Abstract

The evolution of distributed systems has significantly reshaped how modern applications are deployed and managed, with containerization and orchestration technologies at the forefront of this change. Kubernetes stands out as the leading platform for container orchestration and is well-suited for handling complex application clusters. However, its default scheduling mechanism primarily focuses on basic resource metrics like CPU and memory, lacking a comprehensive understanding of the broader context in which applications operate. This literature review examines the limitations of Kubernetes' scheduler in complex and dynamic environments. It explores advanced strategies that enhance scheduling decisions by incorporating network-aware and hardware-aware approaches. This review highlights significant improvements in application performance, resource efficiency, and system reliability by analyzing recent research proposals that integrate real-time network conditions and optimize hardware resource usage—particularly GPUs. The findings underscore the importance of adopting context-aware scheduling in Kubernetes to meet the demands of modern applications across diverse computing environments.

I. Introduction

The rise of distributed systems has fundamentally transformed how modern applications are built and deployed. Containerization lies at the heart of this transformation, enabling services to modularize themselves into small, independent services known as containers. As applications scale, it is critical for operators to be able to efficiently manage hundreds or thousands of concurrent container instances. As a result, a wide array of orchestration technologies have emerged in recent years, This review conducts a deep dive into Kubernetes, the leading industry standard for container orchestration [11]. It examines shortcomings within the Kubernetes scheduling mechanism due to context limitations and explores cutting-edge solutions to combat them.

II. Background and Context

A. Containerization and Modern Software Architecture

The advent of containerization initiated a transformative approach to managing and deploying applications in distributed computing systems. A distributed system can be visualized as a city, with various services like transportation, power grids, and communication networks operating independently while maintaining varying levels of interconnectivity. In this analogy, containerization acts like standardized shipping containers. Much like how shipping containers are constructed to be transported efficiently across different modes of transportation, software containers encapsulate individual services or applications to provide a consistent and portable environment. This "all-in-one" structure simplifies application deployment across different systems by eliminating compatibility issues [11].

Furthermore, containers share a single OS kernel, unlike traditional virtual machines (VMs), which each require a complete operating system. This design reduces their size and start-up time, making them exceptionally lightweight and resource-efficient [10]. This efficiency makes containers highly suited for cloud and edge computing environments, where applications must scale rapidly and consistently with minimal overhead. As a result, containerization has become a crucial component in managing large-scale applications that demand high responsiveness, enabling faster replication and deployment of applications across diverse infrastructures [10].

B. The Role of Container Orchestration

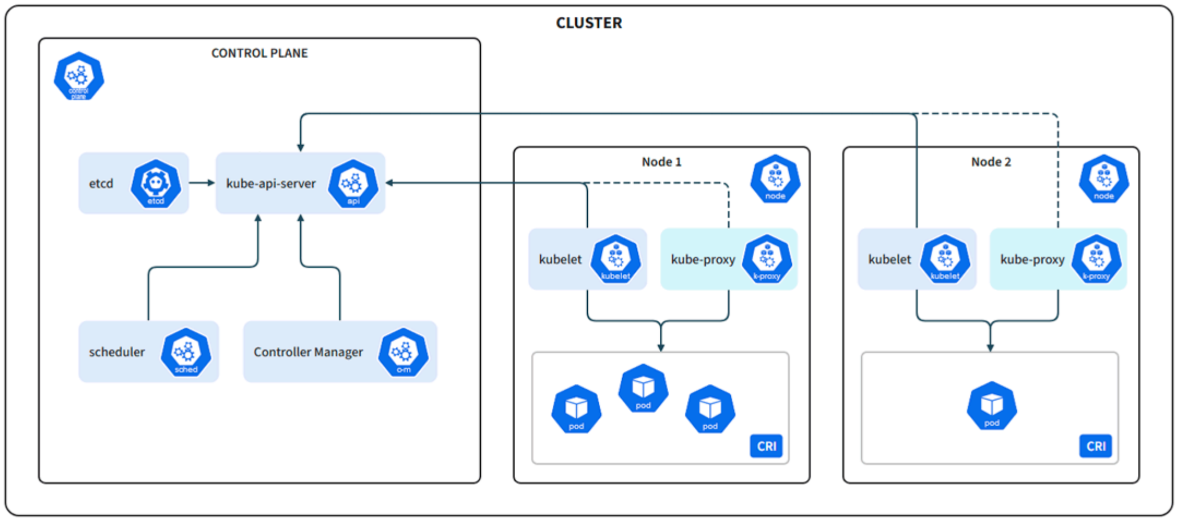
Container orchestration tools like Kubernetes offer an automated approach to managing the lifecycle of the aforementioned containers within distributed systems. Similarly to how an orchestra's conductor ensures all musicians play in harmony, orchestration tools manage containers' deployment, scaling, and operation across multiple servers, managing them as a cohesive unit. This coordination is essential in modern distributed systems, where applications may span hundreds or thousands of containers that must work together seamlessly. [11]

