements. The EMWOS system offers significant potential for deployment in data centers and high-performance computing environments where energy consumption has become a critical operational concern. The demonstrated capability to manage resource contention while maintaining fairness and quality of service guarantees positions EMWOS as a viable solution for modern computing environments seeking to balance computational performance with environmental responsibility.

Chapter 6

Conclusion and Future Work

6.1 Overview

This chapter presents the conclusions of the thesis and outlines the potential future direction of this research. Section 6.2 presents the overview of the thesis and summarizes the outcomes of each chapter of the thesis. Section 6.3 presents the contributions arising from the work presented in this thesis to the domain of energy-aware scientific workflow computation. Section 6.4 discusses potential future work which could be undertaken in this area and how the work can be applied to other domains of computing. Finally, Section 6.5 presents the concluding remarks on the work presented in this thesis.

6.2 Overview of Thesis

This thesis has addressed the critical challenge of energy-aware scheduling for scientific workflows in high-performance computing environments. As computational demands continue to grow exponentially, the environmental impact and operational costs associated with energy consumption have become a serious concern for research institutions worldwide. The research presented in this thesis provides a comprehensive framework for understanding, monitoring, and optimizing energy consumption in scientific workflow execution while maintaining the performance requirements essential for cutting-edge research.

The thesis is structured as a systematic and iterative approach toward developing efficient methods for reducing energy consumption in scientific computation while maintaining computational performance and reliability. This research addressed the critical gap between growing computational demands and environmental sustainability concerns in high-performance computing [37, 62]. This section outlines and summarizes the significant outcomes of each chapter in this thesis.

Chapter 1 established the motivation and context for energy-aware scientific workflow scheduling research, grounding the work within the broader movement toward sustainable computing practices [258, 259]. The chapter addressed the growing challenges of energy consumption in high-performance computing environments, outlined the research objectives and scope, and provided an overview of the methodological approach employed throughout the thesis. The introduction positioned the research within the broader context of sustainable computing practices in the domain of scientific research.

Chapter 2 provided an in-depth analysis of the existing research landscape in energy-aware computing, workflow scheduling, and optimization techniques. The chapter examined the foundational work in scientific workflow management and energy-efficient computing. The chapter also examined current approaches to energy monitoring in computational environments, reviewed existing scheduling strategies for scientific workflows [22, 45], and identified critical gaps in the literature that motivated the research contributions presented in subsequent chapters. The background analysis established the theoretical foundations and contextualized the novel contributions of this thesis within the broader academic domain.

Chapter 3 presented the foundational understanding necessary for energy-aware computing by developing a reliable and accurate energy monitoring framework, addressing fundamental challenges in energy measurement that had limited previous research efforts [64, 65]. This chapter addressed the fundamental challenge of accurately measuring and analyzing energy consumption patterns in scientific workflows. The research identified critical factors affecting both performance and energy consumption, including the relationship between resource allocation and energy efficiency across different workflow types, the trade-offs between execution time and energy consumption for various computational tasks, and the opportunities and challenges presented by heterogeneous computing environments [68]. The development of reliable energy monitoring capabilities provided the essential measurement infrastructure upon which all subsequent optimization efforts were built.

Chapter 4 translated the insights gained from energy monitoring into actionable scheduling strategies. This chapter developed both static and adaptive scheduling frameworks that could intelligently allocate resources to minimize energy consumption while maintaining performance objectives. The research applied the Monitor-Analyze-Plan-Execute (MAPE) model for adaptive scheduling [158], enabling dynamic decision making based on real-time execution data. The experimental validation demonstrated substantial energy consumption savings as compared to standard scheduling approaches, with several policies achieving simultaneous improvements in both energy efficiency and execution performance. The integration of autonomic computing principles with scientific workflow scheduling represented a significant methodological advance that enabled self-optimizing systems responsive to changing execution conditions.

Chapter 5 presented the combination of the research in the form of EMWOS, a comprehensive system that addressed the complex challenges of multi-user computing environments. This chapter advanced beyond the single-user scenarios addressed in previous work to tackle the realistic complexity of production computing environments where multiple users with diverse requirements competed for shared resources [47, 49]. The system incorporated mathematical modeling frameworks for multi-objective optimization [50], real-time energy monitoring capabilities, intelligent multi-user management, and hybrid optimization algorithms that combined the benefits of exact and heuristic approaches [159]. The experimental evaluations demonstrated the system's capability to balance energy efficiency with performance requirements while respecting diverse user preferences and maintaining fairness in resource allocation.

Throughout this progression, the research maintained a consistent focus on practical applicability, ensuring that theoretical advances aligned with real-world deployment requirements in modern computing environments. The work ensured that theoretical advances could be translated into deployable solutions for real-world computing environments. The systematic validation across multiple scientific workflows and computing configurations provided confidence in the generalizability and robustness of the proposed approaches.

6.3 Thesis Contributions

The research presented in this thesis makes several significant contributions to the field of energy-aware scientific computing, advancing both theoretical understanding and practical capabilities for sustainable high-performance computing.

Comprehensive Energy Modeling Framework: The development of mathematical models that capture the complex relationships between workflow characteristics, resource capabilities, and energy consumption patterns provides a rigorous foundation for optimization. The formal representation of workflows as directed acyclic graphs with energy-aware task and resource modeling enables systematic optimization of multi-objective problems comprising of energy efficiency and performance.

Adaptive Scheduling Methodology: The integration of autonomic computing principles through the MAPE model with scientific workflow scheduling represents a significant area of interest. This approach enables dynamic adaptive decisions based on real-time execution data, resulting in superior performance compared to static scheduling approaches. The methodology provides a systematic framework for developing self-adapting systems that can respond to changing computational conditions while maintaining energy efficiency objectives.

Multi-User Optimization Framework: The extension of energy-aware scheduling to multi-user environments addresses a critical gap in existing research that primarily focuses on single-user

scenarios. The mathematical formulation with defined constraints and user preference modeling provides theoretical foundations for managing complex multi-user computing environments while maintaining energy efficiency objectives.

Hybrid Optimization Strategy: The development of optimization approaches that strategically combine Mixed Integer Linear Programming (MILP) with heuristic algorithms addresses fundamental trade-offs between solution quality and computational complexity. This hybrid approach harnesses the benefits of exact optimization methods while maintaining practical scalability for large-scale problems.

Reliable and Accurate Energy Monitoring System: The development of a comprehensive framework for a reliable and accurate energy monitoring and analysis of energy consumption patterns across diverse computational scenarios, supporting both research investigations and operational optimization efforts.

Energy-Aware Scheduling Implementations: The progression from static to adaptive scheduling implementations demonstrates the practical feasibility of energy optimization in scientific computing environments. The seamless integration approach into existing computation management systems ensures compatibility with existing workflow management systems, facilitating adoption in production environments without requiring major infrastructure modifications.

EMWOS System Architecture: The comprehensive Energy Monitoring and Workflow Optimization Scheduler System provides a complete solution for energy-aware computing in multi-user environments. The EMWOS system provides a modular and easy way for users to achieve their energy or performance goals without necessarily needing to understand the underlying complexities needed to achieve them.

6.4 Future Work

The research presented in this thesis establishes a solid foundation for energy-aware scientific workflow scheduling, yet several promising directions remain for future investigation and development. These opportunities span across theoretical advances, practical enhancements, and broader applications that could further enhance the impact and applicability of energy-aware computing.

Machine Learning Integration: The incorporation of machine learning approaches for predictive modeling of energy consumption and performance characteristics represents a significant avenue that can be explored to enhance the effectiveness of the optimization [133, 132]. Advanced techniques such as deep learning, reinforcement learning, and ensemble methods could improve

the accuracy of energy and performance predictions, enabling more sophisticated optimization decisions [147, 53]. The development of adaptive learning mechanisms that can characterize new workflows based on execution patterns and historical data would reduce the dependency on manual configuration and improve system autonomy [52, 202].

Specialized Hardware Integration: Integration of specialized computing hardware such as Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs), and emerging accelerator technologies can be explored as a expansion to the current state of energy-aware scheduling domain. They present significant opportunities for optimization [260, 131]. The incorporation of Dynamic Voltage and Frequency Scaling (DVFS) techniques and hardware-specific power management capabilities could provide additional dimensions for energy optimization [38].

Large-Scale Deployment Studies: The validation of energy-aware scheduling approaches on larger, more diverse computing clusters would strengthen the evidence for practical deployment in production environments [93, 234]. Studies involving hundreds or thousands of nodes, diverse scientific communities, and extended operational periods would provide insights into long-term stability, scalability limitations, and operational considerations [90, 91].

Extended Workflow Coverage: The evaluation of energy-aware scheduling across broader ranges of scientific workflows from diverse research domains would confirm the generalized nature of the system and identify domain-specific optimization opportunities [8, 14]. The investigation of emerging computational paradigms, including machine learning workloads, data analytics pipelines, and real-time processing applications, would expand the relevance of the research [56, 61].

6.5 Concluding Remarks

This thesis has demonstrated that energy-aware scheduling of scientific workflows represents both a critical necessity and an achievable objective for sustainable high-performance computing. The research progression from fundamental energy monitoring through adaptive scheduling to comprehensive optimization systems illustrates the systematic approach required to address the complex challenges of energy efficiency in computational environments.

The substantial reductions in energy consumption achieved across diverse experimental scenarios provide compelling evidence that significant environmental and economic benefits can be realized without compromising scientific objectives. The successful demonstration of these capabilities across single-user and multi-user environments, different workflow types, and various computational scales establishes confidence in the practical viability of energy-aware scheduling for real-world deployment.

