Quality of cloud services determined by the dynamic management of scheduling models for complex heterogeneous workloads

Damián Fernández-Cerero,
Alejandro Fernández-Montes
Department of Computer Languages and Systems
University of Seville, Spain
Emails: damiancerero@us.es, afdez@us.es

Joanna Kołodziej
Department of Computer Science
Cracow University of Technology
Poland
Email: jokolodziej@pk.edu.pl

Laurent Lefèvre
Univ. Lyon, Inria, CNRS
ENS de Lyon
Univ. Claude-Bernard Lyon 1
LIP, France
Email: laurent.lefevre@ens-lyon.fr

***Abstract*—The quality of services in Cloud Computing (CC) depends on the scheduling strategies selected for processing of the complex workloads in the physical cloud clusters. Using the scheduler of the single type does not guarantee of the optimal mapping of jobs onto cloud resources, especially in the case of the processing of the big data workloads. In this paper, we compare the performances of the cloud schedulers for various combinations of the cloud workloads with different characteristics. We define several scenarios where the proper types of schedulers can be selected from a list of scheduling models implemented in the system, and used to schedule the concrete workloads based on the workloads' parameters and the feedback on the efficiency of the schedulers. The presented work is the first step in the development and implementation of an automatic intelligent scheduler selection system. In our simple experimental analysis, we confirm the usefulness of such a system in today's data-intensive cloud computing.**

***Index Terms*—Big Data Quality, Big Data, Cloud scheduling, Dynamic cloud scheduling, Cloud Computing.**

1. Introduction

Big data [6] is an emerging field where innovative technology offers alternatives to resolve the inherent problems that appear when working with huge amounts of data, providing new ways to reuse and extract value from information. In today's data intensive computing world, we need a complete ICT (information and communication technology) paradigm shift in order to support the development and delivery of Big data applications, in a way that applications do not get overwhelmed by incoming data volume, data rate, data sources, and data types. Dealing with this huge amount of data requires a new end-to-end data-management and data-analysis paradigm in which methods need to be efficient, not just as stove-pipe processes, but as part of a well-integrated system. Big-Data analysis tools that may be aware of the significant trade-off that exists between the Computational Time and the Quality of Result are needed in this context. This trade-off between time and accuracy (quality) requires big-data analysis architectures to support both Batch Data Analysis Processes (for latent but quality solutions) as well as Streaming Data Analysis (for near real-time situation awareness).

The deployment and provision of efficient-management systems for b data applications will greatly benefit from a cloud-like middleware infrastructure, which could be formulated to create a combined environment consisting of multiple private (municipalities, enterprises, research organisations) and public (Amazon, Microsoft) infrastructure providers to deliver on-demand access to emergency management applications. Cloud computing (CC) assembles large networks of virtualised ICT services such as hardware resources (such as CPU, storage, and network), software resources (such as databases, application servers, and web servers) and applications. In industry these services are referred to as *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS),* and *Software as a Service (SaaS)*. CC services are hosted in large data centres, often referred to as data farms, operated by companies such as Amazon [21], and Microsoft [8]. Several features of this paradigm, such as elasticity, thin–client user interface as well as resource pools for both data and computing, denote that the CC paradigm is adequate for collecting, hosting and processing complex workloads as well as the associated data needed for their executions.

Cloud scheduling remains a challenging problem especially in the big data era. Although many cluster, grid and cloud scheduling models have been developed over the past two decades [24], most of them cannot be successfully implemented for solving the complex data-intensive realistic problems. The main reason of that is the lack of a simple but effective scalable scheduling management mechanism, which can be the background of the new scheduling adaptation model based on the workload characteristics. In fact, the main role of such mechanism is the selection of the proper scheduler from a set of scheduling models and using that scheduler for the 'proper' workload based on the analysis of the workload structure and the parameters of its jobs, such

as the number of tasks in the job, the interrelations among tasks, and the requirements of the jobs in the workload. An additional challenge in cloud scheduling are the special requirements from the end-users related to the privacy and data protection, which make all the process even more complex.

The main aim of our research is to define and implement the intelligent automatic scheduler selection system, which allows us to apply the proper type of the scheduler to a given workload just arrived at the cloud cluster. There are three main stages of our work:

- **Stage 1** — preliminary experimental comparison of the selected classes of cloud schedulers and simple adaptation of the schedulers to the workload parameters – first implementation of the manual scheduler selection mechanism.
- **Stage 2** – development of an automatic system that can dynamically select and match the proper scheduling model to the specified type of workload depending on the workload parameters and cloud cluster architecture.
- **Stage 3** – improvement the scheduler selection mechanism with machine-learning techniques.

In this paper we present the results achieved in the first stage. We define a sequence of complex workloads with different characteristics and we implemented two centralized scheduling models in a trustworthy cloud simulator. Then we used separately each scheduler for mapping all workloads onto the cloud cluster and monitored the execution times of all jobs in the workloads. Finally, based on the obtained results for the schedulers, we repeated the experiment but we changed dynamically the schedulers used for planning the concrete workflows. The results of such simple experiments show that such scheduler selection mechanism may significantly improve the performance of the cloud physical cluster and also improve the quality of the cloud services, especially for processing the big volumes of data.

The paper is organized as follows. In Section 2 we define a simple classification of the CC schedulers based on the task, data and resource management models. We also survey the examples of the schedulers from each of the defined class. In Section 3 we define the formal notation and explain the background of the scheduling models together with the characteristics of the heterogeneous workloads used in this paper. In Section 4, we provide a simple experimental analysis of the selected schedulers for the three scheduling scenarios. The paper ends with summary of the work and future research plans in Section 5.

