Container Scheduling Methods in Cloud Environments

Max Koskinen
max.koskinen@aalto.fi

Tutor: Antti Ylä-Jääski

Abstract

Containerization has emerged as a key technology in cloud computing due to its lightweight virtualization, making it ideal for both short-lived tasks and long-running services. Its compatibility with the microservice architecture paradigm has further driven adoption, enabling scalable, modular, and efficient service deployment. However, as application demands evolve, effective container scheduling becomes critical for optimizing resource utilization, minimizing costs, and enhancing performance. Traditional scheduling techniques such as bin packing, spread, and random allocation leave room for improvement. This paper explores advanced scheduling methods, including deep reinforcement learning, meta-heuristic approaches, and mathematical modeling, which offer significant improvements over the conventional techniques. While these state-of-the-art strategies increase resource efficiency and quality of service metrics, scalability remains a challenge in large-scale cloud environments. Further research is essential to develop production-ready scheduling frameworks that ensure optimal workload placement while maintaining computational feasibility in modern data centers.

KEYWORDS: Container Scheduling, Cloud computing

1 Introduction

Cloud computing, known for its high-performance capabilities, continues to drive digital transformation by providing on-demand computing resources [6]. Achieving these performance standards requires advanced technical methods and continues research efforts [19]. A key enabler of efficient cloud resource management is virtualization, which has long been fundamental to optimizing resource utilization and ensuring workload isolation [2]. Traditionally, virtualization is implemented using hypervisors, which manage virtual machines (VMs) on a single physical machine. However, running a VM requires an entire operating system (OS) stack, resulting in unnecessary overhead.

While traditional hypervisor-based virtualization has enabled effective isolation and resource utilization, the emergence of containerization—a more lightweight virtualization technology—has reduced the resource overhead associated with running a full OS stack [19]. The shift toward a microservices-based architectural paradigm, combined with the adoption of containerization, has accelerated the development of both technologies, as they complement each other.

In the microservice architecture paradigm, applications are defined as a set of independent and small modular services which communicate through lightweight protocols, with each executing a single task [17]. Together, these services compose the application requirements [19, 7]. Naturally, containerization is a good fit to run microservices as it provides a lightweight virtualization platform [16]. Unlike traditional virtualization through hypervisors, containers encapsulate an application and its dependencies in container images while sharing the host OS [21, 12]. This design significantly reduces overhead, allowing faster startup times, improved scalability and more efficient resource utilization [7].

Despite the advancements made in container technologies and orchestrators there still remains areas for optimization in container placement and task scheduling in cloud environments [6]. How containers are allocated between physical nodes affects system performance, reliability, and scalability [7]. Moreover, the automatic management of containers influence application performance. Elasticity in the context of VM environments has been well studied [4], however, only recently research related to optimizing containerized applications is being published more regularly.

As cloud service demand continues to grow, it is imperative to explore

and evaluate new optimization strategies. Providing cloud services is a costly business and providing users with compute resources as effectively as possible is important for businesses and users. In this paper, we explore and evaluate various optimization strategies for container-based task scheduling in cloud environments. Our analysis focuses on the strengths and weaknesses of these strategies, considering key evaluation metrics such as optimization objectives, ranging from hardware-level resource utilization and energy efficiency to QoS metrics and service level agreement (SLA) assurance.

The structure of this paper is as follows: Section 2 provides background information on resource management, followed by an overview of container scheduling principles and methods in Section 3. Section 4 evaluates the optimization methods presented and discusses possible research directions. Finally, Section 5 concludes the paper.

2 Resource Management

Resource management optimization is a fundamental research challenge in cloud computing. The goal of resource management is to efficiently allocate and schedule computing resources to meet end-user requirements while maximizing system performance and minimizing costs [2]. Cloud service providers offer scalable compute resources, often described as scalable from zero to near infinite resources, but effective resource management is crucial ensuring optimal utilization of these resources [6]. Efficient resource management improves application performance, improves system throughput, reduces energy consumption, and reduces operational cost for both cloud providers and consumers [8].