Perhaps most significantly, the research challenges the traditional assumption that energy optimization necessarily requires performance sacrifices. The demonstration of scheduling policies that achieve simultaneous improvements in both energy efficiency and execution performance indicates that energy awareness can enhance rather than constrain computational capabilities. This finding has profound implications for the future of scientific computing, suggesting that sustainability and performance can be complementary rather than competing objectives.

The comprehensive frameworks, methodologies, and systems developed in this research provide essential tools for the scientific computing community to address growing environmental responsibilities while maintaining the computational capabilities required for cutting-edge research. As computational demands continue to expand and environmental concerns become increasingly urgent, the approaches presented in this thesis offer practical pathways toward sustainable scientific computing practices. By establishing energy awareness as a central consideration in workflow scheduling, this work helps position the scientific computing community to meet both current sustainability challenges and future environmental responsibilities.

The journey toward truly sustainable scientific computing requires continued innovation, broader adoption of energy-aware practices, and ongoing commitment to balancing computational capabilities with environmental stewardship. This thesis provides a foundation for that journey, demonstrating that the path toward sustainable high-performance computing is not only necessary but achievable through intelligent system design, careful optimization, and dedicated research effort. The future of scientific computing lies in systems that seamlessly integrate energy awareness with computational excellence, enabling researchers to pursue ambitious scientific goals while fulfilling their responsibilities towards the environment.

Bibliography

[1] J. C. Jacob, D. S. Katz, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince *et al.*, "Montage: An Astronomical Image Mosaicking Toolkit," *Astrophysics Source Code Library*, pp. ascl–1010, 2010.

[2] 1000 Genomes Project Consortium, "A Global Reference for Human Genetic Variation," *Nature*, vol. 526, no. 7571, pp. 68–74, 2015, DOI: 10.1038/nature15393.

[3] M. R. Zargham, *Computer Architecture: Single and Parallel Systems*. Prentice-Hall, Inc., 1996, ISBN: 978-0134830285.

[4] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields *et al.*, *Workflows for e-Science: Scientific Workflows for Grids*, 1st ed. London: Springer, December 2007, vol. 1, ISBN: 978-1846285196.

[5] B. Barney *et al.*, "Introduction to Parallel Computing," *Lawrence Livermore National Laboratory*, vol. 6, pp. 1–42, 2010.

[6] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*. Cambridge University Press, 1989, ISBN: 978-0521388474.

[7] J. Yu and R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 44–49, 2005, ISSN: 0163-5808, DOI: 10.1145/1084805.1084814.

[8] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and Profiling Scientific Workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013, DOI: 10.1016/j.future.2012.08.015.

[9] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the Use of Cloud Computing for Scientific Workflows," in *2008 IEEE Fourth International Conference on eScience*. IEEE, 2008, pp. 640–645, DOI: 10.1109/eScience.2008.167.

[10] K. A. Berman and J. Paul, *Fundamentals of Sequential and Parallel Algorithms*. PWS Publishing Co., 1996, ISBN: 978-0534943363.

[11] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005, ISSN: 1058-9244.

[12] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: A Tool for Building and Running Workflows of Services," *Nucleic Acids Research*, vol. 34, no. suppl_2, pp. W729–W732, 2006, DOI: 10.1093/nar/gkl320.

[13] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006, DOI: 10.1002/cpe.994.

[14] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of Scientific Workflows," in *2008 Third Workshop on Workflows in Support of Large-Scale Science*. IEEE, 2008, pp. 1–10, DOI: 10.1109/WORKS.2008.4723958.

[15] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, and F. Sommers, "Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers," in *Handbook of Nature-Inspired and Innovative Computing*. Springer, 2006, pp. 521–551, DOI: 10.1007/0-387-27705-6_16.

[16] T. Kaur and I. Chana, "Energy Efficiency Techniques in Cloud Computing: A Survey and Taxonomy," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–46, 2015, DOI: 10.1145/2808797.

[17] K. Maheshwari, E.-S. Jung, J. Meng, V. Morozov, V. Vishwanath, and R. Kettimuthu, "Workflow Performance Improvement Using Model-Based Scheduling Over Multiple Clusters and Clouds," *Future Generation Computer Systems*, vol. 54, pp. 206–218, 2016, DOI: 10.1016/j.future.2015.03.017.

[18] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Scientific Workflow Applications on Amazon EC2," in *2009 5th IEEE International Conference on e-Science Workshops*. IEEE, 2009, pp. 59–66, DOI: 10.1109/ESCIW.2009.5408002.

[19] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60, DOI: 10.1007/10968987_3.

[20] M. J. Flynn, "Very High-Speed Computing Systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966, DOI: 10.1109/PROC.1966.5273.

[21] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny *et al.*, "Pegasus, a Workflow Management System for Science Automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015, DOI: 10.1016/j.future.2014.10.008.

[22] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002, DOI: 10.1109/71.993206.

[23] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. Fernandes, and G. Mehta, "Adaptive Workflow Processing and Execution in Pegasus," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 16, pp. 1965–1981, 2009, DOI: 10.1002/cpe.1434.

[24] M. Geniviva, "Comparing BWA and Bowtie 2 Aligners in RNA-Seq Differential Expression Analyses," Master's thesis, University of Pittsburgh, 2022.

[25] G. de Sena Brandine and A. D. Smith, "Falco: High-Speed FastQC Emulation for Quality Control of Sequencing Data," *F1000Research*, vol. 8, p. 1874, 2021, DOI: 10.12688/f1000research.21142.2.

[26] S. Tuteja, "An Evaluation of Variant Annotation Tools–Alamut Batch, ENSEMBL Variant Effect Predictor (VEP), and ANNOVAR-for Clinical Next Generation Sequencing (NGS) Based Genetic Testing," *Annual Biomedical Research Conference for Minority Students (ABRCMS)*, pp. 1–12, 2022.

[27] Y. Tong, "The Comparison of limma and DESeq2 in Gene Analysis," in *E3S Web of Conferences*, vol. 271. EDP Sciences, 2021, p. 03058, DOI: 10.1051/e3sconf/202127103058.

[28] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow Management in Condor," in *Workflows for e-Science*. Springer, 2007, pp. 357–375, DOI: 10.1007/978-1-84628-757-2_22.

[29] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: A Grid-Enabled Engine for Delivering Custom Science-Grade Mosaics on Demand," in *Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493. International Society for Optics and Photonics, 2004, pp. 221–232, DOI: 10.1117/12.550551.

[30] X. Liu, D. Yuan, G. Zhang, W. Li, D. Cao, Q. He, J. Chen, and Y. Yang, *The Design of Cloud Workflow Systems*, 1st ed., ser. SpringerBriefs in Computer Science. New York: Springer Science & Business Media, January 2012, DOI: 10.1007/978-1-4614-1933-4, ISBN: 978-1-4614-1932-7.

[31] H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014, DOI: 10.1109/TPDS.2013.57.

[32] Q. Wu and V. V. Datla, "On Performance Modeling and Prediction in Support of Scientific Workflow Optimization," in *2011 IEEE World Congress on Services*. IEEE, 2011, pp. 161–168, DOI: 10.1109/SERVICES.2011.28.

[33] K. Lee, N. W. Paton, R. Sakellariou, and A. A. A. Fernandes, "Utility Functions for Adaptively Executing Concurrent Workflows," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 6, pp. 646–666, 2011, DOI: 10.1002/cpe.1673.

[34] S. K. Garg, R. Buyya, and H. J. Siegel, "Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-Off Management," in *Proceedings of the Thirty-Second Australasian Conference on Computer Science*, ser. ACSC '09, vol. 91. Darlinghurst, Australia: Australian Computer Society, Inc, 2009, pp. 151–160, ISBN: 978-1-920682-72-9.

[35] B. Kocot, P. Czarnul, and J. Proficz, "Energy-Aware Scheduling for High-Performance Computing Systems: A Survey," *Energies*, vol. 16, no. 2, p. 890, 2023, DOI: 10.3390/en16020890.

[36] M. Warade, J.-G. Schneider, and K. Lee, "Towards Energy-Aware Scheduling of Scientific Workflows," in *2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*. United States of America: IEEE, 2022, pp. 93–98, DOI: 10.1109/GECOST55694.2022.10010634.

[37] M. Warade, K. Lee, C. Ranaweera, and J.-G. Schneider, "Energy and Scientific Workflows: Smart Scheduling and Execution," *Journal of Information Science and Engineering*, vol. 40, pp. 957–977, 2024, DOI: 10.6688/JISE.202409_40(5).0012.

[38] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An Energy-Efficient Task Scheduling Algorithm in DVFS-Enabled Cloud Environment," *Journal of Grid Computing*, vol. 14, pp. 55–74, 2016, DOI: 10.1007/s10723-015-9334-y.

[39] S. Wang, Z. Qian, J. Yuan, and I. You, "A DVFS Based Energy-Efficient Tasks Scheduling in a Data Center," *IEEE Access*, vol. 5, pp. 13 090–13 102, 2017, DOI: 10.1109/ACCESS.2017.2723285.

[40] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy Aware Consolidation for Cloud Computing," in *USENIX HotPower'08: Workshop on Power Aware Computing and Systems at OSDI*, 2008, pp. 1–5.

[41] S. Singh and R. Kumar, "Energy Efficient Optimization with Threshold Based Workflow Scheduling and Virtual Machine Consolidation in Cloud Environment," *Wireless Personal Communications*, vol. 128, no. 4, pp. 2419–2440, 2023, DOI: 10.1007/s11277-022-10043-4.

[42] X. Xu, W. Dou, X. Zhang, and J. Chen, "EnReal: An Energy-Aware Resource Allocation Method for Scientific Workflow Executions in Cloud Environment," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 166–179, 2015, DOI: 10.1109/TCC.2015.2453969.