2. Classification of the CC Schedulers

In this section, we present a simple classification of the most popular CC schedulers used for the complex workloads generated in big data scenarios. This classification is based on the workload (jobs), data and resource management policies. It defines in fact the categories of the schedulers developed for the cloud physical clusters, which are however the main infrastructures for the implementation of the cloud virtual servers. We also overview the recently developed CC scheduling models related to the defined classification. The section ends with some critical remarks, which better illustrate the motivation of our work.

2.1. Taxonomy of CC schedulers

In Fig. 1, we present a simple taxonomy of the CC schedulers based on the job, resources and data management policies.

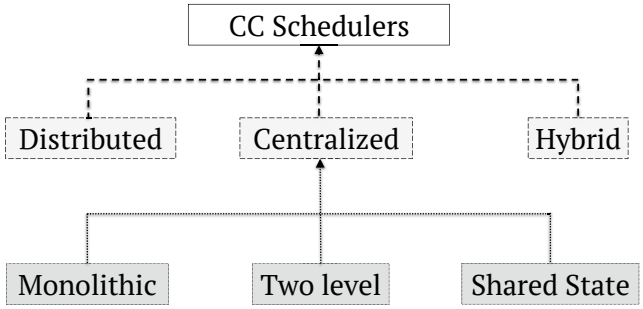


Figure 1: Generic taxonomy of CC schedulers based on the data and task management policies

There are three main categories of schedulers defined in that taxonomy, namely:: a) Centralized schedulers; b) Distributed schedulers; and c) Hybrid schedulers.

In the *centralized scheduling model*, there is a central authority with a full knowledge of the system. Such central scheduler makes the task (job) – machine mapping and data processing. It may work well for workloads with a low number of high–demanding jobs. This scheduling model has usually limited scalability, and some difficulties may occur in the accommodation of multiple local policies imposed by the resource owners.

In the *distributed scheduling model*, there is no central authority responsible for resource allocation and many local schedulers work independently to manage the tasks and jobs. Such distributed schedulers can work optimally for a larger number of jobs with a low task inter–arrival rates. However, the local schedulers have the limited knowledge about the whole system. They usually are able to communicate with the schedulers of the neighbour cloud clusters. The distributed scheduling model is highly scalable, fault–tolerant and easy for reconfiguration of the architectures of the cloud clusters, but the scheduling decisions in such models may be sub-optimal, which can be critical for long-running and high-demanding jobs.

Finally, in the *hierarchical scheduling model*, there is a central meta–scheduler (or metabroker), which interacts with the local job dispatchers. Such meta–scheduler manages the large sets of jobs and resources while the local job managers control small local cloud clusters. The local schedulers have the knowledge about resource clusters, but they cannot monitor the whole system. The main benefits of using hierarchical scheduling is that it incorporates scalability and fault–

tolerance issues while also retaining some of the advantages of the centralized scheme such as co-allocation.

The above classification is very general. Each class may consists of several subcategories such as centralized scheduling model presented in Fig. 1. In the next Section, we shortly survey the concrete models developed for each of the presented categories and summarize their functionalities.

2.2. Characteristics of the CC Scheduler Classes

The quality of the cloud services and the efficiency of the cloud schedulers depend on the workload parameters and the requirements of the cloud end–users. The main workload parameters include security and data privacy issues; task (job) execution deadline constraints; and service utilisation costs. The delivered data files needed in computation and the results of the computation of jobs (tasks) are stored and replicated in the cloud data centers [3]. In order to increase the utilisation of the local cloud clusters and to optimize the usage of the data–center servers, the cloud schedulers must be effective in the processing of the multiple heterogeneous workloads in the physical and virtual cloud infrastructures. In the rest of this section, we provide a detailed characteristics of the schedulers from the classes presented in Fig. 1, describe their main features and limitations.

2.2.1. Centralized Schedulers. Centralized schedulers can be divided into three classes, namely: a) monolithic schedulers; b) two-level schedulers; and c) shared-state schedulers.

Monolithic centralized scheduler is the most popular model of cloud cluster schedulers. Monolithic schedulers [22] work very well under low job-arrival rate conditions. The example of such a class is the Map–Reduce model for long-running jobs [9]. Latencies of seconds or minutes [14] are acceptable in this context. This kind of scheduler can perform high-quality scheduling decisions [12], [40] by examining the whole cluster state. In this process, the best resources can be assigned to each task, and therefore the scheduler can determine the negative performance impact due to hardware heterogeneity and interference because of the utilisation of shared resources [25], [35], [39], among others. This approach leads to higher machine utilization rates [38], shorter execution times, better load balancing, more predictable performance [13], [41], and increased reliability [34]. The scheduling process for the monolithic centralized scheduler is illustrated in Fig. 2

The increasing need of fast-response jobs led to the partitioning of jobs, which implies that a higher number of small and fast tasks are to be scheduled. This new situation may exceed the capabilities of a centralized monolithic scheduler. In order to face this new challenge, two new centralized scheduling approaches that parallelize scheduling decisions were proposed. *Two-level centralized schedulers*, such as Mesos [20], and YARN [37], use a centralized coordinator that blocks the whole cluster every time a scheduler makes a scheduling decision. Then, this coordinator offers resources to the different frameworks, such as Hadoop and MPI. Each

