Cluster resource scheduling in cloud computing: literature review and research challenges

Article *in* The Journal of Supercomputing · April 2022

DOI: 10.1007/s11227-021-04138-z

CITATIONS
12

READS
595

2 authors:

Wael Khallouli
Old Dominion University
16 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Jingwei Huang
University of Texas Southwestern Medical Center
58 PUBLICATIONS **1,011** CITATIONS
SEE PROFILE

The Journal of Supercomputing (2022) 78:6898–6943
https://doi.org/10.1007/s11227-021-04138-z

Cluster resource scheduling in cloud computing: literature review and research challenges

Wael Khallouli[1] · **Jingwei Huang**[1]

Accepted: 11 October 2021 / Published online: 29 October 2021

Abstract
Scheduling plays a pivotal role in cloud computing systems. Designing an efficient scheduler is a challenging task. The challenge comes from several aspects, including the multi-dimensionality of resource demands, heterogeneity of jobs, diversity of computing resources, and fairness between multiple tenants sharing the cluster. This survey provides a multi-perspective overview of the cluster scheduling problem. We present a multi-dimensional classification of existing cluster management solutions based on their scheduling architectures, objectives, and methods. We also survey the recent research works which have employed machine learning solutions in cloud computing resource management. Existing cluster scheduling systems face many challenges, such as achieving a tradeoff between multiple conflicting objectives, finding the balance between jobs' requirements, scaling to the new operational demands, and choosing the appropriate scheduling architecture. Using machine learning in cluster scheduling is a promising direction to go to develop the future generation of intelligent resource schedulers.

Keywords Cloud computing · Data centers · Resource management · Distributed systems · Job scheduling · Resource allocation

1 Introduction

A scheduler is a pivotal component in a cloud computing system. It is a challenging task to design an efficient and effective scheduler. This challenge comes from several aspects, including the characteristics of the cloud environment, such as the heterogeneity of computing resources and jobs, the multi-dimensionality of resource demands, and the dynamicity of submitted workload, and the different expectations

✉ Wael Khallouli
wkhallou@odu.edu

[1] Department of Engineering Management and Systems Engineering, Old Dominion University, 2101 Engineering Systems Building, Norfolk, VA 23529, USA

and goals of cloud providers and users. Cloud providers' primary concern is to reduce all operational costs and, at the same time, meet users' expectations, mainly the quality of service and the availability of computing resources. Many schedulers have been proposed or deployed to tackle different issues such as (1) finding the balance between latency-sensitive and long-running jobs requirements, (2) achieving performance and QoS requirements, (3) achieving a fair and efficient allocation of resources, and (4) scaling to the demands of a growing number of jobs. In this paper, we review representative cluster scheduling solutions, including YARN, Borg, Kubernetes, Tetris, Quincy, Firmament, Mesos, Apollo, Omega, Sparrow, Tarcil, Hawk, Mercury, MEDEA, Open-Stack, Paragon, Quasar, Ernest, Eagle, Gemini, Prophet, Corp, and TetriSched, and others.

The cluster scheduling problem fundamentally exists across different cloud settings, such as data-intensive and distributed computing systems, infrastructure as a service (IaaS) systems, platform as a service (PaaS) systems, and container as a service (CaaS) systems. We investigate the cloud scheduling problem from two different angles: (1) scheduling architectures and (2) scheduling objectives and methods. Cluster resource scheduling includes two main functions: resource allocation and job scheduling. Resource allocation is the process of assigning a certain quantity of computing resources to each user or application at runtime, guided by a global policy to share cluster resources among multiple users based on fairness and/or predefined priority. Job scheduling is the process of assigning a scheduling unit (task, process, container, or virtual machine) with the allocated quantity of resources to an appropriate node in the cluster. The challenge of scheduling in a cloud cluster is centered on targeting multiple objectives: (1) high cluster efficiency; (2) fairness among multiple jobs and multiple tenants; (3) low scheduling latency; (4) maximizing the system throughout; (5) meeting performance requirements (QoS requirements). Additionally, it is critical for cluster scheduling systems to be able to (6) support diverse application frameworks; and (7) scale up with a high rate of incoming jobs. A wide range of scheduling algorithms and policies have been proposed, varying from randomized ad-hoc approaches to more sophisticated optimization algorithms and machine learning-based techniques.

1.1 Motivation and related work

Cloud providers rely on large-scale compute clusters as the physical infrastructure for running cloud computing services and applications. Nowadays, many data analytics, machine learning, AI, and distributed applications use the cloud infrastructure to leverage its computing power. To meet the needs of computing tasks from recent advances of digital technologies, such as IoT, AI/ML, and Blockchain, high-performance computing (HPC) is also increasingly using cloud computing paradigm [2, 41]. Hence, a cloud computing cluster is typically shared by diverse applications with complex operational demands. In the future, we will continue to face an increasing need for cloud computing clusters with potentially new computing demands as many industrial sectors are moving their applications to the cloud. It is of paramount importance to be able to manage computing resources in this

environment effectively. Developing efficient scheduling systems to cope with current and upcoming scheduling challenges requires an understanding of the different aspects of the cluster scheduling problem: (1) what scheduling objective to address, (2) what scheduling design to select, and (3) what is the most effective scheduling approach to use. An up-to-date and comprehensive overview covering the different cluster scheduling aspects is of prime importance.

This survey aims at providing cloud scheduling researchers with a multi-perspective broad view of the essential features of the cluster scheduling problem. It will also help researchers understand the key challenges and issues of the cluster scheduling problem for further investigation. The major contributions of this survey are the following:

- We analyze a broad range of cluster resource management systems, including the most recent ones. We provide a multi-dimensional classification of cluster management systems based on their scheduling designs, objectives, and research approaches.
- We identify the main features of the cluster scheduling problem, and we highlight its major research challenges ahead.
- We explore the use of machine learning approach in cluster scheduling.

Many important survey articles on cluster resource management can be found in the literature. Rodriguez and Buyya [72] surveyed existing container orchestration tools. They proposed a taxonomy of these systems based on their workload types, architectures, granularity, and objectives. This survey only covered ten operational (commercial and open-source) management systems, including Borg, Kubernetes, Swarm, Mesos, Marathon, Yarn, Omega, Apollo, and Fuxi. Weerasiri et al. [92] also surveyed resource orchestration tools (i.e., cluster management systems responsible for scheduling and monitoring container-based jobs) deployed in large-scale production clusters. They presented a multi-dimensional taxonomy of the resource orchestration tools based on their cloud model (IaaS, PaaS, or SaaS), runtime environment (e.g., public, private, hybrid, virtualization-based, container-based), and scheduling strategy (e.g., rule-based or user-defined). This survey primarily focused on a few operational open-source solutions. Although these surveys provided comprehensive views of the container-based orchestration tools, they only covered few systems. Besides, none of these surveys discussed the existing cluster scheduling architectures. Our survey addresses a broader range of cluster management systems including production systems (e.g., YARN, Kubernetes, Mesos, and others) and theoretical systems (e.g., Sparrow, Tarcil, and others) operating in different cloud settings (e.g., data-intensive systems, IaaS systems, container orchestration systems, and others). Malte [5] provided a survey on cluster scheduling architectures. He categorized the cluster management systems into four categories—centralized, two-level, distributed, and hybrid—discussing their advantages and limitations. We extend Malte survey and develop a more detailed taxonomy of cluster scheduling designs.

Singh and Chana conducted two important surveys on resource provisioning [77] and scheduling [78] in cloud computing. Jennings et al. [36] investigated the different resource management problems in cloud computing. Several other survey papers

focused on specific resource scheduling types (e.g., workflow (DAG) scheduling algorithms [94]) or techniques (e.g., load balancing techniques [26]). We propose a multi-perspective survey addressing more than one aspect of the problem. Resource management in cloud computing is a hot topic; the research in this area is rapidly evolving as the complexity of the problem is growing. This survey explores the latest research studies, and at the same time, the prominent earliest scheduling works. This work is also one of the few works introducing how machine learning methods are used in cluster scheduling which is the building block to design the future generations of intelligent scheduling solutions.

1.2 Outline

The remainder of this paper is organized as follows. In Sect. 2, we describe the cluster scheduling problem and identify its main actors, objectives, and constraints. In Sect. 3, we review the different scheduling designs proposed in the field and discuss their major advantages and limitations. In Sect. 4, we review a set of representative cluster scheduling solutions from each design category. We categorize the cluster scheduling research studies according to their scheduling objectives and identify the major scheduling approaches they employ. In Sect. 5, we discuss the application of machine learning techniques to the resource management problem in cloud computing. In Sect. 6, we summarize the main features of the reviewed schedulers and the research challenges ahead. Finally, in Sect. 7, we conclude the paper.

2 Problem characteristics

The cluster scheduling problem fundamentally exists across different cloud settings, including data-intensive, PaaS, CaaS, and IaaS cloud environments. Several actors are involved in public cloud systems [52], mainly the cloud providers and users. The problem comprises several entities: providers, users, and the scheduler.

2.1 Cloud providers

A cloud provider is responsible for managing and maintaining a large-scale pool of computing, storage, and networking resources to offer cloud services. Cloud providers deliver different categories of services; Infrastructure as a service (IaaS), platform as a service (PaaS), and container as a service (CaaS). IaaS providers offer infrastructure resources (computing, network, and storage resources) in the form of virtual machine instances while platform as a service (PaaS) and container as a service (CaaS) providers offer fully managed computing environments on which users can run their applications and/or containers. IaaS providers generally deploy IaaS cluster management systems, such as OpenStack and Amazon EC2, to fully manage the assignment of virtual machine instances to the physical servers upon users' requests. PaaS providers deploy distributed computing platforms, such as Hadoop, Google App Engine, and Azure App service, to offer their services. CaaS providers

deploy container orchestration systems, such as Kubernetes [44] and Docker Swarm [33]. Each of these platforms integrates a scheduler responsible for scheduling users' tasks (PaaS services) or applications' containers (CaaS services) on a compute cluster. Cloud providers rely on large-scale data centers as the cloud computing physical infrastructure. Data centers nowadays are composed of servers with different hardware configurations [45] to support a wide range of applications with various requirements (e.g., data analytics applications and batch applications).

2.2 Cloud users and workload characteristics

The cloud user can be either a person or an organization that maintains a business relationship with the cloud provider. There are two categories of cloud users: end-users and third-party cloud users. Third-party users might also be cloud providers (PaaS, CaaS, or SaaS providers) that allocate computing resources from an IaaS provider to offer PaaS and CaaS services. End-users generate workload by submitting tasks, jobs, or containers. The workload in modern cloud systems is highly heterogeneous in many aspects: priority, size, and demands. The design of an efficient scheduler requires a deep understanding of the workload's characteristics and usage patterns. However, we notice that only few public resource usage traces have been published.

2.2.1 Public resource usage traces

To the best of our knowledge, Google dataset [27] is the largest cluster usage dataset that has been released so far. The dataset includes a one-month resource usage trace (May 2011) on a cluster cell of 12,000 machines in one of Google production systems. It involves the activity of hundreds of thousands of jobs including details about their resource usage, durations, preferences, and states. The dataset was used in several studies (e.g., [69, 16, 17, 48]) to analyze machines' properties (configurations, attributes, states, and loads), jobs' behavior (resource usage, patterns, and states), and users' demands.

More recent public traces from Alibaba clusters were analyzed in [30, 37]. Guo et al. [30] analyzed an 8-day long trace that contains 4000 machines, 9000 online services, and 4 million batch (DAG) jobs. Jiang et al. [37] analyzed a 24-h long trace dating back to 2017.

Cortez et al. [9] studied a VM resource usage trace on Microsoft Azure datacenters (from November 2016 to February 2017). The trace comprises usage data collected from both PaaS and IaaS based VMs, deployed by first-party users (i.e., VMs dedicated for internal use) and external customers (i.e., end-users interacting with VMs deployed by a third-party user). The paper analyzed different aspects of the deployed VMs, such as consumption patterns, sizes, and VM lifetimes.

Talluri et al. [83] collected a 6-month long Spark workload from a big data processing provider. The workload includes a mixture of stream jobs (interactive and non-interactive) and DAG batch jobs. This paper focuses on understanding the

interaction of Spark jobs with the cluster in terms of, for instance, the number of bytes read, and the number of modifications.

2.2.2 Jobs and machines properties in Google data centers

Google data centers host a mixture of long-running services, such as end-user services (e.g., emails and interactive document processing), interactive batch jobs (e.g., machine learning jobs), and latency-sensitive jobs. Long-running services require a certain amount of resources to run permanently and manage end-users' requests. The availability of resources is the major concern for customers using these services; thus, this type of job requires an optimal allocation of resources rather than fast scheduling decisions. Batch jobs in Google clusters typically run from few seconds to few days and require fast scheduling decisions. The size of jobs in Google data centers ranges from a couple of seconds to the entire duration of the trace [69]. The vast majority of jobs run for few minutes; nearly 80% of jobs are shorter than 1000 s [17]. 94% of jobs are executed for less than 3 h [17]. Besides, the majority of Google jobs are not CPU-intensive as they consume only one processor per job. Google jobs are generally single-task jobs (75% of jobs run a single task) [69]. Resource requests are generally small (e.g., 70,000 jobs request less than 0.0001 units of CPU per task) [69]. Google jobs can be categorized into three classes based on their priority: high, middle, and low-priority. High-priority (typically production jobs) jobs have a clear daily pattern and account for more resource usage. Jobs' priorities correlate with durations (e.g., production jobs include the majority of long-running tasks). There is also a correlation between CPU consumption and job priorities (e.g., most of the CPU cores are consumed by low-priority tasks while high-priority tasks are not usually heavy CPU consumers). Several research studies identified a gap between reserved and used resources in Google data centers. While the average reservation rates of memory and CPU are 80% and 100%, the actual resource usage does not exceed 50% and 60% of memory and CPU, respectively. Google production systems are characterized by the high frequency of scheduling events; In peak hours, the scheduler needs to make hundreds of per-second placement decisions [69] due to the high rate of task re-submission caused by hardware failures, evictions, and killed tasks. Although the Google usage trace dates back to 2011, it shows some aspects of workload heterogeneity (e.g., in priority). However, the majority of jobs are homogeneous (batch jobs). We expect that resource management systems nowadays process jobs on a bigger scale with a higher level of heterogeneity, given the current technological capabilities. This adds further complexity to the scheduling process in modern production systems.

