Aalto University
School of Science
Bachelor's Programme in Science and Technology

Multi-objective Computing Cluster Scheduling and Resource Reservation Optimization

Bachelor's Thesis

January 8, 2025

Valtteri Kyber

Aalto-yliopisto
Perustieteiden korkeakoulu
Teknistieteellinen kandidaattiohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Valtteri Kyber
Työn nimi:	Monitavoitteinen laskentaklusterin ajoitusten ja resurssien varauksen optimointi
Päiväys:	8. tammikuuta 2025
Sivumäärä:	21
Pääaine:	Tietotekniikka
Koodi:	SCI3027
Vastuuopettaja:	Professor Lauri Savioja
Työn ohjaaja(t):	PhD, University Teacher Tetiana Malykhina (Department of Computer Science)

Nyky-yhteiskunta on yhä riippuvaisempi laskentaklustereista, jotka tukevat moderneja sovelluksia, kuten tekoälyä, pilvilaskentaa ja suurten tietomäärien analysointia. Klusterit koostuvat toisiinsa liitetyistä palvelimista, jotka jakavat tehtäviä ja resursseja. Niiden hallinta vaatii älykkäitä strategioita, jotka tasapainottavat suorituskyvyn ja energiankulutuksen. Tämä on erityisen tärkeää suurissa datakeskuksissa, joiden energiankulutus vaikuttaa kustannuksiin ja ympäristöön.

Tässä kandidaatintyössä tarkastellaan laskentaklustereiden tehtävien ajoitusta ja resurssien varaamista. Työ analysoi menetelmiä, jotka pyrkivät tasapainottamaan suorituskyvyn, energiatehokkuuden ja resurssien käytön erityisesti pilvi- ja hybridilaskentaympäristöissä.

Kirjallisuustutkimus käsittelee perinteisiä menetelmiä kuten kierrosmenetelmä, sekä edistyneempiä menetelmiä, kuten geneettisiä algoritmeja. Joustavammat algoritmit soveltuvat erityisesti tilanteisiin, joissa tehtävien vaatimukset ja resurssien saatavuus vaihtelevat, mikä on yleistä nykyaikaisissa klustereissa. Lisäksi tarkastellaan tekoäly- ja koneoppimispohjaisia lähestymistapoja.

Tulokset osoittavat, että yksinkertaiset menetelmät toimivat vain vakio-olosuhteissa, kun taas edistyneemmät menetelmät sopeutuvat muuttuviin ympäristöihin paremmin. Geneettiset algoritmit ja hiukkasparvioptimointi ovat mahdollisesti erityisen hyviä ratkaisuja.

Avainsanat:	Klusteriajoitus Resurssivaraukset, Monitavoiteoptimointi, Laskentaklusterit, Energiatehokkuus, Pilvilaskenta, Dynaaminen ajoitus, Heterogeeniset ympäristöt, Kehittyneet algoritmit, Kuormituksen tasapainotus
Kieli:	English

Contents

Abbreviation	**Description**
SLA	Stringent Service-Level Agreements
FIFO/FCFS	First In, First Out / First Come, First Served
RR	Round Robin
GA	Genetic Algorithms
PSO	Particle Swarm Optimization
Starvation	Job waits indefinitely without execution
VM	Virtual Machine
QoS	Quality of Service
HPC	High Performance Computing

1 Introduction

In today's world of technology, the multi-objective computing clusters scheduling and resource reservation optimization have become crucial to support the high performance and diversity demands of modern applications. With the ever growing world of cloud computing, big data analytics, artificial intelligence, and many other applications, the need for efficient resource management in computing clusters is very important. Modern clusters often operate in hybrid environments, combining local infrastructure with cloud resources. This setup requires scheduling that balances resources across different platforms. In addition, with increasing emphasis on sustainability, organizations seek energy-efficient solutions that reduce carbon footprints while maintaining high performance. Thus, multi-objective computing clusters scheduling and resource reservation optimization plays a critical role in meeting these complex demands, making it essential for building responsive, scalable, and cost-effective systems in this computationally intensive world.

The origins of cluster computing date back to the 1960s and 1970s when the experiments with connecting multiple computers to maximize processing power and resource utilization started. One of the foundational algorithms first-come first served (FCFS) was very popular in these early systems, largely because it requires very little computational overhead. FCFS laid good groundwork for later, more complex methods (Jackson, 1957-08). Though clusters were relatively basic during this period, the ideas of efficient resource management and job scheduling introduced a base upon which future optimizations were built.