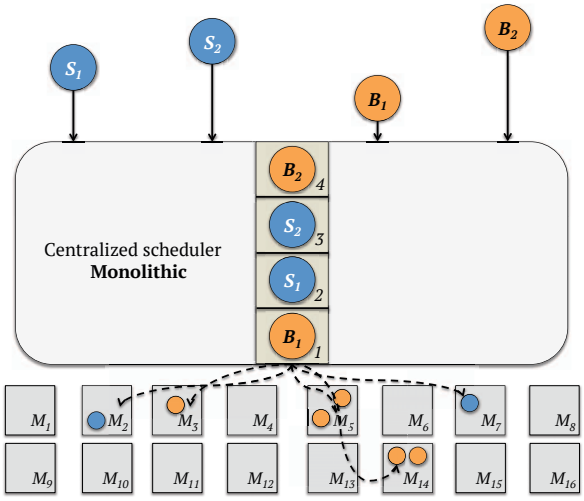


Figure 2: Monolithic centralized scheduling workflow, B - Short-running Batch task, S - Long-running Service task, M - Machine

of these frameworks has its own scheduler responsible for assigning tasks to servers, as shown in Figure 3. In this approach, scheduling decisions are suboptimal, since the total cluster state and task requirements are not available for neither the involved framework scheduler nor the centralized coordinator.

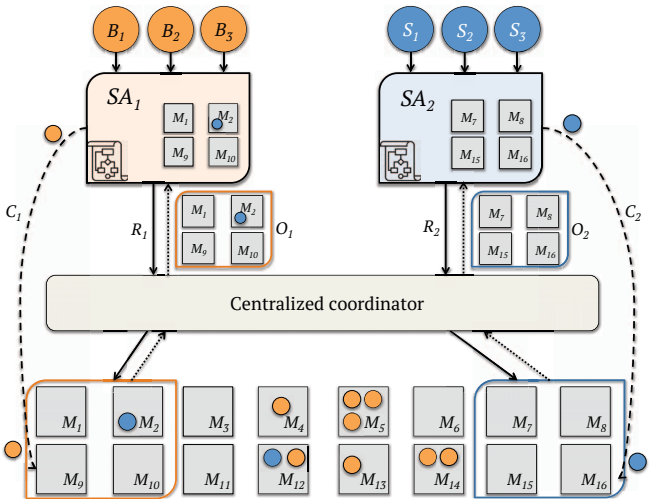


Figure 3: Two-level centralized scheduling workflow, O - Resource offer, SA - Application Scheduler , R - Resource Request, C - Scheduling decision Commit transaction

Shared-state centralized schedulers, such as Omega [34], follow an optimistic approach where a centralized coordinator manages several concurrent schedulers that are able to operate simultaneously. In this model, each scheduler makes scheduling decisions by using a stale copy of the whole cluster state. Then, these schedulers commit atomic scheduling transactions to the centralized cluster. If these transactions

result in a conflict, the local copy of the cluster state used by the scheduler is then updated and the scheduling process retried, as illustrated in Figure 4.

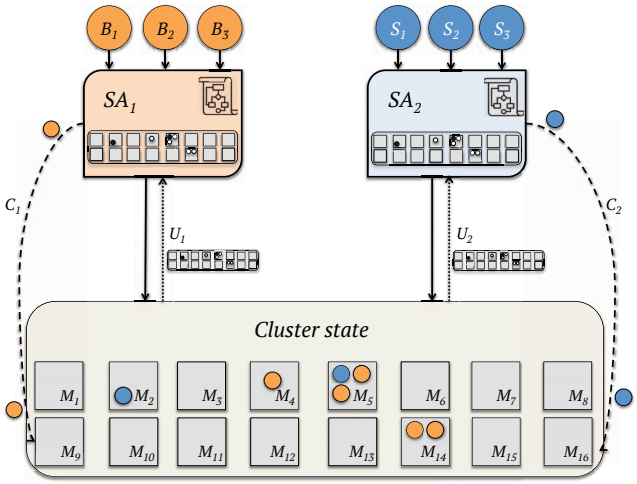


Figure 4: Shared-state centralized scheduling workflow, U - Cluster state Update

Even though centralized schedulers simplify the cloud cluster management, the performance bottleneck caused by the centralized coordinators that are the path for all scheduling decisions makes them suboptimal for environments where tens of thousands or even millions of short, user-facing and latency-sensitive tasks must be served every second [28].

2.2.2. Distributed cloud schedulers. *Distributed schedulers* [16], [28], [30] use simple algorithms that only examine a small part of the cluster architecture (local niches). This kind of scheduler achieves high throughput and low-latency parallel task placement, but the scheduling decisions are inferior. The distributed scheduling process is illustrated in Figure 5

2.2.3. Hybrid cloud schedulers. In many realistic cloud scheduling scenarios, the workload is not homogeneous [7], [32], but is composed of two main groups of jobs:a) Approximately 90% of short and latency-sensitive jobs that consume approximately 15% of the total resources; and b) Approximately 10% of long-running jobs that consume approximately 85% of the total resources [2], [31], [33]. In this environment, the poor placement decisions made by distributed schedulers impact specially on these long-running jobs, which may perform even worse than in stressed centralized scheduling environments.

In order to overcome the above limitations, the *hybrid cloud scheduling models* [11], [10], [23] were developed as the combinations of both centralized and distributed models in the way presented in Figure 6. The hybrid scheduler delivers centralized high-quality scheduling decisions for long-running resource-demanding jobs and sub-optimal fast scheduling decisions for short and latency-sensitive jobs.