[43] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, and B. Luo, "Cost and Energy Aware Scheduling Algorithm for Scientific Workflows with Deadline Constraint in Clouds," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 713–726, 2015, DOI: 10.1109/TSC.2015.2453933.

[44] I. Pietri and R. Sakellariou, "Energy-Aware Workflow Scheduling Using Frequency Scaling," in *Proceedings of 43rd International Conference on Parallel Processing Workshops*. IEEE, 2014, pp. 104–113, DOI: 10.1109/ICPPW.2014.20.

[45] J. J. Durillo, V. Nae, and R. Prodan, "Multi-Objective Workflow Scheduling: An Analysis of the Energy Efficiency and Makespan Tradeoff," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 203–210, DOI: 10.1109/CCGrid.2013.47.

[46] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, "Multi-Objective Approach for Energy-Aware Workflow Scheduling in Cloud Computing Environments," *Scientific World Journal*, vol. 2013, no. 1, p. 350934, 2013, DOI: 10.1155/2013/350934.

[47] M. Reyes-Sierra, C. C. Coello *et al.*, "Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 287–308, 2006.

[48] K. Cao, M. Batty, B. Huang, Y. Liu, L. Yu, and J. Chen, "Spatial Multi-Objective Land Use Optimization: Extensions to the Non-Dominated Sorting Genetic Algorithm-II," *International Journal of Geographical Information Science*, vol. 25, no. 12, pp. 1949–1969, 2011, DOI: 10.1080/13658816.2011.570269.

[49] J. Yu, M. Kirley, and R. Buyya, "Multi-Objective Planning for Workflow Execution on Grids," in *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, ser. GRID '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10–17, DOI: 10.1109/GRID.2007.4354111.

[50] H. Rommelfanger, R. Hanuscheck, and J. Wolf, "Linear Programming with Fuzzy Objectives," *Fuzzy Sets and Systems*, vol. 29, no. 1, pp. 31–48, 1989, DOI: 10.1016/0165-0114(89)90133-8.

[51] A. K. Maurya and A. K. Tripathi, "Deadline-Constrained Algorithms for Scheduling of Bag-of-Tasks and Workflows in Cloud Computing Environments," in *Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications*, 2018, pp. 6–10, DOI: 10.1145/3195612.3195614.

[52] M. Sardaraz and M. Tahir, "A Hybrid Algorithm for Scheduling Scientific Workflows in Cloud Computing," *IEEE Access*, vol. 7, pp. 186 137–186 146, 2019, DOI: 10.1109/ACCESS.2019.2961106.

[53] J. Zhang, L. Cheng, C. Liu, Z. Zhao, and Y. Mao, "Cost-Aware Scheduling Systems for Real-Time Workflows in Cloud: An Approach Based on Genetic Algorithm and Deep Reinforcement Learning," *Expert Systems with Applications*, vol. 234, p. 120972, 2023, DOI: 10.1016/j.eswa.2023.120972.

[54] H. Y. Shishido, J. C. Estrella, and C. F. M. Toledo, "Multi-Objective Optimization for Workflow Scheduling Under Task Selection Policies in Clouds," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8, DOI: 10.1109/CEC.2018.8477903.

[55] W. Chen and E. Deelman, "Workflow Overhead Analysis and Optimizations," in *Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science*, ser. WORKS '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 11–20, DOI: 10.1145/2110497.2110500.

[56] S. Giampà, L. Belcastro, F. Marozzo, D. Talia, and P. Trunfio, "A Data-Aware Scheduling Strategy for Executing Large-Scale Distributed Workflows," *IEEE Access*, vol. 9, pp. 47 354–47 364, 2021, DOI: 10.1109/ACCESS.2021.3068226.

[57] T. Rauber and G. Rünger, *Parallel Programming*. Springer, 2013, ISBN: 978-3642378003.

[58] L. Zhang, K. Li, W. Zheng, and K. Li, "Contention-Aware Reliability Efficient Scheduling on Heterogeneous Computing Systems," *IEEE Transactions on Sustainable Computing*, vol. 3, no. 3, pp. 182–194, 2017, DOI: 10.1109/TSUSC.2017.2682102.

[59] S. Abbasi, A. M. Rahmani, A. Balador, and A. Sahafi, "A Fault-Tolerant Adaptive Genetic Algorithm for Service Scheduling in Internet of Vehicles," *Applied Soft Computing*, vol. 143, p. 110413, 2023, DOI: 10.1016/j.asoc.2023.110413.

[60] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, 2013, pp. 1–16, DOI: 10.1145/2523616.2523633.

[61] J. Bader, L. Thamsen, S. Kulagina, J. Will, H. Meyerhenke, and O. Kao, "Tarema: Adaptive Resource Allocation for Scalable Scientific Workflows in Heterogeneous Clusters," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 65–75, DOI: 10.1109/BigData52589.2021.9671997.

[62] International Energy Agency, "Energy and AI," IEA, Paris, Tech. Rep., 2025, licence: CC BY 4.0.

[63] A. Katal, S. Dahiya, and T. Choudhury, "Energy Efficiency in Cloud Computing Data Centers: A Survey on Software Technologies," *Cluster Computing*, vol. 26, no. 3, pp. 1845–1875, 2023, DOI: 10.1007/s10586-022-03649-7.

[64] M. Warade, J.-G. Schneider, and K. Lee, "Measuring the Energy and Performance of Scientific Workflows on Low-Power Clusters," *Electronics*, vol. 11, no. 11, p. 1801, 2022, DOI: 10.3390/electronics11111801.

[65] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A Measurement-Based Characterization of the Energy Consumption in Data Center Servers," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2863–2877, 2015, DOI: 10.1109/JSAC.2015.2481199.

[66] Q. Zhang, Z. Meng, X. Hong, Y. Zhan, J. Liu, J. Dong, T. Bai, J. Niu, and M. J. Deen, "A Survey on Data Center Cooling Systems: Technology, Power Consumption Modeling and Control Strategy Optimization," *Journal of Systems Architecture*, vol. 119, p. 102253, 2021, DOI: 10.1016/j.sysarc.2021.102253.

[67] A. Choudhary, M. C. Govil, G. Singh, L. K. Awasthi, and E. S. Pilli, "Energy-Aware Scientific Workflow Scheduling in Cloud Environment," *Cluster Computing*, vol. 25, no. 6, pp. 3845–3874, 2022, DOI: 10.1007/s10586-022-03671-9.

[68] H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, and P. Bv, "Energy-and Performance-Aware Scheduling of Tasks on Parallel and Distributed Systems," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 8, no. 4, pp. 1–37, 2012, DOI: 10.1145/2367736.2367741.

[69] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic Initialization of PSO Task Scheduling Algorithm in Cloud Computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2370–2382, 2020, DOI: 10.1016/j.jksuci.2020.11.002.

[70] R. O. Aburukba, T. Landolsi, and D. Omer, "A Heuristic Scheduling Approach for Fog-Cloud Computing Environment with Stationary IoT Devices," *Journal of Network and Computer Applications*, vol. 180, p. 102994, 2021, DOI: 10.1016/j.jnca.2021.102994.

[71] E. Casalicchio and V. Perciballi, "Measuring Docker Performance: What a Mess!!!" in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, 2017, pp. 11–16, DOI: 10.1145/3053600.3053605.

[72] G. Kecskemeti, W. Hajji, and F. P. Tso, "Modelling Low Power Compute Clusters for Cloud Simulation," in *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2017, pp. 39–45, DOI: 10.1109/PDP.2017.67.

[73] M. Warade, K. Lee, C. Ranaweera, and J.-G. Schneider, "Optimising Workflow Execution for Energy Consumption and Performance," in *2023 IEEE/ACM 7th International Workshop on Green And Sustainable Software (GREENS)*. IEEE, 2023, pp. 24–29, DOI: 10.1109/GREENS59328.2023.00011.

[74] M. A. R. Khan, S. N. Shavkatovich, B. Nagpal, A. Kumar, M. A. Haq, V. J. Tharini, S. Karupusamy, and M. B. Alazzam, "Optimizing Hybrid Metaheuristic Algorithm with Cluster Head to Improve Performance Metrics on the IoT," *Theoretical Computer Science*, vol. 925, pp. 87–104, 2022, DOI: 10.1016/j.tcs.2022.05.031.

[75] P. Lindberg, J. Leingang, D. Lysaker, S. U. Khan, and J. Li, "Comparison and Analysis of Eight Scheduling Heuristics for the Optimization of Energy Consumption and Makespan in Large-Scale Distributed Systems," *Journal of Supercomputing*, vol. 56, no. 1, pp. 25–47, 2010, DOI: 10.1007/s11227-010-0504-5.

[76] P. Hosseinioun, M. Kheirabadi, S. R. Kamel Tabbakh, and R. Ghaemi, "A New Energy-Aware Tasks Scheduling Approach in Fog Computing Using Hybrid Meta-Heuristic Algorithm," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 88–96, 2020, ISSN: 0743-7315, DOI: 10.1016/j.jpdc.2020.04.008.

[77] S. Perrin and T. Roncalli, "Machine Learning Optimization Algorithms & Portfolio Allocation," *Machine Learning for Asset Management: New Developments and Financial Applications*, pp. 261–328, 2020, DOI: 10.1002/9781119751182.ch8.

[78] R. V. Aroca and L. M. G. Gonçalves, "Towards Green Data Centers: A Comparison of x86 and ARM Architectures Power Efficiency," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1770–1780, 2012, DOI: 10.1016/j.jpdc.2012.08.005.

[79] S. Kreten, A. Guldner, and S. Naumann, "An Analysis of the Energy Consumption Behavior of Scaled, Containerized Web Apps," *Sustainability*, vol. 10, no. 8, p. 2710, 2018, DOI: 10.3390/su10082710.