2.2.3 Jobs properties in other data centers

Alibaba production systems mainly host two types of jobs: batch jobs and online services. Online services are generally end-user interactive applications while batch jobs are generally composed of a set of parallel tasks either independent or interconnected (e.g., MapReduce and DAG jobs). The majority of batch jobs in Alibaba clusters are relatively 'short' running for a couple of seconds or minutes (80% of

batch jobs finish in less than 100 s [30]). Some jobs are very short (order of sub-seconds). Jiang et al. [37] also pointed out that the large percentage of Alibaba jobs are short-running jobs (order of tens of seconds); the 80th, 90th, and 99th percentiles of jobs run for less than 132, 260, and 1067 s, respectively. Memory utilization of batch jobs in Alibaba data centers is highly fluctuating (compared to CPU consumption). However, the CPU and memory utilization of online services are more stable.

The workload in Microsoft Azure data centers exhibits a different pattern [9]; the deployed VMs generally have very low average CPU utilization (60% of VMs have an average CPU utilization less than 20%), with a slightly higher utilization for PaaS VMs compared to IaaS VMs. The majority of VMs are small-sized VMs that require few virtual CPU cores and memory (80% of VMs require 1 to 2 cores and 70% request less than 4GB). VMs are generally homogeneous in size with a few portions of very large VMs. The majority of VMs have durations ranging from few minutes to few hours, and run a small percentage of very long-running jobs.

2.3 The scheduler entity

The scheduler's role is to assign scheduling units (e.g., task, container, or process) to resource units (e.g., physical machines, resource containers, or slots) upon users' requests. The scheduling problem fundamentally exists in different cloud settings (Table 1).

Scheduling systems cope with multiple requirements coming from the different actors (users and providers). As a result, scheduling in cloud computing is generally formulated as a multi-objective optimization problem in which the scheduler aims to achieve either a single objective or a tradeoff between different categories of objectives. We identify three categories: fairness, efficiency, and performance.

Providers' primary concern is to reduce all operational costs by efficiently utilizing their resources (cluster efficiency), and at the same time, deliver reliable services that meet users' QoS requirements. Multiple scheduling objectives are used in the literature, including maximizing the scheduling throughput (performance), minimizing jobs' waiting times (performance), minimizing the amounts of stranded resources (cluster efficiency), minimizing the energy consumption (efficiency), and maximizing the shares of worst-off users (fairness).

2.4 Scheduling constraints

Resource scheduling is performed subject to several sets of constraints, such as placement constraints, locality constraints, inter-dependency (precedence) constraints, and deadline constraints.

- Placement constraints: placement constraints include users' preferences for specific computing resource types. For example, users might ask for a machine with a particular kernel version or a special hardware requirement, such as GPU. In many cases, jobs are not able to run on machines that cannot meet their computing requirements [25].

Table 1 Resource scheduling problem

Setting	Scheduling unit	Resource granularity	Examples
Big data/distributed systems	Map/Reduce tasks and executors	Resource containers and slots	YARN [87], Spark [98] Sparrow [63], Tarcil [15] Hawk [12]
IaaS systems	Virtual machines	Physical machines	OpenStack Nova [80]
PaaS/CaaS systems	Containers/pods and tasks	Physical machines and virtual machines	Kubernetes [44]
Internal production systems	Tasks and Internal services	Physical machines	Borg [89], OMEGA [75], Apollo [4]

- Locality constraints: users may ask for nodes located on the same rack where input data are stored. In data-intensive computing clusters, placing the jobs close to their input data is critical to meet performance requirements. A congested cross-cluster network significantly delays jobs' completion times [35]. This constraint is known as data locality constraint. Similarly, users in IaaS systems might request physical hosts located in specific geographic locations (e.g., servers in a data center located within the user's country) to assign their virtual machines.
- Workload constraints: jobs are often coupled with precedence, priority, and deadline constraints. In many computing platforms, tasks belonging to the same job may be interdependent. Typical examples include Hadoop jobs [87] and Spark DAG applications [97, 98]. Hadoop jobs are composed of map and reduce tasks; reduce tasks are mostly launched after the completion of reduce tasks. In Spark, Applications' tasks are arranged in a directed acyclic graph (DAG). DAG applications are divided into several stages, where in each stage, tasks are executed in parallel. Tasks belonging to the different stages might also be interdependent. Large production jobs running for hours often come with strict deadline constraints [86].
- Affinity and anti-affinity constraints: affinity constraints require placing tasks or VM instances belonging to the same user on the same node or rack to reduce the network traffic. Anti-affinity constraints require that tasks (or VM instances) are placed on different nodes or racks to avoid interference (e.g., Affinity filters and anti-affinity filters in OpenStack).

Scheduling constraints are also classified into *hard* and *soft constraints* [25, 69]. Hard constraints (e.g., completion deadlines) are the set of mandatory constraints that prevent the execution of a job if they are not satisfied. Soft constraints (e.g., data locality) are the set of non-mandatory constraints that ignoring them does not prevent the execution of a job but reduce its performance [85].

2.5 Cluster scheduling problem: a summary

In summary, cluster management systems operate in a complex cloud environment characterized by:

- Heterogeneity of computing resources: the coexistence of different hardware configurations on the same computing cluster makes computing units have different capacities and performance.
- Multi-dimensionality of resources: each server in the cluster has multiple types of resources such as CPU cores, memory, GPUs, Disk r/w, and network bandwidth. Resource requests, whether they are submitted in the form of virtual machines or resource containers, are always a combination of multiple types.
- Heterogeneity of jobs: jobs have different demands and priority levels. Submitted jobs in cloud computing systems vary from tiny (order of milliseconds) and short jobs to long-running jobs. Some jobs require immediate execution; others

tolerate queueing delays. Jobs have different resource requirements. They may be CPU-intensive or memory-intensive. Some jobs may need a high volume of network traffic, while others need extra GPUs.

- High dynamicity: The availability of resources on servers is highly fluctuating. Some example scenarios include, jobs/tasks running on a server start and complete all the time; available network bandwidth is dependent on network traffic at a particular time. Some dead servers are replaced every day in a large data center. Furthermore, jobs may have fluctuating loads over time, including peak-demand and low-demand periods.

Heterogeneity is one of the most notable features of the workload in cloud computing systems. Several studies tried to characterize the workload in cloud computing systems using public usage traces. Existing cloud scheduling studies categorize jobs in different ways, for instance, based on their size (e.g., short, medium, and long jobs) and/or resource requirements (e.g., CPU-intensive, memory-intensive, and bandwidth-intensive jobs). It should be pointed out that no accurately defined criteria exist to draw the line between these categories; the classification of jobs is still a fuzzy concept. The classification of jobs into short/long is not necessarily based on their temporal characteristics but also on other characteristics, such as tolerance to scheduling delays and scheduling priority. Long jobs are typically characterized by their high priority, tolerance to waiting delays, and the need for resource availability (e.g., long-running Google web services). Short jobs do not generally tolerate scheduling delays and have low priority; examples include big data queries and stream applications. In the same way, classifying jobs based on their resource consumption (e.g., CPU-intensive, memory-intensive, and bandwidth-intensive) is also fuzzy. Examples of CPU-intensive jobs include machine learning tasks and HPC batch jobs, which typically consume a lot of CPU cores. Other jobs, such as deep learning applications, might request additional GPU processors to speed up. As a general rule, any job whose execution heavily depends on the availability of a certain resource type is considered 'intensive' on that resource. Some existing schedulers, such as Hawk [12] split the submitted jobs into several classes as part of their scheduling process. Short/long jobs in Hawk are classified according to a threshold calculated from past jobs' statistical properties.

In this review, we identified many common scheduling issues encountered in cloud computing systems, such as resource contention, over-allocation, job fragmentation, noisy neighbor, head-of-line-blocking, and interference issues. Resource contention refers to the conflict over access to a shared server. In a Hadoop cluster, a large rate of submitted low-priority jobs might prevent other critical jobs (e.g., HBase streaming jobs) from running or using input data located on a specific node [56]. This issue occurs due to the absence of a heterogeneity-aware scheduling mechanism. Over-allocation is another common issue in cloud computing that occurs when resource demands are misestimated or relevant resource types are not explicitly specified in users' requests [29] (e.g., network resources). Job fragmentation refers to the placement of tasks of the same job across different machines; a high job fragmentation rate results in a congested

network which reduces the overall performance. Noisy Neighbor and head-of-line-blocking problems typically occur when a heavy job (or virtual machine) monopolizes the majority of resources [74, 93].

3 Cluster scheduling architectures

Cluster management systems typically belong to four main categories according to their scheduling architecture: centralized, two-level, distributed, and hybrid systems. These categories can be further split into several sub-categories: queue-based centralized, flow-based centralized, offer-based two-level, fully two-level, shared-stated distributed, fully distributed, fully hybrid, shared-state hybrid, and two-scheduler designs. The scheduling design has shifted away from the traditional centralized design toward decentralized and distributed designs. The scheduling architectures vary in many aspects: the visibility assigned to the schedulers, the number of deployed schedulers, the organization of the schedulers, and the implemented scheduling logic.

3.1 Centralized systems

The centralized scheduling design is adopted by a large number of cluster management systems including Hadoop YARN [87], Google Borg [89], Google Kubernetes [44], Tetris [29], Quincy [35], Firmament [28], and Gemini [59]. High-performance computing (HPC) (e.g., slurm [79]) and infrastructure-as-a-service (IaaS) cloud schedulers (e.g., OpenStack Nova [61]) also use this design. Centralized schedulers include three key components:

- Node manager: deployed on each node to report its state (resource availability) and manage the execution of its tasks.
- Job master: assigned to each job to manage its lifecycle and decide where to assign its tasks.
- Per-cluster resource manager: receives requests from running jobs' masters and allocates resources based on an updated and global view of the cluster's state (e.g., YARN resource manager and BorgMaster).

There exist two main categories of centralized schedulers: queue-based schedulers and flow-based schedulers. Queue-based schedulers, such as YARN [87], use multiple hierarchical per-user queues. Computing resources are distributed among these queues based on a global allocation policy (e.g., fairness, capacity, and delay policies). Single-queue schedulers, such as Borg [89] and Kubernetes [44], maintain a central waiting queue in which admitted tasks are placed. The scheduler periodically scans this queue to select the next task to place. Flow-based schedulers (e.g., Quincy [35] and Firmament [28]) make scheduling decisions by solving a min-cost flow-based formulation of the scheduling problem.

The centralized scheduling design might be extended to integrate other components; for example, Gemini [59] integrates an offline training module in its resource manager. This module collects real-time data from the workload to support the scheduler in selecting the appropriate placement policy. Centralized schedulers support a wide range of resource sharing policies, including fairness, capacity, and priority-based policies. Using the cluster-wide view, they can apply sophisticated scheduling algorithms to achieve optimal placement decisions. The centralized design also supports complex scheduling constraints (e.g., placement constraints and locality constraints).

The limited scalability and the inability to cope with job heterogeneity are the two main limitations of this design. The design of early centralized schedulers (e.g., Hadoop YARN [87]) assumed that the workload in data centers is homogeneous consisting mainly of batch jobs. As a result, these schedulers were not built to meet the strict latency demands of latency-sensitive and short jobs [75] and could not scale to the high submission rates of jobs. The centralized scheduling design is not convenient for latency-sensitive and short jobs mainly due to the scheduling overhead coming from the time required to aggregate details about resource availability in the cluster and the time required to elaborate optimal scheduling plans. Modern data centers typically host a mixture of computing frameworks to meets users' computing requirements [31]. It is difficult to implement multiple scheduling policies to meet each computing framework's requirements using a single centralized design.

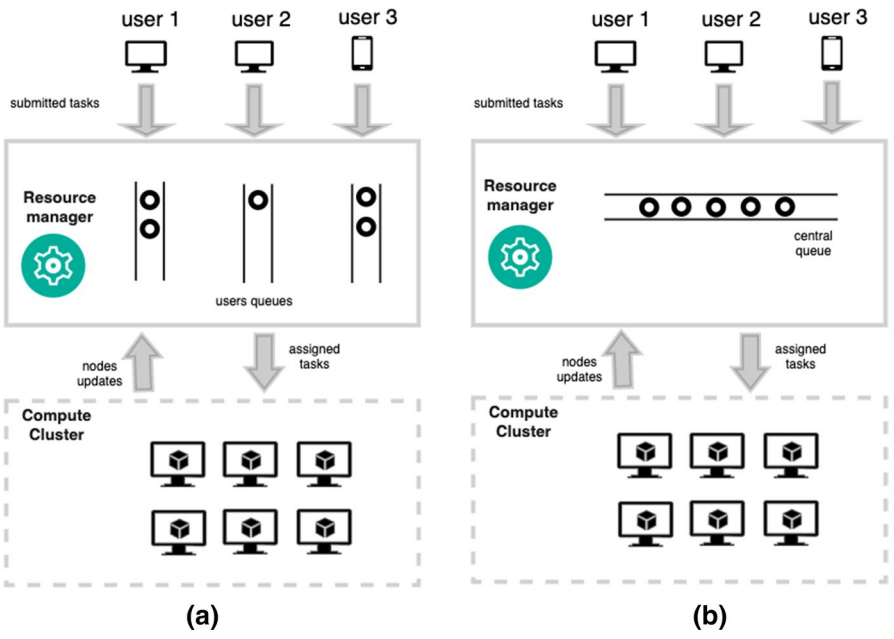

Fig. 1 Queue-based centralized designs: **a** multiple queue-based design (e.g., YARN [87]), and **b** single queue-based design (e.g., Borg [89])

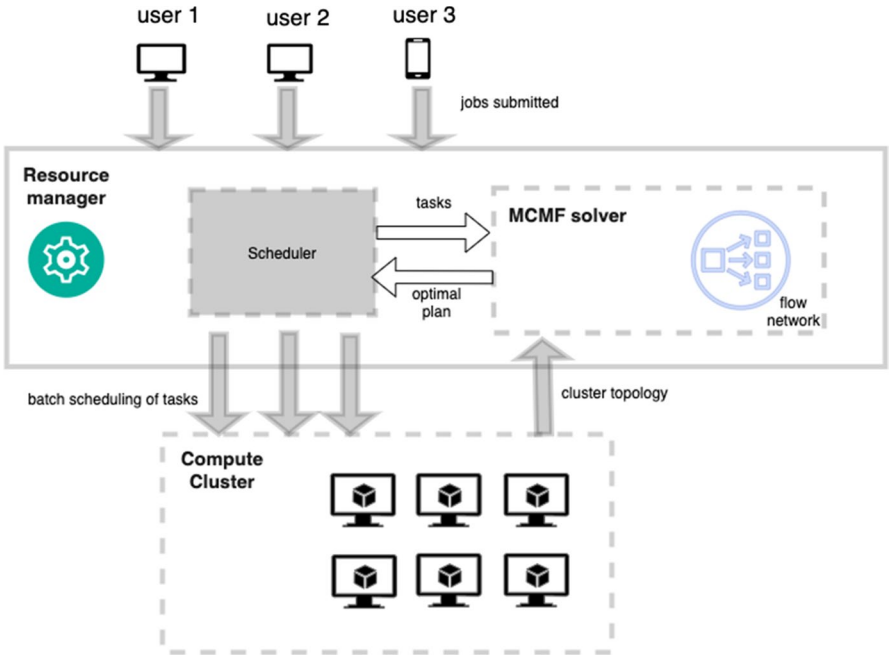

Fig. 2 Flow-based centralized design (e.g., Firmament [28])

Figure 1 shows the architectures of queue-based schedulers (multiple-queue and single-queue based schedulers), while Fig. 2 shows the architecture used by flow-based centralized schedulers. Flow-based centralized schedulers use the same centralized architecture; however, scheduling decisions are made through an MCFC solver, given the nodes' states.