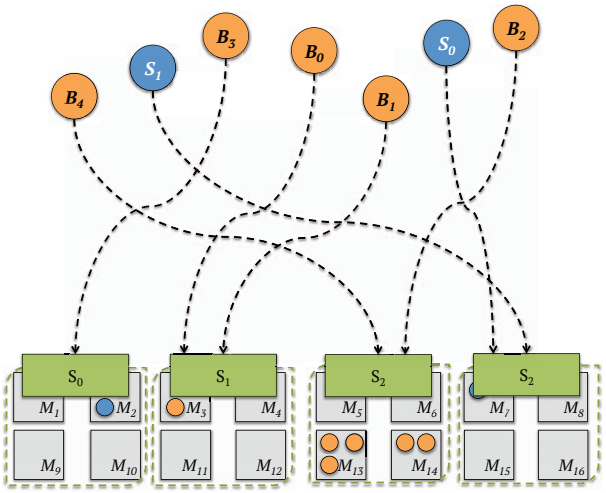


Figure 5: Distributed scheduling workflow, S - Distributed scheduler

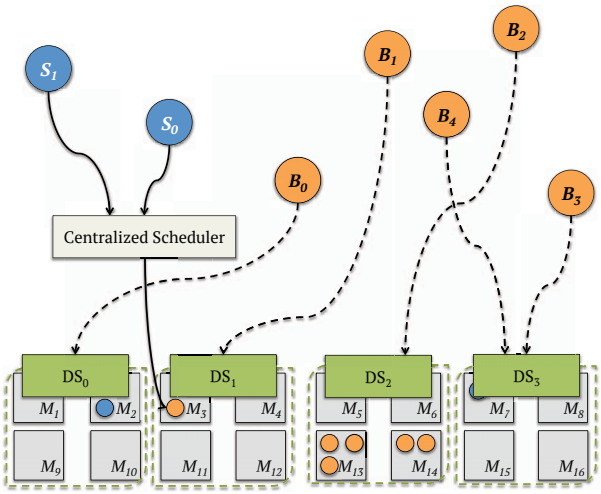


Figure 6: Hybrid scheduling workflow

2.3. Summary of the classification

The summary of the reviewed schedulers is presented in Table 1.

It can be observed from the table, that centralized schedulers work quite well for long-running batch workloads, distributed schedulers are effective in planning the short and user-facing data analytics workloads and hybrid schedulers perform optimally for the mixed type (BW and SW) heterogeneous workloads. The results of that simple comparison analysis shows that the efficiency of the scheduler depends very much of the type of the workload and jobs [18]. Using schedulers just from the one class would not guarantee the high quality of the cloud services offered to the end-users.

Table 1: Evolution of the cluster scheduling strategies

Strategy	Optimal environment	Suboptimal environments	Frameworks
Centralized Monolithic	Low number of long-running and non-latency sensitive jobs	Mid and high number of jobs Mixed workloads	Paragon [12] Quasar [13] Borg [38]
Centralized Two-level	Mid number of diverse long-running, non latency sensitive jobs	Latency-sensitive workloads	YARN [37] Mesos [20]
Centralized Shared-state	Mid number of heterogeneous workloads	High number of short and latency-sensitive jobs	Omega [34]
Distributed	High number of short, non resource-demanding, latency-sensitive jobs	Long, resource-demanding jobs Mixed workloads	Canary [30] Tarcil [14] Sparrow [28]
Hybrid	Mixed workloads composed of 90% of short, latency-sensitive jobs and 10% of long-running, resource-demanding jobs	Homogeneous workloads Other workloads patterns Evolving patterns	Mercury [23] Hawk [11] Eagle [10]

2.4. Adaptive scheduling

The idea of adaptive scheduling has been previously explored in the literature. In [36], Tumanov et al. propose a scheduler named TetriSched which is built on top of YARN and collaborates with the calendaring reservation system to continuously optimize the immediate-term scheduling plan for all pending jobs. This scheduler uses a Mixed Integer Linear Programming (MILP) solver that takes into account job runtime estimates, placement preferences and changing cluster conditions for the batch of pending jobs instead of making job-by-job decisions.

In [29], Polo et al. propose a resource-aware scheduling technique for MapReduce multi-job workloads that aims to rise resource utilization across machines while minimizing the impact in makespan. This scheduler extends the abstraction of 'task slot' to 'job slot'; explores resource profiling information to obtain better utilization of resources and improve application performance; adapts to changes in resource demand by dynamically allocating resources to jobs; seeks to meet soft-deadlines via a utility-based approach; and differentiates between map and reduce tasks when making resource-aware scheduling decisions.

In [27], Niu et al. propose an adaptive scheduler called Gemini for Hadoop YARN focused on the balance between fairness and performance. This scheduler is based on a a model that uses the regression approach to estimate the performance improvement and the fairness loss under the sharing computation compared to the exclusive non-sharing scenario. Gemini applies this model for tasks allocation in order to optimize the performance of the cluster given the user-defined fairness level. Instead of using a static scheduling policy, Gemini adaptively decides the proper scheduling policy according to the particular workload and environment.

In this work, unlike the related work studied, we present a novel model that works at the resource manager framework (scheduling model) level instead of the low-level scheduling algorithms (allocation policies). In addition, our model is not bound to any particular workload/scenario, but changes dynamically the scheduling model according to the environment, cluster and workload under consideration.