Research efforts in recent years have focused on container placement in cloud environments. These optimizations aims to improve resource utilization, reduce energy consumption and ensure high availability and performance for applications deployed as microservices in containerized environments. Zeng et al. [19] categorize optimization efforts for containerized applications into three primary layers illustrated by figure 1. The application layer focuses on performance evaluation and monitoring of running applications, the service layer on resource provisioning, management and scaling of services, and finally the infrastructure layer on system tuning and coordination of the underlying cloud infrastructure.

In this study, we specifically analyze and focus on the service layer,

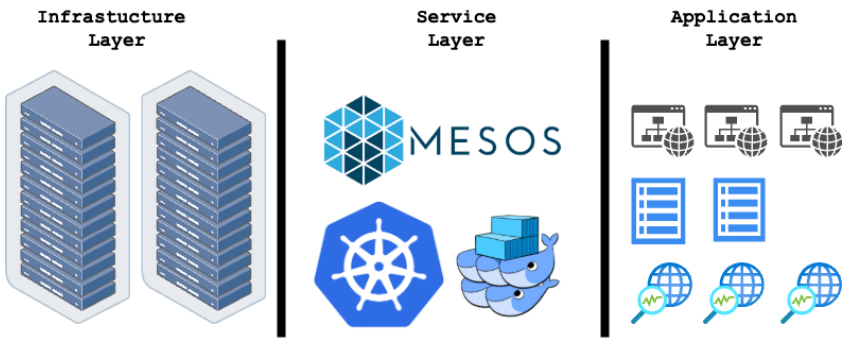

Figure 1. Optimization categories for containerized applications illustration

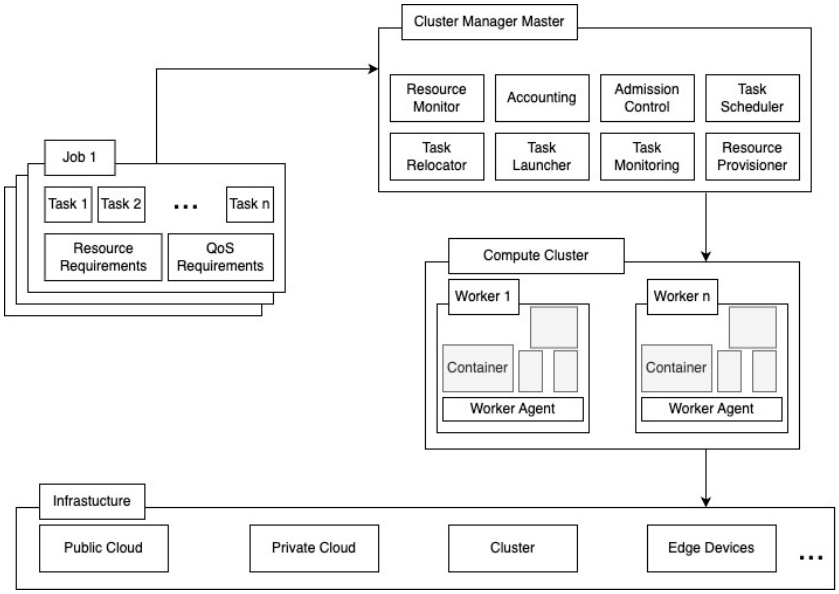

Figure 2. Container orchestration architecture reference [13]

with the assumption that microservices are primarily deployed within containerized environments. Next, we discuss the fundamentals related to container optimization. This includes concepts in container scheduling, scheduling architecture, and current research efforts made in the context of container optimization in the service layer.

3 Container Scheduling

Container Scheduling refers to the process of allocating containerized workloads to the most suitable computing resources within a distributed cluster [1]. It determines not only when container jobs should be executed, but in addition where they should be placed across the available infrastructure [5]. This process is crucial to ensure workloads have access to the necessary CPU, memory, storage, and networking resources while minimizing conflicts and optimizing overall system efficiency [12].

