Resource Management Frameworks in Modern Datacenters

Pengyu Wang
Yale University

1 Introduction

The evolution of modern data centers reflects the dramatic shifts in both computational demands and technological supply. Data centers today serve a diverse range of applications, from latency-sensitive online services to throughput-heavy batch workloads. These applications exhibit dynamic resource requirements, demanding elasticity, heterogeneity, and stringent performance guarantees such as low latency and high throughput. Addressing these challenges has necessitated the development of efficient resource management frameworks capable of handling the increasing complexity and scale of modern data centers.

The advent of Virtual Machines (VMs) marked a critical milestone in resource management by abstracting physical hardware into isolated, flexible units. VMs enabled multi-tenancy, improved security through isolation, and enhanced hardware utilization. However, as applications became more diverse and real-time demands increased the heavyweight nature of VMs introduced inefficiencies. This led to the widespread adoption of containerization technologies, such as Docker and Kubernetes, which offer lightweight, portable, and highly scalable runtime environments, further optimizing resource sharing and orchestration.

To efficiently allocate resources at scale, early cluster management systems such as Google's Borg [10], Apache Mesos [3], and Hadoop YARN [5] pioneered fundamental principles in resource scheduling and orchestration. These systems tackled resource fragmentation, heterogeneous workload scheduling, and introduced fine-grained resource sharing strategies. Later, systems like Omega [6] and Kubernetes [1] decoupled scheduling from resource management, enabling greater scalability, improved fault tolerance, and support for diverse applications. Recently, unified resource management frameworks like Twine [7] and Alibaba's Optum [4] have emerged, aiming to co-locate diverse workloads (batch and latency-sensitive jobs) within a single infrastructure while enforcing global optimization strategies.

This literature review critically examines the state-of-the-art in data center resource management, focusing on frameworks that have driven innovation in virtual machine orchestration and containerized scheduling. Specifically, we analyze seminal systems including Borg, Mesos, YARN, Omega, Kubernetes, Twine, and modern unified schedulers, addressing the problems they were designed to solve, their architectural choices, and their limitations. We further discuss their impact on hardware design, software ecosystems, and industry trends, and highlight how subsequent innovations have built upon these foundational works. Finally, we explore remaining challenges in the field, such as achieving unified resource optimization and accommodating emerging workloads.

2 Requirements of a Datacenter Resource Framework

Modern datacenters have evolved a lot, from the batch-job resource allocator in its early days, to the modern cluster management systems that operate on millions of machines. Therefore, the requirements of a resource management framework have evolved as well. Below are the list of requirements that are essential in modern-day resource frameworks.

- Scalability. Datacenter operators typically operate a cluster on the scale of thousands to millions of machines. Yahoo operated Hadoop with YARN on 2500 nodes in 2023 [9], and Meta orchestrates up to a million machines with Twine [7]. The scalability requirements of modern datacenters are only increasing, and the resource management frameworks should keep up with that.

- Multi-tenancy. As datacenters moved from dedicated bare-metals, to cluster partitioning, and now onto virtualization and containerization, the number of users simultaneously using the cluster has only increased. While VMs and containers abstract the applications away, resource frameworks should only expect that more users are sharing the same set of physical machines.

- Fault-tolerance. This requirement goes hand-in-hand with scalability. As the number of nodes goes up, the number of faults goes up as well. The framework should be able to tolerate all kinds of faults and hide them from the applications.

- Heterogeneity. Different applications require different sets of resources, with varying amounts. Tasks could be CPU-bound, GPU-bound, I/O-bound, or a mixture of them. Some tasks might have performance requirements like latency or throughput. Frameworks should take these into consideration when scheduling tasks for hardware.

3 Implementation Techniques

3.1 Hardware-oriented to Application-oriented

A significant trend in resource management frameworks is the decoupling of hardware concerns from software execution logic, thereby simplifying application development and improving scalability and fault tolerance. This decoupling creates an abstraction layer, allowing applications to operate under the illusion of infinite resources while the underlying frameworks transparently handle resource provisioning.

In Mesos [3], resource allocation is performed via "resource offers" made by the Mesos master to individual framework schedulers. These framework schedulers autonomously decide how to utilize the offered resources, enabling flexibility to support diverse application frameworks. However, Mesos remains primarily hardware-centric, as the resource offers consist of fixed bundles of hardware resources. This approach can result in a mismatch between application demands and resource allocation granularity, limiting overall efficiency.