3. Scheduling model

In this section we will define the basic notation and explain the background of the scheduling models we use in our research.

3.1. Notation and scheduling model backgrounds

Let us denote by j a job arriving to the cloud system for scheduling.

Definition A job j in the scheduling model is the collection of atomic tasks submitted by a given (the same) cloud end-user. Such tasks can be executed independently or present dependencies between them.

In the simplest case, the cloud job may consists of just a single task. Formally, the job which is composed of k tasks is denoted by j_k and can be defined as follows:

$$j_k = (t_1, \ldots, t_k)\,, \tag{1}$$

where t_i denotes the i–th task in the job j_k.

In the case of existence of internal relations among tasks in j_k, such job can be modelled by the *Directed Acyclic Graph* (DAG) model in the following way:

Definition The Directed Acyclic Graph DAG_{j_k} used for modelling the job j_k is defined as the following graph without cycles:

$$DAG_{j_k} = (V_k; E_{e_k}), \quad (2)$$

where $V_k = \{t_1, \ldots, t_k\}$ is the set of vertices which represents the set of tasks in j_k, and E_{e_k} is the set of edges, which defines the tasks dependencies (interrelations).

The cloud jobs can be characterized by a wide set of parameters [26]. In this work, we focus on the following two main attributes of the job j_k:

- **Job inter-arrival time** TI_{j_k}- is the time needed for job j_k to arrive to the system and to make it ready for scheduling and execution. This parameter may define the number of jobs which can be scheduled and executed in a specified time window.
- **Job duration time** TD_{j_k}- is the completion time of the j_k job in the cloud cluster.

The efficiency of the cloud schedulers is monitored in the process of mapping many jobs in a specified time window. This set of jobs is defined as a workflow. The processing time of the whole workflow is the main scheduling quality criterion.

Definition A workflow W_s is the collection of s jobs submitted to the cloud system.

Usually, the term *workload* refers to all cloud system inputs such as applications, services and data transfers. In cloud computing, these inputs usually correspond to online interactions of the cloud users with cloud services hosted in the cloud clusters or to jobs processed in such clusters. It is also important to mention that cloud workloads almost never refer to hard real-time applications.

3.2. Workload classification– batch and service jobs in Google cloud schedulers

The quality of cloud services, especially for big data processing, depends on the quality of scheduling process. The workflows can be organized in the different ways based on the (i) job configuration (workflow internal architecture) or (ii) job processing model. In the first case, the workflow is described by the number and types of services or jobs being instantiated by a cloud application and their mutual dependencies. The jobs in this model can be executed as a pipeline, in parallel or according to the hybrid mode usually defined as a DAG of jobs [4]. In the second case, there are two main policies of the cloud job processing:

- **Batch workload** BW– this workload is defined as a batch of jobs with specified arrival, start and completion times.
- **Service workload** SW – this workload is composed of the long-running jobs with not determined completion times.

There are many practical examples of such workflows, such as typical MapReduce workflows [9] are of BW type, and Web servers or services like BigTable [5] are good examples of SW-s.

In this paper, we focus on the analysis of two types of centralized schedulers mentioned in Section 2.2.1, namely: (i) two-level schedulers, such as Mesos, and (ii) shared-state schedulers, such as Omega, developed by Google [34]. We define realistic workflows, which are composed of BW and SW workloads [1], [15]. A comprehensive simulation analysis is presented in the next section.

4. Experimental analysis

The main aim of our simple experimental analysis is to show the effectiveness of the dynamic selection of the proper cloud scheduler for a given workload depending on its attributes. We considered the two-level and shared-state schedulers for the experimental analysis of this work, namely (i) *Omega* [34]; and (ii) *Mesos* [20], since they are different scheduling strategies inside the centralized scheduling model, which makes them theoretically optimal for the same scenarios. This simple comparison enables us to easily analyse the behavioral differences on the same environment.

We created the sequence of workflows of the both types BW and SW. First, we scheduled all workflows by using the same considered scheduler, then based on the observed scheduling results, we mapped the concrete workflows to the proper scheduler by switching dynamically the scheduling model for the given workload. We wanted to show in our experiments how much the quality of cloud service and data processing depends on the workload attributes (parameters) and the types of the schedulers used for mapping the workload jobs onto the cloud resources.

Our experiments have been conducted for 15 days.

4.1. Simulation tool and environment

The SCORE simulator [17] was used in this work. This tool allows to generate realistic cloud workloads of different types and it is prepared for the implementation of various scheduling models. The SCORE simulator does not simulate all real-life cluster and data center operations, but it was not very essential in the research presented in this paper.

The cloud cluster employed to run the simulations is composed of 10 000 machines of 4 CPU cores and 16 GB of RAM. In the scheduler model we have one centralized node and 5 scheduling agents: 4 may work in parallel for scheduling the BW jobs and 1 is defined for scheduling the SW jobs.

4.2. Workload settings and scenarios

In order to create realistic cloud scheduling scenarios, the following workflow parameters were defined:

- **Job structure**: The number of tasks for each job in BW is sampled from an exponential distribution with a mean value is 180, while the number of tasks for each job in SW is sampled from an exponential distribution whose mean value is 30.

- **Task duration**: The duration of BW-jobs tasks is sampled from an exponential distribution whose mean value is 90 (seconds). For SW-jobs tasks, this duration is sampled from an exponential distribution with a mean value of 2000 (seconds).
- **Resource usage**: in SW-jobs tasks consume 0.3 CPU cores and 0.5 GB of memory, in SW-job tasks consume 0.5 CPU cores and 1.2 GB of memory.

