UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EXPLORING ADAPTIVE JOB SCHEDULERS FOR GEOGRAPHICALLY DISTRIBUTED DATA CENTERS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Daniel Santos Ferreira Alves

June 2022

The Dissertation of Daniel Santos Ferreira Alves
is approved:

Katia Obraczka, Chair

Peter Alvaro

Abdul Kabbani

Peter F. Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Exploring Adaptive Job Schedulers for Geographically Distributed Data Centers

by

Daniel Santos Ferreira Alves

To help meet the ever increasing demand for cloud computing services worldwide, while providing resilience and adequate resource utilization, cloud service providers have opted to distribute their data centers around the world. This trend has been motivating research from the data center management research and practitioner community on new job schedulers that take into account data center geographical distribution. However, designing, testing and benchmarking new schedulers for geo-distributed data centers is complicated by the lack of a common, easily extensible experimental platform. To fill this gap, we propose GDSim, an open-source, extensible job scheduling simulation environment for geo-distributed data centers that aims at facilitating the benchmarking of existing and new geo-distributed schedulers. Using GDSim, job schedulers specifically designed for geo-distributed data centers can be tested, validated, and evaluated under a variety of data center workloads and conditions. We use GDSim to reproduce experiments and results for recently proposed geo-distributed job schedulers, as well as testing those schedulers under new conditions which can reveal trends that have not been previously uncovered. We demonstrate how GDSim can be used to design and evaluate different adaptive job schedulers, which, based on current workload and data center conditions, use heuristics to select the most appropriate scheduler.

To my friends and family, who helped me through this time.

Acknowledgments

I want to express my deepest appreciation to Katia Obraczka, Abdul Kabbani for their general guidance, both in my research and outside of it. I'm also grateful to my lab colleagues for the discussions and general company. I'd like to thank the professors I crossed paths with such as Peter Alvaro, J.J. Garcia-Luna-Aceves, Mike Parsa and Tracy Larrabee. I'd also like to recognize the help of Rick Lindberg with the work of change point detection. I'm also grateful to L.S. Kim for hosting me for more than seven years. Many thanks to my friends and family for supporting me through these seven years. And special thanks to my advisors in Brazil, Felipe França and Priscila Lima, without whom I probably wouldn't be on this path.

Chapter 1

Introduction

Current practice in large computing companies involves the usage of data centers for processing of diverse computing tasks. Data centers are buildings dedicated to hosting hundreds or even thousands of computing systems. Placement of computing jobs on those servers is not done manually but instead relies on algorithms to provide completion of jobs in a reasonable amount of time, according to the expectations of clients, achieving adequate utilization of data centers resources without overloading them.

Data centers handle different types of workloads. Some data centers might be used mostly for compute heavy jobs and thus will be equipped with servers with powerful processors, but proportionally smaller storage capacity, while other data centers may need larger storage capacity. In practice data centers can have a mix of resources such that can handle different types of workloads and thus it is imperative to have efficient algorithms to allocate jobs in order to achieve performance in terms of job completion time.

For large cloud provider companies such as Amazon, Facebook and Google, having an efficient job scheduler is an even more complicated problem when compared to smaller companies that manage centralized data centers located in a single campus or in very close proximity of one another. The scale of large cloud providers operations means that they have to distribute their data centers geographically and schedule jobs across these geographically distributed data centers. Under these circumstances, the effect of the underlying communication network that connects the geo-distributed data center must be taken into account when scheduling jobs. While job schedulers that account for these constraints have been proposed, their evaluation seems to be restricted to a limited set of execution scenarios. In this work we explored schedulers that adapt to different data center conditions such as workload. We developed a set of tools to compare different schedulers using publicly available job traces for evaluation, and also synthetic job traces that mix features from existing traces to allow for more variety of scenarios. Using these tools we can better explore how to build a job scheduler better suited to the particulars of processing job workloads in a set of data centers, such as the ones we explored.

The rest of this dissertation is as follows: Chapter 2 presents the context of the scheduling problem for computing in general and data centers in particular. Chapter 3 describes scheduling in the context of geographically-distributed data centers, in particular showing the algorithms for existing schedulers that are used through our work. Chapter 4 introduces the tools that we used in our study for simulation and generation of synthetic traces. Chapter 5 presents the experiments we used to validate our tools. Chapter 6 brings the exploration of adaptive schedulers and a comparison with some existing schedulers. Chapter 7 concludes this dissertation with a summary of the work done.

1.1 Main Contributions

The main contributions of this research are as follows:

1. A general platform to evaluate job schedulers that specifically target geographically-distributed data centers. This platform allows us to model and test different scheduling algorithms and compare them in varied scenarios, as well as account for the impact of the scheduler on the network.

2. As part of GDSim, we built a workload generator which is able to build synthetic workloads based on existing traces.

3. We investigate different heuristics that can be employed by geo-distributed job schedulers in order to adapt to different workloads, network and data center conditions.

1.2 Prior Accomplishments

Before studying job scheduling in geo-distributed data centers, we researched three distinct topics: delay-tolerant networks, RTT distribution throughout the world, and identification of relevant telemetry in data centers. Delay-tolerant networks (also known as DTNs) are networks that present arbitrarily long-lived delays and/or connectivity disruptions that make the application of traditional transport protocols such as TCP impractical. Examples of DTN applications include space communication, and scenarios where fixed infrastructure is non-existent or has been severely compromised.

The study of RTT distribution was based on data provided by a private traffic management company. The study allowed us to identify trends in RTT distribution based on location as well as patterns that are globally present [4].

Finally, our work on identifying relevant data center telemetry was based on the idea of narrowing which telemetry time series show variations that seem to correspond to relevant events. Telemetry not deemed relevant could be then filtered out, which can reduce substantially the amount of data that needs to be analyzed. We used a method based on change point detection to compare variations in telemetry to those observed during relevant events as way to systematically filter the available data [3].

Chapter 2

Background

In this chapter we present the problem of job scheduling, as well as related work that has already been developed in this area. The continuous growth of cloud computing results in new problems and the need for corresponding solutions. As the popularization of data centers required the development of new techniques for storage, networking and scheduling, among others, the spread of data centers across the globe presents new complications to be solved, as can be seen in surveys covering this topic [12, 6, 26]. In particular, the distance between data centers that cooperate means that the cost of transferring data between those will have to be considered when deciding how to locate data and where to execute programs. For this dissertation we will focus on the latter problem, the planning of execution schedules for jobs that are submitted for execution across those data centers. This problem is an instance of a classical computing problem, the scheduling problem, which we detail in the next section.

2.1 The Scheduling Problem

The scheduling problem is a classical computing problem, which can be NP-hard depending on the constraints imposed. The core of the problem is that someone has a set of jobs to execute and limited resource capacity to execute the jobs, so they have to distribute the jobs over the available resources. That distribution is the aforementioned scheduling that characterizes this class of problems. The main elements in this problem are the jobs, the resources, the constraints connecting those, and the goals. Figure 2.1 shows an example of job scheduling.

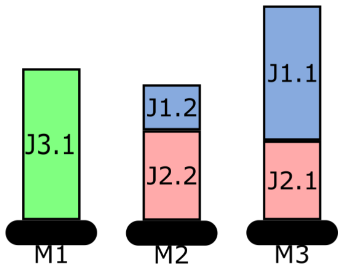


Figure 2.1: Example of job scheduling.

Jobs are the actions to be executed. They can be composed of multiple tasks that can be executed in parallel. The problem might have all jobs available from the start, or have some jobs arrive after others. Jobs require access to resources executed, and they might also have dependencies, either to other jobs, or internally among their tasks. In Figure 2.1, the tasks are identified as blocks with their lengths corresponding to their duration. In this example there are three jobs, J1, J2 and J3, the first two with two tasks, and the last one with only one.

A computing example of a job would be the execution of a query for data. That query can be broken down in multiple subqueries, each one corresponding to a task. The aggregation of the results from all subqueries into one result would be another task. The resources that each task requires are CPU time to process it, memory capacity for the operation, as well as access to the data to the queried.

That leads us to the next element of a scheduling problem, the resources. Those are the parts that make it possible to execute a job to completion. Processors, memory, storage and network are examples of resources involved when scheduling computing jobs. While those resources can be shared by different programs, they can also be partitioned into units for ease of scheduling. For example, multiple tasks could be scheduled to the same server, and each task could use at most a fraction of the total processor time available, those fractions being the corresponding resource unit for that type of resource. In Figure 2.1 there are three equivalent resources being shared, M1, M2, and M3, which are represented by the black bars at the bottom of the figure.