[80] E. Blem, J. Menon, and K. Sankaralingam, "Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 1–12, DOI: 10.1109/HPCA.2013.6522302.

[81] A. Akram, "A Study on the Impact of Instruction Set Architectures on Processor's Performance," Master's thesis, Western Michigan University, 2017.

[82] S. Arora, "Approximation Schemes for NP-Hard Geometric Optimization Problems: A Survey," *Mathematical Programming*, vol. 97, no. 1, pp. 43–69, 2003, DOI: 10.1007/s10107-003-0438-y.

[83] V. A. Strusevich, "Complexity and Approximation of Open Shop Scheduling to Minimize the Makespan: A Review of Models and Approaches," *Computers & Operations Research*, vol. 144, p. 105732, 2022, DOI: 10.1016/j.cor.2022.105732.

[84] M. Hjeij and A. Vilks, "A Brief History of Heuristics: How Did Research on Heuristics Evolve?" *Humanities and Social Sciences Communications*, vol. 10, no. 1, pp. 1–15, 2023, DOI: 10.1057/s41599-023-01542-z.

[85] L. Yang, J. M. Schopf, and I. Foster, "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments," *Proceedings of the ACM/IEEE SC2003 Conference*, pp. 31–31, 2003, DOI: 10.1109/SC.2003.10056.

[86] Q. Wu, M. Zhou, and J. Wen, "Endpoint Communication Contention-Aware Cloud Workflow Scheduling," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 1137–1150, 2021, DOI: 10.1109/TASE.2021.3053765.

[87] R. Ghosh, S. P. R. Komma, and Y. Simmhan, "Adaptive Energy-Aware Scheduling of Dynamic Event Analytics Across Edge and Cloud Resources," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 72–82, DOI: 10.1109/CCGRID.2018.00019.

[88] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011, DOI: 10.1002/spe.995.

[89] A. Nunez, J. L. Vazquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente, "Design of a New Cloud Computing Simulation Platform," in *International Conference on Computational Science and Its Applications*. Springer, 2011, pp. 582–593, DOI: 10.1007/978-3-642-21934-4_47.

[90] T. Coleman, H. Casanova, and R. F. da Silva, "WfChef: Automated Generation of Accurate Scientific Workflow Generators," in *2021 IEEE 17th International Conference on eScience (eScience)*. IEEE, 2021, pp. 159–168, DOI: 10.1109/eScience51609.2021.00025.

[91] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the HPC Challenge Benchmark Suite," *Scientific Programming*, vol. 13, no. 4, pp. 213–219, 2005.

[92] M. Warade, J.-G. Schneider, and K. Lee, "FEPAC: A Framework for Evaluating Parallel Algorithms on Cluster Architectures," in *2021 Australasian Computer Science Week Multiconference*, 2021, pp. 1–10, DOI: 10.1145/3437378.3437394.

[93] P. J. Basford, S. J. Johnston, C. S. Perkins, T. Garnock-Jones, F. P. Tso, D. Pezaros, R. D. Mullins, E. Yoneki, J. Singer, and S. J. Cox, "Performance Analysis of Single Board Computer Clusters," *Future Generation Computer Systems*, vol. 102, pp. 278–291, Jan. 2020, DOI: 10.1016/j.future.2019.07.040, ISSN: 0167-739X.

[94] M. F. Cloutier, C. Paradis, and V. M. Weaver, "A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement," *Electronics*, vol. 5, no. 4, p. 61, 2016, DOI: 10.3390/electronics5040061.

[95] J. C. Davidson, "Clock System for Electronic Computers," Dec. 28 1965, US Patent 3,226,648.

[96] K. Srinivasan, C.-Y. Chang, C.-H. Huang, M.-H. Chang, A. Sharma, and A. Ankur, "An Efficient Implementation of Mobile Raspberry Pi Hadoop Clusters for Robust and Augmented Computing Performance," *Journal of Information Processing Systems*, vol. 14, no. 4, pp. 989–1009, 2018, DOI: 10.3745/JIPS.04.0075.

[97] G. F. Pfister, "An Introduction to the InfiniBand Architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, no. 617-632, p. 102, 2001.

[98] K. Hintze, S. Graham, S. Dunlap, and P. Sweeney, "InfiniBand Network Monitoring: Challenges and Possibilities," in *International Conference on Critical Infrastructure Protection*. Springer, 2021, pp. 187–208, DOI: 10.1007/978-3-030-75652-6_10.

[99] G. Kaur and M. Bala, "RDMA over Converged Ethernet: A Review," *International Journal of Advances in Engineering & Technology*, vol. 6, no. 4, pp. 1890–1900, 2013.

[100] A. C. Castillo, "Various Network Topologies and an Analysis Comparative Between Fat-Tree and BCube for a Data Center Network: An Overview," in *2022 IEEE Cloud Summit*. IEEE, 2022, pp. 1–8, DOI: 10.1109/CloudSummit54781.2022.00009.

[101] N. Schot, "Feasibility of Raspberry Pi 2 Based Micro Data Centers in Big Data Applications," in *Proceedings of the 23th University of Twente Student Conference on IT*, vol. 22, Enschede, The Netherlands, 2015.

[102] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2016, pp. 117–124, DOI: 10.1109/W-FiCloud.2016.36.

[103] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, and W. Dai, "Energy Efficient Task Allocation and Energy Scheduling in Green Energy Powered Edge Computing," *Future Generation Computer Systems*, vol. 95, pp. 89–99, 2019, DOI: 10.1016/j.future.2019.01.041.

[104] Y. Hao, J. Cao, Q. Wang, and J. Du, "Energy-Aware Scheduling in Edge Computing with a Clustering Method," *Future Generation Computer Systems*, vol. 117, pp. 259–272, 2021, ISSN: 0167-739X, DOI: 10.1016/j.future.2020.11.029.

[105] C. M. Rao and K. Shyamala, "Analysis and Implementation of a Parallel Computing Cluster for Solving Computational Problems in Data Analytics," in *2020 5th International Conference on Computing, Communication and Security (ICCCS)*. IEEE, 2020, pp. 1–5, DOI: 10.1109/ICCCS49678.2020.9277086.

[106] C. Jin, B. R. de Supinski, D. Abramson, H. Poxon, L. DeRose, M. N. Dinh, M. Endrei, and E. R. Jessup, "A Survey on Software Methods to Improve the Energy Efficiency of Parallel Computing," *International Journal of High Performance Computing Applications*, vol. 31, no. 6, pp. 517–549, 2017, DOI: 10.1177/1094342016675446.

[107] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of Backfilling Strategies for Parallel Job Scheduling," in *Proceedings. International Conference on Parallel Processing Workshop*. IEEE, 2002, pp. 514–519, DOI: 10.1109/ICPPW.2002.1039793.

[108] F. M. M. ul Islam and M. Lin, "Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning," *IEEE Systems Journal*, vol. 11, no. 2, pp. 931–940, 2015, DOI: 10.1109/JSYST.2015.2472139.

[109] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy Efficient DVFS Scheduling for Mixed-Criticality Systems," in *Proceedings of the 14th International Conference on Embedded Software*, 2014, pp. 1–10, DOI: 10.1145/2656045.2656057.

[110] W. Y. Lee, "Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-Core Processors," in *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. IEEE, 2009, pp. 216–223, DOI: 10.1109/DS-RT.2009.50.

[111] A. Verma, P. Ahuja, and A. Neogi, "Power-Aware Dynamic Placement of HPC Applications," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, 2008, pp. 175–184, DOI: 10.1145/1375527.1375555.

[112] S. K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jun, A. Y. Zomaya, and R. Ranjan, "Energy-Efficient VM-Placement in Cloud Data Center," *Sustainable Computing: Informatics and Systems*, vol. 20, pp. 48–55, 2018, DOI: 10.1016/j.suscom.2018.05.002.

[113] F. Teng, L. Yu, T. Li, D. Deng, and F. Magoulès, "Energy Efficiency of VM Consolidation in IaaS Clouds," *Journal of Supercomputing*, vol. 73, no. 2, pp. 782–809, 2017, DOI: 10.1007/s11227-016-1767-5.

[114] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and Performance Management of Virtualized Computing Environments via Lookahead Control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009, DOI: 10.1007/s10586-008-0070-y.

[115] C. Anderson, "Docker [Software Engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102–c3, 2015, DOI: 10.1109/MS.2015.62.

[116] J. Nickoloff and S. Kuenzli, *Docker in Action*, 2nd ed. Simon and Schuster, 2019, ISBN: 978-1617294761.

[117] J. Cook, "Docker Hub," in *Docker for Data Science*. Springer, 2017, pp. 103–118, DOI: 10.1007/978-1-4842-3012-1_7.

[118] E. A. Santos, C. McLean, C. Solinas, and A. Hindle, "How Does Docker Affect Energy Consumption? Evaluating Workloads in and out of Docker Containers," *Journal of Systems and Software*, vol. 146, pp. 14–25, 2018, DOI: 10.1016/j.jss.2018.07.077, ISSN: 0164-1212.

[119] A. Lingayat, R. R. Badre, and A. K. Gupta, "Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine," in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2018, pp. 1019–1023, DOI: 10.1109/CESYS.2018.8723931.

[120] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, "Sarus: Highly Scalable Docker Containers for HPC Systems," in *International Conference on High Performance Computing*. Springer, 2019, pp. 46–60, DOI: 10.1007/978-3-030-20656-7_4.

[121] M. Xu and R. Buyya, "BrownoutCon: A Software System Based on Brownout and Containers for Energy-Efficient Cloud Computing," *Journal of Systems and Software*, vol. 155, pp. 91–103, 2019, DOI: 10.1016/j.jss.2019.05.011.