.

We consider in our experiments the following four levels of cloud physical cluster utilization:

1) **Low utilization level** – in this case, the cluster utilization is very low and the structure of the considered workloads is as follows: BW_L – 17,000 jobs and SW_L – 1,500 jobs.
2) **Low-Mid utilization level** – in this case, the cluster utilization is higher than in the previous scenario, and the workflow structure is: BW_{LM} – 43,000 jobs and SW_{LM} – 4,300 jobs.
3) **Mid-high utilization level** – in this case, the cluster utilization is noticeably higher. The structure of the workloads to be executed is as follows: BW_{MH} – 52,000 jobs and SW_{MH} – 5,200 jobs.
4) **High utilization level** – in this case, the demand of resources of the workload exceeds the total computational capacity of the cloud cluster. The structure of the workloads under consideration is as follows: BW_H – 130,000 jobs and SW_H – 13,000 jobs.

The detailed parameters of these workloads are presented in Table 2

We consider also the following three scheduling scenarios:

- **Scenario 1** – all the workloads defined in Table 2 are served by a shared-state scheduler, that is, the *Omega* scheduler
- **Scenario 2** – all the workloads defined in Table 2 are served by a two-level scheduler, that is, the *Mesos* scheduler
- **Scenario 3** – in this case cloud cluster utilization fluctuates constantly between a *Mid-High* usage period and a *Low-usage* period every 12 hours during the whole simulation time (15 days). This kind of resource-usage pattern can be found in many web applications where the users are located in one geographical region and suffer from day-night or weekday-weekend patterns. This scenario is designed for the comparison of the efficiency achieved by using only one scheduling model compared to the dynamic application of many scheduling frameworks.

4.3. Scheduling efficiency measures

For the comparison and evaluation of the performance of the considered schedulers, we define the following two scheduling efficiency measures:

- JQT_{fi} **–Job queue times until first scheduled**: This parameter represents the time a job needs to wait in queue until it scheduled for the first time.
- JQT_{full} **– Job queue times until fully scheduled**: This parameter represents the time a job needs to wait in queue until it is fully scheduled.

4.4. Results

Table 3 presents the results of the experiments in scheduling Scenarios 1 and 2 and Table 4 shows the results of the experiment in Scenario 3.

In this work, we only present the results of the batch workloads BW. The results obtained for SW workloads show that the influence of such workloads on the performance of the whole CC cluster is very minor and can be ignored.

4.4.1. Scenarios 1 (Omega) and 2 (Mesos). In this sections, the simulation results are discussed in terms of the utilization periods:

1) **Low-usage period.** In this scenario, the pessimistic blocking approach of *Mesos* causes an overhead which leads to higher response times and worse user experience. On the other hand, *Omega* scheduler achieves a 10x reduction on average, as shown in Table 3.
2) **Low-mid usage period results.** In this period, the *Omega* scheduler reduce the average time that a job spend on queue until its first task is scheduled by 40%, and 20% until all its tasks are scheduled.
3) **Mid-high usage period.** In this scenario, the optimistic approach used by *Omega* causes a high number of conflicts and the related need to repeat those scheduling decisions. The user experience is therefore worsened. It can be observed in Figure 7 that the time a job needs to wait in queue until it scheduled for the first time is approximately 20% longer on average in this period. In the same way, the time a job needs to wait in queue until it is fully scheduled is approximately double when the *Mesos* scheduler is used. These results differ from those presented in the *Low-mid* period, where *Omega* performs better than *Mesos*. The time, when workload pattern fluctuates between *Low* and *High* usage periods, would be the optimal moment for changing the scheduling models.
4) **High-usage period.** In this period, the time a job has to wait in queue until its first task is scheduled is approximately five times higher on average for the *Omega* scheduler. Hence, this result confirms the trend presented for the mid-high usage period. However, a much lesser impact can be observed for the time a job needs to wait until all its tasks are scheduled on average, where the *Omega* scheduler is approximately 20% slower.

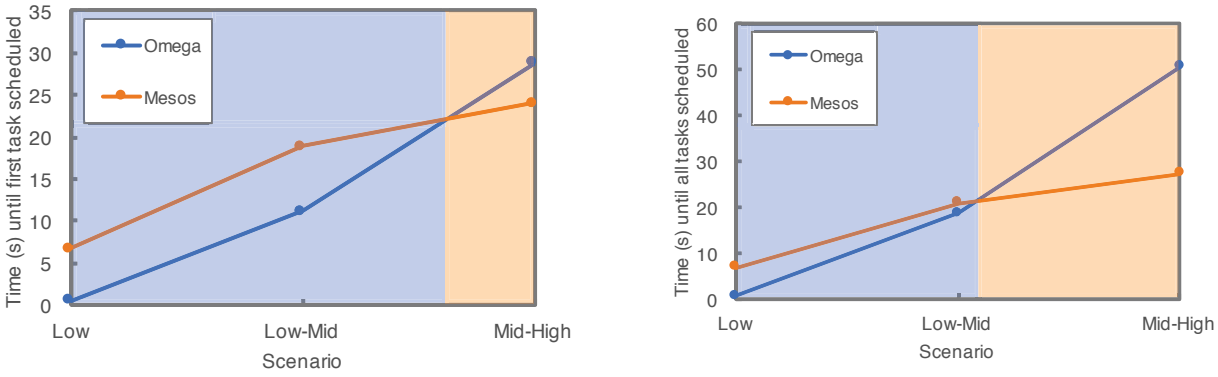
4.4.2. Scenario 3. In this scenario, it can be noticed that the utilization of the *Omega* scheduler for *Low utilization* periods and the *Mesos* scheduler for *Mid-high utilization* periods results in important benefits in terms of performance. Regarding the time for a job to be fully scheduled, the results delivered by this

Table 2: Workload environments. BW - *Batch* workload, SW - *Service* workload.