The constraints then define the relations jobs and resources have. A common constraint is when jobs cannot share a resource. For example, a job might be granted exclusive access to a server for the execution of its tasks. Other types of resource might be accessible to multiple jobs simultaneously, but there's a maximum capacity that these resources can support. Another common constraint is that the resources might be distributed in such a way that accessing one resource limits which other resources are available. Going from our previous example of a query that is to be executed, if our resources are distributed over multiple computers, then a task can only access the resources that are available in the computer it will execute.

Other constraints include dependency relations between tasks of a job, and between jobs. Internal dependencies between the tasks of a job can be simplified by breaking up a job into multiple jobs, such that each resulting job has tasks that have no internal dependencies. In this way we are exchanging internal dependencies for external (between jobs) and can adapt task dependencies for schedulers that do not handle those. Similarly, if we have schedulers that do not handle dependencies between jobs, we can submit a job with dependencies only after the dependencies have completed execution. In our example in Figure 2.1 the constraint is that only one task can use each resource at a time, so the tasks are represented as a stack to indicate the ordering of the queue to access the resource.

As for goals, there are many possibilities, often involving the minimization of a cost metric, with makespan being a common case. Makespan is the measurement of time from the start of the first job to the end of the last job. It provides a simpler way to compare two different schedulings and is often used for that purposed when presenting new algorithms. A user might be also interested in the time that a specific job requires to execute, measured from the moment the job is submitted for scheduling to the conclusion of its last task, which we will call the latency of a job. As there are usually multiple jobs running in a given schedule, the latency of the schedule must be measured as a statistic of the distribution of latencies of all jobs. Common statistics for that are the mean, the median, and n-percentiles such as the 90th- or 99th-percentile. These last two are specially relevant as they represent the latency affecting the tail of the distribution, that is, the results of the jobs that perform worse than 90% or 99% of the jobs. Due to the high amount of jobs that a data center sees in its operation, these percentiles still represent a large number of jobs.

Another interesting metric could describe the operational cost of the resources. A measurement of resource utilization, either during the entire scheduling or as a function of time, is also another useful way to compare scheduling performance, but it must be considered in conjunction with other performance metrics.

Referring once again to our example in Figure 2.1, we can measure the total makespan of the schedule as the length of the tallest stack of tasks, and we can measure the latency of each job similarly, but considering only the blocks that represent the corresponding job.

2.2 Job Scheduling in Computing

Job scheduling is an important problem in computing and present at different contexts. As examples of that importance, job scheduling is used by the operating system to manage concurrent operation of programs [31], in distributed systems to manage execution of programs across the machines that compose the system [23], in data [44, 22] to define which cluster will execute a program, and can also be used to define which data center will receive the program to be executed, which is the focus of this dissertation.

While at a superficial level those applications are similar, a more detailed examination will reveal significant differences. For instance, in the context of an operating system meant for interactive use, the goal is usually to allow the program to have access to resources in a way that user perceives them as operating concurrently. Meanwhile, when allocating a set of tasks in a distributed system, the goal might be to minimize the total time spent executing those. Inside a data center, it would be important not only to minimize the time executing the scheduled jobs, but also minimize the use of energy required in this [1, 25].

2.3 Job Scheduling in Data Centers

Scheduling in data centers is a very important problem for data center operators, as evidenced by the wealth of literature available [44, 22, 47, 13, 25]. There are many factors to consider, such as securing good performance for all jobs, managing access to shared resources such as disk and network, and ensuring that the performance of a job is not heavily impacted by the slowest tasks. Even in terms of considering how to model the unit of execution there are many options, such as packet, flow, coflow, task and job. While the latter two terms refer to the work executed in the computers in the data center, the first three refer to the data that is transferred as part of this work. Scheduling reflects then those different models, either focusing on the application that is being deployed on the data center (jobs and tasks), or on the traffic that will be generated [47]. We also find works that operate in the intersection of both models, such as the model of network tasks proposed by Giroire et al. [14]. The cost of operation is another important factor when considering the usage of a data center, and that has given rise to schedulers that aim to minimize energy consumption while preserving performance [1].

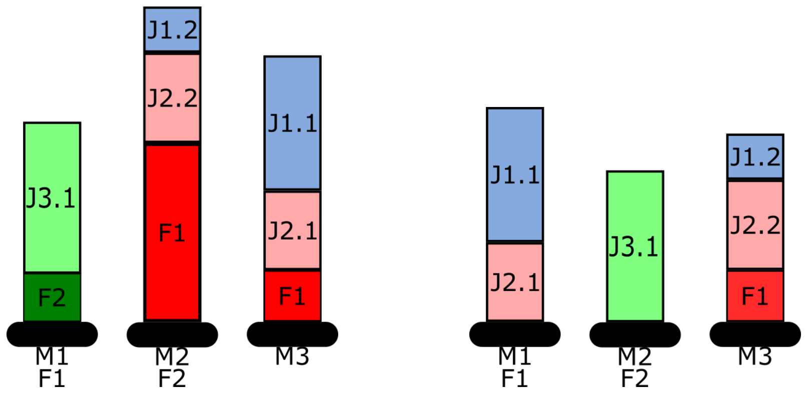
Chapter 3

Job Scheduling in Geo-Distributed Data Centers

The problem of scheduling for geo-distributed data centers refers to allocating tasks to data centers that are spread across the Earth. This goes beyond traditional data center scheduling problems, which focus on the scheduling of jobs inside a data center, because the distance between data centers, and the consequent impact in transferring data, present new complications to estimation of resource utilization. As data is not replicated equally across all data centers, allocating a task to a data center must also consider where the data is required for execution, and how its transfer would impact performance. That impact affects not only the total execution time of the task, since it would have to wait for the data to be available before it can start, but also the overall performance of the data center. A bad schedule could lead to more transfers than the data center has capacity for, reducing the overall performance. It is important then for job scheduling algorithms in geo-distributed data centers to consider not only the cost of data transfer for scheduling, but also the resulting impact on the data center network.

Figure 3.1 shows an example similar to the one we discussed in Chapter 2, but with files (F1 and F2) restricted to two locations, and data transfer times represented as blocks in a darker color. For this example, we define that J1 and J2 both depend on F1, and J3 depends on F2. On the left side, we see the original scheduling as shown in Figure 2.1, but with added times to transmit files F1 and F2. Notice that the transmission times can change according to the destination. On the right side, we switch the tasks around to minimize the transfer times.

First we transfer the tasks between M1 and M2, removing the need to transfer F2 for J3.1 and F1 for J2.2 and J1.2. This already reduces the overall makespan, but we can improve further on this by switching the tasks between M1 and M3. Since the tasks currently in M3 require longer to execute, they would benefit more from not having to transfer the data. The result is what we see on the right side of Figure 3.1.



(a) Bad scheduling **(b)** Better scheduling

Figure 3.1: On the left side we have the result of a naive scheduling, on the right side we have a scheduling that minimizes transfer times.

3.1 Geo-Distributed Job Scheduling Algorithms

In this section we present some of the algorithms that have been already published for scheduling in geo-distributed data centers. Most of those algorithms propose heuristics, while a small group use approximations to model the scheduling as a linear programming model, identifying an optimal solution under certain constraints.

3.1.1 SWAG

Workload-Aware Geo-distributed Scheduling [18] (SWAG) was one of the first proposals for job scheduling in geo-distributed data centers. Its introductory paper presents two heuristics and a comparison against a well-known heuristic that works well in a "single-server-single-queue" scenario, Shortest Remaining Processing Time (SRPT). The proposed heuristics are called Reordering and SWAG, and they are compared against the SRPT algorithm with two extensions to handle multiple servers and queues.

The idea for SRPT is to calculate for each job the total processing time measured as the sum of the expected duration of each task in the job, then schedule the jobs from the smallest processing time to the largest. When a job is scheduled, the tasks are scheduling starting with the longest tasks and ending with the smallest. It is a simple heuristic, but works well for a single data center scenario [5]. The extensions used to adapt it to the geo-distributed scenario are Global-SRPT and Independent-SRPT.

Global-SRPT works by applying the same principles of SRPT scheduling to all data centers. It allocates the tasks of the job with least remaining processing time to data centers with free slots. Independent-SRPT, on the other hand, is applied inside each data center to schedule the tasks that are allocated to that data center.