III. Kubernetes Overview

A. Kubernetes as the Industry Standard

Kubernetes, a now open-source tool originally developed by Google, sources its popularity from its robust scalability, flexibility, and adaptability across diverse environments. Unlike simpler tools like Docker Swarm, Kubernetes is designed to manage large, complex clusters with automated scheduling, resource allocation, and dynamic scaling, which are critical for microservices-based applications [11]. Its modular architecture allows tailored configurations and integrations with plugins, making it versatile for various workloads. Additionally, Kubernetes' strong backing from major cloud providers, including Google, AWS, and Azure, has driven widespread adoption and continuous innovation through its active open-source community. This combination of scalability, customization, and extensive support enables Kubernetes to meet the demands of modern distributed systems more effectively than competing orchestration tools [11].

B. Kubernetes Architecture Overview



Kubernetes Architecture Diagram [9]

Control Plane Components:

- **API Server:** handles all incoming requests and directs them appropriately within the system.
- **Scheduler:** decides which worker should handle a new task based on current workloads and resources.
- **Controller Manager:** monitors the system's state and adjusts to keep everything running smoothly.
- **etcd:** the system's memory, securely storing all the critical data and configurations.

Worker Nodes:

- **Pods:** represents container(s)[1] that share the same network and storage. Placed on nodes.
- **Nodes:** the actual machines (physical or virtual) where applications run.
- **Kubelet:** functions as a supervisor on each node, ensuring tasks are performed correctly.
- **Kube-proxy:** manages communication externally and within the node.

[1] Pods usually contain a single container, but can sometimes be grouped due to tight coupling within services.

IV. Kubernetes Scheduler

A. Overview

The scheduler is one of Kubernetes's most important components, deciding which compute resources should handle incoming tasks and workloads. The default scheduler operates through a two-step process: filtering and scoring. During the filtering phase, it identifies nodes that meet the resource requests and constraints specified by the pods. This involves checking for sufficient CPU, memory, and other resource requirements or node selectors. In the scoring phase, the scheduler assigns scores to the eligible nodes based on various criteria, such as resource availability and affinity rules, to select the most suitable node for the pod [8].

B. Existing Challenges

While the default scheduling algorithm in Kubernetes is adequate for straightforward resource allocation, it falls short in complex and heterogeneous environments, including edge computing and Internet of Things (IoT) applications. This is due to the default algorithm's limited scope of context awareness and primary concentration on isolated resource metrics such as CPU and memory availability on individual nodes. The scheduler's narrow focus overlooks critical factors like network topology, data locality, inter-pod dependencies, and real-time performance metrics, which are vital for optimizing performance in distributed systems [2]. As a result of this basic scheduling model, workloads may not always be optimally placed, leading to potential inefficiencies such as elevated latency and reduced application performance, particularly for data-intensive or latency-sensitive applications.

V. Proposed Strategies

A. Network-Aware Scheduling

Network-aware scheduling is essential for optimizing the performance and reliability of distributed systems, particularly in environments with dynamic and heterogeneous network conditions. By considering the underlying network topology and real-time network states, network-aware scheduling enhances Quality of Service (QoS), reduces latency, and ensures efficient utilization of network resources. This approach is increasingly important in applications such as smart cities, edge computing, and 5G networks, where the demand for low-latency and high-throughput services is essential.

[4] Santos et al. address resource scheduling challenges in Smart City deployments by proposing a network-aware scheduling approach tailored for container-based applications. This enhancement allows Kubernetes to make more informed resource provisioning decisions based on real-time network conditions. The authors validated their approach using container-based applications typical of Smart Cities and demonstrated an 80% reduction in network latency compared to the standard Kubernetes scheduler.

[6] Ogbuachi et al. focus on improving Kubernetes scheduling for edge computing applications by incorporating physical, operational, and network parameters alongside software states. This enhanced scheduler dynamically orchestrates and manages applications by leveraging real-time information about edge devices, thereby improving fault tolerance and adaptability to environmental changes. Through comparative analysis with the default Kubernetes scheduler, the proposed design exhibited superior fault tolerance and more effective dynamic orchestration, addressing the limitations of traditional scheduling methods in rapidly changing edge environments.