3.2 Two-level systems

The two-level architecture is designed to share resources among multiple computing frameworks running on the same cluster (e.g., Spark and MPI). There are two main categories of two-level systems: offer-based systems (e.g., Apache Mesos [31]) and fully controlled systems (e.g., Mira [39]).

Offer-based systems decouple the resource sharing and task placement functions into a two-layer design. Mesos [31] uses a central master to make periodic *resource offers* (i.e., list of free resources) to each computing framework. Computing frameworks' schedulers can either accept or reject the offers and wait for the next ones. The allocation of resources at this level is performed subject to fairness constraints (e.g., DRF) or simple priority rules (e.g., FIFO). The assignment of tasks is granted to the computing frameworks' internal schedulers; each scheduler decides where to run its tasks according to its requirements.

The main advantage of using two-level systems, compared to fully centralized systems, lies in their ability to multiplex a shared compute cluster among multiple computing frameworks in a scalable way. For example, a single Mesos cluster can host Marathon [53] (a service scheduler) to run long-running jobs and services, Apache Spark [82] to run data processing jobs, and Chronos [8] to run batch and repetitive jobs. However, the distribution of resources between clusters (using for instance a fairness-based scheme) is static. Mesos uses a pessimistic concurrency control approach; once computing resources are assigned by the central Mesos master to a scheduler, they become unavailable for the other schedulers even when they are unused. This static allocation of resources leads to inefficient cluster utilization. Furthermore, the two-layer design leads to relatively large scheduling delays. The time taken by the central master to allocate resource offers often impedes the ability of the schedulers to make decisions at small timescales.

It is possible to improve two-level systems by enabling dynamic mechanisms to redistribute resources (between the frameworks' tasks and/or across frameworks) and better integration and communication between the frameworks' internal schedulers and the central master. Indeed, achieving an efficient allocation of resources and reducing the scheduling delays in two-level systems are current research directions. In an attempt to address the above issues, Mira [39] adopts a fully connected two-level design. The system maintains the traditional two-level design that consists of two main components: a per-cluster master and a set of computing frameworks. However, the central master and the computing frameworks' internal schedulers are tightly coupled, the advantage being reduced scheduling delays. The system introduces an execution environment (EX) interface to improve the communication between the computing frameworks' internal schedulers and the central master. Implementing this interface minimizes the delays taken by the computing frameworks to acquire and release computing resources, and thus, overall scheduling latency is minimized. In addition, Mira supports a dynamic allocation of resources by redistributing unused resources in real-time to increase the efficiency of two-level systems in highly-loaded clusters.

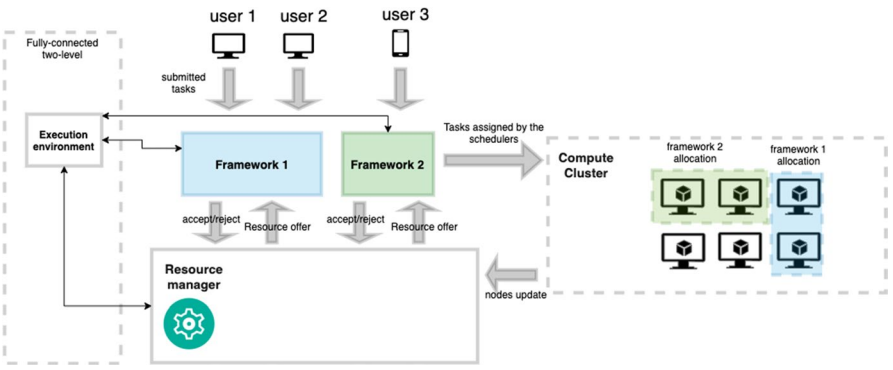

Fig. 3 Two-level scheduling design (e.g., Mesos [31]). The dashed box presents the added component for fully connected designs (e.g., Mira [39])

Using the newly introduced EX component, the master continuously evaluates resource assignments to cope with the changing applications' demands by releasing idle executors.

The two-level scheduling design is illustrated in Fig. 3. Both offer-based and fully connected two-layer designs include a higher-level resource manager (master) and a set of computing frameworks.

3.3 Distributed systems

Distributed scheduling systems consist of a set of parallel per-job schedulers operating independently without any centralized agent that performs a cluster-wide allocation of resources. There are two main categories of distributed systems: shared-state and fully distributed systems. Microsoft Apollo [4], Google Omega [75], and Tarcil [15] are typical examples of shared-state distributed systems. Scheduling decisions are made based on a shared cluster-state; for instance, the shared-state in Omega is a central registry (Paxos-based store) in which all placement transactions are stored. The shared-state in Apollo is a central matrix that includes estimations of the possible waiting times on each node. The key difference between shared-state systems and centralized systems lies in the number and organization of the deployed schedulers in the cluster. The shared-state distributed design involves many parallel schedulers (e.g., per-job schedulers) using the shared-state (either logical or physical shared-state) to coordinate their scheduling decisions. In a centralized design, a single monolithic scheduler is deployed on top of the cluster to make all scheduling decisions. Schedulers in this design generally use an optimistic lock-free concurrency control where all unused resources are always accessible, unlike the pessimistic concurrency control approach used by two-level offer-based systems.

Shared-state distributed systems are unable to make placement decisions at millisecond timescale to meet the latency requirements of time-sensitive tiny tasks. To overcome this problem, fully distributed systems, such as Sparrow [63], fully eliminate the shared-state component. Scheduling decisions in Sparrow are completely decentralized. Each scheduler makes its scheduling decisions based on a limited view of the cluster (i.e., each scheduler has access to a limited number of random nodes).

Distributed systems suffer from several shortcomings. First, it is unclear how to achieve global requirements, such as fairness, without a centralized view of the cluster state. Second, distributed systems generally apply simple placement approaches (e.g., sampling-based placement) that cannot capture complex placement constraints. Finally, in highly-loaded clusters, the absence of coordination between parallel schedulers in fully distributed systems, the outdated shared-states, and the absence of a cluster-wide view results in several efficiency problems, such as race condition, i.e., multiple schedulers simultaneously select the same node.

Figure 4 illustrates the two subcategories of distributed designs. Both designs adopt the a parallel architecture. Shared-state schedulers use a shared register to coordinate their scheduling decisions while fully distributed systems use sampling techniques to assign tasks without using a central shared-state.

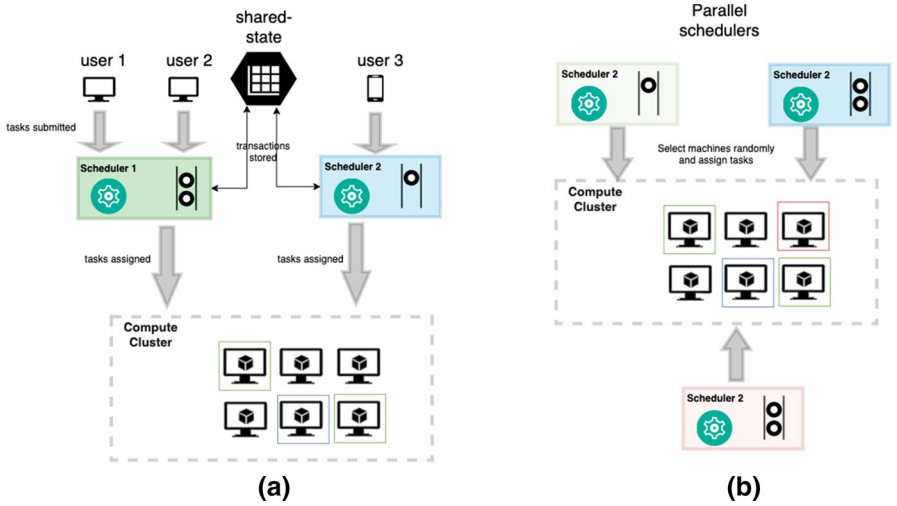

Fig. 4 Distributed scheduling designs: **a** shared-state distributed design (e.g., Omega [75] and Apollo [4], and **b** fully distributed design (e.g., Sparrow [63])

3.4 Hybrid systems

The hybrid design combines both centralized and distributed designs. We identify three categories of hybrid systems: fully hybrid, shared-state hybrid, and two-scheduler systems.

The fully hybrid design (e.g., Hawk [12] and Mercury [38]) is typically composed of a centralized resource manager and a set of distributed schedulers. Fully hybrid systems typically operate in a cluster that hosts a small number of long-running and high-priority jobs consuming the largest fraction of resources, and a large number of low-latency jobs with low resource demands [12]. In this design, a separate scheduler is dedicated to each job type; long-running jobs are optimally placed by the centralized scheduler using sophisticated scheduling algorithms while short jobs are placed in a distributed manner to minimize the scheduling latency. Fully hybrid schedulers leverage the benefits of both centralized (optimal scheduling of long jobs) and distributed designs (fast scheduling of short jobs). However, they have several issues:

- Classification of jobs: hybrid systems classify incoming jobs into short and long jobs. Then, each job is assigned to a dedicated scheduler based on its category. It is not clear how to draw the line between the different job categories (i.e., how to identify which jobs are long-running and which jobs are short). Hawk [12] calculates a threshold by collecting statistics about incoming jobs, assuming that the proportion of each job type is stable over time. This static job classification approach limits the ability of Hawk scheduler to cope with the dynamicity of modern cloud computing clusters.

- Efficiency problem: several workload analytics studies (Sects. 2.2.1 and 2.2.2) pointed out that short jobs dominate the workload in cloud computing data centers. Hybrid schedulers generally use simple and randomized task placement techniques to place short jobs. Given the high rate of short jobs in cloud computing clusters, the randomized placement of jobs leads to efficiency and performance issues. The absence of a centralized view to locate where short jobs are placed typically leads to resource contention issues.

Shared-state hybrid systems (e.g., Eagle [10] and Phoenix [84]) allow distributed schedulers to partially use the information about the nodes' states, which is provided by the centralized scheduler. For instance, Eagle [10] maintains a vector that includes m (m: number of nodes) binary elements, each of which represents a cluster's node. The value of each element indicates whether a long task is assigned to the node. This vector is used by the distributed schedulers to assign their tasks to the nodes with the least waiting times (where no long job is assigned) to reduce jobs' latency. Phoenix [84] maintains a constraint resource vector (CRV) that estimates the scheduling delays on each node taking into account the node configuration and the waiting tasks' requests. The system applies a dynamic reordering technique that periodically redistributes short tasks between nodes based on this vector.

Ray [55] also incorporates a global scheduler and a set of distributed schedulers. However, the centralized global scheduler is not exclusively dedicated to a certain job category. Ray uses a bottom-up hierarchical scheduling approach where tasks are scheduled locally in the first place. The global scheduler is used only when local schedulers are not able to schedule a task (due to the node overload or the incapacity of the scheduler to meet the task's constraints locally).

Finally, the two-scheduler design (e.g., MEDEA [23]) consists only of two centralized schedulers: a centralized scheduler dedicated for long-running jobs (with respect to their placement constraints) and a regular task scheduler to allocate containers for short-running jobs. Figure 5 illustrates the architectures of the fully and shared-state (dashed box) hybrid designs.

3.5 Summary

The evolution of the cluster scheduling architectures was originally driven by two main factors: the increasing heterogeneity of jobs and the limited scalability of centralized schedulers.

- Job heterogeneity: cloud computing data centers host a mixture of job profiles, each of which has its scheduling requirements. The centralized design is the adequate design for the implementation of cluster-wide policies (e.g., fairness, priority, and so on) and elaborated scheduling algorithms to meet fairness, performance, and cluster efficiency requirements. However, this design fails to meet short jobs' latency requirements. Hence, the distributed design eliminates the centralized resource manager to speed up scheduling decisions for short-lived

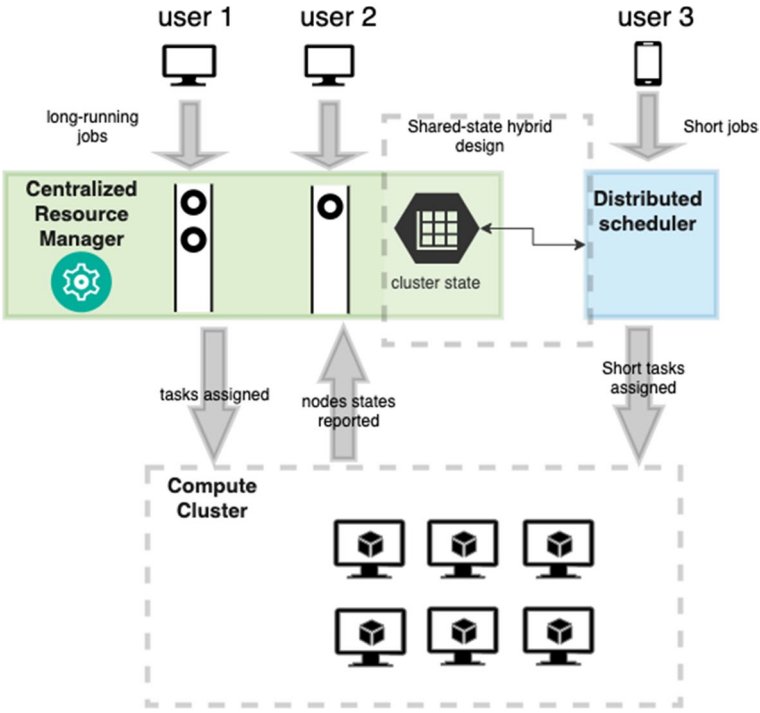

Fig. 5 Fully (e.g., Hawk [12]) and shared-state hybrid (e.g., Eagle [11]) scheduling designs

and interactive jobs. Job heterogeneity is one of the most challenging issues facing cluster scheduling systems. Recently, hybrid architectures were proposed as an alternative to leverage the benefits of both centralized and distributed architectures.

