A Survey of Scheduling Frameworks in Big Data Systems

Ji Liu, Esther Pacitti, Patrick Valduriez

A Survey of Scheduling Frameworks in Big Data Systems

Ji Liu

Inria and LIRMM, University of Montpellier, France
E-mail: ji.liu@inria.fr

Esther Pacitti

Inria and LIRMM, University of Montpellier, France
E-mail: Esther.Pacitti@lirmm.fr

Patrick Valduriez

Inria and LIRMM, University of Montpellier, France
E-mail: Patrick.Valduriez@inria.fr

Abstract: Cloud and big data technologies are now converging to enable organizations to outsource data in the cloud and get value from data through big data analytics. Big data systems typically exploit computer clusters to gain scalability and obtain a good cost-performance ratio. However, scheduling a workload in a computer cluster remains a well-known open problem. Scheduling methods are typically implemented in a scheduling framework and may have different objectives. In this paper, we survey scheduling methods and frameworks for big data systems, propose a taxonomy and analyze the features of the different categories of scheduling frameworks. These frameworks have been designed initially for the cloud (MapReduce) to process Web data. We examine sixteen popular scheduling frameworks and discuss their features. Our study shows that different frameworks are proposed for different big data systems, different scales of computer clusters and different objectives. We propose the main dimensions for workloads and metrics for benchmarks to evaluate these scheduling frameworks. Finally, we analyze their limitations and propose new research directions.

Keywords: Big data, cloud computing, cluster computing, parallel processing, scheduling method, scheduling framework

Reference to this paper should be made as follows: Liu, J. and Pacitti, E. and Valduriez, P. (xxxx) 'A Survey of Scheduling Frameworks in Big Data Systems', *Int. J. Cloud Computing*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Ji Liu is a postdoctoral researcher at Inria in the Zenith Inria team at LIRMM, Montpellier. His research interests include scientific workflows, big data, cloud computing and distributed data management. He obtained his Ph.D. degree from University of Montpellier in 2016 and a Master degree from Télécom SudParis in 2013. He obtained his BSc degree from Xidian University in 2011. He published several papers in international conferences and journals on scientific workflows and cloud computing.

Esther Pacitti is a professor of computer science at University of Montpellier and a senior researcher in the Zenith Inria team at LIRMM. She obtained her Ph.D. from PUC Rio (Brazil) in 1998 and her "Habilitation à Diriger les Recherches" (HDR) degree in 2008 from University of Nantes. Her research interests include scientific workflows, big data, cloud computing and distributed data management. Previously, she was an associate professor at University of Nantes. She has served or is serving as program committee member of major international conferences (VLDB, SIGMOD, CIKM, EDBT, etc.). She currently serves as associate editor of the DAPD journal. She has authored over 100 technical papers and a book in Synthesis Lectures on Data Management on P2P Techniques for Decentralized Applications.

Patrick Valduriez is a senior researcher at Inria, in the Zenith Inria team at LIRMM, working on distributed and parallel data management. He obtained his Ph.D. and Doctorat d'Etat degrees from University Pierre et Marie Curie (Paris) in 1981 and 1985, respectively. He has been a researcher at MCC, Austin, Texas and a professor of computer science at University Pierre et Marie Curie. He has authored over 250 technical papers and several textbooks, among which "Principles of Distributed Database Systems". He currently serves as associate editor of several journals, including VLDBJ, DAPD and Internet and Databases. He has served as PC chair or general chair of major conferences such as SIGMOD and VLDB. He obtained the best paper award at VLDB00. He was the recipient of the 1993 IBM scientific prize in Computer Science in France and the 2014 Innovation Award from Inria - French Academy of Science - Dassault Systems. He is an ACM Fellow.

1 Introduction

Companies and research organizations capture more and more big data for their customers or experiments. For instance, Facebook processed 4 petabytes of data per day in 2016 and IBM estimates that 2.5 quintillion bytes of data were produced every day in 2017. To deal with big data, computer clusters (clusters for short) composed of nodes, *i.e.* computer servers, are widely used because they provide a good cost-performance ratio and the opportunity to scale out. Thus, big data systems deployed in clusters must efficiently manage nodes to store and process the data. A big data system is a data processing system, which can store and process large amount of data, built upon multiple servers.

Cloud and big data technologies are now converging to promise cost-effective delivery of big data services. The ability to produce high-value information and knowledge from big data makes it critical for many applications such as decision support, forecasting, business intelligence, research, and (data-intensive) science. Cloud computing, on the other hand, encompasses on demand, reliable services provided over the Internet with easy access to virtually infinite computing, storage and networking resources. Cloud computing provides computing, storage and network resources through infrastructure (IaaS), platform (PaaS) and software (SaaS) services, with the illusion that resources are unlimited [LPVM15]. Through simple Web interfaces, users can outsource complex tasks, such as data storage, system administration, or application deployment, to very large data centers operated by cloud providers. Although cloud and big data have different goals (big data aims at added value and operational performance while cloud targets flexibility and reduced cost), they can well help each other by (1) encouraging organizations to outsource more and more

strategic internal data in the cloud and (2) get value out of it (for instance, by integrating with external data) through big data analytics.

Web companies such as Facebook, Google, Microsoft, and Yahoo! as well as open source software foundations such as Apache have designed big data frameworks, which can highly parallelize data processing based on data partitioning, *i.e.* using data-based parallelism. For example, MapReduce [DG08], Hadoop [Had], Cosmos [CJL+08] and Spark [ZCF+10] are big data systems that can process huge amounts of Web data and run high numbers of jobs, *e.g.* MapReduce jobs, DAG-based jobs and MPI jobs (large parallel jobs that use the standard Message Passing Interface). These frameworks have been designed initially as a paradigm for I/O intensive distributed computing on commodity hardware and some may be used for the cloud environments. A big data system stores and processes huge amounts of data by exploiting very big clusters. The data is typically stored in a distributed file system[1], such as GFS [GGL03] or HDFS [Had], in a shared-nothing cluster. Shared-nothing is a highly-scalable architecture with no sharing of either disk or memory among nodes [LPVM15, Val09], *i.e.* each node is composed of processor, main memory and disk and communicates with other nodes through the interconnection network [Val09]. The data is partitioned into multiple chunks and each chunk is stored in a node. A job is a unit of work submitted by users. Each job is composed of several stages linked by data dependencies. A stage is a logical data processing unit, such as a Map or Reduce function in a MapReduce job. Each stage consists of multiple tasks, *i.e.* the components of a job with corresponding data chunks, and each task can be scheduled to a node to be executed. A task is the smallest unit of computation, which can be restarted in case of failure to provide fault-tolerance. A DAG-based job, also called a workflow, is composed of a set of stages and data dependencies that can be described as a Directed Acyclic Graph (DAG) [LPVM15].

In order to efficiently process big data with high numbers of nodes, the resources should be efficiently managed. The big data systems exploit scheduling frameworks, such as Yarn [VMD+13], Omega [SKAEMW13] and Jockey [FBK+12], to schedule the workload to nodes. A scheduling framework in a big data system is a structured software that provides both extensible scheduling functionality (scheduling methods) and related functionalities [FKHM96], *e.g.* gathering the information on tasks, jobs and nodes and the interaction among tasks, jobs and nodes, etc. A scheduling method, as implemented in a scheduling framework, is the combination of policies, rules or algorithms to map the workload to the nodes in order to achieve one ore several objectives. The workload can refer to jobs or tasks according to different scheduling granularities. The scheduling problem is to decide how to allocate nodes or slots of nodes to run the corresponding tasks based on some objective function when given a workload of multiple jobs. This is an NP-hard problem [JBM+15]. A slot defines the time period, the amount of memory and the number of CPUs of a node that can be used to perform execution. The objectives can be to reduce execution time or the amounts of data transferred in the network, etc. The scheduling frameworks can be used within a cloud infrastructure to manage the physical computers or within a big data system build upon virtual machines (VMs) provided by the cloud. In the latter case, the scheduling frameworks need to handle specific aspects of the cloud environment, such as elasticity, by managing VMs that are dynamically added or removed.