The first new heurist proposed, Reordering, aims to improve job completion times by modifying the resulting queues of tasks at each data center, which were themselves the result of another scheduling algorithm. The steps to implement it are:

1. identify the data center D with the longest queue;
2. identify the job J of the last task in said queue;
3. add job J to a queue data structure Q;
4. remove all tasks associated with job J from the queue for data center D;
5. repeat from step 2 until no tasks remain in the queue for data center D;
6. reschedule all jobs in N in the queue for data center D in the reverse order they were added to Q.

While Reordering does not affect the total makespan of the scheduling, it can optimize job completion times by switching tasks around such that no job will have a worse completion time, but a job might have a better completion time because of it. In essence, Reordering is sorting all the tasks in the execution queue according to the resulting end time for their jobs. The last ending job will still end at the same moment, but other tasks it has in that queue will be moved to later spots, allowing other jobs to finish earlier, and that process is repeated throughout the queue.

SWAG's heuristic, unlike Reordering, does not require an existing schedule. The idea is similar to SRPT, using expected job makespan instead of just processing time. The way it works is that during scheduling, the algorithm considers the resulting makespan of each job, if that job were the only one to be schedule.

This makespan is calculated by placing the tasks from that job, from the longest to the shortest, in the data center with the shortest queue that has the required data. Once these estimated makespans are calculated, the jobs are actually scheduled, using the same rules, starting from the job that had the smallest expected makespan.

It is important to note however that it restricts job placement to locations that already contain the required data, either through the original copy or replication. The algorithm for SWAG is more formally described in Algorithm 1.

Algorithm 1 SWAG algorithm

```
$J \leftarrow$ set of all jobs
$\{T_j\} \leftarrow$ set of the sets of tasks for each job $j \in J$
$C \leftarrow$ set of all data centers
$L \leftarrow$ empty list
$\{d_{j,i,c}\} \leftarrow$ set of durations, $d_{j,i,c}$ is the duration of the $i$-task of job $j$ when running on data center $c$
$\{q_c\} \leftarrow$ set of queues, $q_c$ is the length of the queue at data center $c$
while $|J| > 0$ do
    for all $j \in J$ do
        $m_j \leftarrow \max_{c \in C, i \in T_j} q_c + d_{j,i,c}$
    end for
    $k \leftarrow \arg\max_{j \in J} m_j$
    add $k$ to end of $N$
    $J \leftarrow J - \{k\}$
end while
schedule all jobs in the reverse order they were inserted in $N$
```

3.1.2 Flutter and Scheduling with Max-Min Fairness

Flutter [17] is an approach based on finding the optimal solution with linear programming, intended for use in the Spark framework. While the job scheduling problem can be NP-hard, Flutter uses an approximation to a problem of lexicographical ordering to create a model that can be solved in linear time. This approximation, however, involves constraints that can limit the application of this mechanism. For instance, the scheduling is restricted to placing tasks only on free servers. That means that two tasks could not use the same servers, one after the other, even if the cost of transferring data would mean that this results in a better schedule. Another restriction that limits Flutter applicability is that it considers all tasks as belonging to the same job, so it makes no attempt at minimizing the latency of individual jobs.

The work was later extended by Chen et al. [8], who removed the limitation of scheduling tasks of only one job. The intuition for this change is to optimize for the worst completion time among all the jobs being scheduled. Their solution then is to continuously identify the optimal placement for all unscheduled jobs, schedule the one with the worst completion time, and repeat until there are no jobs left to schdule. Although those approaches can be promising, the constraints limit applicability.

3.1.3 Workload-Aware Scheduling

Jin et al. [19] proposed a heuristic that identifies a schedule by performing a local search. In a somewhat similar fashion to Flutter, this proposal was presented in the context of a specific framework, MapReduce in this case. It also presents an analysis that shows that the geo-distributed workload-aware scheduling problem is NP-complete.

Similar to Reordering, the scheduling algorithm proposed here starts from an already feasible solution and improves on that result. The idea to find a neighbor solution is to identify the task that will complete last, given the current schedule, and transfer it to another data center if the transmission costs can be offset.

3.1.4 GeoDis

GeoDis [11] follows on the steps of SWAG [18], but it allows for data migration to other data centers, if that results in better scheduling. The basic intuition of GeoDis is that it works as SWAG, but it considers the cost of migrating the data to the location where the task would be executed. Scheduling works then by calculating the best makespan for all jobs, assuming it would be the only job being scheduled under current data center conditions, then proceed by scheduling fist the job with the lowest expected makespan.

In extending the work presented by SWAG, the authors of GeoDis opted to allow for transfer of data between data centers, and define three rationales to guide the development of their algorithm:

1. Data locality should be favored – and if a transfer is necessary it should come from the source with lowest latency;
2. Tasks from a job should be spread across data centers;
3. Total load among data centers should be balanced too.

While those rationales can at times contradict each other, the authors opted to favor better completion times over avoiding data transfer at all costs. The resulting heuristic is described in Algorithm 2.

3.2 Summary

In this chapter we discussed the problem of scheduling jobs in geo-distributed data centers and how that differs from scheduling jobs in a single data center, or a set of data centers in close proximity. We also discussed four algorithms for

Algorithm 2 GeoDis algorithm

```
$J \leftarrow$ set of all jobs
$\{T_j\} \leftarrow$ set of the sets of tasks for each job $j \in J$
$C \leftarrow$ set of all data centers
$\{d_{j,i,c}\} \leftarrow$ set of durations, $d_{j,i,c}$ is the duration of the $i$-task of job $j$ when running on data center $c$
$\{q_c\} \leftarrow$ set of queues, $q_c$ is the length of the queue at data center $c$
$\{b_{c,g}\} \leftarrow$ set of bandwidths from data center $c$ to data center $g$ (can be considered infinitely large when $c = g$)
$\{l_j\} \leftarrow$ location of data required by job $j$
$\{s_j\} \leftarrow$ size of data required by job $j$
for all $j \in J$ do
    for all $c \in C$ do
        $cq_c \leftarrow q_c$
    end for
    for all $t \in T_j$ do
        $target \leftarrow \arg\min_{c \in C} \left[\min_{g \in l_j} d_{j,t,c} + \frac{s_j}{b_{g,c}}\right]$
        $cq_{target} \leftarrow cq_{target} + \min_{g \in l_j} d_{j,t,target} + \frac{s_j}{b_{g,target}}$
    end for
    $m_j \leftarrow \max_{c \in C} cq_c$
end for
sort all jobs in ascending order according to their score $m_j$
schedule all jobs in the resulting ordering
```

scheduling in geo-distributed data centers: SWAG [18], Flutter [17], Workload-Aware Scheduling [19], and GeoDis [11]. SWAG and GeoDis are both greedy heuristics that schedule jobs based on expected completion time given current load of the data centers. Their main difference is that SWAG only schedules jobs in data centers that have the required data, while GeoDis allows data migration, taking that in consideration for the completion time. This makes GeoDis more flexible and capable of finding better solutions than SWAG, both in terms of completion time and utilization of resources, in scenarios that have higher data concentration in fewer data centers. Flutter, on the other hand, provides an optimal solution under certain constraints, however those constraints limit practical applicability of Flutter. Finally, the Workload-Aware Scheduling proposal uses local search to improve an existing scheduling, so it can be applied to an existing schedule, but it does not provide a solution by itself.

Chapter 4

Simulation and Trace Generation

In this chapter we describe our work establishing a framework to study geo-distributed job schedulers. In order to be able to study the impact of different scheduling mechanisms, the first step is to establish how we can compare existing schedulers, and any proposals we make. For that, there are three possible approaches:

- Using real production data centers;
- Using a testbed;
- Using a simulator.

While using real data centers would give the best results to compare the performance of different schedulers in the conditions faced by these data centers, it is not a realistic environment for initial testing of schedulers due the impact on productivity that would have. testbeds would be the next best option, possessing enough similarity to real data centers to produce meaningful results, without the drawback of taking away from production capacity. However, testbeds are still costly to build, maintain and use so that brings us to the third method, simulation.

Simulators allow us to compare ideas without the high infrastructure cost. This is true not only of job scheduling, but also other areas such as networking [43, 24] or hardware development [42, 7]. While simulators are the least accurate of the methods mentioned, a well-built simulator can still provide enough information to guide development, allowing for usage of more accurate methods like testbeds later. So for this work we started with simulation of geo-distributed job schedulers.