Typically, containerized applications are managed using container orchestration platforms such as Kubernetes, Apache Mesos, and Docker Swarm [1, 6, 5]. Figure 2 demonstrates a typical architecture for container orchestrators. These platforms provide an abstraction layer that simplifies cluster management by handling container scheduling, lifecycle management, and automated deployments, including scalability [6]. A core component of these orchestrators is the container scheduler. Schedulers are often modular, pluggable components within orchestrators [1], allowing flexibility in how workloads are assigned to computing nodes.

3.1 Container Scheduling Architecture

Container orchestrators come with their own default schedulers, each implementing different policies and architectures. Rodriguez et al. [13] characterize scheduling architectures to fall under the following three architectures: centralized, decentralized and two-level. Table 1 highlights the advantages and challenges of each scheduling architecture.

Centralized architecture encompasses a single scheduler for making scheduling decisions [13]. The scheduler is monolithic meaning that all scheduling decisions are made in a single code base [13]. The Kubernetes and Docker Swarm schedulers are examples which use monolithic scheduling [5].

In decentralized architecture the schedulers are distributed replicas that can be monolithic or modular. Scheduling request are be partitioned between the schedulers and manage state. State management can be handled in two ways, implementing shared state or having each scheduler have a partial view of the state [5].

Lastly, two-level architecture, as the name suggest, resource management and application framework are decoupled [13]. The two distinct layers handle the following [13, 5]: The bottom layer manages the low level resources in a computing cluster. Depending on design the bottom layer can operate in either offer or request based manner. Once the bottom layer proves or grants resources, the application framework makes placement decisions. They determine which task should run on which resource, based on their own scheduling logic. Apache Meos is an example framework which uses two-level scheduling architecture [5].

Building on scheduling architecture, in the next section, we review existing research efforts made for container optimization in the context of scheduling in cloud environments. We focus on two major categories: deep

Architecture	Advantages	Challenges
Centralized	Global view of system	Single point of failure
Decentralized	Scalable, monolithic or modular	Complexity, potential inefficiencies
Two-level	Flexibility, scalability	Complexity

Table 1. Comparison of Scheduling Architectures [5, 13]

learning-based methods (which include various machine learning models) and traditional optimization techniques such as meta-heuristics and mathematical programming. These techniques can be utilized to make the actual container scheduling decisions i.e. where containers should be placed in cluster, which containers are given run-time resources, and when they are given.

3.2 Deep Learning Methods

Machine learning methods are effective at dynamically adjusting and identifying patterns, which makes reinforcement learning methods effective for container scheduling in cloud environments with diverse work loads. Lorido-Botran and Bhatti [9] leveraged a decentralized actor-critic multi-agent deep reinforcement learning model with action shaping and proposed RLShed. Compared to other model is able to converge and place more containers per session compared to other deep reinforcement learning models. Similarly, Nagarajan et al [11] used deep multi-agent reinforcement learning to optimize resource allocation for greater energy efficiency in data-centers and circumvent Service Level Contract (SLA) damages.

Muniswamy et al. [10] proposed a hybrid optimal deep learning method for dynamic task scheduling (DSTS). In their work, they utilize modified multi-swarm coyote optimization for scaling virtual container resources enhancing SLA. Additionally, a modified pigeon-inspired optimization is utilized for task clustering, and the fast adaptive feedback recurrent neural network is used for pre-virtual CPU allocation. Finally, a deep convolution neural network is used for load monitoring to achieve priority based dynamic scheduling. Numeric result show that the proposed DSTS was able to outperform existing task scheduling models in terms of quality of service (QoS) metrics.

Wang et al [18] explored using hierarchical reinforcement learning techniques to learn optimal placement of long running container applications. They used policy gradient training routine to train their Mentis

model. Their presented model offers quantitative scheduling guidelines and show considerable improvement compared to constraint based schedulers. However, the scheduler still suffers from scalability issues for performance critical applications.