Savvas and Kechadi (2004) emphasize that computing clusters, particularly in cloud environments, require dynamic strategies for managing resource allocation and optimizing performance while addressing multiple objectives, such as reducing task execution time, minimizing costs, and improving load balancing. The authors investigate how these factors can be balanced to achieve optimal performance, even in uncertain environments where variables like network bandwidth and processing speeds may fluctuate. While keeping these objectives in mind, the clusters have to ensure availability and meet the stringent service-level agreements (SLAs).

This bachelor's thesis examines existing literature and studies that explore optimal resource utilization within computing clusters, focusing on minimizing energy consumption while maximizing task performance. Large-scale data centers, such as those operated by Google, consume vast amounts of energy to power their servers, cooling systems, and

network infrastructure. For instance, Google's data centers implement advanced cooling technologies and machine learning algorithms to reduce energy usage, that according to Gao (2014) achieves a Power Usage Effectiveness (PUE) as low as 1.12 that is a benchmark for sustainability. Such optimizations are crucial, as global data centers collectively consumed approximately 240–340TWh in 2022, accounting for around 1–1.3% of global electricity demand (IEA, 2022).

However, energy efficiency must not come at the expense of performance. Computational tasks, especially in fields like AI model training or real-time analytics, demand high throughput and low latency. Addressing this dual challenge of sustainability and performance requires advanced scheduling systems, such as SLURM, that is capable of managing resources effectively in complex computing environments.

Simple Linux Utility for Resource Management (SLURM) is an open-source workload manager widely used in high-performance computing (HPC) environments for resource allocation and job scheduling. Its scalability and flexibility allow it to operate seamlessly across heterogeneous clusters, supporting up to millions of processors. As highlighted in the RJMS-Bull SLURM tutorial (RJMS-Bull, 2024), SLURM's ovverall design and plugin architecture enable users to modify the system to specific needs. SLURM will be discussed more in section 3.1.

The core research problem addressed in this bachelor's thesis is how to balance multiple objectives in task scheduling while managing the uncertainties present in cloud computing environments. The aim is to examine existing research on dynamic and multi-objective scheduling approaches, with a particular emphasis on how different optimization methods address these challenges. The goal is not to develop new algorithms, but rather to explore and analyze existing solutions, drawing attention to the limitations and potential improvements in current research.

The structure of this bachelor's thesis is as follows: Chapter 2 presents a literature review on multi-objective optimization and scheduling in cloud computing. Chapter 3 compares and analyzes the algorithms currently used. Finally, in Chapter 4 the conclusions of this bachelor's thesis are explored.

2 Literature Review

This section focuses on the fundamental principles and evolving strategies of scheduling in cluster computing, which is crucial for optimizing resource utilization and achieving high performance in distributed environments. Scheduling in clusters involves assigning tasks to a set of nodes, with the primary goals of reducing task completion time, improving throughput, and balancing the workload across those nodes. In addition, basic resource allocation strategies are also introduced.

This review of scheduling algorithms in clusters will first cover foundational methods, including First-Come, First-Served (FCFS)(Balharith and Alhaidari, 2019) and Round Robin (RR) (Balharith and Alhaidari, 2019), which, despite their simplicity, serve as benchmarks for understanding more advanced strategies. The discussion will then explore heuristic and metaheuristic approaches, such as Genetic Algorithms (GA)(Forrest, 1996-03) and Particle Swarm Optimization (PSO) (Wang et al., 2018-1), which are widely used in many multi-objective scheduling scenarios. Also some hybrid algorithms are discussed.

2.1 Common scheduling algorithms in clusters

2.1.1 First-Come, First-Served

According to Bibu and Nwankwo (2019) FCFS scheduling algorithm is the simplest and most straightforward approach to scheduling. In this algorithm, jobs are processed in the exact order in which they arrive. The first job to enter the queue is the first to be allocated resources and executed, followed by the next job in line, and so on. There is no prioritization, meaning that the time a job spends waiting is solely dependent on when it entered the queue and the execution times of the jobs ahead of it.