- Scalability: centralized schedulers cannot scale to a large number of submitted jobs. Several studies in the literature have focused on decentralizing the schedul-

Table 2 Scheduling designs taxonomy

Design	Approach	Examples
Centralized	Queue-based	Borg [89], YARN [87], Kubernetes [44], Tetris [29], Gemini [59]
	Flow-based	Quincy [35], Firmament [28]
Two-level	Offer-based	Mesos [31]
	Fully connected	Mira [39]
Distributed	Shared-state	Omega [75], Apollo [4]
	Fully distributed	Sparrow [63]
Hybrid	Two-scheduler	MEDEA [23]
	Fully hybrid	Hawk [12], Mercury [38]
	Shared-state	Eagle [10], Phoenix [84]

Table 3 Scheduling designs - advantages and limitations

Design	Advantages	Limitations
Centralized solutions	Global view of the cluster	Bottleneck for low-latency jobs
	Optimal scheduling of long services and batch jobs	Scalability limitations
	Easy to implement fairness and priority-based sharing policies	Inability to implement multiple scheduling policies
Distributed solutions	Fast scheduling decisions for low-latency jobs	Hard to enforce a global strategy, such as fairness
	High scalability	Inefficient cluster utilization
		Bad performance under high load
Hybrid solutions	Address the requirements of both short and long jobs	Assume the workload is composed of a large number of short jobs and few long jobs
	Improved cluster utilization	

ing design by decoupling the resource allocation and task placement functions into multiple layers to increase the scalability of the scheduling systems.

In Table 2, we report the main categories (and subcategories) of the surveyed cluster management systems.

In Table 3, we summarize the main advantages and drawbacks of each scheduling design.

In summary, we observe a shift from centralized scheduling designs to distributed and hybrid designs to add further flexibility to the scheduling systems and meet the latency requirements of short jobs. However, centralized schedulers are still commonly used in practice since they are well-suited for implementing cluster-wide scheduling policies and meeting complex scheduling constraints.

4 Cluster scheduling approaches

Cluster management systems operate in a complex environment in which they cope with multiple scheduling objectives, sometimes contradicting. A wide range of scheduling approaches has been proposed. In this section, we review studies related to three major classes of scheduling objectives: fairness, cluster efficiency, and performance.

4.1 Scheduling approaches for achieving fairness

The core idea of the cloud computing paradigm is to share computing resources among multiple users or organizations in the most effective way. Fairness is one of

the main scheduling goals, which cloud providers aim to achieve in the resource allocation process. Different fairness variants have been implemented, such as max-min fairness and proportional fairness; however, max-min fairness models are the commonly used ones.

4.1.1 Max-min fairness

The key idea of max-min fairness is to maximize the shares of worst-off users to avoid monopolizing shared resources by a small group of resource-extensive large jobs (i.e., job starvation situation) [93]. Early max-min fairness models were implemented on a single resource type basis. The model was then generalized to multiple resource types. Max-min fairness algorithms generally use progressive filling to assign resources to the jobs.

A notable model for multi-dimensional resource allocation is dominant resource fairness (DRF) [24]. The model relies on the notion of users' "dominant shares"; a user's dominant share is her highest share (i.e., the fraction of allocated resource to the total capacity) of any resource among all her shares. At each scheduling cycle, DRF allocates the next resource unit (e.g., YARN container) to the user (or job) with the smallest dominant share so that all dominant shares are equalized. Therefore, each user receives her highly requested resource type, and worst-of users' shares (in terms of dominant shares) are maximized. For a detailed running example of DRF, readers are referred to the DRF paper [24]. The proposed DRF model has several limitations: (1) the model does not integrate any mechanism that takes into account jobs' constraints, such as placement constraints, (2) users' demands are assumed to be identical, (3) users' allocations are exclusively calculated based on a "fairness" metric while other important metrics, such efficiency, are not considered, and (4) the heterogeneity of computing resources is not taken into account. DRF was further extended [65]. In this work, the authors proposed EDRF (extended dominant resource), which generalizes DRF to deal with some situations appearing in real settings, such as: (1) zero-demand case (i.e., where users do not request every resource type), and (2) weighted demands case (i.e., weights are assigned to each resource type). The authors also discussed other DRF limitations, such as the "indivisibility assumption" of resource demands. DRF works under the assumption that both computing resources and tasks are divisible. If a task receives half of its resource demands, DRF allows the completion of 1/2 of it, which is not a realistic assumption in practice. A sequential Minmax algorithm was proposed to allocate resources under EDRF.

Max-min fairness models have been integrated into several cluster management systems, such as YARN, Mesos, and Mercury. Hadoop YARN [87] fair scheduler uses max-min fairness to allocate resources on both single and multiple resource bases (CPU and Memory) [22, 93]. Apache Mesos [31] applies DRF to allocate resource offers to the computing platforms running on the same Mesos cluster. Mercury [38] integrates DRF in its centralized resource manager to assign long-running containers.

4.1.2 Max-min fairness with placement and locality constraints

The original DRF model does not consider the heterogeneity and diversity of computing resources in the cluster. Computing resources are generally assumed to be homogeneous, and thus, the cluster is regarded as a single entity. Users' dominant shares in DRF are calculated based on this assumption. Hence, many subsequent studies investigated how to generalize DRF.

For instance, DRFH [90] redefined how the cluster-wide users' dominant shares are calculated by taking into account their dominant shares on each of the cluster's nodes. A local dominant share for a user i on a local node m is to her maximum number of tasks that can be scheduled on that node. The new global (cluster-wide) dominant share for the user i is simply the aggregation of her local dominant shares. At each scheduling cycle, DRFH selects the user with the minimum global dominant share. DRFH does not embed any efficient mechanism to decide where to assign tasks in the cluster. PS-DSF [42] generalizes DRF to take into account users' placement constraints. In a similar way to DRFH, the model calculates users' dominant shares on nodes level. PS-DSF defines a per-machine "virtual dominant share" for each user. The "virtual dominant share" for a user i on a node m is defined as the fraction of the number of tasks already allocated to the possible number of tasks that could be allocated if the user i monopolizes the whole node.

Choosy [25] addresses the fairness problem (single resource basis) subject to hard placement constraints. In this paper, placement constraints are encoded in a constraint graph. The graph's vertices represent both users and machines. An edge between a user node and a machine node in the graph represents whether the user could run her jobs on the machine due to the hardware requirements (e.g., presence of GPU). The scheduler aims to find an allocation that achieves the optimal constrained max-min fairness (CMMF). An optimal CMMF is Pareto-optimal; it is simply achieved when the scheduler is not able to increase any user's allocation without reshuffling the machines given to the other users. Two progressive filling variants were proposed to solve the allocation problem: an offline variant (optimal solution) and an online variant (approximate solution). Quincy [35] addresses the fairness problem subject to data locality constraints in data-intensive computing clusters. The cluster scheduling problem is mapped as a standard flow network that encodes the structure of the cluster, waiting tasks, and their locality constraints. A cost is given for each assignment to reflect the amounts of data transferred between tasks. The scheduler performs a global search for the optimal solution that minimizes the total assignment cost, in other words, the total data transfer. Finally, Justice [19] is a fair scheduler that incorporates deadline constraints in its scheduling scheme. Justice estimates the number of CPU cores required to meet the deadline constraints of each job, based on completed jobs' logs. Justice integrates this estimation into its admission control aiming at minimizing the number of jobs that violate their deadlines.

4.1.3 Fairness-efficiency tradeoff and other fairness mechanisms

Most of the existing fair schedulers apply strict and short-term measurements of fairness (e.g., DRF). However, few fair allocation schedulers address fairness with

respect to other objectives, such as efficiency and performance. Under certain conditions, it is practical to allow a slight violation of fairness constraints to improve performance or cluster utilization. Such models apply a "relaxed" fairness metric. Gemini [59] is a workload-aware scheduling model (implemented for Hadoop YARN) that copes with the problem of finding the tradeoff between fairness and performance in data-intensive clusters. Gemini uses a new metric called "complementary degree" to measure the heterogeneity level of resource demands. The Complementary degree (d) is quantified as an entropy (i.e., used in information theory to measure the uncertainty of information) and defined as:

$$d = -\sum_{i \in R} P(i) log_2 P(i) \tag{1}$$

where $P(i)$ is the probability of observing a job whose i is its dominant resource type. Gemini periodically measures the complementarity degree of the running workload. The scheduler constructs a tradeoff regression model that estimates the potential fairness loss and performance improvement, given the workload heterogeneity degree as input. The idea behind this tradeoff model is derived from an observation about the workload in Google production systems [27]. The authors noticed that there exists a relationship between the fairness/efficiency tradeoff and the variation of resource demands. In this paper, the performance improvement was quantified in terms of throughput (percentage reduction of makespan for a given allocation) while fairness loss was quantified in terms of average reduction of job completion times. Gemini adaptively decides between using a load-balancing policy (based on resource imbalance heuristic) or a fairness-based policy (DRF). If the estimated fairness loss is smaller than a user-defined threshold, the resource imbalance heuristic is applied. Otherwise, DRF is applied.

Tetris [29] uses a fairness knob-based mechanism to address the tradeoff between fairness and performance. The fairness knob is a value within the [0,1] range that quantifies to which extent the model can deviate from pure dominant resource fairness to achieve better performance. Let J be the set of jobs sorted in decreasing order of how far they are from their fair shares. A DRF algorithm would typically select a task from the worst-off job among all the jobs in the waiting queue. Using the "fairness knob' mechanism, Tetris selects the next task from the subset of $[(1-f)J]$ jobs instead. The value of f arbitrates how efficient the allocation is; setting f to 0 leads to an efficient allocation while setting f to 1 leads to strict fairness.

4.2 Scheduling approaches for achieving cluster efficiency

4.2.1 Minimizing the amounts of stranded resources

Achieving an efficient utilization of resources is also one of the key scheduling objectives in cloud computing. Inefficiency issues occur for several reasons, such as the multi-dimensionality and heterogeneity of computing resources and demands (ignoring the multi-dimensionality and heterogeneity aspects leads to higher

amounts of stranded resources) and the incapacity of scheduling systems to deal with the complexity of scheduling constraints in modern cloud systems.

Centralized systems (e.g., Borg [89], Kubernetes [44], and OpenStack [61]) typically use a filtering-scoring approach to assign tasks. The approach consists of two phases: filtering (or feasibility checking) and scoring. In the first phase, the scheduler selects the set of feasible nodes (those meeting the submitted job's constraints). Existing filters support a wide range of constraints, such as availability constraints, capacity constraints, affinity and anti-affinity constraints, and preference constraints [43, 80]. In the second phase, the scheduler assigns a weight to each node in the feasible set using a scoring function, and the node with the highest weight is selected. The scoring function generally reflects a scheduling policy (e.g., load spreading or load packing) to optimize a scheduling objective (e.g., minimizing the number of preempted tasks). The filtering-scoring mechanism is generally used to apply load-balancing scheduling policies to improve cluster efficiency. It is also easier to integrate users' constraints in the assignment process using this mechanism.

Examples of cluster management systems using the filtering-scoring include Google Borg [89], Google Kubernetes [44], and OpenStack Nova (OpenStack scheduler) [80]. Google Borg [89] uses a hybrid scoring model (between worst-fit and best-fit) to minimize jobs' fragmentation and reduce the amounts of stranded resources. Kubernetes scheduler [44] supports several scheduling policies, such as load spreading (*ServiceSpreadingPriority* and *LeastRequestedPriority* weighters). OpenStack Nova [80] integrates several scheduling policies to improve the cluster efficiency, such as spreading/packing the virtual machine instances across the cluster (e.g., *ram_weight_multiplier*).

Microsoft Apollo [4] implements an opportunistic task placement mechanism to achieve a higher cluster utilization. The scheduler splits incoming tasks into regular tasks, which are generally high-priority tasks, and opportunistic tasks, which are mainly low-priority tasks. Regular tasks are first scheduled using a waiting times matrix (explained in Sect. 4.3). Then, opportunistic tasks are placed to fill unused spaces in the cluster.

MEDEA [23] uses an ILP-based approach to place long-running jobs and jobs with complex placement constraints (e.g., affinity, anti-affinity, and inter-dependency constraints) with respect to the efficiency and performance objectives. MEDEA introduces a 'tag' mechanism (i.e., a tag given to each node or group of nodes) to allow users' express their locality and preference constraints. MEDEA users can attach node tags to their requests to indicate their preferences (e.g., machines configurations). An expression is generated for each resource request and then integrated into the scheduler's optimization model. MEDEA uses an objective function that includes three objectives: (1) maximizing the number of placed containers, (2) minimizing the number of violated constraints, and (3) minimizing the number of lightly loaded nodes. The system administrator can manually give a weight for each objective.

4.2.2 Energy-efficient scheduling

The massive consumption of energy in data centers is one of the biggest concerns facing cloud providers nowadays. A very active research topic is how to achieve efficient utilization of servers and computing resources in order to reduce energy consumption. Saving energy in data centers sometimes comes at the expense of meeting users' QoS requirements. Hence, energy-aware scheduling plays a key role in finding the balance between energy consumption cost, on the one hand, and users' QoS requirements (e.g., jobs' completion times), on the other hand. Energy saving is generally achieved by reducing the number of active nodes in the cluster through a dynamic allocation of resources and load balancing techniques.