Some scheduling methods are useful for big data systems [GPDC15], such as data location based scheduling [ZBS+10], capacity scheduling [WCZ13], etc. Most of them are designed for MapReduce jobs, which schedule Map or Reduce tasks to nodes. But they have many drawbacks. First, they do not consider other types of jobs, such as DAG-based jobs. Second, they cannot efficiently manage high numbers of nodes [ZLT+14] and

tasks. Third, they lack support for some user-defined objectives, such as job completion deadline. Some other scheduling methods are used in scheduling frameworks, such as job isolation scheduling in Heracles [LCG+15] and the scheduling method based on task requirements in DAGPS [GKR+16]. Scheduling frameworks can implement different scheduling methods for diverse objectives. Scheduling frameworks can be categorized according to different elements, such as granularity, architecture, execution time, adaptation to the execution environment, etc. Both makespan and response time represent execution time while makespan is used for batch jobs and response time is used for real-time requests or streaming data processing [FR98]. Although some scheduling frameworks attempt to achieve fairness among different jobs, they use different techniques at the job and task levels. If fairness among different jobs is not obtained, then there may be starvation, *i.e.* some jobs cannot get executed for a long time while the nodes are busy running other long-running jobs. The objectives of scheduling frameworks can depend on the user's perspective or the service provider's perspective.

Scheduling frameworks have been designed for different big data systems. They are targeted at different objectives and scales of clusters with diverse optimization methods. In this paper, we survey sixteen popular scheduling frameworks, developed by companies or communities, which we classify in four types: resource managers, Spark schedulers, Microsoft Cosmos [CJL+08] scheduler and YARN-Based scheduler.

Some benchmarks have been proposed to evaluate scheduling frameworks in big data systems. They typically focus on execution time, throughput of jobs [AMP] or tasks and the financial cost and energy consumption to execute jobs [TPCb]. The throughput represents the number of jobs or tasks that can be processed by the big data system within a time unit. In order to have a comprehensive comparison of scheduling frameworks in big data systems, some other metrics and diverse types of workloads should be examined. For instance, the mixed workloads of small jobs and long running jobs should be considered. Some other important metrics can be the amount of data transferred over the network, the utilization of nodes, the failures of jobs or tasks, etc.

Sakr *et al.* [SLBA11] give a survey of scheduling methods for data-intensive applications in cloud computing environments. However, they do not focus on big data systems. Several surveys have also been proposed [GPDC15, RL11, YS11, SKR15, GPDC15], but they generally focus on the scheduling methods without considering the scheduling architectures, different types of jobs or multiple objectives used in scheduling frameworks. Masdari *et al.* [MVSA16] survey scheduling methods for DAG jobs. However, they focus on task scheduling and do not consider job scheduling. The survey does not consider different scheduling architectures, different types of jobs either. The scheduling process is also related to some other topics that are not addressed, such as permanent service scheduling and network scheduling, where big amounts of data can be processed or transferred. Permanent service scheduling represents the scheduling process for online services, such as Kubernetes [Ren15], Amazon Lambda [KMM+15] and the Azure function scheduler. The scheduling problem in the network is how to map input ports to output ports in order to speed up the switching networks [KKBM13, PBL+16]. In this paper, we focus on the scheduling frameworks, including the scheduling methods, in big data systems and propose a taxonomy of the existing frameworks and compare the analysis of the features of different categories. The main contributions of this paper are as follows:

1. We propose a taxonomy of scheduling frameworks.
2. We analyze the features of different scheduling methods.

3. We analyze and compare different scheduling frameworks.

4. We propose metrics for benchmarks to evaluate scheduling frameworks in big data systems.

5. Finally, we propose new research directions.

The paper is organized as follows. Section 2 describes the execution process in a big data system. Section 3 proposes a taxonomy of scheduling frameworks and analyzes the features of each category. Section 4 introduces and compares 16 existing scheduling frameworks. Section 5 proposes workload dimensions and metrics for benchmarks to evaluate scheduling frameworks in big data systems. Section 6 discusses the limitations of current scheduling frameworks and proposes new directions of research on scheduling in big data systems. Finally, Section 7 summarizes the main findings of this study.

2 Execution in a Big Data System

In this section, we present the execution process, which is composed of job execution and task execution, in a big data system.

2.1 Job Execution

The job execution process is generally composed of four steps [JCM+16, CDD+14, FBK+12], *i.e.* job estimation, resource provisioning (scheduling), job execution and monitoring and resource release.

In the job estimation step, the execution time of a job can be estimated based on a predefined function, according to Amdahl's Law [SC10], by analyzing the historical statistical parameters of previous execution of jobs or using machine learning algorithms [JCM+16]. A machine learning algorithm learns from experience E with respect to some class of tasks T and performance measure P while its performance at tasks in T, as measured by P, improves with experience E [Mit97]. In addition, a critical path method can be used to estimate the execution time of jobs [AD12]. A critical path is a path composed of a set of stages of a job with the longest average execution time, which is equal to the execution time of a job.

With the estimated execution time and specific numbers of nodes, the system can perform resource provisioning, which is the scheduling process at the job level, *i.e.* allocating nodes to the job to perform the execution. In order to meet the deadline requirement of jobs, an over-provisioning method is used at the beginning of the job execution. Over-provisioning allocates more nodes to the job than required and considers the failures during the execution of different tasks. Some jobs, such as MPI jobs, may need a gang of nodes. The execution of such jobs can start only when all the nodes are available. Thus, the system needs to schedule guaranteed nodes to these jobs [CDD+14].

With the allocated nodes, a job can start its execution. Some scheduling frameworks may omit the previous two steps and directly use all the available nodes to execute the job. During execution, big data systems should monitor the execution and guarantee fault-tolerance, *i.e.* rerun the failed jobs. After execution, the allocated nodes are released to be used by other jobs.

2.2 Task Execution

The task execution process generally contains the following steps [BEL+14, GAK+14]: constructing a ready list, task sort, node sort, scheduling, task execution and failure recovery, task completion and job completion. At the beginning of job execution, the corresponding tasks are generated. When the input data of tasks are ready, the tasks are put into a ready list. Then, the tasks are queued and then sorted according to parameters, such as waited time [GAK+14]. In addition, the nodes are also sorted and can be sampled [DSK15]. After sorting the tasks and nodes, the tasks are scheduled to the nodes according to different scheduling methods. Note that a task can be scheduled to a slot of a node [VMD+13]. The execution time of a task, which is used to make smart scheduling decisions, can be estimated by sampling the execution time of previously executed tasks. Then, the task is executed in the scheduled node. During execution, some tasks may be killed [IPC+09] in order to make room for the tasks of other jobs to achieve job fairness. In addition, the execution of some tasks may fail. The failed or killed tasks will generally be put into the ready list or the queue to be re-executed in order to achieve fault-tolerance. Finally, after the execution of all the tasks, the corresponding jobs finish their execution.

3 Taxonomy of Scheduling Frameworks

In big data systems, the problem of scheduling the workload in a cluster remains a well-known NP-hard problem. In this section, we first present scheduling methods. Then, we propose a taxonomy of scheduling frameworks and analyze the features of different categories.

3.1 Scheduling Methods

Different scheduling methods can be implemented in scheduling frameworks. In this section, we present eight important scheduling methods.

The most basic scheduling method used in big data systems is Data Location Based (DLB). In this method, the jobs are scheduled in FIFO [ZBS+10] order, *i.e.* the later submitted jobs have to wait until the end of the execution of earlier jobs. When a node is free, the tasks of the early submitted jobs are scheduled to the nodes where the input data of the tasks are stored.

In order to schedule tasks to where their input data is stored, a matchmaking algorithm [HLS11] or a delay algorithm [ZBS+10] can be used. The matchmaking algorithm gives every node a fair chance to grab the local tasks before other tasks are scheduled to them. The input data of a local task is stored at the corresponding node. The delay algorithm schedules tasks to a node, where its input data is located. If not successful, the scheduling method delays the job execution and searches for a task from succeeding jobs. However, if a job has waited for a long time, then it is executed no matter whether the input data is located at the nodes or not in order to avoid starvation.

The DLB method can reduce the time to transfer data and tries to achieve fairness by data locality, but if the data is not evenly distributed in the nodes, the load of each node may not be balanced and the execution time may be long. However, both the matchmaking and delay algorithms may suffer from hot data spots, *i.e.* some nodes are filled with tasks because of the stored data in the nodes while others remain idle. If most data is stored at

some nodes, the tasks are scheduled to these nodes. Thus, the data generated by the tasks are always stored in the few nodes. As a result, more and more tasks are scheduled at these nodes and more data is stored in the same nodes, which become hot, while the other nodes remain idle. In this situation, the throughput is low and the execution time of a job is long.