3.3 Meta-heuristic Methods

The Non-Dominated Sorting Genetic Algorithm II (NSGA-II) has been a somewhat popular base for cloud container optimizations. For example, Guerrero et al. [7] proposed using NSGA-II for container allocation and automatic elasticity management, i.e. scaling the microservices up and down based on demand. Their proposed algorithm enhances system provisioning, system performance, system failure and network overhead [7]. Similarly, Shi et al. [14] tackled the container consolidation problem as a single multi-objective optimization problem using NSGA-II to search for solutions. Specifically, the objective of Shi et al. [14] was to minimize total energy consumption and number of container migrations over a period of time.

Al-Malmi et al. [2] address the container and VM placement problem to minimize power consumption in data centers and maximize resource utilization. They present a whale optimization system, a nature-inspired meta-heuristic optimization technique, that optimizes initial container and VM placement that considers container and VM placement as a single optimization problem.

Another popular optimization algorithm for scheduling problems is the Ant colony algorithm, which is also a meta-heuristic approach. Lin et al. [8] applied the ant colony optimization approach to container scheduling. They present a multi-objective container scheduling model, which considers physical machine resource constrains, network overhead, system load and service reliability.

3.4 Mathematical Methods

Integer linear programming methods have also been explored to optimize container placement and job scheduling. Zhang et al. [20] considered job scheduling in terms of containerized workload dispatching, aiming to reduce image transition costs from image registry to container host and energy costs, with success. Furthermore, Boukadi et al. [3] consider the same problem and presenting a algorithm combining first-fit and linear

programming technique.

Zhou et. al [21] took a practical approach to online container placement. They explicitly took the perspective of a cloud service provider. Zhou et. al formulated the container placement problem as a quadratic integer program (IP) that capture inter-container traffic and reformulated it into a compact exponential integer-linear program (ILP) to improve computational feasibility. Furthermore, they designed an online primal-dual framework that dynamically adjusts resource prices and evaluates request values in real-time to make efficient placement decisions.

4 Comparison of Optimization Methods

Optimizing container scheduling is essential for both cloud service providers and users due to its direct impact on cost efficiency, resource utilization, and overall system performance [1, 2, 3, 8]. Most cloud platforms operate on a pay-as-you-go model, meaning users are charged based on the computing resources they consume [2, 19]. From the provider's perspective, inefficient scheduling can lead to resource under-utilization, increasing operational costs and reducing profitability. Non-efficient scheduling can lead to resource fragmentation, excessive provisioning, or service bottlenecks, impacting both costs and performance. For users, optimal scheduling ensures their workloads run efficiently, reducing expenses while maintaining performance, availability, and reliability.

In the following subsections, we analyze the presented optimization techniques from key perspectives. Given the diverse nature of workloads and tasks executed in cloud environments, there are several metrics that should be considered when evaluating container scheduling. It is important to recognize that achieving an optimal balance among these factors is crucial, as it is typically impractical to maximize all objectives simultaneously.

4.1 Method-Specific Evaluation

Deep reinforcement and hierarchical learning have the ability to explore a wide range of solutions for multi-objective problems. These methods leverage experience-based learning to dynamically adapt to and learn how the system reacts to decisions the RL agent makes [11]. Furthermore, deep learning-based methods are able to capture and learn more complex pat-

terns that emerge in high-dimensional state and action spaces, allowing the RL agent to generalize better across different scenarios. This capability is particularly beneficial for multi-objective optimization problems, where trade-offs between competing objectives must be dynamically balanced [15, 11].

In the studies conducted by [11, 9, 10, 18] utilizing forms of deep reinforcement learning, all were able to outperform existing methods in their respective optimization metric. The primary metric used in these studies was energy consumption and resource utilization. Additionally, some other metrics, were used such as SLA in the study by Nagarajan et al [11] and QoS metrics in the study by Muniswamy et al [10].

Meta-heuristic methods are well-suited for multi-objective optimization problems across various fields, including container scheduling [1]. Across the studies [2, 7, 8], all proposed meta-heuristic methods demonstrated improved performance in their respective optimization areas. This highlights their effectiveness in container scheduling for cloud environments, as they can efficiently optimize multiple competing objectives. However, a key drawback of meta-heuristic methods is their inherent computational complexity when dealing with large problems, requiring additional hardware resources to remain viable [1].