While it would be ideal to start with a well-established simulator, we found no simulator intended for scheduling of tasks across geo-distributed data centers. We found job scheduling simulators for other scenarios, such as job scheduling in clusters [21, 20, 35, 32, 36], but after examining their implementations, we found they would need significant modifications in order to be useful for our research. For these reasons we decided to build a simulator for geo-distributed job scheduling.

Before doing any simulation, however, we will need data to drive these simulations. This data will need to describe the work to be executed by the simulated servers, with description of jobs that are submitted and what data they require. If we used instead completely random values to drive the activity in the simulation, that would be unlikely to match real patterns of usage. It follows that it is important to use workload traces derived from real data center operation, but there are few traces available to the public. That motivated us to also work on a trace generator to create synthetic traces from real workloads in order to be able to run more diverse simulations that still demonstrate realistic features.

Figure 4.1 shows how the trace generator and simulator work together: the simulator will consider two inputs, a description of the topology to be modeled, and of the workload to be executed; the workload can be either real or created by the generator, which still uses original workloads as a basis for the distributions in the synthetic trace. This is represented in the figure by the extractor, which reads from the original traces and provides information that the generator uses to create synthetic traces of the specified size.

Another important feature that we have not mentioned yet is in what mode the scheduler will be configured in the simulation. Since different scheduling approaches can benefit from executing their scheduling in different moments, how the scheduler is called should also be an option, which we will call a hook. Two examples of how we could execute scheduling differently are running the scheduling algorithm every X seconds or every time a new job arrives.

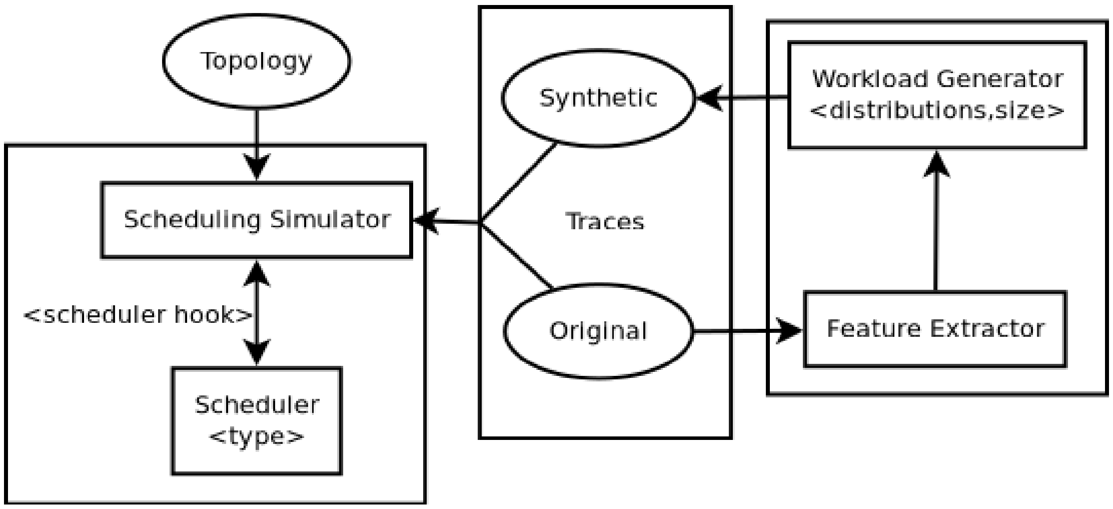


Figure 4.1: Structure for evaluation framework.

With those ideas in consideration, we can discuss the design of our trace generator and our simulator.

4.1 Simulator Design

As mentioned before, when investigating the state of the art in this research we found no specific simulator referenced. Given the lack of geo-distributed data center infrastructure for us to test those scheduling mechanisms, we opted for developing an event-oriented simulator so we can compare different schedulers under the same conditions. Figure 4.2 illustrates the flow of operation for the simulator.

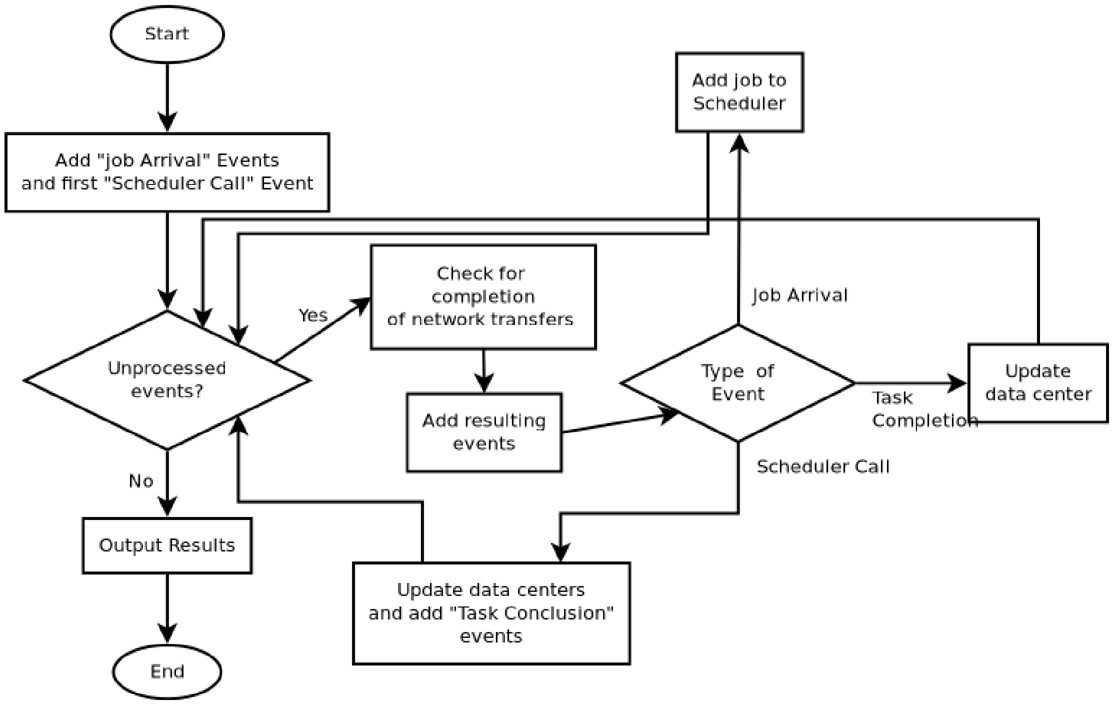


Figure 4.2: Simulation flow.

The simulator works by maintaining a heap of upcoming events, with the next event to occur on top. Events have two main properties, a time when it happens, and what effects it has on the simulated environment. This provides us with a stable base to extend our simulator, an important feature for continued work. Modeled events are arrival of jobs, calls to the scheduler, conclusion of tasks, and requests to transfer data.

The simulated environment includes the scheduler, the data centers and the network. The schedulers follow to the description in Chapter 3. The data center model is simple, keeping track of available and occupied resources, and the tasks occupying these resources, while abstracting the data center's internal structure. We decided for this simple model due to the scope of this simulator, which must be able to handle multiple data centers working together. This limits the applicability of the simulator as a tool to do scheduling for real data centers, but it was an acceptable compromise for the development of this simulator.

Given the importance of preserving realistic networking characteristics, it would be beneficial to leverage existing network simulators instead of duplicating their work. Because of that our simulator separates network events, allowing for future work to move this work to a proper network simulator. This is represented in Figure 4.2 in the step that checks for the status of network events before executing the next event. Currently our simulator only use network events to model data transfers. The network model is simple and attempts to consider bandwidth and latency to estimate how long a transfer will take.

Another important type of event to consider are calls to the scheduler. These trigger the processing of the scheduling mechanism, which sends jobs to the simulated data centers for execution. The specifics of this process will change according to the particular scheduler in use, but in broad strokes the scheduler will consider all jobs that were registered in the job submission since the last scheduler call, create a schedule, and place them on data centers according to that schedule. Placement of tasks on data centers may result in data transfers, which are handled in the network part of the simulation, as described above, and also in task conclusion events, which free resources from data centers once the tasks are done.