[122] J. Luo, L. Yin, J. Hu, C. Wang, X. Liu, X. Fan, and H. Luo, "Container-Based Fog Computing Architecture and Energy-Balancing Scheduling Algorithm for Energy IoT," *Future Generation Computer Systems*, vol. 97, pp. 50–60, 2019, DOI: 10.1016/j.future.2019.02.016.

[123] Kubernetes Team, "Kubernetes: Production-Grade Container Orchestration," https://kubernetes.io/, 2019, retrieved May 24, 2019.

[124] N. Marathe, A. Gandhi, and J. M. Shah, "Docker Swarm and Kubernetes in Cloud Computing Environment," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 179–184, DOI: 10.1109/ICOEI.2019.8862763.

[125] S. Long, W. Wen, Z. Li, K. Li, R. Yu, and J. Zhu, "A Global Cost-Aware Container Scheduling Strategy in Cloud Data Centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2752–2766, 2021, DOI: 10.1109/TPDS.2021.3133458.

[126] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and R. Gaire, "Energy-Aware Resource Scheduling for Serverless Edge Computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 190–199, DOI: 10.1109/CCGrid54584.2022.00027.

[127] B. Barker, "Message Passing Interface (MPI)," in *Workshop: High Performance Computing on Stampede*, vol. 262. Houston, TX, USA: Cornell University, 2015.

[128] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-A Hunter of Idle Workstations," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep. CS-TR-1987-726, 1987.

[129] H. Chen, M. C. Caramanis, and A. K. Coskun, "Reducing the Data Center Electricity Costs Through Participation in Smart Grid Programs," in *International Green Computing Conference*. IEEE, 2014, pp. 1–10, DOI: 10.1109/IGCC.2014.7039172.

[130] T. Saillant, J.-C. Weill, and M. Mougeot, "Predicting Job Power Consumption Based on RJMS Submission Data in HPC Systems," in *International Conference on High Performance Computing*. Springer, 2020, pp. 63–82, DOI: 10.1007/978-3-030-59851-8_4.

[131] P. Shah, R. G. Shenoy, V. Srinivasan, P. Bose, and A. Buyuktosunoglu, "TokenSmart: Distributed, Scalable Power Management in the Many-Core Era," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 42–45, 2021, DOI: 10.1109/LCA.2021.3055560.

[132] H. Yang, "Energy Prediction for I/O Intensive Workflow Applications," PhD thesis, University of British Columbia, 2014.

[133] M. Nazeri, M. Soltanaghaei, and R. Khorsand, "A Predictive Energy-Aware Scheduling Strategy for Scientific Workflows in Fog Computing," *Expert Systems with Applications*, vol. 247, p. 123192, 2024, DOI: 10.1016/j.eswa.2024.123192.

[134] X. Mei, Q. Wang, X. Chu, H. Liu, Y.-W. Leung, and Z. Li, "Energy-Aware Task Scheduling with Deadline Constraint in DVFS-Enabled Heterogeneous Clusters," *arXiv preprint arXiv:2104.00486*, 2021.

[135] M. N. Bux, "Scientific Workflows for Hadoop," Ph.D. dissertation, Institute for Computer Science, Humboldt University, Berlin, August 2018, DOI: 10.18452/19321.

[136] B. P. Harenslak and J. de Ruiter, *Data Pipelines with Apache Airflow*. Simon and Schuster, 2021, ISBN: 978-1617296901.

[137] J. Köster and S. Rahmann, "Snakemake—A Scalable Bioinformatics Workflow Engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012, DOI: 10.1093/bioinformatics/bts480.

[138] E. Tsamtsurov and N. Balashov, "HTCondor Cluster Monitoring," *Physics of Particles and Nuclei*, vol. 55, no. 3, pp. 606–608, 2024, DOI: 10.1134/S1063779624030572.

[139] J. P. Jones, "PBS: Portable Batch System," in *Beowulf Cluster Computing with Windows*. MIT Press, 2001, pp. 363–383, ISBN: 978-0262133241.

[140] Y. Hao, J. Cao, Q. Wang, and J. Du, "Energy-Aware Scheduling in Edge Computing with a Clustering Method," *Future Generation Computer Systems*, vol. 117, pp. 259–272, 2021, DOI: 10.1016/j.future.2020.11.029.

[141] F. Sun, J. Cao, and Z. Lu, "HEFT-Dynamic Scheduling Algorithm in Workflow Scheduling," in *2022 34th Chinese Control and Decision Conference (CCDC)*. IEEE, 2022, pp. 4885–4890, DOI: 10.1109/CCDC55256.2022.10033778.

[142] J. Chen, Y. He, Y. Zhang, P. Han, and C. Du, "Energy-Aware Scheduling for Dependent Tasks in Heterogeneous Multiprocessor Systems," *Journal of Systems Architecture*, vol. 129, p. 102598, 2022, DOI: 10.1016/j.sysarc.2022.102598.

[143] J. Roeder, B. Rouxel, S. Altmeyer, and C. Grelck, "Energy-Aware Scheduling of Multi-Version Tasks on Heterogeneous Real-Time Systems," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 501–510, DOI: 10.1145/3412841.3441931.

[144] S. Singh, M. Dutta *et al.*, "Critical Path Based Scheduling Algorithm for Workflow Applications in Cloud Computing," in *2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Spring)*. IEEE, 2016, pp. 1–6, DOI: 10.1109/ICACCA.2016.7578885.

[145] R. Medara, R. S. Singh, and M. Sompalli, "Energy and Cost Aware Workflow Scheduling in Clouds with Deadline Constraint," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 13, p. e6922, 2022, DOI: 10.1002/cpe.6922.

[146] L. Ye, Y. Xia, and L. Yang, "A Cost-Aware Scheduling Algorithm for Reliable Workflow in IaaS Clouds," in *2021 33rd Chinese Control and Decision Conference (CCDC)*. IEEE, 2021, pp. 275–280, DOI: 10.1109/CCDC52312.2021.9602316.

[147] F. Cheng, Y. Huang, B. Tanpure, P. Sawalani, L. Cheng, and C. Liu, "Cost-Aware Job Scheduling for Cloud Instances Using Deep Reinforcement Learning," *Cluster Computing*, vol. 25, no. 6, pp. 4123–4140, 2022, DOI: 10.1007/s10586-022-03631-3.

[148] J. Ding, S. Schulz, L. Shen, U. Buscher, and Z. Lü, "Energy Aware Scheduling in Flexible Flow Shops with Hybrid Particle Swarm Optimization," *Computers & Operations Research*, vol. 125, p. 105088, 2021, DOI: 10.1016/j.cor.2020.105088.

[149] G. Xie, X. Xiao, H. Peng, R. Li, and K. Li, "A Survey of Low-Energy Parallel Scheduling Algorithms," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 1, pp. 27–46, 2021, DOI: 10.1109/TSUSC.2021.3057983.

[150] J. A. Liakath, P. Krishnadoss, and G. Natesan, "DCCWOA: A Multi-Heuristic Fault Tolerant Scheduling Technique for Cloud Computing Environment," *Peer-to-Peer Networking and Applications*, vol. 16, no. 2, pp. 785–802, 2023, DOI: 10.1007/s12083-022-01412-3.

[151] F. Liu, K. Hu, J. He, W. Hu, H. Li, M. Peng, and Y. He, "A Fault-Tolerant Scheduling Algorithm that Minimizes the Number of Replicas in Heterogeneous Service-Oriented Cloud Computing Systems," *Journal of Supercomputing*, vol. 80, no. 8, pp. 11 234–11 259, 2024, DOI: 10.1007/s11227-023-05847-2.

[152] M. Warade, K. Lee, C. Ranaweera, and J.-G. Schneider, "Energy Aware Adaptive Scheduling of Workflows," in *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2022, pp. 562–570, DOI: 10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00074.

[153] H. Wang and J. Deng, "A Multi-Objective Cloud Workflow Scheduling Strategy Based on the Modified HEFT Algorithm," in *Journal of Physics: Conference Series*, vol. 2504. IOP Publishing, 2023, p. 012060, DOI: 10.1088/1742-6596/2504/1/012060.

[154] M. Sardaraz and M. Tahir, "A Parallel Multi-Objective Genetic Algorithm for Scheduling Scientific Workflows in Cloud Computing," *International Journal of Distributed Sensor Networks*, vol. 16, no. 8, p. 1550147720949577, 2020, DOI: 10.1177/1550147720949577.

[155] Y. Qin, H. Wang, S. Yi, X. Li, and L. Zhai, "An Energy-Aware Scheduling Algorithm for Budget-Constrained Scientific Workflows Based on Multi-Objective Reinforcement Learning," *Journal of Supercomputing*, vol. 76, no. 1, pp. 455–480, 2020, DOI: 10.1007/s11227-019-02992-z.

[156] D. Papakyriakou, D. Kottou, and I. Kostouros, "Benchmarking Raspberry Pi 2 Beowulf Cluster," *International Journal of Computer Applications*, vol. 179, no. 32, pp. 21–27, Apr 2018, DOI: 10.5120/ijca2018916728, ISSN: 0975-8887.

[157] N. Balakrishnan, "Building and Benchmarking a Low Power ARM Cluster," Master's thesis, University of Edinburgh, 8 2012.

[158] P. Dehraj and A. Sharma, "A Review on Architecture and Models for Autonomic Software Systems," *Journal of Supercomputing*, vol. 77, no. 1, pp. 388–417, 2021, DOI: 10.1007/s11227-020-03298-6.