The Fair Scheduling (FS) method from Facebook [GPDC15] assigns nodes to jobs such that all jobs have an equal share of nodes. This method can ensure that each job can get its nodes to execute its tasks so that there is no starvation of short running jobs. However, this method may assign small numbers of nodes to a long running job, which may take much time to complete execution.

The Capacity Scheduling (CS) method from Yahoo! [WCZ13] allocates a guaranteed capacity of nodes to each user. The jobs of each user are scheduled in FIFO to the nodes without exceeding the guaranteed capacity of each user. However, this method does not have a global view of all the submitted jobs and cannot generate an optimal or near-optimal scheduling plan.

DLB, FS and CS are simple to implement, but they have several limitations. First, they are YARN-based and have poor performance on complex workloads, such as DAG-based jobs, or workloads with high velocity, such as real-time requests and stream data processing. Second, they do not support important user-defined objectives, such as Service Level Agreement (SLA), deadline and stable services. Third, they have no support for very big clusters, *e.g.* more than dozens of thousands of nodes [ZLT+14]. Some other scheduling methods are designed for scheduling frameworks.

The Function Based (FB) method schedules jobs to nodes based on a predefined function. Based on the predefined function and the parameters of nodes, the jobs are scheduled to the nodes in order to maximize their utilization or to balance the load of a cluster. The Job Isolation (JI) method schedules jobs to different parts of the cluster or isolates the usage of CPU, memory or network resources of the shared nodes for the execution of different types of jobs in order to achieve different user-defined objectives, *e.g.* deadline. This method can satisfy different requirements of different types of jobs. The method to schedule jobs to available nodes (JN) chooses a job based on the job priority and waited time. It is different from FB since it chooses jobs for nodes while FB chooses nodes for jobs. The Task Requirement (TR) method chooses nodes for tasks based on the requirements of tasks, *e.g.* CPU frequency, memory, etc. A weighted score for different requirements is calculated for each node and the node with the highest score is chosen. The Node Quality (NQ) method schedules a task to the node with the best quality, which is evaluated based on data location, minimum finish time, response time, etc. The minimum finish time is the time when the current tasks are executed. This time can be calculated based on the tasks to execute in the queue of that node and the time to calculate the current task.

3.2 Taxonomy of Scheduling Frameworks

The scheduling frameworks can be classified based on different features. In this section, we propose a taxonomy of scheduling frameworks and analyze the features of different categories. The taxonomy is shown in Figure 1.

3.2.1 Granularity

During the scheduling process, the granularity can be job or task. If the granularity is job, the jobs submitted by users will be scheduled to one or multiple nodes. If the granularity is task, each task of a job will be scheduled to a node. Some scheduling frameworks schedule

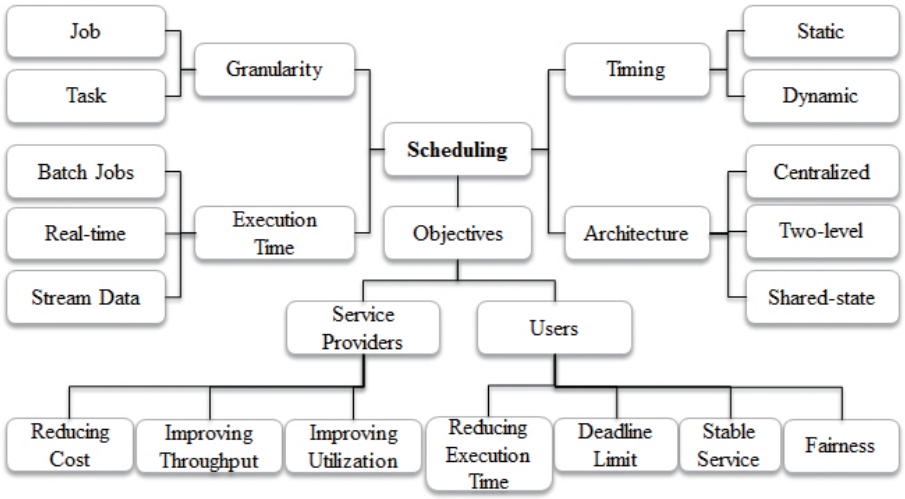

Figure 1: **A Taxonomy of Scheduling frameworks.**

both the jobs and tasks at the same time, which is called hybrid scheduling. If the submitted jobs have special requirements of nodes, the granularity should be job or hybrid [CDD+14].

In addition, the tasks of jobs can be scheduled to nodes in order to achieve load balancing of all the nodes or to fill the chosen nodes tightly [CDD+14]. The load balancing based method schedules the workload to each node so that the workload at each node is balanced. When the jobs are of the same type, and no job needs a gang of nodes, load balancing based methods can be used. However, this type of methods may occupy many nodes in order to execute small jobs. The method with the objective of filling the chosen nodes can make the utilization of selected nodes very high while the number of the occupied nodes remains small. This way, the remaining nodes can be scheduled to other jobs. When the jobs need a gang of nodes, the scheduling frameworks should fill the selected nodes first.

3.2.2 Execution Time

We distinguish the scheduling frameworks according to the requirements of execution time. We classify the jobs into three categories, *i.e.* batch job, real-time request and stream data processing. The execution time of batch jobs can be up to several hours or days while the response time of real-time requests should be a few seconds or milliseconds. The difference between batch jobs and real-time request is the time when the data is processed [Bar13]. If the data is stored in the system and processed at some points in the future, the jobs are batch jobs. If the data is processed at the moment when the data arrive, the jobs are real-time requests. The response time of stream data processing can be several milliseconds or less than one millisecond [CF02]. Stream data processing is also real-time but different since it has continuous incoming data and the process generally exploits a window to capture the data to process [ÖV11]. For batch jobs, the corresponding scheduling frameworks can take much time to generate better scheduling plans so that the total execution time is reduced. A scheduling plan defines the matching relationship between each task or job and nodes. However, the scheduling frameworks of real-time requests or stream data processing should generate scheduling plans within little time since the response time should be very short.

Scheduling frameworks generally take all the nodes as candidates to schedule, but some scheduling frameworks for real-time requests or stream data processing [OWZS13] randomly choose several nodes from all the nodes and schedule the tasks or jobs to one of the chosen nodes, using sampling. The sampling method can reduce the time to generate scheduling plans, *i.e.* scheduling time, which is critical to the real-time requests or stream data processing. However, sampling methods may yield sub-optimal scheduling plans. As a result, a trade-off between the scheduling time and the quality of the scheduling plans should be made by the scheduling framework by analyzing statistical information of previous executions [DSK15].

3.2.3 Scheduling Timing

According to the timing at which scheduling is done, we can distinguish between static and dynamic scheduling. Static scheduling generates scheduling plans before the execution of jobs and the scheduling plans are strictly abided by the execution module during the execution. Because it is before execution, static scheduling yields little overhead at runtime. It is efficient if the system can accurately predict the workload of tasks or jobs, and when the execution environment varies little during the execution, and the system has enough information about the computing and storage capabilities of the corresponding nodes. However, when the execution environment experiences dynamical changes, it is very difficult to achieve load balancing among available nodes with static scheduling.

Dynamic scheduling generates scheduling plans during the execution with the real-time information on the execution environment. This kind of scheduling framework is appropriate for environments where the workload is difficult to estimate, or the capabilities of the nodes vary much during execution. Dynamic scheduling can achieve good load balancing but may increase execution time because of the overhead of doing scheduling. In order to dynamically generate scheduling plans, the frameworks have knowledge of the workload before execution, *i.e.* the execution time limit of jobs [FBK+12], or they may dynamically get the knowledge about the execution environment, *e.g.* the status of nodes [RKK+16] and data location [ZBS+10].

During dynamic scheduling, the scheduling framework needs to provision nodes to jobs. Most frameworks believe in the users' requests and schedule the required number of nodes to the submitted jobs, which is called static provisioning. However, the users may not have enough knowledge of the execution environments or the execution status of the submitted jobs. They may ask for more or less nodes than the real needs of the submitted jobs. Thus, the static provisioning method may incur SLA failures or low efficiency of the scheduled nodes. Some scheduling frameworks estimate the workload of submitted jobs and dynamically provision a proper number of nodes using heuristics or by analyzing the historical execution performance [JCM+16], which is called dynamic provisioning.