Currently our simulator implements the SWAG [18] and GeoDis [11] job schedulers, as well as a naive algorithm based on Shortest Remaining Processing Time (Global-SRPT). The naive algorithm applies the same Global-SRPT extension described by Hung et al. [18] without the restriction of placement in the same location as the data. Since none of these schedulers consider intra- or inter-job dependencies, the simulator currently does not implement dependencies. The scheduler sorts all jobs by the total estimated processing time, not considering possible transfer times that would vary on placement, and places jobs on available data centers starting from the job with the least remaining total processing time. The descriptions for the algorithms employed by SWAG and GeoDis are available in Sections 3.1.1 and 3.1.4, respectively. We selected SWAG and GeoDis because of their relevance as an early geo-distributed job scheduling algorithm and as the first proposal to consider network and locality-awareness, respectively. It is also useful for the purpose of validating the work of our simulator that the original description for GeoDis included results of an experimental comparison against SWAG. We will further discuss this in Chapter 5. Next we describe the workloads that are used to drive our simulations.

4.2 Data Center Workload

In order to have reproducible simulations that model different scenarios, we need to provide our simulator with data that models the jobs that will be scheduled, as well as the files that those jobs require and the topology of the data centers in which the jobs will be scheduled.

The job trace information contains one job description per line with the following information:

1. job identifier: an identifier for referencing the job, but without impact on simulator execution.
2. CPU requirements: a number of computer slots that each task of the job will need to execute.
3. inter-arrival interval: a number measuring the interval after the submission of the previous job.

4. file name: a file identifier.

5. task list: a list of estimated duration, in seconds, for each task that the job has.

Similarly to the job trace information, the file information file contains one description per line with the following attributes:

1. file name: an identifier to allow joining with jobs that use that file.

2. file size: an integer representing the size of the file in bytes.

3. file locations: a list of integers, each one identifying the file as present in the n-th data center.

Lastly, the topology description must contain descriptions for all data centers, following this format:

1. number of data centers: how many data centers are in the topology.

2. number of servers: how many servers are present in the simulated data center.

3. CPU slots per server: how many CPU slots a given server has.

4. list of bandwidths: a list describing the capacity of the links going from a data center to each of the other data centers, measure in kbps.

4.3 Publicly Available Data Sets

A simulator needs workload traces to drive the behavior that will be simulated. This type of data is usually owned by large companies that prefer to keep it private, but some of these companies make it available for research purposes. In this section we describe these execution traces that have been made available by those companies and can be freely used to model realistic production behavior. We also note what information they might lack and how we adapted them for our use.

4.3.1 Facebook

The first data set that we considered was published by Facebook in 2010 and shows activity from Hadoop traces on a 3000-server cluster. They were provided as part of the SWIM Project [9, 10], which covered analysis of historical Hadoop traces and synthesis of new traces based on those. The SWIM project itself presents a methodology for workload synthesis, but restricted to MapReduce systems, in particular Hadoop. The traces provided cover a period of 24 hours and they have information on how many jobs were scheduled, when they were scheduled during that period, what file they required and how much data they moved in the mapping, shuffling and reducing phases. It does not have information on the size of the files, the number of tasks in a given job, or the duration of these same tasks. We estimate file sizes based on the amount of data that is received in the mapping phase, and we estimate the number of tasks and their duration using statistical models as described by Convolbo et al. for the evaluation of GeoDis [11] using the same data.

4.3.2 Google

Another data set that we considered was first published by Google [33] in 2011 and has information about job scheduling in a Borg cell with 12500 servers. The traces map events in the execution of tasks, such as scheduling, start, and conclusion of those. From those we identify tasks grouped under a job and we can calculate the duration of the tasks. This trace does not provide any file usage information, so we estimate file size from storage requirements for the jobs, and we use the job user identity as a proxy for the file name to better model jobs that require access to the same data.

4.3.3 Alibaba

The Alibaba trace [15] contains information about server and container usage, and task execution. Those include resource usage, even network, and task duration, as well as mapping of tasks to jobs. It does not include file information, so we used the job owner information as a proxy for the file id, and we used disk and memory usage to estimate file sizes.

4.4 Data Center Workload Generator

In order to evaluate how the scheduler performs with different workload patterns, we also developed a data center workload generator. Our workload generator can create synthetic data center workloads by combining properties from real workload traces. It consists of two parts, the generator proper and a feature extractor. The feature extractor can parse a workload trace and collect the measurements for sampling or generation of an empirical cumulative distribution, storing them in a file so that it can be used by the generator. The generator can combine information from these empirical distributions or use more traditional distributions (such as the Zipf distribution) for each random parameter in an extensible way.

The feature extractor reads a trace and generates a population object for each of the elements in the trace. We can then sample from these objects, obtaining results with similar behavior to the original data distribution. We can also use those population objects to calculate the empirical cumulative distribution, which is a useful tool for comparing different behaviors.

The data center workload generator creates a new job trace and file trace on invocation. It can create all the parameters described in the data center workload trace format in Section 4.2. Those parameters can have values generated from a well-known distribution, or from an empirical distribution created by the distribution identifier. Examples of well-known distributions in this context are the Zipf distribution for placement of files in data centers, or the Pareto distribution to estimate the duration of tasks Both of those are examples taken to the methodology presented by Hung et al. [18]. We call this method of creating a synthetic trace by combining empirical distributions *attribute sampling*, as opposed to the *job sampling* approach described by Chen et al. [9].

4.4.1 Attribute Sampling and Job Sampling

Chen et al. [9] presented a methodology to create synthetic workloads from existing workloads by sampling from jobs in a preexisting workload trace. This method has the benefits of reproducing the characteristics of the original trace and was demonstrated to be representative of the original data. In our situation, with limited workloads, we find that we want to explore more diverse scenarios. This approach, however, is limited in this aspect.

We built *attribute sampling* for this purpose. The idea is that while building a synthetic job that samples the attributes individually we are creating a job that is less representative of the original workloads, but which still has its behavior grounded in observed patterns.

Figure 4.3 provides an example to illustrate the differences between job- and attribute sampling. While this example shows all synthetic jobs as created from the same source, it becomes more interesting when we consider that we can combine attributes from two or more different original traces. So if we have a trace with jobs that have many short tasks, and another trace with jobs that have few long tasks, we can combine them to create a trace with jobs that have many long tasks, or few short tasks. Being able to explore diverse scenarios like this is the reason that we have adopted attribute sampling as part of our framework. We must note however that we have not examined how attribute sampling affects the diversity of patterns in the workload itself and it would be interesting to examine it in the future through formal means such as entropy. It's also important to observe that attribute and job sampling are not exclusive, and can be combined to create more diverse groups of jobs.

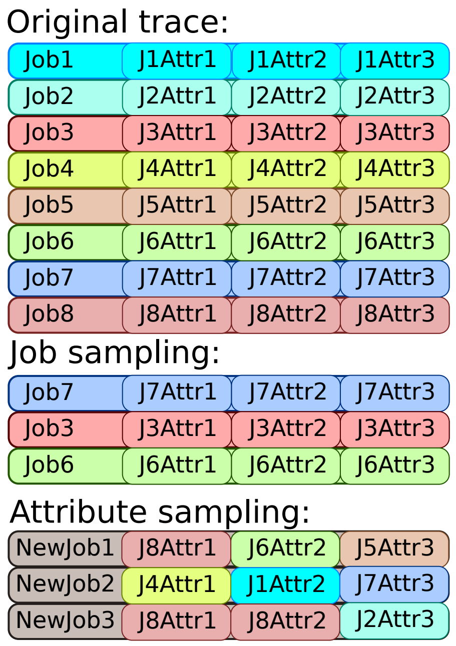
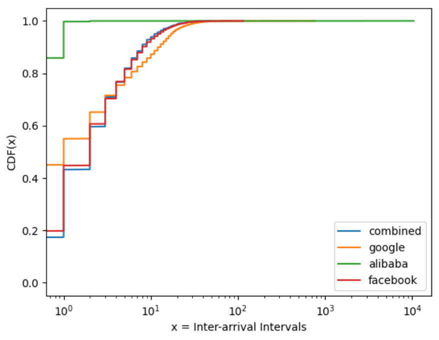

Original trace:
Job1 J1Attr1 J1Attr2 J1Attr3
Job2 J2Attr1 J2Attr2 J2Attr3
Job3 J3Attr1 J3Attr2 J3Attr3
Job4 J4Attr1 J4Attr2 J4Attr3
Job5 J5Attr1 J5Attr2 J5Attr3
Job6 J6Attr1 J6Attr2 J6Attr3
Job7 J7Attr1 J7Attr2 J7Attr3
Job8 J8Attr1 J8Attr2 J8Attr3
Job sampling:
Job7 J7Attr1 J7Attr2 J7Attr3
Job3 J3Attr1 J3Attr2 J3Attr3
Job6 J6Attr1 J6Attr2 J6Attr3
Attribute sampling:
NewJob1 J8Attr1 J6Attr2 J5Attr3
NewJob2 J4Attr1 J1Attr2 J7Attr3
NewJob3 J8Attr1 J8Attr2 J2Attr3

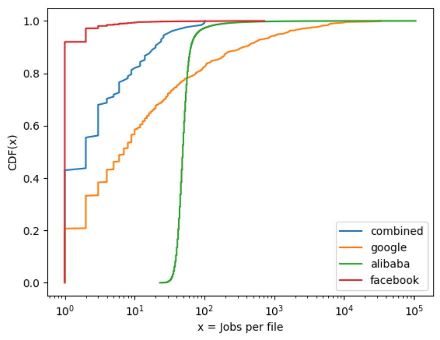
Figure 4.3: Comparison of job sampling and attribute sampling.