Bibu and Nwankwo (2019) Illustrate that FCFS offers both advantages and disadvantages, with simplicity and fairness being among the most notable in common use cases. One significant benefit is that every job is guaranteed execution, ensuring there is no risk of starvation. However, despite these benefits, FCFS has notable drawbacks, as Rogiest et al. (2015-09) notes that in some cases, FCFS can lead to excessive waiting times for shorter jobs when they are queued behind longer tasks. This effect, often referred to as the convoy effect significantly reduces system efficiency by causing delays across the queue. Furthermore, the study highlights that FCFS does not account for job sizes, which limits its ability to minimize mean waiting times compared to scheduling methods

like Shortest-Processing-Time-First (SPTF), which prioritize shorter tasks and improve overall throughput in systems with varied job sizes.

Process	Arrival Time	Execution Time	Finish Time	Wait Time
P1	0	15	15	0
P2	1	12	27	14
P3	2	3	30	25
Average wait time: (0+14+25)/3 = 13				

Figure 1: Wait times of example processes using FCFS

Figure 1 illustrates an example of the FCFS scheduling algorithm applied to a set of three processes (P1, P2, and P3) with different arrival and execution times. In this example, each process is executed in the order it arrives, without consideration for its execution time. Process P1 arrives first and completes after 15 units of time, followed by P2, which starts immediately afterward and completes by time 27. Finally, P3, the shortest process, begins at time 27 and finishes at time 30. The waiting times for each process are calculated based on the time each process spends waiting in the queue before execution. In this case, P1 has a waiting time of 0, P2 waits for 14 units of time, and P3 waits for 25 units. The average waiting time, calculated as (0 + 14 + 25) / 3, results in 13 units. This example highlights a key limitation of FCFS: shorter processes, such as P3, can experience significant delays if they are queued behind longer processes, resulting in an increased average waiting time.

2.1.2 Round Robin

According to Balharith and Alhaidari (2019), RR scheduling algorithm is one of the most common approaches for time-sharing systems, designed to ensure that all processes receive a fair share of CPU time. In Round Robin scheduling, each process is assigned a fixed time slice called quantum, that is usually 10 to 100 milliseconds in length, during which it can execute, ensuring that no single task monopolizes system resources. They also note that if a process does not finish within its allocated quantum, it is preempted and placed at the back of the queue, allowing the next process to take its turn. This cycle continues until all processes have completed.

Although RR could possibly be very good at certain tasks, Balharith and Alhaidari (2019) note that the efficiency of the algorithms entirely depends on the size of the quantum. The authors also mention that the RR algorithm is widely used due to its simplicity and inherent fairness in distributing CPU time among processes. They also note that if the quantum is too large, the algorithm begins to resemble the FCFS approach, leading to longer wait times for shorter tasks. Conversely, if the quantum is too small, frequent

context switches increase overhead and reduce overall efficiency.

The study further highlights the application of RR in both CPU scheduling and cloud computing environments, where it enhances performance and improves QoS In cloud computing, for instance, the RR algorithm is often used at two levels: first, to allocate CPU time within VMs for different tasks and second, to distribute tasks among VMs in a balanced manner. This dual-level setup helps maintain fairness and prevents resource monopolization across multiple tasks and users, which is crucial in a multi-tenant cloud setting.

Process	Arrival Time	Execution Time	Finish Time	Wait Time
P1	0	15	30	15
P2	1	12	27	14
P3	2	3	11	6
Average wait time: (15+14+6)/3 ≈ 11.67				

Figure 2: Wait times of example processes using Round Robin

In Figure 2, the processes with the same characteristics as in Figure 1 (i.e., identical arrival times and execution durations) are scheduled using the RR algorithm. In this example, a time quantum of 4 units is used. To simplify the example, it is assumed that switching between processes takes no time. This approach also assumes no prioritization among processes, aside from their order of arrival. As a result, each process receives multiple turns to execute until completion, leading to fair distribution of CPU time among all tasks. The RR algorithm mitigates the bottleneck effect that can occur in FCFS by allowing shorter processes to complete without being blocked indefinitely by longer ones. The table in Figure 2 illustrates each process's finish time and total wait time, resulting in an average wait time of approximately 11.67 units. This example demonstrates the balance that RR achieves between fairness and efficiency.