Finally, the nodes may be heterogeneous, *i.e.* with differences in CPU, memory or hard disk [RTG+12]. Some scheduling frameworks [IPC+09, OWZS13] treat all the nodes as though they have the same capacity while some other scheduling frameworks [HKZ+11, LCG+15, ZLT+14] take different capacities of each node into consideration to generate better scheduling plans. The scheduling frameworks generally exploit state information to know if nodes have free resources, *e.g.* CPU, memory, or not [VMD+13].

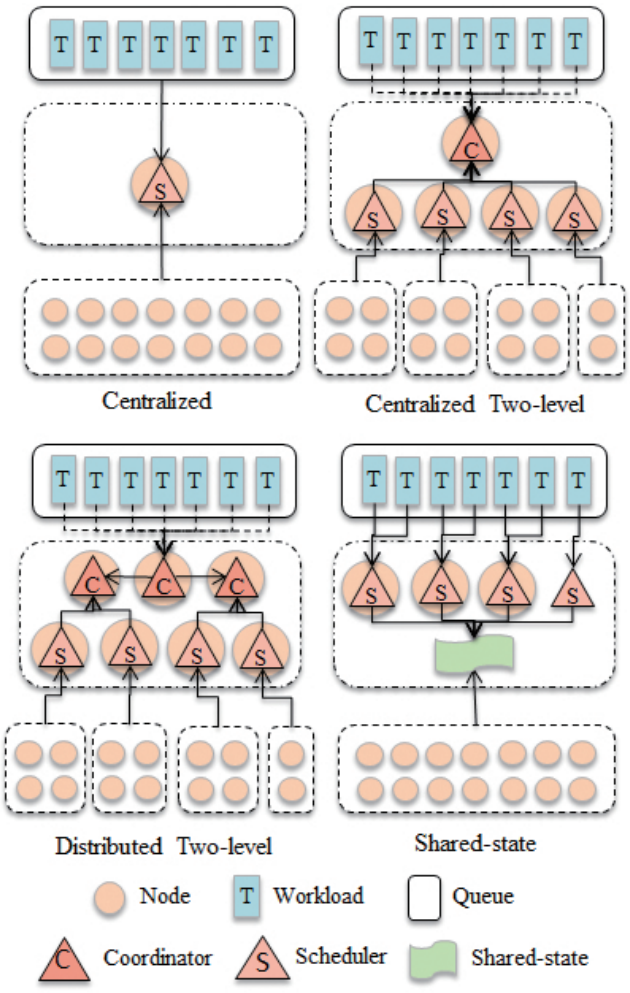

Figure 2: **Architectures of Scheduling Frameworks.**

3.2.4 Architecture

The architecture of scheduling frameworks can be centralized, centralized two-level, distributed two-level or shared-state (see Figure 2). In the centralized architecture, a scheduler, the module that generates scheduling plans, schedules tasks or jobs to nodes. The centralized scheduler has a global view of the whole system and can generate optimal or near-optimal scheduling plans. However, the centralized architecture has poor throughput and scalability since the scheduler may become a single point of congestion in the system. In the two-level architecture, a coordinator module allocates nodes to distributed schedulers. Each scheduler can schedule the tasks of each job to its allocated nodes [DSK15]. Since the coordinator module and the distributed schedulers do not have a global view of the whole system, the generated scheduling plans are suboptimal [SKAEMW13]. In the two-level architecture, the coordinator module can be centralized (centralized two-level) or distributed (distributed two-level). The centralized two-level architecture has one coordinator while the distributed two-level architecture has a centralized coordinator and multiple distributed coordinators to accelerate the process of allocating nodes to distributed schedulers. In the distributed two-level architecture, the centralized coordinator assigns a coordinator to allocate nodes to a job. After allocating nodes to a job by the coordinator, a scheduler is created to schedule the tasks of the jobs to the allocated nodes. In the shared-state architecture, there are distributed schedulers to schedule the tasks of different jobs to nodes. Each scheduler has full access to all nodes and can get all the status of the nodes. There is no coordinator but a master copy of the status of the nodes is maintained. When one scheduler updates the master copy for one node, the other schedulers cannot update the status of the node until the node is released by the occupied task. Since all the schedulers have a global view of the nodes, they can generate optimal scheduling plans for its tasks. But since each scheduler does not consider the tasks scheduled by other schedulers, the final scheduling plans are not optimal in terms of all tasks.

Although only the centralized architecture can generate an optimal or near-optimal scheduling plan, it takes time to schedule all the tasks of different jobs to all the nodes, *i.e.* the latency to generate scheduling plans is high and scalability is poor. In addition, when the jobs are executed within different frameworks, it is complicated and requires expensive refactoring to use a centralized architecture [HKZ+11]. Therefore, the centralized architecture is a good option only if: the jobs need the same big data system to execute, the average execution time of tasks is not short and the number of nodes is small. If the jobs need different big data systems to execute, two-level scheduling can be used. YARN is a scheduling framework with a two-level architecture and the frameworks designed based on YARN also exploit the two-level architecture. However, the two-level scheduling architecture has limited parallelism and restricted visibility of nodes [SKAEMW13]. In order to achieve not only high scalability and high throughput of tasks but also optimal scheduling plans, the shared-state architecture should be exploited. The centralized scheduling architecture is the simplest, since it just requires a centralized scheduler for all the tasks or jobs. Two-level scheduling is somehow more complex to implement, since there should be communication between the centralized coordinator (centralized two-level) or coordinators (distributed two-level) and the distributed schedulers besides the normal scheduling process. Shared-state has the most complex implementation, since it needs to maintain a master copy of the status of all the nodes and each scheduler has to compete to get nodes.

3.2.5 Objectives

There are two kinds of objectives, *i.e.* the user's perspective and the service provider's perspective. From the user's perspective, the SLA should be achieved to fulfill the users' requirements, *e.g.* reducing execution time [GKR+16], executing jobs within the deadline limit [CDD+14], maintaining the stable service and achieving fairness among different jobs. Stable service means that the execution time should be stable so that the users can easily predict the execution time of a job. From the service provider's perspective, the objective can be to reduce computation cost and improve the throughput of jobs [IPC+09] or the utilization of nodes [LCG+15, GKR+16]. The computing cost can be data transfer [IPC+09], which uses the network resources, or power consumption.

The scheduling frameworks should achieve fairness among different jobs in order to solve the starvation problem, which yields bad user experience. Some scheduling frameworks may have potential starvation problems [FBK+12] while others achieve fairness at the job and task levels. At the job level, the system can block long running jobs and schedule the released nodes to the jobs that have waited for a long time. Blocking the jobs can lead to bigger memory or storage occupation to store the running status of the jobs. The system can also reduce allocated nodes for long-running jobs [OWZS13, GKR+16]. In addition, it can allocate user quotas, *i.e.* a number of nodes for each user. At the task level, the system can attribute high priority to the tasks that have waited for a long time [RKK+16] or kill the long running tasks [LCG+15, IPC+09, KRC+15, HKZ+11]. The tasks with high priority will be executed first. Killing the running tasks may lead to a waste of completed computations, which may reduce throughput. Since some scheduling frameworks [GAK+14] are designed and used for the jobs or tasks of limited workload, they may not have starvation problems.

The scheduling frameworks may have multiple objectives [CDD+14], *e.g.* reducing computing cost, maximizing node utilization, achieving job fairness, reducing execution time, etc. Most of the existing frameworks share the objectives of reducing execution time by maximizing the utilization of nodes and achieving load balancing among nodes while maximizing parallelism. However, the high utilization of nodes may lead to hardware failures, which may increase the total execution time. Thus, some objectives cannot be all achieved at the same time. For instance, if the objective of reducing execution time of a job is achieved, some other jobs may need to wait much time to get nodes to perform execution, which reduces job fairness. In addition, in some situations, the service provider tries to reduce both execution time and energy consumption. However, in order to reduce execution time, more nodes are used to achieve high parallelism, which may lead to higher energy consumption. Furthermore, some users want to achieve short execution time and stable service. Since the workloads among nodes vary a lot, it is easy to ensure the stable services with longer execution time while it is difficult to offer stable services with very short execution time. For the scheduling problems with conflicting objectives, we can use two methods. The first method [LPV+16] is to use a weighted function to calculate the overall cost with consideration of all objectives. Then, the tasks can be scheduled to nodes, so that there is the highest reduction of the weighted function. The second method is to generate a Pareto front, which is composed of a set of Pareto optimal scheduling plans [ZLT+14]. Each Pareto optimal scheduling plan is better than the other solutions with respect to at least one objective while it is no worse than other scheduling plans in terms of remaining objectives. The Pareto optimal solutions can be achieved by tuning the scheduling plans in order to

improve the performance of some objectives while maintaining the good performance of other objectives.