ILP-based approaches offer precise solutions under clearly defined constraints, but their scalability is an issue in large-scale, dynamic cloud environments. Despite ILP theoretical rigor, their models are inherently NP-hard [1] and computationally expensive to compute. To address this computational complexity ILPs are used in conjunction with other techniques. For example, the proposed container scheduler by Zhou et al. [21] utilized the primal-dual technique to get an approximate result for the scheduling problem to achieve computational efficiency.

4.2 Research Opportunities and Challenges

Both meta-heuristic and deep RL methods demonstrate potential for container scheduling, as they are capable of finding near-optimal or improved solutions compared to traditional scheduling policies like bin-packing, random allocation, and spread strategies [1]. A key advantage of these methods is their ability to consider multiple optimization metrics, that may be in competition against one another. Currently infrastructure level metrics are given more attention. Schedulers could utilize application level metrics such as QoS with more weight [12]. Exploring varying combinations

of optimization metrics with different RL and meta-heuristic techniques for container scheduling still offers a variety of research opportunities to find competitive combinations.

While ILP-based approaches offer the advantage of providing mathematically optimal solutions, their computational complexity makes them less suitable for real-time decision-making in dynamic cloud environments. However, ILP methods can complement meta-heuristic and deep RL approaches by providing offline optimal solutions, which can be leveraged as guidance for more efficient real-time scheduling decisions as demonstrated by Zhou et al [21].

A common challenge across all these methods — whether deep RL, meta-heuristic, or ILP-based — is scalability. As cloud environments grow in size and complexity, ensuring that these approaches remain computationally viable is a critical research direction. While deep learning-based methods and meta-heuristics have shown promising results, they are still susceptible to getting trapped in local optima. For deep RL, leveraging high-quality real-world data traces can help improve learning efficiency and generalization across different workload patterns [12].

5 Conclusion

This paper discussed the importance of resource management in container scheduling for cloud environments. The popularity of containerization is largely due to its lightweight virtualization, which aligns well with the microservices architecture. In addition, we explore scheduling architectures, optimization approaches, and key elements of container-based scheduling.

The techniques reviewed effectively optimized container placement and task scheduling, with energy efficiency and resource utilization, particularly CPU and memory, being the most frequently addressed metrics. Other factors like SLA assurance, network transmission overhead, and QoS were considered but with less emphasis. A major challenge across all approaches was scalability, a crucial factor for the future of container-based cloud systems. Without scalable solutions, the benefits of containerization risk being undermined, highlighting the need for further research and innovation.

In conclusion, innovative multi-objective techniques show promising results for effective and efficient resource management in container-based

cloud environments. Furthermore, container scheduling in cloud environments is a complex endeavor. Workloads range from short-lived to long-running services. This diversity makes it challenging to develop a universal scheduling technique that fits all scenarios. Instead, scheduling policies should be adaptive and tailored to the specific characteristics of the environment, workload patterns, and optimization objectives.

References

[1] Imtiaz Ahmad, Mohammad Gh. AlFailakawi, Asayel AlMutawa, and Latifa Alsalman. Container scheduling techniques: A survey and assessment. *Journal of King Saud University - Computer and Information Sciences*, 34(7):3934 – 3947, 2022. doi: 10.1016/j.jksuci.2021.03.002.

[2] Ammar Al-Moalmi, Juan Luo, Ahmad Salah, Kenli Li, and Luxiu Yin. A whale optimization system for energy-efficient container placement in data centers. *Expert Systems with Applications*, 164:113719, 2021. doi: https://doi.org/10.1016/j.eswa.2020.113719.

[3] Khouloud Boukadi, Rima Grati, Molka Rekik, and Hanêne Ben Abdallah. From vm to container: A linear program for outsourcing a business process to cloud containers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10573 LNCS:488 – 504, 2017. doi: 10.1007/978-3-319-69462-7_31.