[159] J. Yen, J. C. Liao, B. Lee, and D. Randolph, "A Hybrid Approach to Modeling Metabolic Systems Using a Genetic Algorithm and Simplex Method," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 2, pp. 173–191, 1998, DOI: 10.1109/3477.662758.

[160] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of Interior-Point Methods to Model Predictive Control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998, DOI: 10.1023/A:1021711402723.

[161] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[162] S. Sivanandam and S. Deepa, "Genetic Algorithm Optimization Problems," in *Introduction to Genetic Algorithms*. Springer, 2008, pp. 165–209, DOI: 10.1007/978-3-540-73190-0_7.

[163] S. Z. Mirjalili, S. Mirjalili, S. Saremi, H. Faris, and I. Aljarah, "Grasshopper Optimization Algorithm for Multi-Objective Optimization Problems," *Applied Intelligence*, vol. 48, no. 4, pp. 805–820, 2018, DOI: 10.1007/s10489-017-1019-8.

[164] R. Poli, J. Kennedy, and T. Blackwell, "Particle Swarm Optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007, DOI: 10.1007/s11721-007-0002-0.

[165] X.-S. Yang and S. Deb, "Cuckoo Search: Recent Advances and Applications," *Neural Computing and Applications*, vol. 24, no. 1, pp. 169–174, 2014, DOI: 10.1007/s00521-013-1367-1.

[166] S. Chinnasamy, M. Ramachandran, M. Amudha, and K. Ramu, "A Review on Hill Climbing Optimization Methodology," *Recent Trends in Management and Commerce*, vol. 3, no. 1, pp. 85–92, 2022.

[167] Z. He, S. Deng, X. Xu, and J. Z. Huang, "A Fast Greedy Algorithm for Outlier Mining," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2006, pp. 567–576, DOI: 10.1007/11731139_67.

[168] S. Wiseman and A. M. Rush, "Sequence-to-Sequence Learning as Beam-Search Optimization," *arXiv preprint arXiv:1606.02960*, 2016.

[169] B. Selman, H. A. Kautz, B. Cohen *et al.*, "Local Search Strategies for Satisfiability Testing," *Cliques, Coloring, and Satisfiability*, vol. 26, pp. 521–532, 1993.

[170] D. Karaboga, "Artificial Bee Colony Algorithm," *Scholarpedia*, vol. 5, no. 3, p. 6915, 2010, DOI: 10.4249/scholarpedia.6915.

[171] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3668–3681, 2019, DOI: 10.1109/TCYB.2019.2950779.

[172] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," *TIK-Report*, vol. 103, 2001.

[173] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani, "Optimization with EM and Expectation-Conjugate-Gradient," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 672–679.

[174] L. Clarke, S. Fairley, X. Zheng-Bradley, I. Streeter, E. Perry, E. Lowy, A.-M. Tassé, and P. Flicek, "The International Genome Sample Resource (IGSR): A Worldwide Collection of Genome Variation Incorporating the 1000 Genomes Project Data," *Nucleic Acids Research*, vol. 45, no. D1, pp. D854–D859, 01 2017, DOI: 10.1093/nar/gkw829, ISSN: 0305-1048.

[175] W. McLaren, L. Gil, S. E. Hunt, H. S. Riat, G. R. Ritchie, A. Thormann, P. Flicek, and F. Cunningham, "The Ensembl Variant Effect Predictor," *Genome Biology*, vol. 17, no. 1, pp. 1–14, 2016, DOI: 10.1186/s13059-016-0974-4.

[176] I. Assayad, A. Girault, and H. Kalla, "A Bi-Criteria Scheduling Heuristics for Distributed Embedded Systems Under Reliability and Real-Time Constraints," in *International Conference on Dependable Systems and Networks*, ser. DSN '04. Firenze, Italy: IEEE, 2003, pp. 347–356, DOI: 10.1109/DSN.2004.1311902.

[177] E. Ilavarsan and P. Thambidurai, "Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments," *Journal of Computer Science*, vol. 3, no. 2, pp. 94–103, 2007, DOI: 10.3844/jcssp.2007.94.103.

[178] X. Wang, R. Buyya, and J. Su, "Reliability-Oriented Genetic Algorithm for Workflow Applications Using Max–Min Strategy," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 108–115, DOI: 10.1109/CCGRID.2009.51.

[179] S. Neuckamp, R. Z. Frantz, F. Roos-Frantz, and S. Sawicki, "A Mathematical Model to Compute Makespan for Application Integration Processes Using Graph Theory Approach," *International Journal of Computer Applications in Technology*, vol. 69, no. 3, pp. 193–205, 2022, DOI: 10.1504/IJCAT.2022.125530.

[180] J. Rudy, "Parallel Makespan Calculation for Flow Shop Scheduling Problem with Minimal and Maximal Idle Time," *Applied Sciences*, vol. 11, no. 17, p. 8204, 2021, DOI: 10.3390/app11178204.

[181] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A Parallel Bi-Objective Hybrid Metaheuristic for Energy-Aware Scheduling for Cloud Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 71, pp. 1497–1508, 2011, DOI: 10.1016/j.jpdc.2011.04.007.

[182] J. Kolodziej, S. U. Khan, and F. Xhafa, "Genetic Algorithms for Energy-Aware Scheduling in Computational Grids," in *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE, 2011, pp. 17–24, DOI: 10.1109/3PGCIC.2011.12.

[183] J. Kolodziej, S. U. Khan, L. Wang, and A. Zomaya, "Energy Efficient Genetic-Based Schedulers in Computational Grids," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1497–1508, 2012, DOI: 10.1002/cpe.1839.

[184] C. O. Diaz, M. Guzek, J. E. Pecero, G. Danoy, P. Bouvry, and S. U. Khan, "Energy-Aware Fast Scheduling Heuristics in Heterogeneous Computing Systems," in *2011 International Conference on High Performance Computing and Simulation*, ser. HPCS. IEEE, July 2011, pp. 478–484, DOI: 10.1109/HPCSim.2011.5999868.

[185] N. Min-Allah, H. Hussain, S. U. Khan, and A. Y. Zomaya, "Power Efficient Rate Monotonic Scheduling for Multi-Core Systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 1, pp. 48–57, 2012, DOI: 10.1016/j.jpdc.2011.09.004.

[186] A. Kassab, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo, "Green Power Aware Approaches for Scheduling Independent Tasks on a Multi-Core Machine," *Sustainable Computing: Informatics and Systems*, p. 100590, 2021, DOI: 10.1016/j.suscom.2021.100590.

[187] A. Kassab, J.-M. Nicod, and L. Philippe, "Green Power Constrained Scheduling for Sequential Independent Tasks on Identical Parallel Machines," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2019, pp. 132–139, DOI: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00028.

[188] K. Ji, F. Zhang, C. Chi, P. Song, B. Zhou, A. Marahatta, and Z. Liu, "A Joint Energy Efficiency Optimization Scheme Based on Marginal Cost and Workload Prediction in Data Centers," *Sustainable Computing: Informatics and Systems*, p. 100596, 2021, DOI: 10.1016/j.suscom.2021.100596.

[189] M. Khaleel and M. M. Zhu, "Energy-Aware Job Management Approaches for Workflow in Cloud," in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 506–507, DOI: 10.1109/CLUSTER.2015.87.

[190] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "Joint Computation Offloading and Scheduling Optimization of IoT Applications in Fog Networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 3266–3278, 2020, DOI: 10.1109/TNSE.2020.2999070.

[191] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-Efficient Dynamic Computation Offloading and Cooperative Task Scheduling in Mobile Cloud Computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 319–333, 2018, DOI: 10.1109/TMC.2018.2831230.

[192] J. Li, Y. Fan, and M. Zhou, "Performance Modeling and Analysis of Workflow," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 34, no. 2, pp. 229–242, 2004, DOI: 10.1109/TSMCA.2003.820576.

[193] M. Ghose, P. Verma, S. Karmakar, and A. Sahu, "Energy Efficient Scheduling of Scientific Workflows in Cloud Environment," in *Proceedings of 19th International Conference on High Performance Computing and Communications; 15th International Conference on Smart City; 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2017, pp. 170–177, DOI: 10.1109/HPCC-SmartCity-DSS.2017.22.

[194] T. Thanavanich and P. Uthayopas, "Efficient Energy Aware Task Scheduling for Parallel Workflow Tasks on Hybrids Cloud Environment," in *2013 International Computer Science and Engineering Conference (ICSEC)*. IEEE, 2013, pp. 37–42, DOI: 10.1109/ICSEC.2013.6694752.

[195] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic Energy-Aware Scheduling for Parallel Task-Based Application in Cloud Computing," *Future Generation Computer Systems*, vol. 78, pp. 257–271, 2018, DOI: 10.1016/j.future.2017.06.029.

[196] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, and R. Sakellariou, "Energy-Constrained Provisioning for Scientific Workflow Ensembles," in *2013 International Conference on Cloud and Green Computing*. IEEE, 2013, pp. 34–41, DOI: 10.1109/CGC.2013.14.

[197] E. N. Watanabe, P. P. Campos, K. R. Braghetto, and D. M. Batista, "Energy Saving Algorithms for Workflow Scheduling in Cloud Computing," in *2014 Brazilian Symposium on Computer Networks and Distributed Systems*. IEEE, 2014, pp. 9–16, DOI: 10.1109/SBRC.2014.17.

[198] P. Hosseinioun, M. Kheirabadi, S. R. Kamel Tabbakh, and R. Ghaemi, "A New Energy-Aware Tasks Scheduling Approach in Fog Computing Using Hybrid Meta-Heuristic Algorithm," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 88–96, 2020, DOI: 10.1016/j.jpdc.2020.04.008.