4 Analysis of Scheduling Frameworks

In this section, we classify and compare sixteen popular scheduling frameworks based on our taxonomy. We end the section with some lessons learnt.

4.1 Classification

The means features of the sixteen popular scheduling frameworks are shown in Tables 1 and 2.

Table 1 Main Features of Scheduling Frameworks. "Hybrid" in granularity means that both jobs and tasks are scheduled while it represents that both static and dynamic methods are used in adaptation. "Kill" represents that long running tasks or jobs are killed in order to achieve fairness.

Framework	Granularity	Adaptation	Fairness	Architecture
YARN	Hybrid	Dynamic	Kill	Centralized two
Tetris	Task	Dynamic	No starvation	Centralized two
Corral	Hybrid	Hybrid	No starvation	Centralized two
Omega	Task	Dynamic	Kill	Shared-state
Heracles	Task	Dynamic	Kill	Centralized
Jockey	Job	Dynamic	Potential starvation	Centralized
Apollo	Task	Dynamic	Potential starvation	Shared-state
Quincy	Hybrid	Dynamic	Kill	Centralized
DAGPS	Task	Dynamic	Reduce nodes	Distributed two
Mercury	Hybrid	Dynamic	Kill	Distributed two
Rayon	Job	Hybrid	User quota	Centralized two
Sparrow	Task	Dynamic	Reduce nodes	Shared-state
Mesos	Task	Dynamic	Kill	Centralized two
Tarcil	Task	Dynamic	Task priority	Shared-state
Fuxi	Job	Dynamic	User quota	Centralized two
Yaq	Task	Dynamic	Task priority	Centralized two Distributed two

Table 1 shows four main features, *i.e.* granularity, adaptation, fairness and architecture, of the sixteen scheduling frameworks. The granularity of most scheduling frameworks is task while some scheduling frameworks (Quincy [IPC+09], Jockey, Rayon [CDD+14] and Fuxi [ZLT+14]) schedule jobs only. Three scheduling frameworks (YARN, Corral [JBM+15] and Mercury [KRC+15]) schedule both jobs and tasks. Almost all scheduling frameworks use dynamic scheduling while two scheduling frameworks (Corral and Rayon) exploit a hybrid method, *i.e.* the combination of static and dynamic scheduling. The scheduling frameworks take advantage of different methods to achieve fairness among different jobs. Most scheduling frameworks use the centralized two-level architecture. Heracles, Jockey and Quincy exploit the centralized architecture. Yaq [RKK+16] has two implementations,

i.e. Yaq-c with the centralized two-level architecture and Yaq-d with the distributed two-level architecture. Mercury also exploits the distributed two-level architecture. Omega, Apollo [BEL+14], DAGPS [GKR+16], Sparrow [OWZS13] and Tarcil [DSK15] exploit the shared-state architecture.

Table 2 Main Features of Scheduling Frameworks (cont.). The heterogeneous column indicates whether the scheduling method supports or not the different capacities of different nodes. "-" indicates that different scheduling methods can be implemented in the scheduling framework. "* " indicates the default scheduling method (other scheduling methods can also be implemented in the scheduling framework).

Framework	Objectives	Heterogeneous	Scheduling method
YARN	Throughput	Support	DLB*
Tetris	Exec. time	Support	TR
Omega	Resource utilization	Support	-
Heracles	Resource utilization	Support	JI
Jockey	Deadline	Support	FB
Mercury	Throughput	Support	NQ
Sparrow	Exec. time	Not	NQ
Mesos	Exec. time	Support	FS
Tarcil	Throughput	Not	NQ
Yaq	Exec. time	Support	NQ
Corral	Exec. time data transfer	Support	JI
Apollo	Throughput Resource utilization	Support	NQ
Quincy	Throughput Data transfer	Not	DLB or FS or CS
DAGPS	Exec. time Resource utilization	Support	TR
Rayon	SLA & deadline Resource utilization	Support	FB and CS
Fuxi	Throughput Resource utilization	Support	JN and CS

Table 2 shows the other main features, *i.e.* objectives, heterogeneous and scheduling method, of the sixteen scheduling frameworks. Different scheduling frameworks have different objectives while all scheduling frameworks share the objective of reducing execution time. Almost all the scheduling frameworks consider different capacities of nodes except Quincy, Sparrow and Tarcil. Most frameworks implement one scheduling method while different scheduling methods can be implemented in YARN and Omega. There are three scheduling methods that can be selected in Quincy while Rayon and Fuxi combine two scheduling methods to schedule jobs. When there are multiple scheduling methods for the scheduling process, machine learning algorithms can be used [MSJ14] to choose a proper one.

4.2 Comparisons

All the scheduling frameworks are designed to be used in big data systems, but why are there so many scheduling frameworks? The first reason is that different scheduling frameworks are used for different requirements of execution time. For instance, full scheduling is more appropriate for batch jobs while sampling scheduling is exploited for real-time requests or stream data processing, *e.g.* Sparrow and Tarcil. The second reason is the scheduling layer. For instance, some scheduling frameworks (Omega, Heracles, Mesos [HKZ+11] and Fuxi) are resource managers, *i.e.* they schedule all the jobs of different applications, *e.g.* MPI jobs and MapReduce jobs, while directly interacting with nodes. This kind of scheduling frameworks is the base of all the other execution systems, *e.g.* file system or MapReduce system. Some scheduling frameworks are based on existing scheduling frameworks or big data systems, *e.g.* YARN, such as Tetris [GAK+14], Corral, Quincy (for Dryad [IBY+07]), Apollo and Jockey (for Cosmos [CJL+08]), DAGPS, Mercury, Rayon, Yaq and Sparrow (implemented with Spark). Dryad and Cosmos have been developed by Microsoft. The third reason is that different companies or organizations propose their own scheduling frameworks, which are adapted to their own big data systems. For instance, Omega and Heracles have been designed by Google; Sparrow, Mesos and Tarcil by universities; Fuxi has been designed for the cloud infrastructure of Alibaba [ZLT+14] and others within Microsoft. Most of the scheduling frameworks are adapted to YARN jobs but Jockey and Apollo are adapted to Cosmos jobs and Quincy is adapted to DAG-based jobs. Finally, different scheduling frameworks have optimizations for different objectives. For instance, each of the YARN-based frameworks (Rayon, Tetris, Corral, Mercury, Yaq, DAGPS) has its own objectives. We discuss the scheduling frameworks according to four types: resource managers, Spark schedulers, Microsoft Cosmos schedulers, YARN-Based schedulers. Resource managers can schedule the jobs or tasks of different types. Spark schedulers and Microsoft Cosmos schedulers are designed for their own systems, *i.e.* Spark or Cosmos. In addition, the YARN-Based schedulers are designed on top of YARN and they are only compatible with the systems that can exploit YARN to schedule jobs or tasks.

Resource managers include four scheduling frameworks. Mesos was designed for Hadoop and MPI jobs in 2011 and Spark can run on top of Mesos. Fuxi was designed for Apsara Cloud Platform, which is a distributed computing system, in 2014. The scale of Mesos is fifty thousand emulated nodes, while Fuxi is used in a production environment with up to hundreds of thousands of nodes. Designed by Google, Omega was proposed in 2013 and Heracles was proposed in 2015. Mesos, Fuxi and Omega can achieve good performance to execute heterogeneous jobs while they cannot guarantee the SLA. Heracles can achieve SLA with high utilization of nodes by proposing hardware isolation.

There are two scheduling frameworks for Spark, *i.e.* Sparrow and Tarcil. These two scheduling frameworks can be used for real-time requests or stream data processing. Sparrow was proposed in 2013. It can reduce the scheduling time by sampling the nodes. However, it may produce low quality scheduling plans since it ignores other non-sampled nodes and the interference of different tasks. Tarcil was proposed in 2015. Tarcil also exploits sampling to accelerate the scheduling process, taking advantage of statistical information and adjusting the dynamic sampling size to achieve better scheduling quality.