The majority of energy-efficient studies have focused on IaaS systems tackling several problems, such as VM allocation and provisioning. For instance, Chen et al. [6] developed a dynamic provisioning approach to reduce energy consumption on servers hosting long-running internet services (e.g., web services) in cloud data centers. The authors analyzed a usage trace collected from Windows Live Messenger servers and noted that users' loads have periodic usage patterns. The authors proposed to dynamically change the number of active servers in the cluster by shutting down inactive servers during off-peak periods through the integration of a short-term load forecasting model.

Other studies focused on energy efficiency in data-intensive clusters, such as Hadoop clusters. For example, Shao et al. [76] proposed an energy-aware fair scheduling approach implemented on Hadoop YARN clusters. The proposed scheduling approach uses dynamic node management to find the balance between efficiency (energy saving) and jobs' deadlines constraints; the scheduler dynamically expands or reduces the number of available nodes according to the incoming jobs' requirements (in terms of deadline constraints). Unused nodes are turned off. A multidimensional knapsack (Multidimensional Knapsack Problem 'MKP') formulation is used to model the problem and a greedy heuristic is proposed to assign online Mapreduce tasks to the cluster's nodes. Kaur et al. [40] proposed an energy-aware scheme (called EnLoc) to find the balance between energy efficiency and users' locality constraints in Hadoop clusters. The paper formulates the scheduling problem as a multi-objective optimization problem, which was solved using an evolutionary algorithm with Tchebycheff decomposition. The scheduler aims at mapping the Hadoop MapReduce tasks and their related input data to a minimal number of nodes for the purpose of reducing energy consumption. Piraghaj et al. [67] investigated the energy-saving problem in CaaS systems. They proposed a scheduling model that assigns containers to the smallest possible number of virtual machines in the cluster. The scheduler evaluates the state of each node in the cluster (whether it is overloaded) and reassigns the containers deployed on overloaded nodes based on a correlation coefficient. This correlation coefficient quantifies the degree of correlation between each container and the load level on the node that would host it. Dong et al. [20] proposed an efficient task placement scheme that minimizes energy consumption through collecting information about the servers' energy profiles (e.g., the amount of power used by a server). The scheduler assigns tasks to the most energy-efficient servers to keep a limited number of nodes active. The proposed scheduling

model was evaluated using Google trace data and shown to be more effective than Google's internal scheduling model.

4.3 Scheduling approaches for achieving performance

Quality-of-service (QoS) is a fundamental concept in cloud computing. Users' QoS requirements are generally measured along three dimensions: performance, availability, and reliability. Scheduling plays a central role in meeting users' QoS requirements and finding the optimal balance between users' and providers' objectives.

4.3.1 Placement of latency-sensitive tasks

Data centers nowadays host a diverse set of jobs including a large number of latency-sensitive jobs, such as interactive data analytics jobs and online processing jobs. Latency-sensitive jobs generally require real-time interaction with computing resources. Hence, achieving low scheduling latency is becoming increasingly important in modern cloud computing systems.

Distributed scheduling systems, such as Sparrow [63] and Hawk [12], implement a sampling-based task placement approach to make scheduling decisions at milliseconds timescale. The central idea of sampling-based scheduling is to randomly select nodes and assign tasks to the least loaded one. Sampling-based scheduling is simply an adaptation of the multiple choices approach (proposed in [64]) to the cluster scheduling field.

Sparrow [63] uses a batch sampling approach; each parallel scheduler sends probes (i.e., a lightweight RPC) to $m.d$ random machines to place d tasks ($d \geq 1$). Tasks are then assigned to the least loaded machines in the entire batch. Several mechanisms, such as late binding (i.e., machines do not immediately reply to the probes but wait until their local queue becomes idle to request a task) and proactive cancellation, are proposed to improve the batch scheduling approach. Other subsequent decentralized schedulers aiming at achieving fast scheduling of latency-sensitive jobs include Ray [55] and Canary [68]. Ray [55] is a computing framework dedicated to machine learning applications. Ray's scheduler uses a distributed bottom-up scheduling approach to dynamically schedule millions of tasks per second. Tasks are scheduled locally first (i.e., on the nodes where they are launched). If the local node is overloaded (or cannot meet local tasks' requirements), the unscheduled tasks are then forwarded to a global scheduler to make scheduling decisions based on the nodes' states. Canary [68] scheduling approach reverses the roles of the global scheduler (called controller) and distributed schedulers (called workers). The global controller in Canary is not responsible for scheduling tasks. However, it is only responsible for data partition—sending partitions to the workers. The workers launch their own tasks based on their assigned data. Canary is also dedicated for running high-performance and data analytics jobs.

Hawk [12] introduces the notion of "task stealing" to improve the sampling-based scheduling approach in over-loaded clusters. Empty machines (out-of-tasks) are allowed to randomly send probes to other machines in the cluster to run some of

their waiting tasks. The "stolen" tasks are generally short tasks, placed behind long tasks in over-loaded waiting queues.

Microsoft Apollo [4] uses an estimation-based approach to achieve fast placement of short tasks. Scheduling in Apollo is performed based on a central wait-time matrix, in which the expected waiting times on each node are estimated. The estimation of waiting times takes into account several parameters, such as locality constraints and tasks' demands. Apollo's parallel schedulers use this matrix to assign their tasks to the nodes with the least estimated waiting times.

Mira [39] achieves low-latency scheduling in distributed analytics clusters by focusing on reducing resource acquisition and release delays, which significantly deteriorate the responsiveness of the cluster at a small time scale. To this end, a communication interface is proposed to guarantee better communication between the resource manager (e.g., YARN) and the application frameworks responsible for task scheduling (e.g., Spark).

4.3.2 Achieving scheduling quality

The term 'scheduling quality' implies selecting the machines' configurations which meet users' QoS requirements. Co-scheduling heavy jobs on the same node result in interference issues, and thus, jobs' completion times are delayed. Hence, a challenging scheduling issue is to find the appropriate machines for hardware-sensitive jobs to improve the overall performance.

Classification-based schedulers, such as Paragon [13] and Quasar [14], apply collaborative-filtering techniques to minimize interference. Both schedulers identify preference similarities between incoming and previously scheduled jobs using singular value decomposition (SVD) and PQ-reconstruction (PQ) methods before scheduling the submitted jobs. Quasar [14] classifies incoming jobs to determine the minimum amount of resources required to meet their QoS constraints. Two types of classification are proposed: scale-up and scale-out. On the one hand, using scale-up classification, the scheduler determines what would be the impact of allocating more resources (e.g., CPU core, memory, and storage capacity) from a single server on the performance of a job. On the other hand, using scale-out classification, the scheduler determines what would be the impact of assigning more servers on the performance of a job. Based on the classification results, the scheduler aims to maintain jobs' performances (quality) and at the same time improve the cluster utilization by avoiding the assignment of extra resources to the jobs.

Tarcil [15] combines both classification and sampling-based techniques to achieve fast and high-quality scheduling decisions. To this end, the scheduler examines a subset of machines instead of the whole cluster to determine the appropriate resources which meet a submitted job performance requirements. Tarcil adjusts the size of the sample according to a "quality target" set by each job; jobs with high-quality targets need a larger sample size than jobs with lower quality targets. Then, the scheduler performs a batch scheduling of tasks within the selected subset of machines. For m requested RUs (resource units), the scheduler assigns tasks to the best $R.m$, where R is the sample size.

Firmament [28] generalizes Quincy's scheduling approach [35]. The cluster scheduling problem is mapped into a min-cost max-flow (MCMF) problem. The scheduler aims to perform an optimal and fast assignment of tasks using a centralized scheduling design. Firmament introduces a flow network (a directed graph) that encodes the cluster's structure, placement constraints, and assignment costs (i.e., the cost of each scheduling decision). The flow network can be adjusted to model one of the following scheduling policies: achieving fairness under data locality constraints, spreading loads, and avoiding bandwidth overload. Firmament uses an incremental solution to solve the derived MCMF problem; the scheduler combines several heuristics such as cycle canceling, relaxation algorithm, successive shortest path, and cost scaling. To reduce the scheduling latency, Firmament builds approximate solutions using the previous graph states rather than optimally solving the problem at each scheduling run.

4.3.3 Achieving performance subject to soft placement constraints

TetriSched [86] is a user-facing reservation system integrated into Hadoop YARN to translate jobs' requests (received at each scheduling cycle) into an expressive language that encodes a large set of constraints including hard constraints (e.g., deadlines constraints), soft constraints (e.g., preference constraints), and combinatorial constraints (e.g., affinity and locality constraints). A Space-Time Request Language (STRL) is introduced to translate jobs' SLO requirements into expression trees using logical operators (e.g., OR, MAX, MIN, AND, and SUM). TetriSched automatically generates an STRL expression that jointly describes all submitted jobs' requests at each scheduling cycle. The expression is automatically transformed into a MILP model; solving this model generates an optimal scheduling plan for all pending jobs. TetriSched is one of the few schedulers that provides a user-interface (based on an expressive language) to integrate information about jobs' requirements and constraints into its scheduling process. The scheduler uses an adaptive plan-ahead scheduling approach to optimize both short-term and long-term decisions. The execution of jobs can be deferred to allow jobs with strict deadline constraints to run. Placement decisions are continuously re-evaluated to adapt the scheduling process with incoming new jobs' demands.

4.4 Evolution of cluster scheduling systems

This paper analyzes the state-of-the-art cluster management systems throughout the last decade. The earliest cluster management systems (e.g., Quincy [35] and early Hadoop versions) used to operate in simple scheduling environments. Resources were defined in terms of fine-grained slots, and the scheduling process was based on a single resource type (generally CPU cores). Compute clusters generally had a small number of nodes. Early cluster schedulers managed homogeneous jobs, predominantly batch jobs. Google Borg was operating in a more complex environment (Google production systems), in which jobs have different priorities. However, the

analysis of Google trace (2011) has shown that the majority of jobs are homogeneous in size (short jobs).

Subsequent cluster management systems were dealing with a higher degree of diversity (short and long jobs), different scheduling requirements, and multi-dimensional resource demands. YARN [87] and Tetris [29] addressed the multi-dimensionality issue of resource requests. OMEGA [75] and Sparrow [63] adopted a distributed design to meet short jobs' requirements. Quasar [14] and Paragon [13] used a classification approach to meet jobs' QoS requirements. Apollo [4] managed the heterogeneity of jobs in Microsoft productions systems using several techniques (e.g., opportunistic scheduling and wait-time matrix).

The development of containerization technology led to the emergence of container orchestration systems, such as Google Kubernetes [44]. Kubernetes managed containers in typically medium-sized computing clusters. Also, the widespread use of data computing frameworks (e.g., Spark [98] and Flink [21]) introduced other categories of jobs, such as interactive analytics, ad-hoc exploratory queries, machine learning iterative jobs, and stream processing jobs. Therefore, cluster management systems started to deal with a tradeoff between multiple objectives, such as fairness/performance tradeoff in data-intensive clusters (Gemini [59]) or speed/quality tradeoff (Tarcil [15]). Hybrid systems, such as Hawk [12] and Mercury [38], principally addressed the tradeoff between long jobs and short jobs requirements. Other hybrid systems focused on improving the completion time of short jobs (e.g., Eagle [10] and Firmament [28]).

The increasing diversity of users' requirements and computing resources led to a new category of jobs: jobs coupled with complex scheduling constraints (e.g., preference constraints). This led to the design of constraint-aware schedulers, such as justice [19] and Tetrisched [86], which can operate in heterogeneous data centers.

4.5 Summary

A wide range of scheduling approaches, including simple and priority-based, randomized, optimization-based, and game theoretic-based approaches, have been implemented to achieve either a single scheduling objective or a tradeoff of multiple objectives. We categorize the surveyed scheduling methods into four different classes:

- sampling and priority-based methods
- optimization-based methods
- classification-based methods
- game theoretic-based methods

Sampling-based scheduling methods are completely randomized. They are generally used in distributed designs to assign short and latency-sensitive jobs. Optimization-based methods are more sophisticated and they are mostly implemented in centralized systems. They are used to achieve an optimal allocation of resources and meet jobs' hard and soft constraints. Optimization-based scheduling decisions

are too slow and as a result they do not meet short jobs requirements. Classification-based scheduling methods are prediction-based; they are generally used to speed up the scheduling process while meeting QoS requirements. Finally, game-theoretic approaches are implemented mostly to model fairness. In Table 4, we provide a classification of the main reviewed cluster scheduling methods and their related scheduling objectives.

5 Machine learning approach for cluster scheduling

The new research trend in cloud computing resource management is shifting to developing adaptive scheduling algorithms by using machine learning, which is a promising direction to go. This section explores the different areas where the machine learning approach has been used to address the challenges in cluster scheduling. We split this section into four major areas: (1) learning scheduling policies, (2) host load prediction, (3) workload estimation and performance modeling, and (4) resource demand estimation. We present the representative research works from each research area.

5.1 Learning scheduling policies

The number of active research works on the use of machine learning techniques for automating the resource management process in cloud computing is steadily growing. The main goal of these research works is to reduce the need for manual tuning of the scheduling schemes and achieve dynamic online scheduling of jobs. The proposed research is centered on two main directions:

- Autonomous scheduling: using the deep reinforcement learning methodology to achieve a fully automatic allocation of computing resources.
- Online selection of scheduling heuristics: using supervised learning to learn how to dynamically select an optimal scheduling heuristic among many alternatives.

Table 4 Scheduling Methods

Scheduling approach	Techniques	Objectives
Randomized/priority-based	Sampling-based task assignment	Low-latency
	Task stealing	Low-latency/load balancing
	Adjustable sampling	Low-latency/quality
Optimization-based	ILP	Cluster efficiency/ quality
	Min-cost max-flow (MCMF)	Fairness/load-balancing/ network-aware
Classification-based	Collaborative filtering	Quality
Game theoretic-based	DRF, DRFH, PS-DSF, max-min	Fairness

5.1.1 Autonomous scheduling

Reinforcement learning has recently drawn attention thanks to its successful domain applications. A recent research trend in cloud computing is to leverage deep reinforcement learning (DRL) to fully automate the allocation of computing resources. The key idea is to map the cluster scheduling process into a reinforcement learning framework; the scheduler agent learns how to optimize the placement of incoming jobs(or tasks) based on a defined reward function (e.g., average completion time). The major challenge in the application of this methodology lies in the design of the appropriate action and state spaces that can model the complexity of the cloud environment (e.g., heterogeneity of computing resources, heterogeneity and dynamicity of workload) as well as the appropriate reward function that can reflect one (or more) scheduling objective(s). RL agents do not rely on historical data for learning the optimal scheduling policy contrary to the supervised learning approaches.