YARN (Yet Another Resource Negotiator) [9] addressed these limitations by decoupling application execution logic from resource management, marking a major architectural advancement. Traditional Hadoop tightly coupled MapReduce with resource management, restricting flexibility for emerging workloads. YARN introduced a central Resource Manager (RM) to handle global resource management and ApplicationMasters (AM) to oversee task execution for individual jobs. This separation allows for heterogeneous workloads, such as MapReduce, Spark [11], and Tez [5], to coexist on a shared cluster. YARN orchestrates resource allocation at the task level, executing tasks within containers managed by the corresponding AM. This architecture increased resource utilization and enabled multi-application support but introduced challenges in adapting to new frameworks due to the additional integration effort required.

Building on YARN's progress, Borg further abstracts resource management by leveraging Linux's `cgroup` mechanism. Each application is allocated isolated namespaces, processes, and file systems, achieving strong security and performance isolation. Borg's open-source successor, Kubernetes [1], expanded this model by favoring lightweight Docker containers over direct `cgroup` management, further enhancing portability and ease of deployment. Similarly, frameworks like Twine [7] manage tasks at the container level, while other systems such as Protean [2] operate on the coarser abstraction of virtual machines (VMs).

From the 2010s to the 2020s, resource allocation frameworks have progressively shifted from exposing hardware details directly to applications to abstracting hardware away entirely. This abstraction, however, does not imply unlimited resource usage. As we will explore in the next section, frameworks continue to impose limits on hardware consumption, but these constraints are handled transparently, allowing applications to remain agnostic to their underlying hardware environments.

3.2 Monolithic to Shared-State Scheduling

An essential component of any resource management framework is the scheduler, which determines task placement and execution order while aiming to meet hardware requirements and achieve near-optimal resource allocation. In this section, we analyze how scheduler architectures have evolved to address the growing scale of modern data centers. While scheduling heuristics are crucial, their extensive breadth merits its independent discussion and is beyond the scope of this section.

Early systems employed monolithic schedulers, where a single central scheduler had complete visibility into the cluster state. This design enabled systems such as Borg to efficiently achieve globally optimal task placement [10]. For example, memory-intensive and CPU-intensive tasks could be colocated to ensure balanced resource utilization with minimal contention. Furthermore, monolithic schedulers could enforce cluster-wide policies for load balancing and resource efficiency. However, as cluster sizes increased, this centralized approach introduced significant limitations: every scheduling decision had to pass through the central scheduler, creating a bottleneck. Additionally, the complexity of maintaining increasingly sophisticated scheduling logic in a single scheduler became impractical.

To address these challenges, two-level schedulers were introduced, decoupling global resource management from application-specific scheduling logic. Systems like Mesos [3] and YARN [9] adopted this design. In Mesos, a central master manages hardware resources and offers subsets of resources to individual application-level schedulers, which independently decide task placement. This design significantly improves scalability and simplifies the implementation of new schedulers. However, it sacrifices optimality: the resource offers provided by the Mesos master may not align well with application demands, leading to suboptimal utilization. By contrast, YARN introduced a request-based model, where Resource

Managers (RMs) manage the global hardware resource pool and Application Masters (AMs) request resources and schedule tasks within allocated limits. This design allows job-level schedulers to employ customized scheduling logic, but the resulting allocations are typically optimal only at the job level. The RM can make post-request adjustments, such as preempting containers, but this process is reactive rather than globally optimized.

The limitations of two-level schedulers paved the way for shared-state schedulers, also referred to as concurrent schedulers, such as Omega [6] and Protean [2]. Unlike two-level schedulers that use pessimistic concurrency (where resource contention is avoided by locking), shared-state schedulers adopt optimistic concurrency. Multiple parallel schedulers operate on a globally shared view of cluster resources, make scheduling decisions independently, and later resolve any detected conflicts. This approach allows each scheduler to consider the full cluster state, enabling more efficient task placement and better overall resource utilization.