This use of attribute sampling is more evident when we compare distributions for features of different traces. Figure 4.4 shows multiple graphs with the cumulative distributions for some of the features we described in Section 4.2. In these graphs we are comparing the behavior of these features for the Alibaba, Facebook, and Google workload traces, and for a fourth trace which was created by our trace generator.

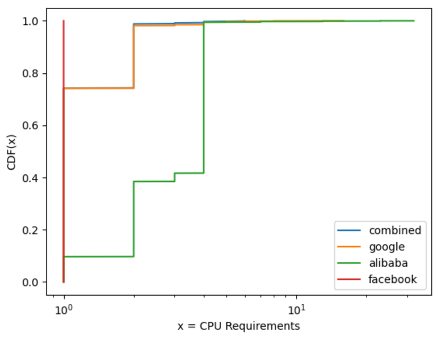
To create this synthetic trace our trace generator considered the inter-arrival times distribution in the Facebook trace, the number of tasks and task durations in the Alibaba trace, and referenced the Google trace for the remaining features. We can observe that for all features but one the cumulative distribution function of the synthetic trace (represented in blue and labeled as "combined") matches the corresponding original trace. The only exception is for the cumulative distribution function that describes how many jobs share a file, where the difference is caused by how many jobs are in the full Google trace (261967) compared to how many jobs are in the synthetic trace (1000). As a result, we have a new trace that shows similarity to real behaviors, but combined in such a way that we have a distinct situation to examine.



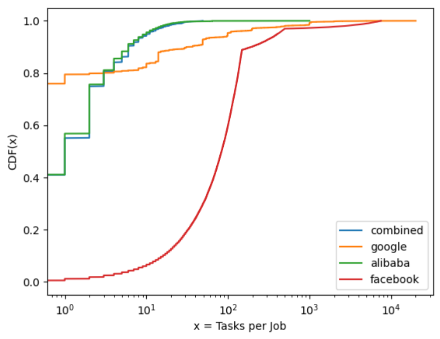
(a) Inter-arrival intervals



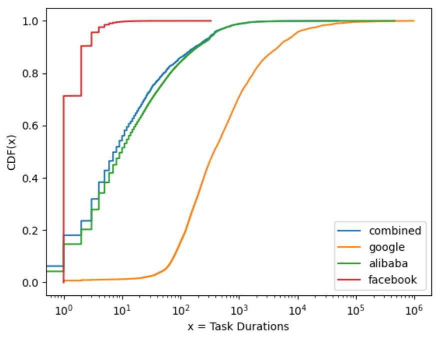
(b) Jobs using same file



(c) CPU requirements for jobs



(d) Number of tasks per job



(e) Task durations

Figure 4.4: Comparison of features for Alibaba, Facebook and Google traces, and a synthetic trace derived from all three.

Chapter 5

Experimental Evaluation

With the simulation and trace generation framework established, we set out first to create experiments that help us validate the functioning of our simulator. We decided to do this by reproducing the experiments described in the proposals for SWAG and GeoDis. These experiments compared the resulting makespan of using the SWAG scheduler against an adaptation of SRPT for geo-distributed data centers (Global-SRPT), and GeoDis against SWAG. Afterwards we executed experiments to measure the scalability of our simulator, since we want to ensure that our simulations execute in a reasonably small amount of time.

5.1 Reproducibility experiments

Given our goal to reproduce experiments, we will be using the same workloads used in the original experiments, the Facebook and Google workload traces. We will reproduce the topology described in the GeoDis proposal [11] for our experiments, as well as a version with reduced capacity. Table 5.1 shows the number of servers and cores per server in both topologies, and Table 5.2 shows the bandwidth between simulated data centers. The main reason for the topology with reduced capacity is the low density of jobs in the Facebook workload trace: the inter-arrival intervals are so large, that we observed no difference in performance between schedulers. In terms of file placement, we did not have that information in the traces, so we defined that each file would have three copies spread across data centers, with the use of a Zipf distribution [18] to determine which data centers to place the files in. We used different configurations for the Zipf distribution varying its skew parameter from 1.01 to 10, with the result that smaller skew parameter values create almost uniform placement of files, while a higher value will cause most files to be placed in fewer data centers.

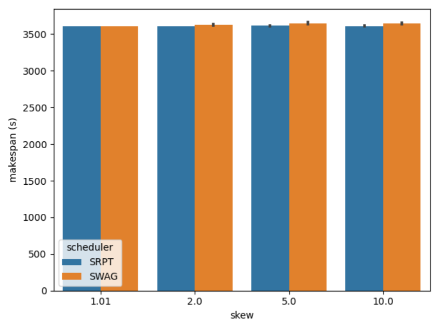
	Experiment 1		Experiment 2	
Data Center	Computers	CPUs	Computers	CPUs
DC 1	13	23	1	1
DC 2	7	12	1	1
DC 3	7	8	1	32
DC 4	12	8	1	1
DC 5	31	32	1	1
DC 6	31	32	1	1
DC 7	10	16	1	1
DC 8	8	12	1	2

Table 5.1: Data center topology in experiments – number of computers and CPUs per computer in each data center

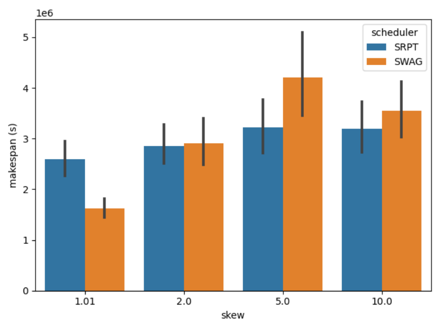
	DC 1	DC 2	DC 3	DC 4	DC 5	DC 6	Dc 7	DC 8
DC 1	–	931	376	822	99	677	389	935
DC 2	931	–	97	672	381	82	408	93
DC 3	376	97	–	628	95	136	946	175
DC 4	822	672	628	–	945	52	77	50
DC 5	99	381	95	945	–	822	685	535
DC 6	677	82	136	52	822	–	69	639
DC 7	389	408	946	77	685	69	–	243
DC 8	935	93	175	50	535	639	243	–

Table 5.2: Data center bandwidths (Mbps)

Figure 5.1 shows the results for the comparison between SWAG and our implementation of Global-SRPT in the original topology, while Figure 5.2 shows the same results using the smaller topology. The first thing we notice in these graphs are the near constant results in Figure 5.1a, which is caused due to the small number of concurrent jobs in the Facebook workload trace proportionally to the number of computer cores available. We also observe that SWAG shows better makespan when files are more evenly distributed, which is what we expected from the original results, but our experiments also show that when there's high concentration of files in few locations the Global-SRPT scheduler can produce equivalent or possibly even better results if it allows for data transfer.



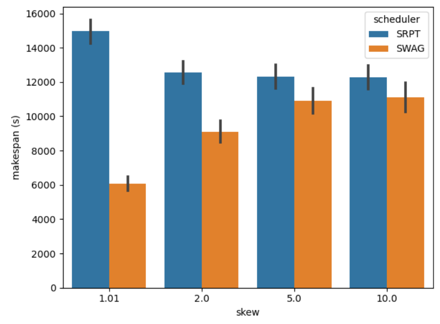
(a) Facebook workload



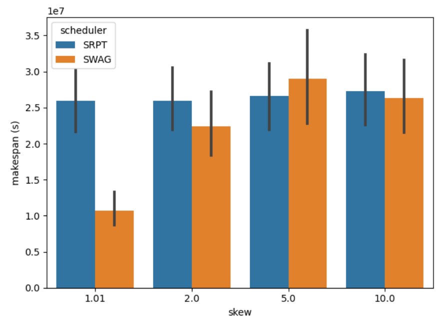
(b) Google workload

Figure 5.1: Measured makespan (in seconds) for schedules built using Global-SRPT and SWAG in original topology.