Mao et al. [49] was the first work to explore the feasibility of using DRL structure for multi-resource task scheduling in distributed computing systems. The authors introduced an online DRL-based scheduler, called "DeepRM", which achieved comparable performance to several state-of-the-art heuristics, such as Shortest Job First (SJF), and Tetris [29]. The state space in "DeepRM" is described by distinct cluster images, each of which representing the assignment of a job to each of the computing resources (e.g., CPU, memory) for several timesteps ahead. The reward function was designed based on a single objective: average jobs slowdown. Gradient-descent was used for policy learning. In this work, the authors made several assumptions to reduce the problem to a much simpler level. They presumed that computing resources are homogeneous—i.e., the cluster was viewed as a single pool of resources–and jobs were treated as a black box; their internal architectures (e.g., tasks inter-dependencies) were not taken into account.

Subsequent research works have considered more complex cluster scheduling settings and introduced new techniques to achieve scalable DRL-based scheduling.

Orhean et al. [62] proposed a DRL-based framework that takes into account DAG jobs requirements and the cluster state. Hence, the state space defined in this work was larger than "DeepRM" encoding several parameters, such as the load level on each node (i.e., number of running tasks) and the arrangement of the tasks. Two learning algorithms were used: Q-learning and SARSA. Although this model captures a more complex cluster setting, the large state and action spaces restrain the capability of the proposed DRL-based scheduler to manage resources on larger scale (i.e., big number of jobs and machines). The simulation results showed that the proposed scheduler outperforms the Heterogeneous Earliest Finish Time (HEFT) scheduling scheme in small-scale clusters. Mao et al. [50] employed a graph neural network to simplify the representation of the states' information (e.g., inter-dependency constraints, nodes' status, and jobs' status at each DAG stage), and thus, reduce the DRL design complexity. The graph neural network learns how to convert the states' information into embedding vectors; the embedding vector is then used by the DRL framework (Decima) for learning an optimal scheduling policy. Decima was implemented as a pluggable scheduler on top of Spark to determine how many executors are needed for each incoming job in an online manner. The evaluation results

show the effectiveness of the proposed scheduler compared to several baseline heuristics (e.g., FIFO, short-job-first critical path heuristic, fair sharing, Tetris, and others) in minimizing the average job completion times. Cheong et al. [7] proposed an attention-based DRL job scheduler, called "SCARL". This study also addresses the large operating cluster state representation problem. An attention-based translator is added to the DRL to transform the input raw data (e.g., jobs and nodes parameters) into embedding vectors that can capture long-term dependencies between jobs. This mechanism is effective to learn dynamic representations of the operating machines and jobs, which keep changing over time. The obtained embedding vectors are then used to learn the optimal scheduling policy through Q-learning. The goal is to minimize job completion times.

Most of the above works focus on a single objective which is typically minimizing average jobs' completion times. Wang et al. [91] is one of the few works addressing multiple objectives within a RL scheduling framework. The authors proposed a multi-agent based RL framework for scheduling DAG applications in IaaS systems. The proposed scheduler aims to find the tradeoff between two conflicting objectives: jobs' execution cost and completion times. The tradeoff between these objectives is achieved by modeling them in a Markov game model.

Other research studies used hybrid approaches; for instance, applying both supervised learning and reinforcement learning in a unified job scheduling framework. Harmony [3] is a RL-based scheduler aiming at minimizing the interference issues arising from co-locating jobs on the same servers in GPU clusters. The proposed scheduler uses a more complex reward function by training a deep supervised learning model. The supervised neural network used to model the reward function takes two input data types: (1) jobs/ machines availability, and (2) jobs' placement decisions. The model predicts in return the jobs' speed (defined reward). Then, the RL framework uses the samples generated by the reward model to learn the optimal scheduling policy that minimizes jobs' execution times. The framework was evaluated on a production GPU cluster and the results showed that Harmony can reduce the average job completion time by 25% compared to Tetris scheduler [29].

DL2 [66] is an adaptive RL-based scheduler dedicated for machine learning clusters. The scheduler also employs a supervised learning/reinforcement learning hybrid approach. The scheduling system consists of two main modules: an offline supervised deep neural network module and an online reinforcement learning module. The supervised deep neural network is trained using training data about past jobs' properties and past scheduling decisions to learn an optimal placement of jobs. The DRL framework is trained online using real-time feedback from the cluster to adapt the scheduling process according to the workload properties. The offline scheduler is used in the early phase of the online RL training process to avoid the poor scheduling decisions made by the RL agent before it converges to the optimal scheduling policy.

5.1.2 Online selection of scheduling heuristics

Several studies indicate the importance of switching scheduling policies in real-time to achieve performance improvements [73]. However, only a few studies have used

machine learning for this task. Online policy selection methods (policy selectors) proceed as follows: a prediction module is implemented to learn how to select the appropriate scheduling heuristic given historical workload data (and/or other features such as the scheduling objective) and past scheduling decisions. Then, given the running workloads (and possibly the cluster state) at a specific scheduling time window, the trained predictor decides which scheduling heuristic to use among a list of alternatives. The Supervised classification methods are typically used in the learning process. This approach is useful for achieving a tradeoff between multiple conflicting objectives where each objective requires a different scheduling strategy to optimize. Gemini [59] and Flex [60] are two typical examples where a dynamic policy selection approach was applied to achieve the tradeoff between fairness and efficiency (defined in terms of makespan).

Gemini [59] uses a regression model to guide the cluster resource manager (in this case, YARN) to select a scheduling heuristic among two alternatives: DRF for achieving fairness and resource imbalance heuristic (i.e., a load-balancing technique) to achieve efficiency. The proposed approach uses the "complementary degree" of the workload and users' fairness SLAs (see Sect. 4.1) as inputs to predict fairness loss and performance improvement. Flex [60] is a general-purpose meta-scheduler deployed on top of YARN and leverages its schedulers (e.g., FIFO, fair schedulers, and others). Flex dynamically selects a scheduler according to the running workload characteristics. To this end, Flex employs a classification-based approach. The policy selection problem is formulated as a multi-class classification problem where each candidate scheduler is represented by a distinct label. Flex decides the appropriate scheduler using decision tree learning.

SantAna et al. [73] proposed an online scheduling policy selection algorithm for HPC clusters. The proposed algorithm selects in real-time a queue ordering heuristic among 8 candidates including First-Come-First-Served (FCFS), Smallest Resource Requirement First (SQF), Smallest Estimated Processing Time (SPF), and others. The authors trained a supervised learning model for the prediction using simulation traces from six HPC clusters. The proposed ML model takes as input statistical features of the existing waiting queues and machine states (e.g., queues' processing time estimation, task waiting times in the queues, number of available processors, and others) and outputs a class corresponding to a scheduling policy. Two classification models were evaluated in this work: Logistic Regression (LR) and Support Vector Machine (SVM). The proposed approach showed that it can reduce the total waiting time in the jobs' queue by up to 40% compared to the FCFS scheduling heuristic. Rjoub and Bentahar [70] developed an ML-based framework to automate the selection of the optimal scheduling policy from a list of three swarm optimization heuristics—Ant colony optimization (ACO), Artificial bee colony optimization (ABC))and particle Swarm optimization (PSO). The proposed system aims at optimizing the system's performance—minimizing the jobs' makespans. The problem was formulated as a multi-label classification problem for which an ensemble chain classifier is proposed. The simulation results using different clusters and task sizes showed that the proposed approach outperforms standard optimization methods and reduces the average makespan by up to 75%.

5.2 Machine learning models for load prediction

Load prediction has drawn a lot of attention in the cloud computing literature. Cloud scheduling researchers have proposed statistical and learning approaches to accurately estimate the load on jobs' and nodes' level. Load prediction plays a central role in the design of future generations of dynamic and proactive resource management systems in cloud computing. However, the problem is arguably more challenging as the load on the cloud systems has much more variance compared to the traditional grid and HPC systems [18]. We investigate research studies proposed for two load prediction tasks: (1) host load prediction and (2) workload prediction.

5.2.1 Host load prediction

Host load prediction refers to estimating the computing resources usage (e.g., CPU and memory) on physical (or virtual) nodes or globally on the cluster. Several approaches were proposed for the problem, including time-series, LSTM and recurrent neural networks, Bayesian models, and others. Early host load prediction works have focused on estimating the CPU load on computing grids and HPC clusters using statistical approaches, such as moving averages, auto-regression, and noise filters or time-series approaches [18]. Due to the diversity of applications running on the cloud environment (see Sect. 2.2), these traditional models fell short of achieving accurate predictions. Researchers are moving towards using deep neural networks to improve the prediction accuracy in this environment.

For instance, Song et al. [81], Mason et al. [51], and Yang et al. [95] employed end-to-end deep learning models to predict future hosts' load. The advantage of using deep neural networks is that no manual feature engineering is needed to process historical jobs' traces. The deep learning approach takes as input the historical load information—often represented in one-dimensional time series—and predicts resource usage (or mean resource usage) values in future time intervals.

Yang et al. [95] proposed an RNN-based Echo State Network (ESN) to predict CPU usage in cloud computing systems using Google trace data. The authors introduced an auto-encoder neural network, which was used to learn high-level features from the input CPU usage data and construct a pre-recurrent feature layer of the ESN. The approach achieved higher accuracy than several baseline models, including auto-regressive, bayesian, and simple ANN models in actual and average CPU loads over several time windows.

Song et al. [81] proposed a LSTM-based recurrent neural network to predict future CPU load. The evaluation results using Google usage trace displayed the effectiveness of the proposed LSTM-based model in accurately predicting in advance future actual and mean CPU loads compared to the ESN approach. Mason et al. [51] proposed a CPU consumption prediction model for IaaS cloud systems using an evolutionary neural network. They used a recurrent neural network architecture and employed three evolutionary optimization techniques (PSO, co-variance matrix adaptation, and differential evolution) to optimize the neural network weights during training. Nguyen et al. [57] developed an LSTM-based encoder-decoder

(LSTM-ED) model. The proposed model used an architecture that consists of a pair of LSTM modules: an encoder and a decoder. On the one hand, the encoder learns a compressed representation of the key information from the input data (e.g., dependencies). On the other hand, the decoder decodes the learned representation to feed it back to the output layer. This method showed a better generalization capability than the previous RNN-based models using Google trace data.

Di et al. [51] proposed a bayesian-based method to predict load fluctuation patterns over a relatively long-term period up to 16 h, focusing on two critical metrics: mean CPU and memory load values. Using Google's one-month usage trace, the authors extracted 10 useful features that capture the key predictive properties, such as previous mean loads, fairness index, last load, etc. The Bayesian classifier uses these features to predict a load level among a defined range of 50 load level classes. Baig et al. [34] used a multi-model approach to adaptively select a resource consumption predictor among alternative models, including Linear Regression (LR), Support Vector Machine (SVM), Kriging (KR), Gradient Boosting Tree (GBT), and others. A random decision forest (RDF) is trained to learn which model yields the lowest prediction error on each prediction time window instead of employing a single prediction model for all time windows.

5.2.2 Workload prediction

Workload prediction refers to estimating the resource usage pattern of a given task/job over future time intervals. Using historical jobs' runtime information to estimate future usage patterns can significantly benefit cluster scheduling systems, which in turn will be able to make informed, and thus, more robust scheduling decisions without relying on hard-coded assumptions. Yu et al. [96] used a clustering-based learning approach to predict short tasks' future workload. The authors used the K-medoid algorithm to create groups of workload usage patterns from the historical tasks' traces. Then, a deep neural network model is trained for each cluster. Once a new task is submitted, the system identifies to which cluster it belongs and predicts its future usage pattern using the cluster's trained neural networks.

CORP [47] integrates deep learning and Hidden Markov Modeling (HMM) into its opportunistic task scheduling. CORP achieves higher cluster utilization by reallocating short tasks to the unused resources. Hence, the scheduling system relies on a deep neural network to predict temporarily allocated but unused resources of running jobs over future time intervals. The authors introduce hidden Markov model to enhance the prediction accuracy of the deep learning model, which sometimes fails to predict unexpected usage fluctuations. The experimental results based on data collected from a real Amazon EC2 cluster showed that packing tasks using deep learning-based prediction of unused resources leads to higher cluster utilization than several baseline schedulers.

5.3 Machine learning for performance modeling

Modern cloud computing clusters consist of computing resources with different configurations. The performance of cloud applications, particularly advanced analytics applications, is sensitive to the amount and types of allocated resources. For instance, as illustrated in [1], the performance of a regression job running on a Spark cluster—in terms of running time—significantly diminishes on a 120 GB RAM machine configuration compared to 256 GB RAM configuration. Also, it is more expensive—i.e., execution cost—to run the regression job on a cluster of 10 hosts than a cluster of 15 hosts. Therefore, identifying the optimal configuration (e.g., the optimal number of VMs necessary to run an application and the optimal VM configurations) for each submitted job has become essential to reduce execution costs and guarantee performance requirements in cloud computing.

Ernest [88] uses a statistical approach to build performance models for advanced analytics jobs. Ernest runs many instances of each incoming job on a small sample of input data (i.e., computing resources' configurations) in an offline manner and uses the derived performance values (in terms of execution costs and running times) to build a prediction model. Ernest applies an "optimal experiment design" approach to minimize the sizes of the training input samples, hence, reducing the training time overhead. Ernest also uses non-negative least squares (NNLS) to fit its performance model.

CherryPick [1] uses a Bayesian optimization-based approach to select the optimal (or near-optimal) VM configurations that minimize the execution costs and running times for recurring big data analytics jobs. Instead of accurately predicting the optimal VM configuration for a data-analytic job, CherryPick uses an approximate solution by calculating "confidence intervals" based on samples of few VMs. The goal is to reduce the extremely large search space to avoid the computational overhead.

The interference caused by co-locating jobs on the same host (see Sect. 4.3)is another significant issue that might affect the performance of jobs on production cloud computing clusters. Quasar [14] and Paragon [13]—a class of QoS-aware schedulers (see Sect. 4.3)—use collaborative filtering to profile the incoming applications (during their early stage) based on similar previously-scheduled applications. Collaborative-based schedulers construct a sparse utility matrix in a similar way to the collaborative filtering-based recommendation systems. For instance, in [13], the matrix rows (users) represent a set of applications while the matrix columns (preferences) represent the different servers' configurations. The matrix values (ratings) correspond to the performance of the applications on each server (e.g., instructions committed per second). Singular Value Decomposition (SVD) is applied to extract hidden similarities between applications' preferences.