2.2 More complex and general purpose algorithms

2.2.1 Genetic Algorithms

Genetic algorithms (GAs) were originally introduced by John Holland in the 1970s. According to Busetti (2007) GAs are optimization techniques designed to solve problems that are too complex for traditional methods. The author mentions that they are based on the principles of biological evolution, GAs use processes like selection, crossover, and mutation to explore large solution spaces. The author implies that GAs are effective

for problems with multiple objectives or constraints, where balancing different factors is key. According to Beasley et al. (1993), GAs operate using a population of solutions, iteratively improving them over generations. This makes them adaptable and possibly suitable for dynamic or uncertain problems, with applications in areas such as machine learning, scheduling, and resource allocation.

Lambora et al. (2019) describe GAs as search-based algorithms rooted in natural selection and heredity principles. They explain that GAs belong to a broader category of evolutionary computation, where solutions are represented as chromosomes and evolve over multiple generations through genetic operations like selection, crossover, and mutation. Each solution, or individual, in the population is assigned a fitness score based on how well it performs against a specified objective, with fitter individuals having a higher probability of passing their traits to the next generation. This iterative process ensures that each generation is progressively more optimized towards finding the best solution, the authors highlight that GAs are particularly advantageous in scheduling applications due to their flexibility in handling both constrained and unconstrained optimization problems. By encoding each potential solution as a chromosome and iterating through generations, GAs can explore large search spaces efficiently. This ability is especially useful in dynamic scheduling scenarios, where task requirements and resource availability may frequently change, as the algorithm can adaptively search for near-optimal solution

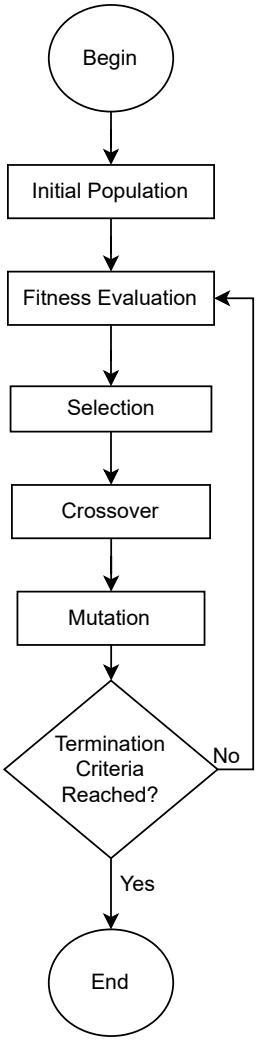

Figure 3: A flowchart illustrating the Genetic Algorithm process.

Katoch et al. (2021-02) explain that GAs offer considerable advantages in optimization tasks, particularly noting their adaptability to a wide range of complex problems where traditional algorithms may fall short. The authors state that GAs can efficiently handle both continuous and discrete optimization problems without needing gradient information, making them versatile across diverse domains. They highlight that GAs' ability to explore the solution space broadly allows for highly optimized solutions, especially valuable in large, multi-dimensional search spaces. However, the authors also point out notable disadvantages, including the risk of premature convergence, where the algorithm may settle on a local optimum instead of exploring potentially better solutions. To mitigate this issue, they recommend techniques like mutation and hybridization with other algorithms to encourage greater exploration. Additionally, the authors caution that GAs can be computationally intensive, especially when dealing with large populations and multiple generations, which can impact their efficiency in some applications.

As Katoch et al. (2021-02) mentions, GAs require precise tuning of parameters such as population size, crossover rate, and mutation rate to achieve optimal results. Improper tuning can lead to premature convergence on suboptimal solutions or inefficient exploration of the solution space. The authors also mention that addressing these challenges is crucial for improving the performance of GAs, with adaptive methods being a key area of research to enhance their robustness and efficiency.

One adaptation of GAs was introduced by Fonseca et al. (1993), multi-objective genetic algorithms (MOGAs) that were formulated to address the challenges of optimizing multiple conflicting objectives simultaneously. Unlike single-objective genetic algorithms (SOGAs), MOGAs aim to generate a diverse set of solutions along the Pareto front, representing optimal trade-offs among objectives without requiring predefined weights or priorities.

Even though Murata and Ishibuchi (1995) demonstrated that MOGAs outperform SOGAs in finding the Pareto front, Katoch et al. (2021-02) highlight the Niched Pareto Genetic Algorithm (NPGA) as a more suitable alternative. NPGA uses a tournament-like selection process, which not only preserves diversity within the population but also enhances its ability to efficiently explore and exploit the solution space for multi-objective optimization problems.