Workload	Load	Inter-arrival time (s)		#Jobs (10^3)		#Tasks per job		Duration (s)		CPU task (core)		RAM task (GB)	
		BW	SW	BW	SW	BW	SW	BW	SW	BW	SW	BW	SW
Low	∼30%	75	750	∼17	∼1.7	180	30	90	2000	0.3	0.5	0.5	1.2
Low-Mid	∼45%	30	300	∼43	∼4.3	180	30	90	2000	0.3	0.5	0.5	1.2
Mid-High	∼50%	25	250	∼52	∼5.2	180	30	90	2000	0.3	0.5	0.5	1.2
High	∼75%	10	100	∼130	∼13.0	180	30	90	2000	0.3	0.5	0.5	1.2

Table 3: Performance indicators for all Batch workload scenarios (BWs) and Scheduling Scenarios (SS) 1 and 2.

Indicator (s)	BW_L		BW_{LM}		BW_{MH}		BW_H	
	S1	SS2	SS1	SS2	SS1	SS2	SS1	SS2
JQT_{fi} (avg.)	0.51	6.72	11.18	18.89	28.80	24.00	117,040.50	22,968.87
JQT_{fi} (90p.)	0.01	20.78	39.22	55.49	102.59	69.05	210,608.19	75,633.55
JQT_{full} (avg.)	**0.67**	6.95	**18.65**	20.88	50.45	**27.30**	193,811.39	**158,647.73**
JQT_{full} (90p.)	0.01	21.50	53.47	61.31	139.16	78.31	405,835.29	441,712.19



(a) Average queue time for jobs until their first task is scheduled (b) Average queue time for jobs until all their tasks are scheduled

Figure 7: Performance of Mesos and Omega scheduling frameworks in each scenario. The periods included in blue background area achieve better results when the Omega scheduler is used, while the orange background area means the same for Mesos. It can be noticed that, between the *Low-Mid* and *Mid-High* period Omega starts to perform worse than Mesos. Thus, a switch to Mesos would deliver the optimal results. The exact threshold depends on the parameter (objective function) desired.

scheduler-switching approach are approximately 50% better on average than those of *Mesos* or *Omega*, as shown in Table 4.

5. Conclusions and Future work

In this work, we tried to confirm the high dependency of the quality of the cloud services on the results of the scheduling of complex heterogeneous workloads in the physical cC clusters. The results of our simple experiments show the high motivation for development of an intelligent automated scheduler management system, which allows to adapt the scheduling model to the concrete workloads based on their characteristics and the cluster architectures.

The research presented in this paper is just the first step in our work on such scheduler management system. The system should use the feedback of using the implemented schedulers together with the results of monitoring of the workload executions in the cloud clusters. We believe that the machine learning methods, already used for supporting the cloud resource management [19], would be the proper tools for solving the scheduler management problem.

Acknowledgement

The research is supported by the VPPI - University of Sevilla and COST Action IC1406 ”High-Performance Modelling and Simulation for Big Data Applications” (cHiPSet).

References

[1] Abdul-Rahman, O.A., Aida, K.: Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon.

Table 4: Performance indicators for the Batch workload (BW) in Scenario 3.

Indicator (s)	Omega fixed	Mesos fixed	Omega-Mesos
JQT_{fi} (avg.)	14.65	15.36	12.25
JQT_{fi} (90p.)	51.29	44.89	34.75
JQT_{full} (avg.)	25.56	22.13	**13.99**
JQT_{full} (90p.)	69.58	49.90	39.15

In: IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 272–277. Singapore (Dec 2014)

[2] Ananthanarayanan, G., Ghodsi, A., Wang, A., Borthakur, D., Kandula, S., Shenker, S., Stoica, I.: Pacman: Coordinated memory caching for parallel jobs. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. pp. 20–20. USENIX Association (2012)

[3] Barroso, L.A., Clidaras, J., Hölzle, U.: The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis lectures on computer architecture 8(3), 1–154 (2013)

[4] Calzarossa, M.C., Della Vedova, M.L., Massari, L., Petcu, D., Tabash, M.I., Tessera, D.: Workloads in the clouds. In: Principles of Performance and Reliability Modeling and Evaluation, pp. 525–550. Springer (2016)

[5] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) 26(2), 4 (2008)

[6] Chen, M., Mao, S., Liu, Y.: Big data: A survey. Mobile networks and applications 19(2), 171–209 (2014)

[7] Chen, Y., Alspaugh, S., Katz, R.: Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. Proceedings of the VLDB Endowment 5(12), 1802–1813 (2012)

[8] Corporation, M.: Microsoft azure (2018), https://azure.microsoft.com

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

[10] Delgado, P., Didona, D., Dinu, F., Zwaenepoel, W.: Job-aware scheduling in eagle: Divide and stick to your probes. In: Proceedings of the Seventh ACM Symposium on Cloud Computing. No. EPFL-CONF-221125 (2016)