In response to increasingly diverse workloads, modern frameworks have moved toward a unified scheduler model, which combines the benefits of monolithic and modular architectures. Unified schedulers retain a global view of the cluster but modularize scheduling logic to improve extensibility and scalability. They accommodate heterogeneous workloads by using techniques such as sharding, replication, and background rebalancing for fault tolerance and scalability. For instance, Twine [7] implements a three-level scheduling hierarchy: a Resource Broker manages machines, a scheduler handles containers, and application-level schedulers oversee specialized workloads. Twine unifies all resource requests and processes them collectively, leveraging sharding and a background ReBalancer to optimize task placement incrementally. Similarly, systems like Optum [4] and the recent unified scheduler in Borg [8] adopt similar unified architectures, ensuring scalability while addressing the heterogeneity of modern workloads.

3.3 Semantic-agnostic to Semantic-aware

A notable trend in modern resource allocation frameworks is the increasing responsibility these systems take on in managing applications. Early frameworks such as YARN and Mesos focused solely on task placement—deciding where to run applications and facilitating their execution. However, with the adoption of containers and virtual machines (VMs), resource management frameworks have evolved to orchestrate additional layers of application runtime management. This includes spinning up the execution environment, resolving domain names, and elastically scaling applications in or out based on demand. These capabilities require the frameworks to gain a deeper understanding of workload characteristics [2, 4, 8].

For instance, Twine introduces host-level profiles that can be dynamically switched on physical machines to meet workload-specific requirements [7]. Similarly, Optum employs a combination of offline and online profiling mechanisms to accurately capture the resource demands of various jobs [4]. Such workload-aware techniques enable more efficient resource allocation by tailoring resources to the nuanced needs of different applications. Moving forward, understanding workload characteristics and integrating this knowledge into scheduling decisions will remain a critical focus for future research, enabling frameworks to balance performance, efficiency, and scalability in increasingly heterogeneous environments.

4 Conclusion

In conclusion, the evolution of data center resource management frameworks has been instrumental in efficiently multiplexing physical infrastructure to support diverse and dynamic workloads. While recent designs have largely stabilized, significant open challenges remain. One key issue is achieving better isolation between colocated services to minimize interference while improving resource utilization. Although frameworks like Borg, Twine, and Optum employ various techniques for performance isolation, ensuring predictable behavior in multi-tenant environments continues to be a complex problem requiring further exploration.

Another emerging challenge lies in the global coordination of resource allocation. Most existing frameworks operate at the granularity of a single data center or region [4,7]. However, modern applications are increasingly globally distributed, often deployed at the edge to serve latency-sensitive workloads. Coordinating resource allocation and task placement across geographically distributed clusters introduces new opportunities and challenges, particularly in balancing latency, fault tolerance, and resource availability.

Addressing these challenges will drive the next generation of resource management frameworks, ensuring they can meet the demands of increasingly diverse workloads, heterogeneous hardware, and globally distributed infrastructures.

References

[1] Kubernetes: Production-grade container orchestration, 2024. Open-source container orchestration platform.

[2] Hadary, O., Marshall, L., Menache, I., Pan, A., Greeff, E. E., Dion, D., Dorminey, S., Joshi, S., Chen, Y., Russinovich, M., et al. Protean:{VM} allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 845–861.

[3] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. Mesos: A platform for {Fine-Grained} resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)* (2011).

[4] Lu, C., Xu, H., Ye, K., Xu, G., Zhang, L., Yang, G., and Xu, C. Understanding and optimizing workloads for unified resource man-

agement in large cloud platforms. In *Proceedings of the Eighteenth European Conference on Computer Systems* (2023), pp. 416–432.

[5] Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., and Curino, C. Apache tez: A unifying framework for modeling and building data processing applications. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data* (2015), pp. 1357–1369.

[6] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems* (2013), pp. 351–364.

[7] Tang, C., Yu, K., Veeraraghavan, K., Kaldor, J., Michelson, S., Kooburat, T., Anbudurai, A., Clark, M., Gogia, K., Cheng, L., et al. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 787–803.

[8] Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., and Wilkes, J. Borg: the next generation. In *Proceedings of the fifteenth European conference on computer systems* (2020), pp. 1–14.

[9] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), pp. 1–16.

[10] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the tenth european conference on computer systems* (2015), pp. 1–17.

[11] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing. In *9th USENIX symposium on networked systems design and implementation (NSDI 12)* (2012), pp. 15–28.