[199] R. Medara and R. S. Singh, "Energy Efficient and Reliability Aware Workflow Task Scheduling in Cloud Environment," *Wireless Personal Communications*, vol. 119, no. 2, pp. 1301–1320, 2021, DOI: 10.1007/s11277-021-08263-z.

[200] H. Raghu, S. K. Saurav, and B. S. Bapu, "PAAS: Power Aware Algorithm for Scheduling in High Performance Computing," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE, 2013, pp. 327–332, DOI: 10.1109/UCC.2013.63.

[201] Y. Liu, C. Du, J. Chen, and X. Du, "Scheduling Energy-Conscious Tasks in Distributed Heterogeneous Computing Systems," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 1, p. e6520, 2022, DOI: 10.1002/cpe.6520.

[202] S. Tu, S. U. Rehman, M. Waqas, O. U. Rehman, Z. Shah, Z. Yang, and A. Koubaa, "ModPSO-CNN: An Evolutionary Convolution Neural Network with Application to Visual Recognition," *Soft Computing*, vol. 25, pp. 2165–2176, 2021, DOI: 10.1007/s00500-020-05288-7.

[203] Q. Shang, "A Dynamic Resource Allocation Algorithm in Cloud Computing Based on Workflow and Resource Clustering," *Journal of Internet Technology*, vol. 22, no. 2, pp. 403–411, 2021, DOI: 10.3966/160792642021032202019.

[204] J. Saffran, G. Garcia, M. A. Souza, P. H. Penna, M. Castro, L. F. Góes, and H. C. Freitas, "A Low-Cost Energy-Efficient Raspberry Pi Cluster for Data Mining Algorithms," in *European Conference on Parallel Processing*. Springer, 2016, pp. 788–799, DOI: 10.1007/978-3-319-43659-3_57.

[205] M. d'Amore, R. Baggio, and E. Valdani, "A Practical Approach to Big Data in Tourism: A Low Cost Raspberry Pi Cluster," in *Information and Communication Technologies in Tourism 2015*. Springer, 2015, pp. 169–181, DOI: 10.1007/978-3-319-14343-9_13.

[206] B. Qureshi and A. Koubaa, "Power Efficiency of a SBC Based Hadoop Cluster," in *International Conference on Smart Cities, Infrastructure, Technologies and Applications*. Springer, 2017, pp. 52–60, DOI: 10.1007/978-3-319-94180-6_6.

[207] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures," in *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*. IEEE, 2013, pp. 108–112, DOI: 10.1109/ICDCSW.2013.25.

[208] G. Pomaska, "Stereo Vision Applying OpenCV and Raspberry Pi," *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. XLII-2/W17, pp. 265–269, 2019, DOI: 10.5194/isprs-archives-XLII-2-W17-265-2019.

[209] N. Salman, "Image Segmentation Based on Watershed and Edge Detection Techniques," *International Arab Journal of Information Technology*, vol. 3, no. 2, pp. 104–110, 2006.

[210] S. Patel, M. Potdar, and B. Gohil, "A Survey on Image Processing Techniques with OpenMP," *International Journal of Engineering Development and Research*, vol. 3, no. 4, pp. 837–839, 2015.

[211] R. F. Rahmat, T. Saputra, A. Hizriadi, T. Z. Lini, and M. K. Nasution, "Performance Test of Parallel Image Processing Using Open MPI on Raspberry PI Cluster Board," in *2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)*. IEEE, 2019, pp. 32–35, DOI: 10.1109/ELTICOM47379.2019.8943848.

[212] S. S. Tadesse, F. Malandrino, and C.-F. Chiasserini, "Energy Consumption Measurements in Docker," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2017, pp. 272–273, DOI: 10.1109/COMPSAC.2017.117.

[213] D.-K. Kang, G.-B. Choi, S.-H. Kim, I.-S. Hwang, and C.-H. Youn, "Workload-Aware Resource Management for Energy Efficient Heterogeneous Docker Containers," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 2428–2431, DOI: 10.1109/TENCON.2016.7848467.

[214] Z. Li, H. Wei, C. Lian, and S. Qin, "Docker-Based Energy Management System Development and Deployment Methods," in *2020 4th International Conference on Power and Energy Engineering (ICPEE)*. IEEE, 2020, pp. 165–168, DOI: 10.1109/ICPEE51316.2020.9311050.

[215] M. Sureshkumar and P. Rajesh, "Optimizing the Docker Container Usage Based on Load Scheduling," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, 2017, pp. 250–253, DOI: 10.1109/ICCCT2.2017.7972269.

[216] M. Chae, X. THANGONGSAK, Y. GUANG, and H. Lee, "Energy Efficient Web Load Balancer Using Docker," in *Proceedings of the Korea Information Processing Society Conference*. Korea Information Processing Society, 2018, pp. 43–45.

[217] N. VasanthaKumari and R. Arulmurugan, "Reorganizing Virtual Machines as Docker Containers for Efficient Data Centres," in *3rd EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing*. Springer, 2022, pp. 201–211, DOI: 10.1007/978-3-030-96040-7_22.

[218] I. M. Murwantara and P. Yugopuspito, "Evaluating Energy Consumption in a Different Virtualization within a Cloud System," in *2018 4th International Conference on Science and Technology (ICST)*. IEEE, 2018, pp. 1–6, DOI: 10.1109/ICSTC.2018.8528695.

[219] . E. Demirkol and A. Demirkol, "Energy Efficiency with an Application Container," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 26, no. 2, pp. 1129–1139, 2018, DOI: 10.3906/elk-1705-253.

[220] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, "FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2020, pp. 859–873, DOI: 10.1145/3373376.3378508.

[221] H. Djigal, J. Feng, and J. Lu, "Task Scheduling for Heterogeneous Computing Using a Predict Cost Matrix," in *Workshop Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10, DOI: 10.1145/3339186.3339220.

[222] F. Shrouf, J. Ordieres-Meré, A. García-Sánchez, and M. Ortega-Mier, "Optimizing the Production Scheduling of a Single Machine to Minimize Total Energy Consumption Costs," *Journal of Cleaner Production*, vol. 67, pp. 197–207, 2014, DOI: 10.1016/j.jclepro.2013.12.024.

[223] X. Ding and J. Wu, "Study on Energy Consumption Optimization Scheduling for Internet of Things," *IEEE Access*, vol. 7, pp. 70 574–70 583, 2019, DOI: 10.1109/ACCESS.2019.2919736.

[224] A. V. Dhumane and R. S. Prasad, "Multi-Objective Fractional Gravitational Search Algorithm for Energy Efficient Routing in IoT," *Wireless Networks*, vol. 25, no. 1, pp. 399–413, 2019, DOI: 10.1007/s11276-017-1569-x.

[225] I. Strumberger, N. Bacanin, M. Tuba, and E. Tuba, "Resource Scheduling in Cloud Computing Based on a Hybridized Whale Optimization Algorithm," *Applied Sciences*, vol. 9, no. 22, p. 4893, 2019, DOI: 10.3390/app9224893.

[226] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud Task Scheduling Based on Ant Colony Optimization," in *2013 8th International Conference on Computer Engineering & Systems (ICCES)*, IEEE. IEEE, 2013, pp. 64–69, DOI: 10.1109/ICCES.2013.6707172.

[227] G. Bindu, K. Ramani, and C. S. Bindu, "Optimized Resource Scheduling Using the Meta Heuristic Algorithm in Cloud Computing," *IAENG International Journal of Computer Science*, vol. 47, no. 3, pp. 360–366, 2020.

[228] S. Ghanavati, J. Abawajy, and D. Izadi, "An Energy Aware Task Scheduling Model Using Ant-Mating Optimization in Fog Computing Environment," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2007–2017, 2020, DOI: 10.1109/TSC.2020.2993332.

[229] Docker, Inc, "Docker Container Platform," *Docker Documentation*, 2020, accessed: June 2020.

[230] E. Carter, "2018 Docker Usage Report," Nov 2022. [Online]. Available: https://sysdig.com/blog/2018-docker-usage-report/

[231] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in Docker Development: A Large-Scale Study Using Stack Overflow," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11, DOI: 10.1145/3382494.3410febc.

[232] I. Miell and A. Sayers, *Docker in Practice*, 2nd ed. Simon and Schuster, 2019, ISBN: 978-1617294808.

[233] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar, "Provenance Trails in the Wings/Pegasus System," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 587–597, 2008, DOI: 10.1002/cpe.1228.

[234] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, and S. Bugoloni, "Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment," in *Proceedings of 5th International Conference on Cloud Computing Technology and Science*, Bristol, UK, December 2013, pp. 170–175, DOI: 10.1109/CloudCom.2013.121.

[235] K. Doucet and J. Zhang, "Learning Cluster Computing by Creating a Raspberry Pi Cluster," in *Proceedings of the SouthEast Conference*, 2017, pp. 191–194, DOI: 10.1145/3077286.3077336.

[236] B. Qureshi and A. Koubaa, "On Energy Efficiency and Performance Evaluation of Single Board Computer Based Clusters: A Hadoop Case Study," *Electronics*, vol. 8, no. 2, p. 182, February 2019, ISSN: 2079-9292, DOI: 10.3390/electronics8020182.

[237] M. Schwartz, "Internet of Things with ESP8266," in *Internet of Things with ESP8266*. Packt Publishing Ltd, 2016, ch. 13, pp. 190–200, ISBN: 978-1786468710.

[238] W. Chen and E. Deelman, "Integration of Workflow Partitioning and Resource Provisioning," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 2012, pp. 764–768, DOI: 10.1109/CCGrid.2012.121.

[239] J. K. Konjaang and L. Xu, "Cost Optimised Heuristic Algorithm (COHA) for Scientific Workflow Scheduling in IaaS Cloud Environment," in *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2020, pp. 162–168, DOI: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00038.