[11] Delgado, P., Dinu, F., Kermarrec, A.M., Zwaenepoel, W.: Hawk: Hybrid datacenter scheduling. In: USENIX Annual Technical Conference. pp. 499–510 (2015)

[12] Delimitrou, C., Kozyrakis, C.: Paragon: Qos-aware scheduling for heterogeneous datacenters. In: ACM SIGPLAN Notices. vol. 48, pp. 77–88. ACM (2013)

[13] Delimitrou, C., Kozyrakis, C.: Quasar: resource-efficient and qos-aware cluster management. In: ACM SIGPLAN Notices. vol. 49, pp. 127–144. ACM (2014)

[14] Delimitrou, C., Sanchez, D., Kozyrakis, C.: Tarcil: reconciling scheduling speed and quality in large shared clusters. In: Proceedings of the Sixth ACM Symposium on Cloud Computing. pp. 97–110. ACM (2015)

[15] Di, S., Kondo, D., Franck, C.: Characterizing cloud applications on a Google data center. In: 42nd International Conference on Parallel Processing (ICPP). Lyon, France (Oct 2013)

[16] Dogar, F.R., Karagiannis, T., Ballani, H., Rowstron, A.: Decentralized task-aware scheduling for data center networks. In: ACM SIGCOMM Computer Communication Review. vol. 44, pp. 431–442. ACM (2014)

[17] Fernández-Cerero, D., Fernández-Montes, A., Jakóbik, A., Kołodziej, J., Toro, M.: Score: Simulator for cloud optimization of resources and energy consumption. Simulation Modelling Practice and Theory 82, 160–173 (2018), https://doi.org/10.1016/j.simpat.2018.01.004

[18] Gog, I., Schwarzkopf, M., Gleave, A., Watson, R.N., Hand, S.: Firmament: Fast, centralized cluster scheduling at scale. Usenix (2016)

[19] Gondhi, N.K., Gupta, A.: Survey on machine learning based scheduling in cloud computing. In: Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence. pp. 57–61. ACM (2017)

[20] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S., Stoica, I.: Mesos: A platform for fine-grained resource sharing in the data center. In: NSDI. vol. 11, pp. 22–22 (2011)

[21] Inc., A.: Amazon web services description (2018), https://aws.amazon.com/es/documentation

[22] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., Goldberg, A.: Quincy: fair scheduling for distributed computing clusters. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. pp. 261–276. ACM (2009)

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

[24] Kołodziej, J.: Evolutionary hierarchical multi-criteria metaheuristics for scheduling in large-scale grid systems, vol. 419. Springer (2012)

[25] Mars, J., Tang, L.: Whare-map: heterogeneity in homogeneous warehouse-scale computers. In: ACM SIGARCH Computer Architecture News. vol. 41, pp. 619–630. ACM (2013)

[26] Nallakumar, R., Sengottaiyan, N., Priya, K.S.: A survey on scheduling and the attributes of task scheduling in the cloud. Int. J. Adv. Res. Comput. Commun. Eng 3(10), 8167–8171 (2014)

[27] Niu, Z., Tang, S., He, B.: Gemini: An adaptive performance-fairness scheduler for data-intensive cluster computing. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 66–73 (Nov 2015)

[28] Ousterhout, K., Wendell, P., Zaharia, M., Stoica, I.: Sparrow: distributed, low latency scheduling. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. pp. 69–84. ACM (2013)

[29] Polo, J., Castillo, C., Carrera, D., Becerra, Y., Whalley, I., Steinder, M., Torres, J., Ayguadé, E.: Resource-aware adaptive scheduling for mapreduce clusters. In: Proceedings of the 12th International Middleware Conference. pp. 180–199. Middleware '11, International Federation for Information Processing, Laxenburg, Austria, Austria (2011), http://dl.acm.org/citation.cfm?id=2414338.2414352

[30] Qu, H., Mashayekhi, O., Terei, D., Levis, P.: Canary: A scheduling architecture for high performance cloud computing. arXiv preprint arXiv:1602.01412 (2016)

[31] Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing. p. 7. ACM (2012)

[32] Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA (Nov 2011), revised 2012.03.20. Posted at url-http://code.google.com/p/googleclusterdata/wiki/TraceVersion2

[33] Ren, K., Kwon, Y., Balazinska, M., Howe, B.: Hadoop's adolescence: an analysis of hadoop usage in scientific workloads. Proceedings of the VLDB Endowment 6(10), 853–864 (2013)

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

[35] Shue, D., Freedman, M.J., Shaikh, A.: Performance isolation and fairness for multi-tenant cloud storage. In: OSDI. vol. 12, pp. 349–362 (2012)

[36] Tumanov, A., Zhu, T., Park, J.W., Kozuch, M.A., Harchol-Balter, M., Ganger, G.R.: Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Proceedings of the Eleventh European Conference on Computer Systems. p. 35. ACM (2016)

[37] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing. p. 5. ACM (2013)

[38] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at google with borg. In: Proceedings of the Tenth European Conference on Computer Systems. p. 18. ACM (2015)

[39] Yang, H., Breslow, A., Mars, J., Tang, L.: Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In: ACM SIGARCH Computer Architecture News. vol. 41, pp. 607–618. ACM (2013)

[40] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European conference on Computer systems. pp. 265–278. ACM (2010)

[41] Zhang, X., Tune, E., Hagmann, R., Jnagal, R., Gokhale, V., Wilkes, J.: Cpi 2: Cpu performance isolation for shared compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems. pp. 379–391. ACM (2013)