Continuing with the GeoDis experiments, we have similar graphs in Figures 5.3 and 5.4. Again we observe near constant results when using the Facebook workload trace in the full topology, but for the remainder of the graphs we observe that GeoDis outperforms SWAG, as expected. The advantage that GeoDis has due to allowing data transfers is even more pronounced as we concentrate the files in fewer locations, evidenced by the distance between measurements as we increase the skew parameters.

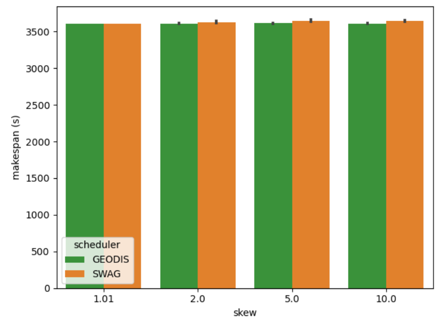


(a) Facebook workload

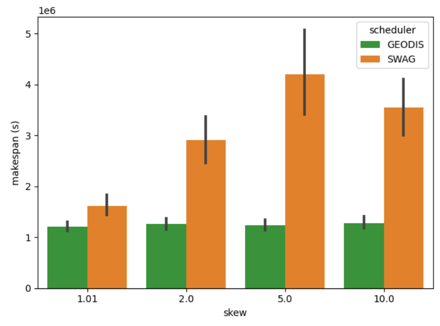


(b) Google workload

Figure 5.2: Measured makespan (in seconds) for schedules built using Global-SRPT and SWAG in smaller topology.

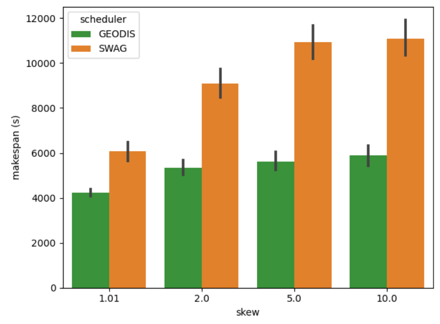


(a) Facebook workload

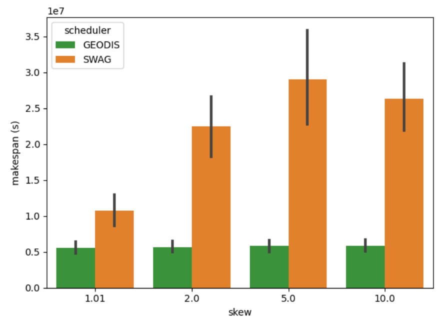


(b) Google workload

Figure 5.3: Measured makespan (in seconds) for schedules built using SWAG and GeoDis in original topology.



(a) Facebook workload



(b) Google workload

Figure 5.4: Measured makespan (in seconds) for schedules built using SWAG and GeoDis in smaller topology.

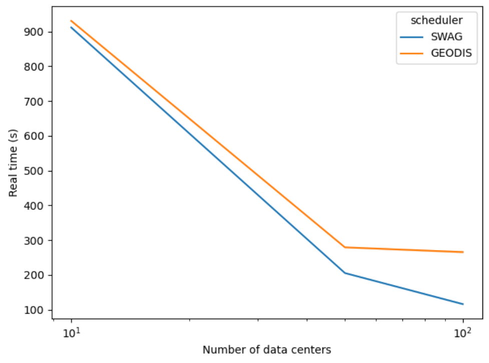
5.2 Scalability experiment

For this experiment we create topologies and workload traces that are completely uniform, so that we can vary the scale of certain parameters and observe how that impacts the performance of our simulations. We considered three main parameters that could impact performance:

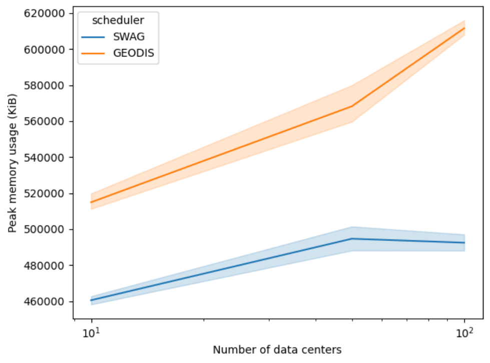
- Total number of data centers in the topology – varied between 10, 50 and 100;
- Number of jobs that arrive in the same batch – varied between 100, 500, 1000, 5000;
- Number of tasks in each job – varied between 10, 100, 1000.

When examining each parameter, we consider the other two as set at their highest values. All executions were ran on a Linux computer with an Intel i5-650 3.2GHz processor and 16GB of RAM. We measured both execution time and maximum memory usage for performance comparisons.

Figure 5.5 shows the performance impact of the total number of data centers, Figure 5.6, the number of jobs, and Figure 5.7, the number of tasks. The first thing we notice is that increasing the number of data centers resulted in a decrease in execution times, which is more pronounced between the two smaller amounts of data centers we tested. This happens because having more data centers makes it easier for the scheduler to place jobs and reduces the number of calls to the scheduler, which appears as the main cause of execution time. Correspondly, increasing the number of jobs increases the load on the scheduler and causes an increase in execution time. Increasing the number of tasks per job also causes an increase in execution time, but less pronounced than when increasing the number of jobs, most likely because both schedulers rely on ordering all jobs according to a heuristic.

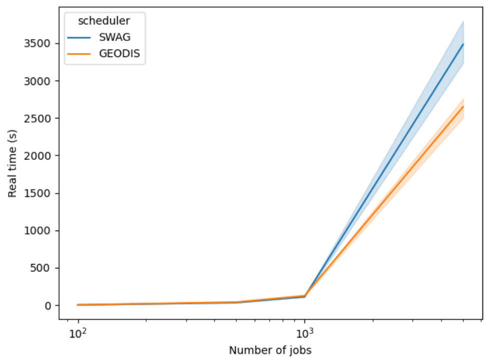


(a) Execution time (s)

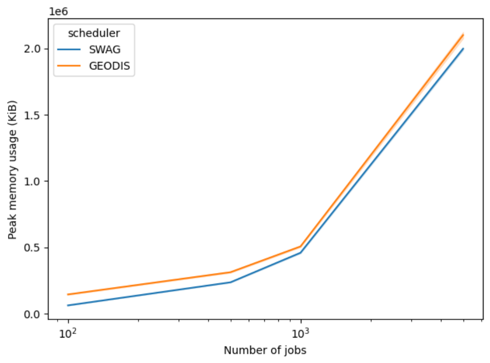


(b) Memory Usage (KiB)

Figure 5.5: Performance impact of changing the total number of data centers in topology.



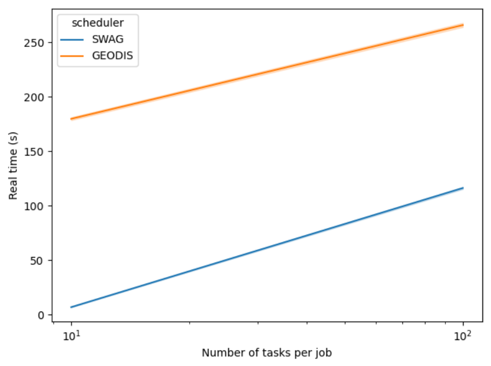
(a) Execution time (s)



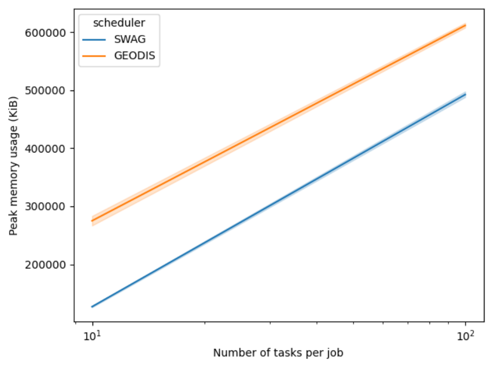
(b) Memory Usage (KiB)

Figure 5.6: Performance impact of changing the total number of jobs in workload trace (jobs arrive simultaneously).