[7] Wojciechowski et al. introduce NetMARKS, a novel Kubernetes pod scheduling method that utilizes dynamic network metrics collected via the Istio Service Mesh. This approach optimizes scheduling decisions by automating the reduction of inter-node bandwidth usage by up to 50% and decreasing application response times by as much as 37%. NetMARKS ensures backward compatibility with existing Kubernetes setups and is particularly beneficial for 5G use cases, where high throughput and low latency are critical for multi-access edge computing and machine-to-machine communication applications.

By integrating these advanced network-aware scheduling strategies, Kubernetes can significantly enhance its performance and reliability across diverse applications. These improvements are crucial for meeting the stringent QoS requirements of modern technological environments, ensuring that distributed systems operate efficiently and responsively.

B. Hardware-Aware Scheduling

Hardware-aware scheduling is critical for maximizing the performance and efficiency of distributed systems by intelligently managing and allocating hardware resources such as GPUs. This approach ensures optimal utilization of computational resources, reduces energy consumption, and enhances the execution speed of resource-intensive applications, particularly in environments like high-performance computing (HPC) and machine learning [13][14]. By aligning task scheduling with the underlying hardware capabilities, hardware-aware scheduling addresses the unique demands of modern applications, leading to significant improvements in system performance and resource efficiency.

[14] Thinakaran et al. confront the challenge of GPU orchestration in contemporary data centers by developing Knots, a GPU-aware resource orchestration layer that forms Kube-Knots when

integrated with Kubernetes. This enhanced scheduler dynamically provisions GPU resources by application demands, effectively mitigating resource fragmentation and interference. By employing two scheduling techniques—Correlation-Based Prediction (CBP) and Peak Prediction (PP)—Kube-Knots significantly enhances GPU utilization by up to 80% for high-performance computing (HPC) workloads and reduces average job completion times for deep learning tasks by up to 36% compared to existing schedulers. Furthermore, Kube-Knots achieves an average of 33% energy savings across diverse workloads and ensures end-to-end Quality of Service (QoS) by reducing QoS violations by up to 53% for latency-critical queries.

[13] Song et al. introduce GaiaGPU, a topology-based GPU scheduling framework that extends the conventional Kubernetes GPU scheduling algorithm. In prevailing scheduling methods, GPUs are allocated as indivisible units, leading to inefficient resource utilization and uneven load distribution across clusters. GaiaGPU addresses these shortcomings by reconstructing the GPU cluster topology into a resource access cost tree, facilitating dynamic scheduling and adjustment of GPU resources based on varying application scenarios. This approach optimizes the scheduling process, enhancing load balancing and resource utilization. Experimental evaluations demonstrate that GaiaGPU improves load balance and increases GPU cluster resource utilization by approximately 10% in Tencent's production environment.

[12] Liu et al. propose KubFBS, an advanced scheduling system that optimizes deep learning (DL) workloads within GPU-accelerated environments. Traditional GPU scheduling algorithms typically focus solely on the number of GPUs or their memory capacity, often leading to diminished performance for DL tasks. Moreover, existing balance-aware scheduling strategies assume that once a DL task commences, it continuously consumes resources, thereby failing to alleviate resource contention and limiting overall execution efficiency. To address these challenges, the authors introduce a Fine-grained and Balance-aware Scheduling Model (FBSM) that meticulously considers the specific resource consumption characteristics of DL tasks. Building upon FBSM, they develop two specialized modules: GPU-Sniffer (GPU-S) and Balance-Aware Scheduler (BAS), collectively forming the KubFBS system. Experimental results demonstrate that KubFBS significantly accelerates the execution of DL tasks while also enhancing the load-balancing capabilities of the GPU cluster.

Kubernetes can significantly improve resource utilization, energy efficiency, and application performance by incorporating hardware-aware scheduling techniques. These advancements are essential for meeting the rigorous demands of modern computational workloads and ensuring that distributed systems operate at peak efficiency and effectiveness.

VI. Conclusion

Overview

While effective for basic resource allocation, Kubernetes' default scheduler is inadequate for optimizing performance in complex distributed systems due to its limited context awareness. It neglects critical factors such as network conditions, data locality, and specialized hardware capabilities, resulting in inefficient workload placement, increased latency, and underutilized resources. The advanced scheduling strategies examined—network-aware and hardware-aware scheduling—address these limitations by incorporating a broader spectrum of considerations into the scheduling process.