At Microsoft, Cosmos used to exploit Dryad to perform job execution before using Hadoop [LJZ17]. There are three frameworks designed for Cosmos-based jobs, *e.g.* Quincy (2009), Jockey (2012) and Apollo (2014). The three frameworks are designed for different scales of clusters. Quincy has been evaluated with a few hundred nodes by the designers,

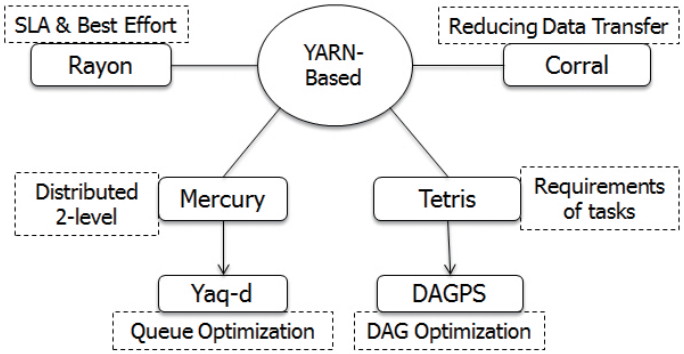

Figure 3: **YARN-Based Frameworks.**

Jockey with several thousands of nodes and Apollo with tens of thousands of nodes. Compared with Quincy and Apollo, Jockey can achieve SLA of jobs by estimating the execution time of jobs and dynamic provisioning for jobs.

Microsoft has also developed several scheduling frameworks based on YARN, *e.g.* Rayon (2014), Tetris (2014), Corral (2015), Mercury (2015), Yaq (2016) and DAGPS (2016), as shown in Figure 3. The objectives of Rayon, Tetris and Corral are different. Rayon focuses on SLA and best-effort jobs based on a reservation-based method while Tetris reduces the execution time of jobs by matching tasks to nodes based on the requirements of the tasks. The reservation-based method schedules the nodes according to the reservation made by users. Corral addresses the problem of largely transferred data during execution by jointly scheduling data and tasks. It schedules the tasks of the same jobs in the same parts of nodes, which reduces the execution time of jobs and the amounts of data transferred in the network. Mercury is an extension of YARN, which exploits the distributed two-level architecture, to achieve scalable scheduling with high throughput. Yaq reduces the execution time of jobs by managing tasks queues. Yaq has a centralized implementation (Yaq-c) based on YARN and a distributed implementation (Yaq-d) based on Mercury. DAGPS is an optimized version of Tetris for DAG-based jobs, which considers the data dependencies of tasks.

4.3 Lessons learnt

1. The number of nodes is increasing. The number of nodes in a data center can range from several hundreds to hundreds of thousands. Different scheduling frameworks are proposed for different numbers of nodes. In the future, the number of nodes will be continuously increasing and new frameworks will be needed to address high numbers of nodes. Thus, a distributed architecture and sampling methods should be used in order to address such scales.

2. SLA and some other user-defined objectives should be supported. In order to offer stable services and good user experience, SLA or user-defined objectives should be supported by the scheduling frameworks. In addition, dynamic scheduling and the isolation of the execution of jobs should be exploited to address the jobs with different deadline requirements of different types of jobs.

3. As the scale of clusters gets higher, the data transferred in the network should not be ignored. The combination of data placement and task scheduling should be exploited in big data systems. In addition, the network topology should be considered during execution so that the time to transfer data is reduced in very big clusters.

5 Benchmarks for Scheduling Methods

Standard benchmarks can be used to evaluate scheduling frameworks in big data systems, *e.g.* TPC-H [TPCb] for the evaluation of Tarcil [DSK15]. A large-scale data analysis benchmark is also proposed in [PPR+09]. However, diverse workloads should be used and different metrics should be measured to evaluate scheduling frameworks in big data systems. In this section, we present the different dimensions for workloads and propose metrics for benchmarks.

5.1 Workload Dimensions in Benchmark

Public data sets are available as described in the TPC-H benchmark [TPCb], TPC-DS benchmark [TPCa] , BigBench benchmark [BDB], Quantcast workloads [ORR+13], Hadoop GridMix workloads [Gri]. Furthermore, traces of Google workload and Facebook workload are presented in [RTG+12] and [GAK+14]. In addition, the workload can also be generated from general applications, such as join operation, PageRank, WordCount, etc. [IPC+09]. In this section, we present the workload dimensions that can be used to evaluate scheduling frameworks in big data systems.

- Heterogeneity. The workload of the benchmark should contain different types of jobs, *e.g.* MapReduce jobs [GAK+14], DAG-based jobs [JBM+15], etc. The execution time of jobs can range from several minutes to several days [CGGK11, GAK+14, RTG+12]. In addition, the execution time of each task should also range from several milliseconds to several hours [GAK+14].

- Task constraints [RTG+12]. Some jobs may contain tasks with special constraints on the nodes, *i.e.* hardware and software. The constraints on the hardware can be the number of CPUs, CPU frequency, GPU, memory size, etc. The constraints on software can be the type of Operating System (OS), the version of OS, programs, etc. The scheduling frameworks should consider these constraints and allocate proper nodes to the tasks.

- Priority. The workload processed in big data systems may consist of latency-sensitive jobs or jobs that do not require short execution time. Thus, the jobs can be associated with different priorities [RTG+12] to be executed. During execution, the scheduling frameworks should enable that the jobs with high priorities are executed first.

- Workload velocity. The jobs or tasks can be generated or submitted at different speeds. The velocity of the workload in big data systems varies a lot. For instance, during peak hours, the scheduling framework needs to schedule hundreds of tasks per second while it may just need to address several tasks per second during quieter times [RTG+12].

5.2 Metrics for Benchmarks

Although some benchmarks exist, they have a few metrics, *e.g.* the throughput of SQL-based workloads [PPR+09], execution time [AMP], financial cost and energy consumption [TPCb]. Big data systems also have some other important metrics to measure. In this section, we propose the following metrics to make a comprehensive comparison of scheduling frameworks in big data systems.

- **Execution time and throughput**. The most important metric is the execution time and the throughput of the jobs or tasks. A good scheduling framework should be able to yield good execution time and high throughput of tasks and jobs. In addition, the standard deviation of the execution time of submitted jobs should also be measured, which indicates the predictability of the execution time. This is critical to the users who need to predict the execution time of submitted jobs.

- **Scheduling time**. It is important to measure the scheduling time, which indicates how fast a scheduling framework can generate a scheduling plan. This may not be very important for batch jobs but it is critical for real-time requests and stream data processing. Good scheduling frameworks need short scheduling time.

- **Scalability**. The scalability in terms of nodes is a very important metric for the scheduling frameworks in big data systems. The scheduling framework should be evaluated for different scales of clusters. For instance, the scale can range from several hundred nodes to dozens of thousands or several hundred thousand nodes because of many different jobs (and users) in a big data system. We can measure at which cluster scale the performance of scheduling starts getting worse, *i.e.* the execution time of jobs increases or the throughput of jobs or tasks decreases.

- **Fairness and job priority**. The fairness of job execution should be addressed in terms of user experience. We assume that there are multiple tenants, *i.e.* multiple users. The jobs of each user should have the corresponding numbers of nodes to perform execution according to the quota of each user and the priority of jobs. This can ensure that the big data system can be shared among different users. Some jobs are submitted with their own priorities and the users prefer that the jobs with high priorities are executed before the jobs with low priorities. In this situation, priority inversion, *i.e.* the jobs with low priorities are executed before the jobs with high priorities, should be avoided and the ratio of priority inversion should also be measured.

- **Execution cost**. The job execution cost should also be measured. The cost includes the financial cost to buy machines or rent Virtual Machines (VMs) in cloud computing environments, the energy and network resources [PAB+15]. All these costs are critical for organizations or companies, which are maintaining the big data systems.

- **Network transfer**. The amount of data transferred over the network should also be measured. This metric can indicate the amount of data transferred in order to execute tasks scheduled to the nodes where the input data of tasks need to be transferred from other nodes. Smart scheduling frameworks should lead to low data transfer during execution. This metric also indicates how much the quality of the network can affect the performance of scheduling.