2.2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based stochastic optimization algorithm motivated by the social behavior of animal groups, such as bird flocks and fish schools. Originally PSO was proposed by Kennedy and Eberhart (1995), since then PSO has been extensively refined over the years to address a wide range of optimization problems. As Wang et al. (2018-1) explain, PSO leverages collective intelligence by enabling particles to explore the solution space collaboratively, updating their velocities and positions based on personal and global best solutions.

PSO shares similarities with GAs (covered above 2.2.1), as both are population-based metaheuristic optimization algorithms inspired by natural phenomena. They aim to solve complex optimization problems by iteratively improving a population of candidate solutions, making them widely used for tasks like resource allocation and scheduling in computing clusters. Wang et al. (2018-1) highlight, PSO relies on social and cognitive interactions between particles to explore and exploit the solution space, which parallels GA's reliance on evolutionary processes such as selection, crossover, and mutation.

Unlike GAs, PSO does not rely on genetic operators like crossover and mutation to generate new solutions. Instead, PSO simulates social interactions, where particles in a swarm adjust their positions in the solution space based on their personal and the swarm's collective best experiences and the knowledge. As highlighted by Juneja and Nagar (2016), PSO mimics the natural goal-seeking behavior of animal groups, such as birds in a flock or fish in a school, by iteratively moving particles toward the best-known solutions in the search space.

This approach allows PSO to achieve efficient optimization without employing the survival-of-the-fittest principle that defines GAs. The absence of selection operations ensures that all particles in the swarm contribute to the search process, maintaining diversity throughout. By balancing exploration and exploitation through its velocity and position update rules, PSO has demonstrated great success in handling complex, multimodal optimization problems across various domains and other algorithms. An excellent example is the hybrid PPTS-PSO algorithm, which combines PSO's evolutionary capabilities with the heuristic strengths of the Predict Priority Task Scheduling (PPTS) approach (Talha and Malki, 2023).

According to Wang et al. (2018-1), one of the significant advancements in PSO research has been the introduction of multi-objective PSO (MOPSO), which brings up Pareto

dominance principles to optimize conflicting objectives simultaneously. MOPSO enables the generation of a Pareto front, providing decision-makers with a diverse set of optimal solutions to choose from, tailored to their specific priorities and constraints. The authors also emphasize PSO's versatility in adapting its topology and parameter configurations to enhance performance across various domains. They discuss the significance of dynamic and static neighborhood structures, such as global and local best strategies, in balancing exploration and exploitation during optimization. Moreover, they highlight the role of parameter tuning, particularly inertia weight and acceleration coefficients, in influencing convergence rates and solution quality. The authors delve into advanced variants, including hybrid PSO models that integrate concepts like chaos theory, genetic algorithms, and quantum-inspired mechanisms, to address limitations such as premature convergence and to improve robustness. Additionally, the article outlines PSO's applications in engineering, multi-objective optimization, and real-world scenarios like network routing, energy management, and job scheduling, showcasing its possibly continued evolution as an optimization tool.

3 Analyzing And Dialog

In this section, the discussion focuses on workload managers and their role in enabling scheduling algorithms to function effectively, alongside a detailed comparison of the algorithms themselves. Topic will use Simple Linux Utility for Resource Management as an example workload manager. Additionally, the section compares the performance and adaptability of scheduling algorithms studied in section 2.

3.1 SLURM

Scheduling algorithms cannot function without a system to run on, known as a workload manager, which provides the framework for resource allocation, job queuing, and execution. Workload managers like the Simple Linux Utility for Resource Management (SLURM) enable these algorithms to interact with hardware, enforce scheduling policies, and monitor system states, making theoretical strategies practical for real-world applications. For example, while FCFS is often used as a baseline algorithm, its limitations are addressed in SLURM by combining it with other techniques such as backfilling. According to the Cluster-Handbook (2024), SLURM uses backfilling by building on FCFS by allowing smaller tasks to utilize idle resources while waiting for larger jobs, thus improving resource utilization and reducing wait times. Without workload managers like SLURM, even the most sophisticated algorithms would remain theoretical, unable to contribute to

efficient task scheduling in high-performance computing environments.