Future Research Directions

Research within cutting-edge orchestration tools is integral as modern-day distributed systems continue to scale. To further advance the field, additional research is needed in several key areas:

1. **Integration of Machine Learning Techniques:** Exploring the use of machine learning and artificial intelligence to predict workload patterns, network conditions, and resource demands can enable proactive and more efficient scheduling decisions. Predictive models could help the scheduler anticipate changes and adjust allocations dynamically to optimize performance.
2. **Security and Privacy Considerations:** As schedulers become more context-aware, they may require access to sensitive system information. Research into securing this data and protecting against potential vulnerabilities is critical to ensure that enhanced scheduling does not compromise system security or data privacy.
3. **Scalability and Overhead Assessment:** It is important to investigate the scalability of advanced scheduling strategies and their impact on system overhead. Studies should quantify the trade-offs between the benefits of context-aware scheduling and the computational resources required to implement them, especially in large-scale deployments.
4. **Cross-Layer Optimization:** Research into cross-layer optimization that considers interactions between the application, network, and hardware layers could lead to more holistic scheduling solutions. Understanding how decisions at one layer affect others can help design schedulers that optimize overall system performance.

Closing Remarks

In conclusion, integrating context-aware scheduling approaches into Kubernetes is imperative to maximize its potential in diverse and demanding computing environments. By extending the scheduler's capabilities to include considerations of network conditions and hardware resources, Kubernetes can better support the evolving needs of modern distributed applications. Continued research and development in this area are essential to ensure that Kubernetes remains adaptable and efficient in the face of growing complexity in distributed systems.

VII. References

[1] Khaldoun Senjab, S. Abbas, and N. Ahmed, "A survey of Kubernetes scheduling algorithms," *Journal of Cloud Computing*, vol. 12, no. 1, Jun. 2023, doi: https://doi.org/10.1186/s13677-023-00471-1.

[2] C. Carrión, "Kubernetes Scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 0360–0300, Jun. 2022, doi: https://doi.org/10.1145/3539606.

[3] I. Čilić, P. Krivić, I. Podnar Žarko, and M. Kušek, "Performance Evaluation of Container Orchestration Tools in Edge Computing Environments," *Sensors*, vol. 23, no. 8, p. 4008, Apr. 2023, doi: https://doi.org/10.3390/s23084008.

[4] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications," *IEEE Xplore*, Jun. 01, 2019. https://ieeexplore.ieee.org/document/8806671 (accessed Jul. 26, 2021).

[5] S.-H. Kim and T. Kim, "Local Scheduling in KubeEdge-Based Edge Computing Environment," *Sensors*, vol. 23, no. 3, p. 1522, Jan. 2023, doi: https://doi.org/10.3390/s23031522.

[6] Michael Chima Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovacs, "Context-Aware K8S Scheduler for Real-Time Distributed 5G Edge Computing Applications," Sep. 2019, doi: https://doi.org/10.23919/softcom.2019.8903766.

[7] Ł. Wojciechowski *et al.*, "NetMARKS: Network Metrics-AwaRe Kubernetes Scheduler Powered by Service Mesh," *IEEE Xplore*, May 01, 2021. https://ieeexplore.ieee.org/abstract/document/9488670?casa_token=UY_A83IjKmUAAAAA:yWAczGU4lv5Ug2TNRZRP9xXCi12HXoMJsY-VP0phVuhjV_5t8pkQoXjyQU-a6tnPpyxoPhFB6fLW (accessed Jan. 14, 2023).

[8] "Kubernetes Documentation," *Kubernetes.io*, 2019. https://kubernetes.io/docs/home/

[9] *Moqups.com*, 2024.
https://landing.moqups.com/img/templates/diagrams/network/kubernetes-cluster-architecture.png
(accessed Nov. 04, 2024).

[10] J. Watada, A. Roy, R. Kadikar, H. Pham, and B. Xu, “Emerging Trends, Techniques and Open Issues of Containerization: A Review,” *IEEE Access*, vol. 7, no. 152443–152472, pp. 152443–152472, 2019, doi: https://doi.org/10.1109/access.2019.2945930.

[11] E. Casalicchio, “Container Orchestration: A Survey,” *Systems Modeling: Methodologies and Tools*, pp. 221–235, Oct. 2018, doi: https://doi.org/10.1007/978-3-319-92378-9_14.

[12] Z. Liu, C. Chen, J. Li, Y. Cheng, Y. Kou, and D. Zhang, “KubFBS: A fine‐grained and balance‐aware scheduling system for deep learning tasks based on Kubernetes,” *Concurrency and Computation Practice and Experience*, vol. 34, no. 11, Jan. 2022, doi: https://doi.org/10.1002/cpe.6836.

[13] S. Song, L. Deng, J. Gong, and H. Luo, “Gaia Scheduler: A Kubernetes-Based Scheduler Framework,” Dec. 2018, doi: https://doi.org/10.1109/bdcloud.2018.00048.

[14] Prashanth Thinakaran, Jashwant Raj Gunasekaran, B. Sharma, Mahmut Taylan Kandemir, and C. R. Das, “Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters,” Sep. 2019, doi: https://doi.org/10.1109/cluster.2019.8891040.