- **Node utilization**. If the utilization of nodes stays low, most of the nodes remain idle during a long time while they are busy running workloads during a short time. This may lead to long execution time of jobs or low throughput of jobs or tasks. Thus, the utilization of nodes can also be measured to know if the nodes are fully used. If they are, and the system cannot process the jobs or tasks within the required time, more nodes should be added and the scalability of the scheduling frameworks should be improved. However, if the utilization of nodes is very low and the system needs to reduce execution time, it is better to improve the utilization of nodes than to add new nodes.

- **Fault tolerance**. In big data systems, the execution of a job or a task may fail and the failed jobs or tasks must be re-executed. Thus, the possibility of failures of jobs or tasks should be measured. Some scheduling frameworks may yield high utilization of nodes, but they may produce many failures of task execution because of hardware failures incurred by the workloads within each node. In this situation, the execution time may be long and the throughput of jobs or tasks may be low. Scheduling frameworks should make a trade-off between the high utilization of nodes and the failures of tasks or jobs.

- **Execution efficiency**. Some systems may duplicate the execution of tasks [AKG+10] to reduce execution time. However, this duplication may reduce the efficiency of execution since some nodes are wasted on calculating some useless results. Thus, if there are duplicated tasks, the ratio of duplicated tasks should also be measured.

- **SLA violation**. Some jobs have user predictability expectation defined by SLAs. The SLA may consist of two elements: a deadline on the execution of the submitted jobs, and a cost, expressed in terms of resources (number of nodes and occupied time) provisioned for the job [JCM+16]. Smart scheduling frameworks should meet the SLA of the submitted jobs [JMNY15]. Thus, the ratio of the SLA violation should be measured to indicate how well the scheduling frameworks can support users' expectation.

These metrics should be measured when the system can produce correct results. Throughput, scalability, node utilization, fault-tolerance are much related to scheduling frameworks while the others are related to both scheduling frameworks and scheduling methods implemented in scheduling frameworks. When one of the metrics is changed, the other metrics can be measured again. Through these metrics, we can perform a comprehensive comparison of scheduling frameworks in big data systems.

6 Research Directions

Although much work has been done on scheduling in big data systems, there remain some limitations, *e.g.* dynamic provisioning in cloud environments, stream data processing, heterogeneous network environments and predictability of job execution time. This section discusses the limitations of the existing frameworks and proposes new research directions.

6.1 Dynamic Adaptation to Execution Environments

The existing approaches provide the methods to schedule jobs or tasks in a static environment or a homogeneous network environment as discussed in Section 3.2.3. However, the

dynamic adaptation to dynamic provisioning during execution and heterogeneous networks in multiple data centers could be further studied.

6.1.1 Dynamic Provisioning in the Cloud

The existing frameworks focus on the provisioning with existing nodes for execution, by scheduling part of or all the nodes to the submitted jobs. However, these scheduling frameworks do not consider dynamic provisioning by adding some additional nodes from the cloud environment during execution. For instance, after estimating the workload of jobs, the scheduling framework can dynamically create some VMs in the cloud in order to reduce the execution time of the jobs of big workloads when the existing nodes are not enough. In addition, the nodes that are not scheduled and remain idle can be shut down to reduce energy consumption [LRD+16].

6.1.2 Scheduling in Heterogeneous Network Environments

The existing scheduling frameworks consider a homogeneous network environment, where the bandwidth is high and the same for each node. However, as the scale becomes very big, the nodes need to be placed in different clusters or even different data centers. In this environment, the bandwidths among different nodes may have much difference. For instance, the bandwidth between two nodes located in the same cluster is very high while the bandwidth between two nodes located in different data centers can be much lower [LPVM16]. Thus, the time to transfer data among different nodes should be carefully considered when scheduling jobs or tasks in heterogeneous network environments.

6.2 Scheduling in Stream Data Processing

As continuous data is being generated by sensors, social networks and other areas, stream data processing becomes important [IZE11]. Some batch jobs can also be converted to data stream processing [FY11]. However, the existing frameworks focus on batch jobs and real-time requests [DSK15, OWZS13] as explained in Section 3.2.2. Some scheduling methods have been proposed for stream data processing, but they focus on a shared memory architecture [FY11, FKB+14] or even a single node [CcR+03]. The methods designed for batch jobs or real-time requests can be used in big streaming data systems, *e.g.* Twitter Heron [KBF+15], Apache Samza [Sam] and Spark Streaming [Spa]. Sparrow and Tarcil only focus on the sampled nodes for scheduling. However, there are several specific dimensions of the streaming data. First, big data velocity can lead to big workloads. Second, the tasks may be replicated onto independently failing nodes to guarantee fault-tolerance of the execution [BHS09] and the data is also replicated because of the replicated tasks. Third, stream data processing requires very short response time. To the best of the authors' knowledge, there is no existing scheduling framework that schedules tasks with consideration of data replication and dynamic provisioning to execute big workloads incurred by big data velocity within a short time.

6.3 Predictability of Job Execution Time

Most scheduling frameworks strive to reduce execution time as explained in Section 3.2.5. However, users may also want to predict the execution time of submitted jobs. When there are many workloads in the system, the execution time of submitted jobs may be very long

and when there are a few workloads in the system, the execution time of submitted jobs may be very short. If the scheduling frameworks focus only on reducing execution time, the difference of the longest execution time and the shortest execution time of the same submitted jobs will be very high, thus making it very difficult to predict the execution time for the users. In order to enable users to predict the execution time of submitted jobs, the predictability of jobs should also be ensured by smart scheduling frameworks. For instance, the scheduling frameworks can accelerate the execution of jobs by adding nodes when there are many workloads and remove some nodes when there are few workloads so that the execution time of the same submitted jobs stays stable.

7 Conclusion

In this paper, we surveyed the scheduling frameworks used in big data systems. First, we described the execution process of jobs and tasks, existing scheduling methods to be implemented in scheduling frameworks and proposed a taxonomy of scheduling frameworks. The scheduling frameworks can be classified by granularity, execution time, adaptation to execution environments, architecture and objectives. There are four kinds of architectures, *i.e.* centralized, centralized two-level, distributed two-level and shared-state. Shared-state can generate optimal scheduling plans while distributed two-level is suitable for large number of nodes and high arrival rate of workloads. Second, we studied sixteen popular scheduling frameworks. They are designed for different requirements of execution time, scheduling layers and big data systems of web companies or software foundations. By analyzing these frameworks, we found that the scheduling frameworks should be adapted to the increasing number of nodes, user-defined objectives and optimized for reducing the transferred data in the network. Third, we proposed the workload dimensions and metrics for benchmarks to evaluate scheduling frameworks. The workload dimensions should include heterogeneity, diverse task constraints, different priorities and velocities of the workload. Some metrics, such as throughput, scalability, node utilization and fault tolerance are much related to scheduling frameworks while others are related to both scheduling frameworks and scheduling methods implemented in scheduling frameworks.

The scheduling techniques for big data systems have focused on batch jobs in cluster environments. Thus, there is much room for research in this area and we proposed several directions. First, scheduling frameworks can be adapted to the dynamic provisioning of VMs in cloud. Second, they should be optimized for streaming data processing with consideration of data replication and dynamic provisioning to efficiently process big workloads. Third, they should also be adapted to environments where high numbers of nodes are distributed in different data centers and the network is heterogeneous. Fourth, they should be designed to maintain stable execution time for the same job so that it is easy for users to predict the job execution time.

Acknowledgements

This work was partially funded by the European Commission (H2020 HPC4e and CloudDBappliance projects) and the Microsoft-Inria Joint Centre (ZCloudFlow project) and performed in the context of the Computational Biology Institute (`www.ibc-montpellier.fr`). We would like to thank Vinod Nair for his review and useful

comments, Markus Weimer for the fruitful discussions, and Sriram Rao and Raghu Ramakrishnan for their help in the collaboration with Microsoft.

References

[AD12] K. Vaswani A. Desai, K. Rajan. Critical path based performance models for distributed queries. Microsoft Technical Report:MSR–TR– 2012–121, 2012.

[AKG+10] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 265–278, 2010.

[AMP] Amplab benchmark. `https://amplab.cs.berkeley.edu/benchmark/`.

[Bar13] M. Barlow. *Real-Time Big Data Analytics: Emerging Architecture*. O'Reilly Media, 2013.

[BDB] Big-data-benchmark. `http://bit.ly/1HlFRH0`.

[BEL+14] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 285–300, 2014.

[BHS09] M. Balazinska, J. Hwang, and M. A. Shah. Fault-tolerance and high availability in data stream management systems. In *Encyclopedia of Database Systems*, pages 1109–1115. 2009.