[240] J. Meena and M. Vardhan, "Cost-Effective Heuristic Workflow Scheduling Algorithm in Cloud Under Deadline Constraint," *Recent Advances in Computer Science and Communications*, vol. 13, no. 6, pp. 1302–1317, 2020, DOI: 10.2174/2666255813666200428114701.

[241] W. Reese, "Nginx: The High-Performance Web Server and Reverse Proxy," *Linux Journal*, vol. 2008, no. 173, p. 2, 2008.

[242] Datadog, "9 Insights on Real World Container Use," https://www.datadoghq.com/container-report/, 2022, accessed: 2022-12-10.

[243] G. Tene, "wrk2: A HTTP Benchmarking Tool Based Mostly on wrk," https://github.com/giltene/wrk2, 2018.

[244] O. H. Jader, S. Zeebaree, and R. R. Zebari, "A State of Art Survey for Web Server Performance Measurement and Load Balancing Mechanisms," *International Journal of Scientific & Technology Research*, vol. 8, no. 12, pp. 535–543, 2019, ISSN: 2277-8616.

[245] A. Krishnan, "NGINX vs Apache: Head to Head Comparison," https://hackr.io/blog/nginx-vs-apache, 2022, accessed: 2022-12-10.

[246] johnlpage, "johnlpage/POCDriver: Workload Driver for MongoDB in Java," https://github.com/johnlpage/POCDriver, Aug 2021.

[247] E. Geschwinde and H.-J. Schönig, *PostgreSQL Developer's Handbook*. Sams Publishing, 2002, ISBN: 978-0672322609.

[248] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003, DOI: 10.1109/MC.2003.1160055.

[249] K. Lee, N. W. Paton, R. Sakellariou, and A. A. Fernandes, "Utility Driven Adaptive Workflow Execution," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2009, pp. 220–227, DOI: 10.1109/CCGRID.2009.69.

[250] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler *et al.*, "Apache Airavata: A Framework for Distributed Applications and Computational Workflows," in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, 2011, pp. 21–28, DOI: 10.1145/2110486.2110490.

[251] E. Deelman *et al.*, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005, ISSN: 1058-9244.

[252] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005, DOI: 10.1177/1094342005051521.

[253] L. Dalcin and Y.-L. L. Fang, "mpi4py: Status Update After 12 Years of Development," *Computing in Science & Engineering*, vol. 23, no. 4, pp. 47–54, 2021, DOI: 10.1109/MCSE.2021.3083216.

[254] J. Liu, K. Wang, and F. Chen, "Understanding Energy Efficiency of Databases on Single Board Computers for Edge Computing," in *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2021, pp. 1–8, DOI: 10.1109/MASCOTS53633.2021.9614267.

[255] R. Garg, M. Mittal, and L. H. Son, "Reliability and Energy Efficient Workflow Scheduling in Cloud Environment," *Cluster Computing*, vol. 22, no. 4, pp. 1283–1297, 2019, DOI: 10.1007/s10586-017-1535-z.

[256] R. R. Barton and J. S. Ivey Jr, "Nelder-Mead Simplex Modifications for Simulation Optimization," *Management Science*, vol. 42, no. 7, pp. 954–973, 1996, DOI: 10.1287/mnsc.42.7.954.

[257] F. Glover, M. Laguna, and R. Martí, "Fundamentals of Scatter Search and Path Relinking," *Control and Cybernetics*, vol. 29, no. 3, pp. 653–684, 2000.

[258] H. Ritchie, P. Rosado, and M. Roser, "Energy Production and Consumption," https://ourworldindata.org/energy-production-consumption, Jul 2020.

[259] D. Gielen, R. Gorini, R. Leme, G. Prakash, N. Wagner, L. Janeiro, S. Collins, M. Kadir, E. Asmelash, R. Ferroukhi *et al.*, "World Energy Transitions Outlook: 1.5° C Pathway," *International Renewable Energy Agency (IRENA): Masdar City, United Arab Emirates*, 2021.

[260] S. Iserte, A. Castelló, R. Mayo, E. S. Quintana-Ortí, F. Silla, J. Duato, C. Reaño, and J. Prades, "SLURM Support for Remote GPU Virtualization: Implementation and Performance Study," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2014, pp. 318–325, DOI: 10.1109/SBAC-PAD.2014.47.

Appendices

Appendix I Definitions of terms

Term	Definition
Access	The reading or writing of data
Algorithm	A set of rules for solving a problem in a given number of steps
Bandwidth	The maximum data transfer rate of a network or Internet connection
Code	A language for expressing operations to be performed by a computer
Computing	Any goal-related activity comprising of computers
Configuration	The particular hardware elements and their interaction in a computer system for a particular period of operation
Database	Accessible collection of information
Dataset	A file or group of files associated with one part of a study
Energy consumption	The amount of power used in a particular time
Framework	Computer programs that perform various tasks
Instance	A particular occurrence of an object defined by a class
Master-slave	A relationship in which slave software obtains services from a master on behalf of a person
Memory	The fastest storage device of a computer
Microprocessor	The main computer chip providing the speed and capabilities of the computer
Node	A member of a network
Operating System	Software that controls the basic, low-level hardware operations, and file management
Parallel computing	Computations performed parallelly
Parameter	A value supplied to an algorithm
Port	The portion of a computer through which a peripheral device may communicate

Appendix II List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FEPAC	Framework for Evaluation of Parallel Algorithms on Clusters
FLOPS	Floating point Operations Per Second
FORTRAN	FOrmula TRANslation
GHz	GigaHertz
GUI	Graphical User Interface
I/O	Input/Output
IC	Integrated Circuits
IP	Internet Protocol
Js	JavaScript
MB	MegaByte
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
OpenCV	Open Computer Vision
RAM	Random Access Memory
ROM	Read Only Memory
SBC	Single Board Computer
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Measuring the Energy and Performance of Scientific Workflows on Low-Power Clusters		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

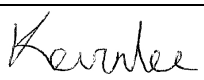
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Towards Energy-aware Scheduling of Scientific Workflows		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

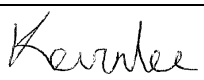
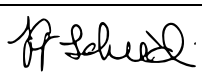
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Energy Aware Adaptive Scheduling of Workflows		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

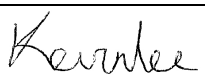
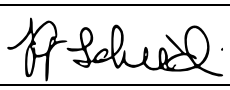
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Monitoring the Energy Consumption of Docker Containers		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

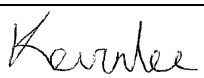
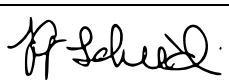
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Energy and Scientific Workflows: Smart Scheduling and Execution		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

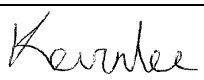
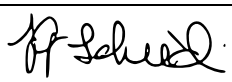
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Optimising workflow execution for energy consumption and performance		**Published**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

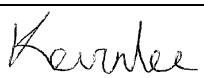
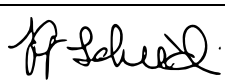
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
A Multi-User Energy-Aware Scheduler for Scientific Workflows		**Submitted**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

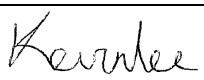
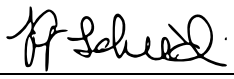
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

AUTHORSHIP STATEMENT

1. Details of publication and executive author

Title of Publication		Publication details
Energy-Aware Optimization and Scheduling of Scientific Workflows		**Submitted**
Name of executive author	School/Institute/Division if based at Deakin	Email or phone
Mehul Warade	**School of IT**	**mehul.warade@research.deakin.edu.au**

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?	**Yes**	If Yes, please complete Section 3 If No, go straight to Section 4.

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	School/Institute/Division if based at Deakin	Thesis title
As above	**School of IT**	**Energy-Aware Scientific Workflow Scheduling**
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication (for example, how much did you contribute to the conception of the project, the design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)		
HDR thesis author contributed to the conception of the project, the design of methodology, implementation, testing, evaluation and drafting the manuscript. They partly contributed to the revision of manuscript.		
I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.	Signature and date	 **Date: 03/06/2025**

4. Description of all author contributions

Name and affiliation of author	Contribution(s) (for example, conception of the project, design of methodology or experimental protocol, data collection, analysis, drafting the manuscript, revising it critically for important intellectual content, etc.)
Mehul Warade, Deakin University	Conception of the project, design, implementation, experimental, data collection and drafting the manuscript.
Kevin Lee, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Chathurika Ranaweera, Deakin University	Project supervision, intellectual content review, manuscript feedback and review.
Jean-Guy Schneider, Monash University	Project supervision, intellectual content review, manuscript feedback and review.

5. Author Declarations

I agree to be named as one of the authors of this work, and confirm:

i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,

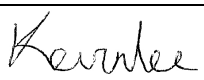
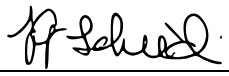
ii. that there are no other authors according to these criteria,

iii. that the description in Section 4 of my contribution(s) to this publication is accurate,

iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Name of author	Signature*	Date
Mehul Warade		03/06/2025
Kevin Lee		03/06/2025
Chathurika Ranaweera		03/06/2025
Jean-Guy Schneider		03/06/2025

6. Other contributor declarations

I agree to be named as a non-author contributor to this work.

Name and affiliation of contributor	Contribution	Signature* and date

***** If an author or contributor is unavailable or otherwise unable to sign the statement of authorship, the Head of Academic Unit may sign on their behalf, noting the reason for their unavailability, provided there is no evidence to suggest that the person would object.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Storage Location	Date lodged	Name of custodian if other than the executive author
N/A	N/A	N/A	N/A

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.