[4] Blanquer I. Moltó G. et al. Caballer, M. Dynamic management of virtual infrastructures. *J Grid Computing*, 2015. doi: https://doi.org/10.1007/s10723-014-9296-5.

[5] Carmen Carrión. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Computing Surveys*, 55(7), 2023. doi: 10.1145/3539606.

[6] Neelima Gogineni and Saravanan Madderi Sivalingam. A systematic review on recent methods of scheduling and load balancing for containers in distributed environments. *International Journal of Advanced Technology and Engineering Exploration*, 11(116):1030 – 1048, 2024. doi: 10.19101/IJATEE.2023.10102431.

[7] Carlos Guerrero, Isaac Lera, and Carlos Juiz. Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. *Journal of Grid Computing*, 16(1):113 – 135, 2017. doi: 10.1007/s10723-017-9419-x.

[8] Miao Lin, Jianqing Xi, Weihua Bai, and Jiayin Wu. Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE Access*, 7:83088 – 83100, 2019. doi: 10.1109/ACCESS.2019.2924414.

[9] Tania Lorido-Botran and Muhammad Khurram Bhatti. Adaptive container scheduling in cloud data centers: A deep reinforcement learning approach. *Lecture Notes in Networks and Systems*, 227:572 – 581, 2021. doi: 10.1007/978-3-030-75078-7_57.

[10] Saravanan Muniswamy and Radhakrishnan Vignesh. Dsts: A hybrid optimal and deep learning for dynamic scalable task scheduling on container cloud environment. *Journal of Cloud Computing*, 11(1), 2022. doi: 10.1186/s13677-022-00304-7.

[11] S. Nagarajan, P. Shobha Rani, M.S. Vinmathi, V. Subba Reddy, Angel Latha Mary Saleth, and D. Abdus Subhahan. Multi agent deep reinforcement learning for resource allocation in container-based clouds environments. *Expert Systems*, 42(1), 2025. doi: 10.1111/exsy.13362.

[12] Anil Prajapati. Container scheduling: A taxonomy, open issues and future directions for scheduling of containerized microservices in cloud environments. *International Journal of Intelligent Systems and Applications in Engineering*, 12(22s):1108–1122, Jul. 2024. doi: http://dx.doi.org/10.2139/ssrn.4699558.

[13] Maria A. Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Software - Practice and Experience*, 49(5):698 – 719, 2019. doi: 10.1002/spe.2660.

[14] Tao Shi, Hui Ma, and Gang Chen. Multi-objective container consolidation in cloud data centers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11320 LNAI:783 – 795, 2018. doi: 10.1007/978-3-030-03991-2_71.

[15] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. doi: https://doi.org/10.1017/S0263574799271172.

[16] Ye Tao, Xiaodong Wang, Xiaowei Xu, and Yinong Chen. Dynamic resource allocation algorithm for container-based service computing. page 61 – 67, 2017. doi: 10.1109/ISADS.2017.20.

[17] Johannes Thönes. Microservices. *IEEE Software*, 32(1):116–116, 2015.

[18] Luping Wang, Qizhen Weng, Wei Wang, Chen Chen, and Bo Li. Metis: Learning to schedule long-running applications in shared container clusters at scale. volume 2020-November, 2020. doi: 10.1109/SC41405.2020.00072.

[19] Hou X. Zhang L. et al. Zeng, R. Performance optimization for cloud computing systems in the microservice era: state-of-the-art and research opportunities. *Springer Nature Link*, 2022. doi: https://doi.org/10.1007/s11704-020-0072-3.

[20] Dong Zhang, Bing-Heng Yan, Zhen Feng, Chi Zhang, and Yu-Xin Wang. Container oriented job scheduling using linear programming model. In *2017 3rd International Conference on Information Management (ICIM)*, pages 174–180, 2017. doi: 10.1109/INFOMAN.2017.7950370.

[21] Ruiting Zhou, Zongpeng Li, and Chuan Wu. An efficient online placement scheme for cloud container clusters. *IEEE Journal on Selected Areas in Communications*, 37(5):1046 – 1058, 2019. doi: 10.1109/JSAC.2019.2906745.