More recently, Meyer et al. [54] proposed an interference-aware scheduling approach that combines clustering and classification techniques. The proposed method collects the interference metrics of a target application based on its behavior during runtime within a given period. A support vector machine (SVM) classifier is then used to categorize the interference level among four possible levels—that is, absent, low, moderate, and high—over each resource type class (e.g., CPU, memory, caches, etc). The SVM results (i.e., queues of interference levels on each resource

type) are used as an input sent to a K-Means clustering algorithm to identify the job's overall interference level .

Li et al. [46] developed a deep neural network model in their GPU scheduler (DeepSys) to build a training speed models for deep learning applications in cloud computing. A speed prediction model for a given deep learning job takes the resource allocation and the task placement configurations as input features and outputs its estimated completion time. The estimated speed is used by the scheduler to minimize tasks' average completion time.

Rjoub et al. [71] employed clustering techniques within their trust-aware scheduling system (BigTrustScheduling). The scheduler determines the tasks' priority, according to their execution costs and resource requirements, so that high-priority tasks are assigned to the highly trusted virtual machines. The authors used Percentile and K-Mean methods for clustering tasks based on their requirements and costs, respectively.

5.4 Discussion

In summary, several machine learning techniques were proposed to address different tasks related to the cluster scheduling problem: dynamic selection of scheduling policy, autonomous scheduling, host load prediction, performance modeling, and workload prediction. Researchers have used different machine learning methodologies:

- Deep reinforcement learning: the cluster scheduling is mapped to a deep reinforcement learning framework in several research works. The scheduler agent learns the optimal scheduling policy using a predefined reward function (e.g., average makespan). Researchers usually define the state and action spaces, reward functions, and other parameters based on the characteristics of the scheduling problem they are targeting.
- Supervised learning: supervised learning techniques are widely used in host load prediction, workload prediction, performance modeling, and tasks' classification. For instance, a supervised learning model for host load prediction takes the historical load data (or any other feature type) as input and outputs a load value during a given time window. Researchers have employed two types of supervised learning algorithms: regression (i.e., the output is the actual load value) and classification (i.e., the output is a load level, for instance, low or high levels).
- Unsupervised learning: Unsupervised learning techniques are less common in cluster scheduling. A few research studies have employed clustering techniques, such as K-Means, to group running tasks and jobs.

Since Mao et al. [49] study, a significant number of research works employed deep reinforcement learning, addressing different variants of the cluster scheduling problem. One of the main challenges in using DRL is the extremely large state and action spaces that need to be defined to represent the cluster scheduling problem. For instance, to map a DAG scheduling problem to a DRL framework, the state and action spaces should capture the jobs' inter-task dependencies relations,

nodes' properties, nodes states, and others. The heterogeneity of computing clusters adds another layer of complexity to the problem. Many research studies addressed these challenges by integrating several techniques, such as graph neural networks and auto-encoders. Nevertheless, applying a DRL-based approach for scheduling has many practical drawbacks: first, it is time-consuming—it takes some time to converge to an optimal scheduling policy—and second, it requires a lot of computational power. To avoid the low-quality scheduling decisions obtained in the early training phase of the DRL, several works employed a hybrid approach: a simple heuristic (or a supervised deep learning-based scheduler) is used for scheduling until the DRL learns the optimal scheduling strategy. Most of the existing DRL models are trained offline using simulations. However, it is important to explore how to integrate the feedback received from the cluster to adjust the DRL learning policy in an online manner.

Deep learning is a practical approach to use for load prediction tasks. More specifically, recurrent neural networks (RNN) are suitable for sequence modeling tasks. Hence, it is the perfect candidate to apply for load prediction in cluster scheduling. The RNN model consumes sequences of historical observations and output future estimations. LSTM-RNN models outperformed other deep learning models thanks to their ability to learn long-term dependencies. Generally speaking, deep learning models require large volumes of data to achieve accurate results. However, we notice that the absence of enough training data is one of the main limitations of the proposed prediction models. Existing load prediction works are essentially validated based on public benchmarks (e.g., Google one-month trace data set). Exploring the performance of machine learning techniques using case studies and data benchmarks from more real production clusters is needed to advance research in this area.

6 Summary and research challenges

We summarize the main characteristics of the reviewed cluster scheduling systems in Table 5. In this table, we report the scheduling architecture, implemented scheduling schemes, allocation granularity (e.g., single resource type, multiple resource types), major scheduling objectives (e.g., fairness, cluster efficiency, quality, and scheduling speed) and constraints (e.g., locality and placement constraints) for each cluster scheduling system,.

We identify ten research challenges in cluster scheduling: (1) performance-aware scheduling; (2) multi-dimensional allocation of resources; (3) over(under) allocation of resources; (4) trade-offs between scheduling objectives; (5) fairness in cloud computing systems; (6) energy efficiency; (7) conflicting operational requirements; (8) scalability; (9) online scheduling; and (10) DAG scheduling.

Performance-aware scheduling The performance of submitted workloads, in particular data analytics and machine learning applications, is highly sensitive to the allocated resources (e.g., types of virtual machines, number of CPU cores, number of GPU cores, etc). Furthermore, in many cases, placing two applications on the same host negatively affects their performance due to the incurred interference. Hence, performance and heterogeneity-aware schedulers aim to allocate the optimal

Table 5 Cluster management systems: key characteristics

System	Design	Scheduling approaches	Granularity	Objectives	Constraints	Online/offline
YARN [87]	Centralized (queue-based)	Fair scheduler capacity scheduler delay scheduler FIFO	Single resource (CPU) multi-dimensional (CPU and mem)	Fairness	Data locality	Offline
Borg [89]	Centralized (queue-based)	Coarse-grained allocation (based on quotas) filtering-scoring	Multi-dimensional (all resource types)	Cluster efficiency load balancing	Placementjob priorities	Offline
Tetris [29]	Centralized (queue-based)	fFairness knob dot-product SRTF	Multi-dimensional (CPU,mem,bandwidth,disk)	Fairness performance throughput	–	Online
Firmament [28]	Centralized (flow-based)	MCMF optimization	Multi-dimensional (CPU, mem, bandwidth)	Fairness load-balancing bandwidth-aware	Data locality placement	Online
OpenStack [80]	Centralized	Filtering-scoring	Multi-dimensional (all resource types)	Load balancing QoS requirements	Placement availability locality affinity	Offline
Mesos [31]	Two-level (offer-based)	DRF FIFO	Multi-dimensional (CPU and mem)	Fairness scalability	–	Offline
Apollo [4]	Shared-state distributed	Token-based allocation wait-time matrix opportunistic placement	Multi-dimensional (CPU and mem)	Cluster efficiency throughput fairness	Placement	Offline
Sparrow [63]	Fully distributed	Priority-based schemes (FIFO/ EDF/ SJF...) sampling-based placement	–	Scheduling speed	Data locality placement	Offline
Tarcil [15]	Shared-state distributed	Adjustable Sampling	Multi-dimensional (CPU and mem)	Scheduling speed scheduling quality	Interference preference	Offline
Hawk [12]	Fully hybrid	Sampling-based placement task stealing	–	Scheduling speed	–	Offline
Eagle [11]	Shared-state hybrid	Least work left scheduling sampling-based placement	–	Scheduling speed	–	Offline

Table 5 (continued)

System	Design	Scheduling approaches	Granularity	Objectives	Constraints	Online/offline
Gemini [59]	Centralized	Regression-based trade-off model	Multi-dimensional (all resource types)	Fairness load balancing	–	Offline
MEDEA [23]	Two-scheduler	ILP-based scheduling	Multi-dimensional(all resource types)	Efficiency/performance	Affinity anti-affinity interdependednc y	Offline
TetriSched [86]	Centralized	ILP-based scheduling	Multi-dimensional(all resource types)	Performance	Deadline preference affinity and locality	Offline

amount (and types) of computing resources to the jobs. Machine learning-based performance modeling techniques play a central role in overcoming this challenge.

Multi-dimensionality of resource demands Traditionally, computing resources are allocated on a single resource type basis, typically CPU or memory (e.g., DRF [24]); other relevant resource types, such as GPU, HD r/w, and network bandwidth generally remain unconstrained. In a multi-tenant environment, when there are resources which are not controlled by the cluster scheduler, some jobs can overuse unconstrained resources causing a noisy neighbor problem to the jobs sharing the same nodes. In worse cases, those noisy jobs can be possibly launched not only by legitimate users but also by malicious users intended for denial-of-service attacks. Even a small amount of compromised capacity can be used to launch a DoS attack and cause a significant drop in Hadoop performance [32].

Over-allocation (under) allocation of resources Several research studies have noted that computing resources in data centers are either under-allocated or over-allocated (e.g., observations from Google trace). An accurate prediction of jobs' demands and resource consumption can benefit the resource allocation process and improve cluster utilization. Machine learning models also play a central role in this research direction.

Tradeoff between multiple objectives The cluster scheduling problem is inherently a multi-objective problem due to the diversity of users and cloud providers' requirements. These requirements are sometimes contradicting. Cloud scheduling researchers have investigated how to balance multiple objectives (e.g., fairness/efficiency trade-off, fairness/performance trade-off, and efficiency/performance trade-off). It is still one of the major research problems in cloud computing.

Fair allocation of resources Fairness is a highly desired property in cloud computing primarily because of the shared nature of cloud computing paradigms. Current fairness models are static; they do not consider relevant factors, such as the diversity of computing resources, users' preferences, jobs' priorities, and workloads' dynamicity. Fairness should also be defined in conjunction with other objectives, such as efficiency, over longer time intervals.

Energy efficiency Cloud computing data centers consume a significant amount of energy. For instance, in 2013, U.S. data centers consumed alone 2.2% of the country's total electricity consumption [58]. Energy efficiency is one of the major concerns for cloud providers. Machine learning techniques can be leveraged in several ways towards achieving this goal; for instance, dynamically adjusting load-balancing strategies using a ML-based policy selector, efficient packing of tasks based on accurate prediction of resource consumption over future time windows, and others.

Conflicting operational requirements The diversity of applications in cloud computing systems results in different operational requirements, which adds further complexity to the scheduling process. For instance, short-running jobs (e.g., short-live queries in online data processing applications) require fast scheduling. However, long-running jobs require high-quality allocation of resources for long-duration. An efficient scheduler should handle all jobs' requirements.

Scalability Given the increasing demands for cloud technologies and services, cloud computing systems face a scalability challenge. One of the main motivations behind decentralizing the scheduling architectures (moving towards distributed and

two-level designs) is the scalability bottleneck of centralized schedulers. Exploring new scheduling designs is needed.

Online scheduling Most of the scheduling systems reviewed in this research employ scheduling heuristics in an offline manner. However, it is not a practical approach in real production systems. Cluster scheduling systems should be able to allocate resources and assign jobs in real-time. Besides, they should be able to continuously update their scheduling models to adapt with the workload change. Machine learning techniques can be used for dynamic and adaptive scheduling.

DAG scheduling DAG scheduling is a special category of task scheduling in cloud computing. Several application types, such as data analytics and scientific workflows applications, are structured in direct acyclic graphs (DAG). Scheduling DAG jobs is non-trivial; the scheduler must consider the complex dependency constraints between tasks.

Although a wide range of scheduling techniques has been proposed in the literature including advanced optimization techniques, production cluster management systems are still relying on simple heuristics to allocate resources and assign jobs, which ultimately fail to cope with the complexity in the scheduling process. To address the above scheduling challenges, hybrid approaches combining machine learning (e.g., deep learning, reinforcement learning, and others) and optimization algorithms (e.g., heuristics and/or meta-heuristic) are needed.

7 Conclusion

In this paper, we surveyed a representative cluster scheduling systems from different design categories, including centralized, decentralized, distributed, and hybrid designs. We observed a shift from the traditional centralized schedulers to distributed and hybrid schedulers. This shift is mainly driven by the emerging characteristics of nowadays' cloud computing systems: (1) the unprecedented high level of heterogeneity of jobs ranging from long-run to sub-second jobs and requiring different types and sizes of computing resources; (2) the dynamicity of resource demands and hardware configurations; and (3) the multi-tenancy feature of cloud computing, and (4) the scalability requirements. It will be very challenging if not impossible to use a single scheduling design to meet all of the needs of cluster resource management for today's cloud computing. We identified ten scheduling challenges in the literature representing current research directions including performance-aware scheduling, multi-dimensional allocation of resources, over(under) allocation of resources, trade-offs between scheduling objectives, scalability to the demands, conflicting operational requirements, online scheduling, and DAG scheduling. Researchers have developed a plethora of techniques, ranging from simple randomized heuristics to advanced optimization and machine learning techniques, to deal with these challenges.

The main stream of research on cluster resource scheduling has focused on designing scalable and heterogeneity-aware scheduling architectures and algorithms. The new research trend is shifting to develop adaptive scheduling algorithms by using machine learning, which is a promising area to explore.

Acknowledgements This work is partially supported by the National Science Foundation under grant CNS-1828593.