The Triton cluster is Aalto University's HPC system, designed to support a wide range of research activities across various disciplines (Aalto Scientific Computing, 2024). Operating on a Linux-based environment and managed by Aalto Scientific Computing, Triton provides researchers with substantial computational resources, including numerous CPU and GPU nodes, to facilitate complex simulations, data analysis, and other computationally intensive tasks. Hundreds of users use Triton simultaneously, thus pure FCFS approach would lead to significant inefficiencies. The cluster utilizes the SLURM workload manager for efficient job scheduling and resource allocation (Aalto Scientific Computing, 2024).

SLURM was originally developed by Lawrence Livermore National Laboratory to meet the resource management needs of large-scale HPC systems. It was released under an open-source license and has since evolved into one of the most widely used workload managers, supported by a global user community and organizations such as SchedMD, that it was passed to in 2011, that continues its development and maintenance. (RJMS-Bull, 2016; Cluster-Handbook, 2024)

SLURM has energy-aware scheduling features that optimize power consumption across compute nodes, aligning with the growing emphasis on sustainability and making it a valuable tool for research into green computing and resource efficiency (RJMS-Bull, 2016). Additionally, its modular plugin architecture supports customization, enabling the development of specialized plugins for tasks like multi-objective optimization and resource reservation, which enhances its flexibility for diverse workload management scenarios (SchedMD, 2024).

SLURM uses several foundational scheduling algorithms, including FCFS and variants of RR, to manage job execution and resource allocation effectively (Cluster-Handbook, 2024). FCFS offers simplicity and predictability but occasionally leads to inefficiencies when shorter jobs are delayed by longer ones (SchedMD, 2024). To improve resource fairness and utilization, SLURM employs RR-like approaches in specific contexts, such as time-sharing scenarios, where tasks are allocated CPU time slices cyclically (Cluster-Handbook, 2024).

3.2 Comparing algorithms and results

This section will compare four algorithms representing both simple and complex scheduling approaches addressed in this bachelors thesis: First-Come, First-Served (FCFS), Genetic Algorithm (GA), Round Robin (RR), and Particle Swarm Optimization (PSO). These algorithms were chosen to highlight a range of strategies, from straightforward methods to advanced optimization techniques, relevant to multi-objective computing cluster scheduling.

The comparison focuses on key performance aspects, including fairness, scalability, and ability to handle complex resource constraints. FCFS and RR are explored as foundational scheduling methods, while GA and PSO represent advanced, optimization-driven techniques. By analyzing these algorithms, this chapter provides a structured understanding of their strengths and limitations, offering insights into their practical implications for resource reservation and cluster scheduling.

Algorithm	FCFS	RR	GA	PSO
Type	Static, Non-Preemptive	Static, Preemptive	Metaheuristic, Adaptive	Metaheuristic, Adaptive
Time Complexity	O(N)	O(N)	O(G*P*E)	O(P*E*I)
Fairness	None	Strong	*	*
Adaptability	Low	Medium	High	High

Figure 4: Summary of charasteristics of studied algorithms

The comparison of time complexity, fairness, and adaptability for the studied algorithms, FCFS, RR, GA, and PSO, is summarized in Figure 4. This figure provides a concise overview of the characteristics of these algorithms, highlighting their differences and similarities in approach and computational requirements. As shown, the time complexity for FCFS and RR is O(N), indicating their minimal computational overhead and simplicity.

On the other hand, we have the more complex algorithms, with their time complexity approximated to three main components. In the figure, GAs' time complexity is expressed as O(G*P*E), where G is the number of generations or iterations, P is the population size, and E is the evaluation cost of the fitness function. These components represent the main components of the algorithm.

As marked in Figure 4, PSO has the time complexity of O(P*E*I), that is also

approximated to three main components. Here, P represents the number of particles in the swarm, E is the evaluation cost of the fitness function, and I denotes the number of iterations required for convergence.

When using more complex algorithms for optimization, choosing the correct one depends highly on the problem and its characteristics. GA are particularly effective for problems with diverse, multi-modal search spaces and strict constraints. On the other hand, PSO excels in problems requiring faster convergence and fewer parameter adjustments, particularly in continuous or less constrained environments. As demonstrated by Roberge et al. (2013) in a real-time UAV route planning example, GA performed better in generating high-quality, refined solutions for complex trajectories, while PSO offered faster initial convergence but struggled in fine-tuning solutions under similar conditions. This comparison underscores the importance of aligning algorithm selection with the specific objectives and constraints of the optimization task.