[CcR+03] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 838–849, 2003.

[CDD+14] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In *ACM Symp. on Cloud Computing (SOCC)*, pages 2:1–2:14, 2014.

[CF02] S. Chandrasekaran and M. J. Franklin. Streaming queries over streaming data. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 203–214, 2002.

[CGGK11] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz. The case for evaluating mapreduce performance using workload suites. In *Annual IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 390–399, 2011.

[CJL+08] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1265–1276, 2008.

[DG08] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DSK15] C. Delimitrou, D. Sanchez, and C. Kozyrakis. Tarcil: Reconciling scheduling speed and quality in large shared clusters. In *ACM Symp. on Cloud Computing (SOCC)*, pages 97–110, 2015.

[FBK+12] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *ACM European Conf. on Computer Systems (EuroSys)*, pages 99–112, 2012.

[FKB+14] Z. Falt, M. Krulis, D. Bednárek, J. Yaghob, and F. Zavoral. Locality aware task scheduling in parallel data stream processing. In *Int. Symp. on Intelligent Distributed Computing (IDC)*, pages 331–342, 2014.

[FKHM96] R. F. Freund, T. Kidd, D. A. Hensgen, and L. Moore. Smartnet: a scheduling framework for heterogeneous computing. In *Int. Symp. on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 514–521, 1996.

[FR98] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing (IPPS/SPDP Workshop)*, pages 1–24, 1998.

[FY11] Z. Falt and J. Yaghob. Task scheduling in data stream processing. In *Dateso: Annual Int. Workshop on Databases*, pages 85–96, 2011.

[GAK+14] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. *ACM Conf. on Special Interest Group on Data Communication (SIGCOMM)*, 44(4):455–466, 2014.

[GGL03] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *ACM Symp. on Operating Systems Principles (SOSP)*, pages 29–43, 2003.

[GKR+16] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni. Do the hard stuff first: Scheduling dependent computations in data-analytics clusters. *CoRR*, abs/1604.07371, 2016.

[GPDC15] J. V. Gautam, H. B. Prajapati, V. K. Dabhi, and S. Chaudhary. A survey on job scheduling algorithms in big data processing. In *IEEE Int. Conf. on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–11, 2015.

[Gri] Hadoop gridmix. `http://hadoop.apache.org/docs/r1.2.1/gridmix.html`.

[Had] Hadoop. `http://hadoop.apache.org/`.

[HKZ+11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, pages 295–308, 2011.

[HLS11] C. He, Y. Lu, and D. Swanson. Matchmaking: A new MapReduce scheduling technique. In *IEEE Int. Conf. on Cloud Computing Technology and Science (CLOUDCOM)*, pages 40–47, 2011.

[IBY+07] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM European Conf. on Computer Systems (EuroSys)*, pages 59–72, 2007.

[IPC+09] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *ACM SIGOPS Symp. on Operating Systems Principles (SOSP)*, pages 261–276, 2009.

[IZE11] IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[JBM+15] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *ACM Conf. on Special Interest Group on Data Communication (SIGCOMM)*, pages 407–420, 2015.

[JCM+16] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, and S. Rao. Morpheus: Towards automated SLOs for enterprise clusters. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 117–134, 2016.

[JMNY15] N. Jain, I. Menache, J. Naor, and J. Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing (TOPC)*, 2(1):3, 2015.

[KBF+15] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 239–250, 2015.

[KKBM13] G. Kumar, S. Kandula, P. Bodík, and I. Menache. Virtualizing traffic shapers for practical resource allocation. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[KMM+15] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *IEEE Int. Conf. on Big Data (Big Data)*, pages 2785–2792, 2015.

[KRC+15] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga. Mercury: Hybrid centralized and distributed scheduling in large shared clusters. In *USENIX Annual Technical Conf. (USENIX ATC 15)*, pages 485–497, 2015.

[LCG+15] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving resource efficiency at scale. In *Annual Int. Symp. on Computer Architecture (ISCA)*, pages 450–462, 2015.

[LJZ17] K. C. Li, H. Jiang, and A. Y. Zomaya. *Big Data Management and Processing*. Chapman & Hall/CRC, 2017.

[LPV+16] J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso. Multi-objective scheduling of scientific workflows in multisite clouds. *Future Generation Computer System*, 63:76–95, 2016.

[LPVM15] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[LPVM16] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. Scientific Workflow Scheduling with Provenance Support in Multisite Cloud. In *Int. Meeting on High-Performance Computing for Computational Science (VECPAR)*, page 8, 2016.

[LRD+16] W. Lang, K. Ramachandra, D. J. DeWitt, S. Xu, Q. Guo, A. Kalhan, and P. Carlin. Not for the timid: On the impact of aggressive over-booking in the cloud. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 1–12, 2016.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1st edition, 1997.

[MSJ14] I. Menache, O. Shamir, and N. Jain. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *Int. Conf. on Autonomic Computing (ICAC)*, pages 177–187, 2014.

[MVSA16] M. Masdari, S. ValiKardan, Z. Shahi, and S. Imani Azar. Towards workflow scheduling in cloud computing: A comprehensive analysis. *Journal of Network and Computer Applications*, 66:64 – 82, 2016.

[ORR+13] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly. The quantcast file system. *Proceedings of the VLDB Endowment (PVLDB)*, 6(11):1092–1101, 2013.

[ÖV11] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

[OWZS13] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *ACM Symp. on Operating Systems Principles (SOSP)*, pages 69–84, 2013.

[PAB+15] Q. Pu, G. Ananthanarayanan, P. Bodík, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. In *ACM Conf. on Special Interest Group on Data Communication (SIGCOMM)*, pages 421–434, 2015.

[PBL+16] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese. Scaling network verification using symmetry and surgery. In *Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)*, pages 69–83, 2016.

[PPR+09] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 165–178, 2009.

[Ren15] D. K. Rensin. Kubernetes - scheduling the future at cloud scale. In *O'Reilly Open Source CONvention (OSCON)*, 2015.

[RKK+16] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao. Efficient queue management for cluster scheduling. In *ACM European Conf. on Computer Systems (EuroSys)*, pages 1–15, 2016.

[RL11] B.Thirumala Rao and Dr. L.S.S.Reddy. Article: Survey on improved scheduling in hadoop mapreduce in cloud environments. *Int. Journal of Computer Applications*, 34(9):29–33, 2011.

[RTG+12] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symp. on Cloud Computing (SOCC)*, pages 7:1–7:13, 2012.

[Sam] Apache Samza. `http://samza.incubator.apache.org`.

[SC10] X. Sun and Y. Chen. Reevaluating amdahl's law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, 2010.

[SKAEMW13] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *ACM European Conf. on Computer Systems (EuroSys)*, pages 351–364, 2013.

[SKR15] C. Sreedhar, N. Kasiviswanath, and P. Chenna Reddy. Article: A survey on big data management and job scheduling. *Int. Journal of Computer Applications*, 130(13):41–49, 2015.

[SLBA11] S. Sakr, A. Liu, D. M. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys Tutorials*, 13(3):311–336, 2011.

[Spa] Spark Streaming. `https://spark.apache.org/streaming/`.

[TPCa] Tpc-ds benchmark. `http://bit.ly/1J6uDap`.

[TPCb] Tpc-h benchmark. `http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp`.

[Val09] P. Valduriez. Shared-nothing architecture. In *Encyclopedia of Database Systems*, pages 2638–2639. 2009.

[VMD+13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Annual Symp. on Cloud Computing (SOCC)*, pages 5:1–5:16, 2013.

[WCZ13] D. Wang, J. Chen, and W. Zhao. A task scheduling algorithm for hadoop platform. *Journal of Computers*, 8(4):929–936, 2013.

[YS11] D. Yoo and K. M. Sim. A comparative review of job scheduling for mapreduce. In *IEEE Int. Conf. on Cloud Computing and Intelligence Systems*, pages 353–358, 2011.

[ZBS+10] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *ACM European Conf. on Computer Systems (EuroSys)*, pages 265–278, 2010.

[ZCF+10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *USENIX Conf. on Hot Topics in Cloud Computing (HotCloud)*, pages 10–10, 2010.

[ZLT+14] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *Proceedings of the VLDB Endowment (PVLDB)*, 7(13):1393–1404, 2014.