References

1. Alipourfard O, Liu HH, Chen J, Venkataraman S, Yu M, Zhang M (2017) Cherrypick: adaptively unearthing the best cloud configurations for big data analytics. In: 14th {USENIX} symposium on networked systems design and implementation ({NSDI} 17), pp 469–482
2. Asch M, Moore T, Badia R, Beck M, Beckman P, Bidot T, Bodin F, Cappello F, Choudhary A, de Supinski B et al (2018) Big data and extreme-scale computing: pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry. Int J High Perform Comput Appl 32(4):435–479
3. Bao Y, Peng Y, Wu C (2019)Deep learning-based job placement in distributed machine learning clusters. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, pp 505–513
4. Boutin E, Ekanayake J, Lin W, Shi B, Zhou J, Qian Z, Wu M, Zhou L (2014) Apollo: scalable and coordinated scheduling for cloud-scale computing. In: 11th USENIX symposium on operating systems design and implementation (OSDI 14), pp 285–300
5. Cambridge U (2016) The evolution of cluster scheduler architectures. http://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2016-03-09-scheduler-architectures.html
6. Chen G, He W, Liu J, Nath S, Rigas L, Xiao L, Zhao F (2008) Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: NSDI, vol 8, pp 337–350
7. Cheong M, Lee H, Yeom I, Woo H (2019) Scarl: attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster. IEEE Access 7:153432–153444
8. Chronos: Chronos: a fault tolerant job scheduler for mesos which handles dependencies and iso8601 based schedules. https://mesos.github.io/chronos/docs/
9. Cortez E, Bonde A, Muzio A, Russinovich M, Fontoura M, Bianchini R (2017) Resource central: understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th symposium on operating systems principles, pp 153–167
10. Delgado P, Didona D, Dinu F, Zwaenepoel, W.:ACM, (2016) Job-aware scheduling in eagle: divide and stick to your probes. In: Proceedings of the seventh ACM symposium on cloud computing. ACM, pp 497–509
11. Delgado P, Dinu F, Didona D, Zwaenepoel W (2016) Eagle: a better hybrid data center scheduler. Tech, Rep
12. Delgado P, Dinu F, Kermarrec AM, Zwaenepoel W (2015) Hawk: hybrid datacenter scheduling. In: 2015 USENIX Annual Technical Conference (USENIX ATC 15), pp 499–510
13. Delimitrou C, Kozyrakis C (2013) Paragon: Qos-aware scheduling for heterogeneous datacenters. ACM SIGPLAN Notices 48(4):77–88
14. Delimitrou C, Kozyrakis C (2014) Quasar: resource-efficient and qos-aware cluster management. ACM SIGPLAN Notices 49(4):127–144
15. Delimitrou C, Sanchez D, Kozyrakis C (2015) Tarcil: reconciling scheduling speed and quality in large shared clusters. In: Proceedings of the sixth ACM symposium on cloud computing. ACM, pp 97–110
16. Di S, Kondo D, Cappello F (2014) Characterizing and modeling cloud applications/jobs on a google data center. J Supercomput 69(1):139–160
17. Di S, Kondo D, Cirne W (2012) Characterization and comparison of cloud versus grid workloads. In: 2012 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 230–238
18. Di S, Kondo D, Cirne W (2014) Google hostload prediction based on bayesian model with optimized feature combination. J Parallel Distrib Comput 74(1):1820–1832
19. Dimopoulos S, Krintz C, Wolski R (2017) Justice: a deadline-aware, fair-share resource allocator for implementing multi-analytics. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 233–244
20. Dong Z, Zhuang W, Rojas-Cessa R (2014) Energy-aware scheduling schemes for cloud data centers on google trace data. In: 2014 IEEE Online Conference on Green Communications (OnlineGreenComm). IEEE, pp 1–6

21. flink: Apache flink. https://flink.apache.org/
22. Foundation AS (2012) Hadoop: fair scheduler. https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
23. Garefalakis P, Karanasos K, Pietzuch PR, Suresh A, Rao S (2018) Medea: scheduling of long running applications in shared production clusters. In: EuroSys, pp 1–13
24. Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: fair allocation of multiple resource types. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, pp 323–336
25. Ghodsi A, Zaharia M, Shenker S, Stoica I (2013) Choosy: max-min fair sharing for choosy: max-min fair sharing for data-center jobs with constraints. In: Proceedings of the 8th ACM European Conference on Computer Systems. ACM, pp 365–378
26. Ghomi EJ, Rahmani AM, Qader NN (2017) Load-balancing algorithms in cloud computing: a survey. J Netw Comput Appl 88:50–71
27. github: google/cluster-data. https://github.com/google/cluster-data
28. Gog I, Schwarzkopf M, Gleave A, Watson RN, Hand S (2016) Firmament: fast, centralized cluster scheduling at scale. In: 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), pp 99–115
29. Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A (2014) Multi-resource packing for cluster schedulers. ACM SIGCOMM Comput Commun Rev 44(4):455–466
30. Guo J, Chang Z, Wang S, Ding H, Feng Y, Mao L, Bao Y (2019) Who limits the resource efficiency of my datacenter: an analysis of alibaba datacenter traces. In: 2019 IEEE/ACM 27th international symposium on quality of service (IWQoS). IEEE, pp 1–10
31. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, pp 295–308
32. Huang J, Nicol DM, Campbell RH (2014)Denial-of-service threat to hadoop/yarn clusters with multitenancy. In: 2014 IEEE international congress on big data (BigData Congress). IEEE, pp 48–55
33. Inc D (2019) Docker documentation. https://docs.docker.com/
34. Iqbal W, Berral JL, Erradi A, Carrera D et al (2019) Adaptive prediction models for data center resources utilization estimation. IEEE Trans Netw Serv Manage 16(4):1681–1693
35. Isard M, Prabhakaran V, Currey J, Wieder U, Talwar K, Goldberg A (2009) Quincy:fair scheduling for distributed computing clusters. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles. ACM, pp 261–276
36. Jennings B, Stadler R (2015) Resource management in clouds: survey and research challenges. J Netw Syst Manage 23(3):567–619
37. Jiang C, Han G, Lin J, Jia G, Shi W, Wan J (2019) Characteristics of co-allocated online services and batch jobs in internet data centers: a case study from alibaba cloud. IEEE Access 7:22495–22508
38. Karanasos K, Rao S, Curino C, Douglas C, Chaliparambil K, Fumarola GM, Heddaya S, Ramakrishnan R, Sakalanaga S (2015) Mercury: hybrid centralized and distributed scheduling in large shared clusters. In: USENIX Annual Technical Conference, pp 485–497
39. Kaufmann M, Kourtis K, Schuepbach A, Zitterbart, M (2018) Mira: sharing resources for distributed analytics at small timescales. In: IEEE International Conference on Big Data. IEEE
40. Kaur K, Kumar N, Garg S, Rodrigues JJ (2018) Enloc: data locality-aware energy efficient scheduling scheme for cloud data centers. In: 2018 IEEE International Conference on Communications (ICC). IEEE, pp 1–6
41. Keahey K, Parashar M (2014) Enabling on-demand science via cloud computing. IEEE Cloud Comput 1(1):21–27
42. Khamse-Ashari J, Lambadaris I, Kesidis G, Urgaonkar B, Zhao Y (2017) Per-server dominant-share fairness (ps-dsf): a multi-resource fair allocation mechanism for heterogeneous servers. In: 2017 IEEE International Conference on Communications (ICC). IEEE, pp 1–7
43. kubernetes: kube-scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/
44. kubernetes: Production-grade container orchestration. https://kubernetes.io/
45. Lee G, Katz RH (2011) Heterogeneity-aware resource allocation and scheduling in the cloud. In: HotCloud
46. Li Q, Xu J, Cao C (2020) Scheduling distributed deep learning jobs in heterogeneous cluster with placement awareness. In: 12th Asia-Pacific symposium on internetware, pp 217–228

47. Liu J, Shen H, Chen L (2016) Corp: cooperative opportunistic resource provisioning for short-lived jobs in cloud systems. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 90–99
48. Liu Z, Cho S (2012) Characterizing machines and workloads on a google cluster. In: 2012 41st International Conference on Parallel Processing Workshops (ICPPW). IEEE, pp 397–403
49. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM workshop on hot topics in networks. ACM, pp 50–56
50. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM special interest group on data communication, pp 270–288
51. Mason K, Duggan M, Barrett E, Duggan J, Howley E (2018) Predicting host cpu utilization in the cloud using evolutionary neural networks. Future Generation Comput Syst 86:162–173
52. Mell P, Grance T et al (2011) The nist definition of cloud computing. Computer security division. Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg
53. Mesosphere I (2018) Marathon: a container orchestration platform for mesos and dc/os. https://mesosphere.github.io/marathon/
54. Meyer V, Kirchoff DF, da Silva ML, De Rose CA (2020) An interference-aware application classifier based on machine learning to improve scheduling in clouds. In: 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE, pp 80–87
55. Moritz P, Nishihara R, Wang S, Tumanov A, Liaw R, Liang E, Elibol M, Yang Z, Pau, W, Jordan MI et al (2018) Ray: a distributed framework for emerging {AI} applications. In: 13th {USENIX} symposium on operating systems design and implementation ({OSDI} 18), pp 561–577
56. Nair V (2016) Quality of service for hadoop: it's about time. https://www.oreilly.com/ideas/quality-of-service-for-hadoop-its-about-time
57. Nguyen HM, Kalra G, Kim D (2019) Host load prediction in cloud computing using long short-term memory encoder-decoder. J Supercomput 75(11):7592–7605
58. Nishtala R, Carpenter P, Petrucci V, Martorell X (2017) The hipster approach for improving cloud system efficiency. ACM Trans Comput Syst (TOCS) 35(3):8
59. Niu Z, Tang S, He B (2015) Gemini: An adaptive performance-fairness scheduler for data-intensive cluster computing. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp 66–73
60. Niu Z, Tang S, He B (2016) An adaptive efficiency-fairness meta-scheduler for data-intensive computing. IEEE Trans Serv Comput 12(6):865–879
61. openstack: openstack. https://www.openstack.org/
62. Orhean AI, Pop F, Raicu I (2018) New scheduling approach using reinforcement learning for heterogeneous distributed systems. J Parallel Distrib Comput 117:292–302
63. Ousterhout K, Wendell P, Zaharia M, Stoica, I (2013) Sparrow: distributed, low latency scheduling. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM, pp 69–84
64. Park G (2011) A generalization of multiple choice balls-into-bins. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing, pp. 297–298. ACM (2011)
65. Parkes DC, Procaccia AD, Shah N (2015) Beyond dominant resource fairness: extensions, limitations, and indivisibilities. ACM Trans Econ Comput 3(1):3
66. Peng Y, Bao Y, Chen Y, Wu C, Meng C, Lin W (2021) Dl2: a deep learning-driven scheduler for deep learning clusters. IEEE Trans Parallel Distrib Syst 32(8):1947–1960
67. Piraghaj SF, Dastjerdi AV, Calheiros RN, Buyya R (2015) A framework and algorithm for energy efficient container consolidation in cloud data centers. In: 2015 IEEE International Conference on Data Science and Data Intensive Systems. IEEE, pp 368–375
68. Qu H, Mashayekhi O, Terei D, Levis P (2016) Canary: a scheduling architecture for high performance cloud computing. arXiv preprint arXiv:1602.01412
69. Reiss C, Tumanov A, Ganger GR, Katz RH, Kozuch, MA (2012) Heterogeneity and dynamicity of clouds at scale: google trace analysis. In: Proceedings of the third ACM symposium on cloud computing. ACM, p. 7
70. Rjoub G, Bentahar J (2017) Cloud task scheduling based on swarm intelligence and machine learning. In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, pp 272–279

71. Rjoub G, Bentahar J, Wahab OA (2020) Bigtrustscheduling: trust-aware big data task scheduling approach in cloud computing environments. Future Generation Comput Syst 110:1079–1097
72. Rodriguez MA, Buyya R (2019) Container-based cluster orchestration systems: a taxonomy and future directions. Softw Pract Exp 49(5):698–719
73. Sant'Ana L, Carastan-Santos D, Cordeiro D, De Camargo R (2019) Real-time scheduling policy selection from queue and machine states. In: 2019 19th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID). IEEE, pp 381–390
74. Scharf M, Stein M, Voith T, Hilt V (2015) Network-aware instance scheduling in open-stack. In: 2015 24th International Conference on Computer Communication and Networks (ICCCN). IEEE, pp 1–6
75. Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J (2013) Omega: exible, scalable schedulers for large compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems. ACM, pp 351–364
76. Shao Y, Li C, Gu J, Zhang J, Luo Y (2018) Efficient jobs scheduling approach for big data applications. Comput Indus Eng 117:249–261
77. Singh S, Chana I (2016) Cloud resource provisioning: survey, status and future research directions. Knowl Inform Syst 49(3):1005–1069
78. Singh S, Chana I (2016) A survey on resource scheduling in cloud computing: issues and challenges. J Grid Comput 14(2):217–264
79. slurm: slurm workload manager. https://slurm.schedmd.com/documentation.html
80. Software OC Scheduling. https://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html#filter-scheduler
81. Song B, Yu Y, Zhou Y, Wang Z, Du S (2018) Host load prediction with long short-term memory in cloud computing. J Supercomput 74(12):6554–6568
82. Spark A Apache spark. https://spark.apache.org/
83. Talluri S, Łuszczak A, Abad CL, Iosup A (2019) Characterization of a big data storage workload in the cloud. In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, pp 33–44
84. Thinakaran P, Gunasekaran JR, Sharma B, Kandemir MT, Das CR (2017) Phoenix: a constraint-aware scheduler for heterogeneous datacenters. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp 977–987
85. Tumanov A, Cipar J, Ganger GR, Kozuch MA (2012) alsched: algebraic scheduling of mixed workloads in heterogeneous clouds. In: Proceedings of the third ACM symposium on cloud computing. ACM, p 25
86. Tumanov A, Zhu T, Park JW, Kozuch MA, Harchol-Balter M, Ganger GR (2016) Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Proceedings of the Eleventh European Conference on Computer Systems. ACM, p 35
87. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth, S et al (2013) Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. ACM, p 5
88. Venkataraman S, Yang Z, Franklin MJ, Recht B, Stoica I (2016) Ernest: efficient performance prediction for large-scale advanced analytics. In: NSDI, pp 363–378
89. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes, J (2015) Large-scale cluster management at google with borg. In: Proceedings of the tenth European Conference on Computer Systems. ACM, p 18
90. Wang W, Li B, Liang B (2014)Dominant resource fairness in cloud computing systems with heterogeneous servers. In: INFOCOM, 2014 Proceedings IEEE. IEEE, pp 583–591
91. Wang Y, Liu H, Zheng W, Xia Y, Li Y, Chen P, Guo K, Xie H (2019) Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE Access 7:39974–39982
92. Weerasiri D, Barukh MC, Benatallah B, Sheng QZ, Ranjan R (2017) A taxonomy and survey of cloud resource orchestration techniques. ACM Comput Surv (CSUR) 50(2):1–41
93. White T (2012) Hadoop: The definitive guide. "O'Reilly Media, Inc.",
94. Wu F, Wu Q, Tan Y (2015) Workflow scheduling in cloud: a survey. J Supercomput 71(9):3373–3418
95. Yang Q, Zhou Y, Yu Y, Yuan J, Xing X, Du S (2015) Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing. J Supercomput 71(8):3037–3053

96. Yu Y, Jindal V, Yen IL, Bastani F (2016) Integrating clustering and learning for improved workload prediction in the cloud. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). IEEE, pp 876–879
97. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. HotCloud 10:95
98. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ et al (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65