While the more complex algorithms can be made fair to meet the needed requirements, even very specific ones, by tuning their parameters, this flexibility comes with added complexity in configuration and execution. In Figure 4, the fairness of GA and PSO is unmarked because the fairness of these algorithms is not inherent but rather dependent on how they are configured. By carefully tuning parameters like selection in GA or social and cognitive coefficients in PSO, the algorithms can be adjusted to balance exploration and exploitation, ensuring fairness in resource distribution or solution prioritization. However, without proper configuration, these algorithms may favor certain solutions disproportionately, making their fairness highly variable and problem-specific. This adaptability contrasts with simpler algorithms like FCFS and RR, where fairness is hardwired but lacks the customization to address more complex requirements, making complex algorithms more suitable for tailored and multi-objective optimization tasks. Usually the complex methods work better for example in cloud computing environments that tend to have very diverse hardware and problem cases.

In Figure 4, the adaptability refers to the algorithms ability to adjust to dynamic environments, varying constraints, or changing objectives. Simpler algorithms like FCFS and RR are inherently rigid; they operate based on predefined rules and cannot adapt to changes in priorities or resource availability. While this simplicity ensures predictability, and requires less computational overhead, it makes these algorithms unsuitable for environments where adaptability is crucial. In the figure, RR has been marked with medium adaptability because, while it is primarily a static scheduling algorithm, it exhibits some flexibility in handling tasks with varying execution times or workloads.

RR's preemptive nature, where tasks are assigned fixed time slices, that can be tuned to fit certain problems, and re-queued if incomplete, allows it to respond to dynamic workloads more effectively than non-preemptive algorithms like FCFS.

4 Conclusions

This bachelor's thesis has explored the comparative performance and applicability of different optimization algorithms, focusing on Particle Swarm Optimization and Genetic Algorithms in the context of multi-objective scheduling and resource allocation, and using First Come First Served and Round Robin as simpler algorithms to compare task scheduling capabilities. This bachelors thesis has highlighted the trade-offs inherent in selecting an optimization algorithm for specific problems. The work emphasizes that while all the algorithms have their strengths, their effectiveness is highly dependent on the nature of the task, the constraints involved, and the level of complexity required in the problem space.

To specify, FCFS is adequate in simple systems with uniform workloads, RR is well suited in Time-sharing systems where responsiveness and fairness are required. Neither of the simple algorithms by themselves are able to tackle multiple objectives simultaneously. Genetic Algorithms are well suited for scenarios where multiple objectives need to be optimized, such as cloud cluster task scheduling. Particle Swarm Optimization works best with tasks that have continuous solution spaces and faster convergence is prioritized over exhaustive exploration. Indicating that PSO would work best of the studied algorithms in real-time scenarios.

In the context of cluster scheduling and resource allocation, this thesis also reflects on the use of SLURM, a widely used job scheduler and workload manager for HPC clusters. SLURM's design, which allows for customizable scheduling plugins, makes it a robust framework to implement and test optimization algorithms like PSO and GA. While SLURM primarily employs policies that are almost like FCFS and prioritization queues for task scheduling, it provides flexibility for integrating more advanced optimization methods, including those studied in this thesis. This makes it a valuable tool for exploring the practical deployment of sophisticated scheduling algorithms in real-world computing clusters.

The findings suggest that combining SLURM's modular scheduling capabilities with

advanced optimization techniques could offer significant improvements in energy efficiency, resource utilization, and task execution time, especially in environments with hybrid cloud and local resources. Future work could focus on integrating algorithms like PSO and GA into SLURM's scheduling framework to assess their impact on real-world cluster performance.

In conclusion, while simple algorithms like FCFS and RR offer ease of implementation and adequate performance in specific scenarios, advanced optimization methods, such as GA and PSO, provide the flexibility and adaptability required for more modern, multi-objective scheduling tasks. The integration of these methods into existing cluster management tools like SLURM holds promising potential for addressing the challenges of next-generation computing clusters.

References

Aalto Scientific Computing (2024), 'Triton Cluster'. [Accessed 3 Nov. 2024].
URL: *https://scicomp.aalto.fi/triton/*

Balharith, T. and Alhaidari, F. (2019), Round robin scheduling algorithm in cpu and cloud computing: A review, *in* '2019 2nd International Conference on Computer Applications Information Security (ICCAIS)', pp. 1–7.

Beasley, D., Bull, D. R. and Martin, R. R. (1993), 'An overview of genetic algorithms: Part 1, fundamentals', *University computing* **15**(2), 56–69.

Bibu, G. D. and Nwankwo, G. C. (2019), 'Comparative analysis between first-come-first-serve (fcfs) and shortest-job-first (sjf) scheduling algorithms'.

Busetti, F. (2007), 'Genetic algorithms overview', *Retrieved on December* **1**.

Cluster-Handbook (2024), *Cluster Handbook: SLURM*. Accessed: 2024-11-28.
URL: *https://en.wikibooks.org/wiki/Cluster-Handbook/SLURM*

Fonseca, C. M., Fleming, P. J. et al. (1993), Genetic algorithms for multiobjective optimization: formulationdiscussion and generalization., *in* 'Icga', Vol. 93, Citeseer, pp. 416–423.

Forrest, S. (1996-03), 'Genetic algorithms', *ACM computing surveys /* **28**(1).

Gao, J. (2014), 'Machine learning applications for data center optimization', *Google Research* . [Accessed: 26-Nov-2024].
URL: *https://www.google.com/about/datacenters/efficiency/internal/*

IEA (2022), 'Data centres and data transmission networks'. [Accessed: 26-Nov-2024].
URL: *https://www.iea.org/reports/data-centres-and-data-transmission-networks*

Jackson, J. R. (1957-08), 'Networks of waiting lines', *Operations research.* **5**(4).

Juneja, M. and Nagar, S. K. (2016), Particle swarm optimization algorithm and its parameters: A review, *in* '2016 International Conference on Control, Computing, Communication and Materials (ICCCCM)', pp. 1–5.

Katoch, S., Chauhan, S. S. and Kumar, V. (2021-02), 'A review on genetic algorithm: past, present, and future', *Multimedia tools and applications.* **80**(5).

Kennedy, J. and Eberhart, R. (1995), Particle swarm optimization, *in* 'Proceedings of the IEEE International Conference on Neural Networks (ICNN)', IEEE, Perth, Australia, pp. 1942–1948.

Lambora, A., Gupta, K. and Chopra, K. (2019), Genetic algorithm- a literature review, *in* '2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)', pp. 380–384.

Murata, T. and Ishibuchi, H. (1995), Moga: multi-objective genetic algorithms, *in* 'Proceedings of 1995 IEEE International Conference on Evolutionary Computation', Vol. 1, pp. 289–.

RJMS-Bull (2016), *State of the Art in Slurm.* Accessed: 2024-11-28.
URL: *https://github.com/RJMS-Bull/slurm-tutorial/blob/master/StateOfTheArt$_S$lurmIEEECluste*

RJMS-Bull (2024), 'Slurm tutorial', `https://github.com/RJMS-Bull/slurm-tutorial`. Accessed: 26-Nov-2024.

Roberge, V., Tarbouchi, M. and Labonte, G. (2013), 'Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning', *IEEE Transactions on Industrial Informatics* **9**(1), 132–141.

Rogiest, W., Laevens, K., Walraevens, J. and Bruneel, H. (2015-09), 'When random-order-of-service outperforms first-come-first-served', *Operations research letters* **43**(5).

Savvas, I. and Kechadi, M.-T. (2004), Dynamic task scheduling in computing cluster environments, *in* 'Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks', pp. 372–379.

SchedMD (2024), *Slurm Overview.* Accessed: 2024-11-28.
URL: *https://slurm.schedmd.com/slurm.html*

Talha, A. and Malki, M. O. C. (2023), 'Ppts-pso: a new hybrid scheduling algorithm for scientific workflow in cloud environment', *Multimedia Tools and Applications* **82**(21), 33015–33038.
URL: *https://www.proquest.com/scholarly-journals/ppts-pso-xa0-new-hybrid-scheduling-algorithm/docview/2859377467/se-2*

Wang, D., Tan, D. and Liu, L. (2018-1), 'Particle swarm optimization algorithm: an overview', *Soft computing : a fusion of foundations, methodologies and applications.* **22**(2).