The resulting execution time were in the range from a few minutes to an hour, meaning that it was possible to run multiple simulations in a day. Moreover, since in this case the simulator's performance is mainly tied to the performance of the scheduler, while a long simulation time is not desirable, it may be indicative of issues with the scheduler in the scenarios used.



(a) Execution time (s)



(b) Memory Usage (KiB)

Figure 5.7: Performance impact of changing the total number of tasks in each job.

In terms of memory usage, Figures 5.5, 5.6 and 5.7 show that the simulator requires between 200MiB and 2GiB in the scenarios used. Memory requirements increase proportionally to the size of the topology and the amount of jobs and tasks involved. The number of jobs seems to have a larger effect on the memory usage than the other parameters, likely due to the need to estimate the makespan of each job as part of the scheduling process.

Chapter 6

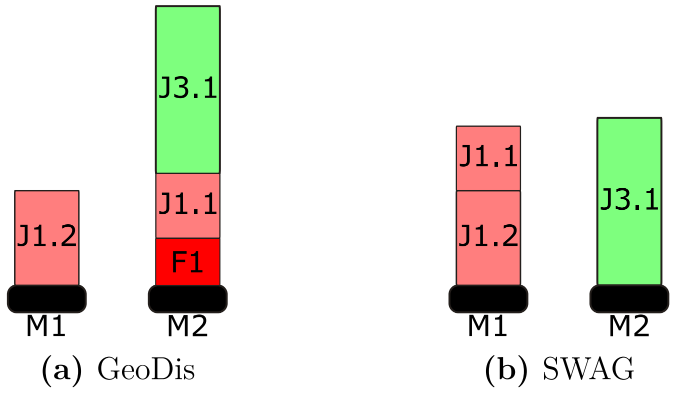
Adaptive Job-Scheduler Exploration

In this chapter we examine how the simulator can be used to configure a meta-scheduler. First we examine how different scenarios can result in different schedulers providing the best performance (e.g. makespan), even for schedulers based on similar ideas such as SWAG and GeoDis. Next we show how we can test a proposal for a meta-scheduler that tries to use the most appropriate scheduler based on current workload and data center conditions.

6.1 Influence of scenario on schedulers

When considering the use of a geo-distributed scheduler, we might ask ourselves how much the conditions of our problem influence that choice. For example, if we examine only SWAG and GeoDis, we expect that GeoDis would provide lower makespan because it builds on the ideas first presented by SWAG and goes a step further by allowing data transfers between data centers for the purpose of executing jobs. It has been reported as performing similarly or better than SWAG [11].

Figure 6.1, however, shows there are scenarios where SWAG yields superior performance. In this example we have two data centers with a single available server (M1 and M2) each and two jobs to be scheduled, one of them (J1) with multiple tasks, the other (J3) with a single task. The files that J1 requires to execute are available in M1 and the files that J3 requires are available in M2. The GeoDis scheduler will place first J1's tasks, occupying all servers, then J3 has to wait for the task J1.1 in the second server to end before it can start in order to avoid another file transfer and thus delay J3 even further. SWAG, on the other hand, will restrict each job to a server and, in this example, both jobs willexecute at the same time. This will result in a schedule with lower makespan and also lower worst case completion time for the jobs.



(a) GeoDis **(b)** SWAG

Figure 6.1: Makespan for two jobs, J1 and J3, using GeoDis and SWAG in two data centers with one server each, M1 and M2.

We can extend to cases with more jobs, which we demonstrate by building synthetic workloads as follows. We define two types of jobs, namely "elephant" and "mouse", where elephant jobs have fewer tasks (between 1 and 5), with long durations (between 20 and 50 seconds), and mouse jobs have many short tasks (between 30 to 100 tasks with durations of 1 to 5 seconds)[1]. These values are ar-

[1]The names "elephant" and "mouse" are usually applied to flows, but we felt that they made for an apt analogy of the types of jobs we want to describe.

bitrary, but inspired by the properties of the Facebook trace. We ran experiments with synthetically-generated [2], 1000-job workloads varying the percentage of elephant jobs. As shown in Figure 6.2, we observe that, when compared to GeoDis, SWAG exhibits superior performance in terms of makespan as the percentage of elephant jobs increase.

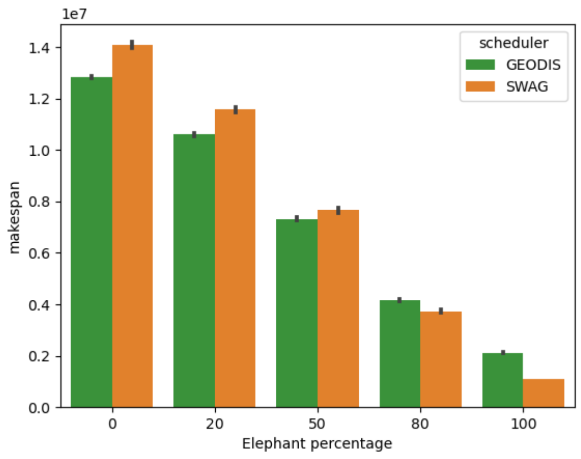
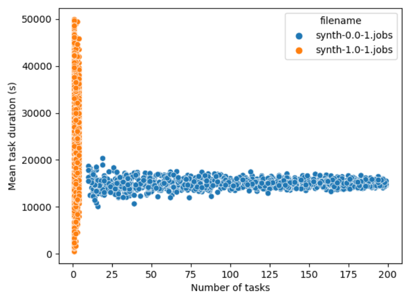


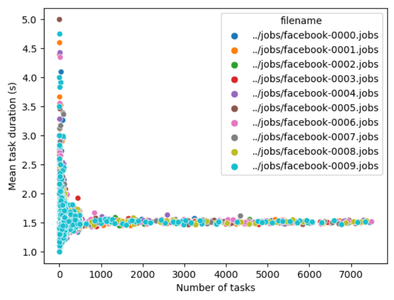
Figure 6.2: Comparison of makespan performance for scheduler under different job mixes.

While these traces were artificially constructed, we find that real workloads may also exhibit similar properties. Figure 6.3 shows the relationship between the average duration of tasks in a job and the number of tasks in the same job for the syntehtic workload trace on the left hand side and the Facebook trace on the right side. with the synthetic jobs on the left side and jobs from the Facebook workload trace on the right side. We observe in both cases a division between jobs that have many short tasks and jobs that have few long tasks, although the uniform distribution in the synthetic data means that in the Facebook workload trace most jobs have few short tasks.

[2]We did not use the workload generator to create these synthetic traces due to their very simple nature – the workload generator is better suited to generating traces that reproduce behavior from other traces via empirical distributions



(a) Synthetic workload trace



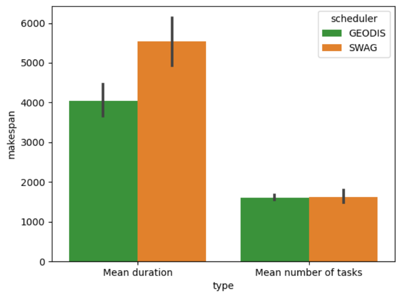
(b) Facebook workload trace

Figure 6.3: Average task duration per job in terms of number of tasks per job.

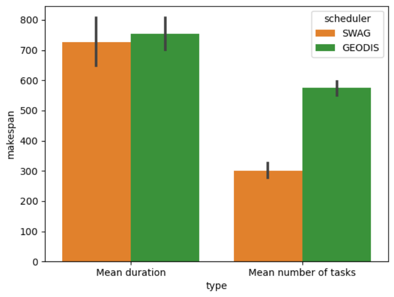
If we create traces using the jobs from the Facebook workload, but separating the jobs according to whether they fit better with our definition of mouse or elephant traces to obtain a similar separation to what we had in our synthetic traces, we can compare how SWAG and GeoDis perform in these scenarios. We create then two separate traces, one with jobs that have high mean duration for their tasks (similar to elephant jobs) and another with jobs that have many tasks (similar to mice jobs). Figure 6.4 shows the average makespan and 95% confidence interval when GeoDis and SWAG are used to schedule jobs using both the small, reduced data center topology (left-hand side graph) and the larger, ful, data center topology (right-hand side graph). We observe that GeoDis outperforms SWAG when there is more contention for data center resources, but particularly when we are using the trace that has jobs with fewer, longer tasks, while SWAG shows better results when there are more resources available, especially when scheduling jobs with many short tasks.

6.2 Building a meta-scheduler

Based on the insights provided by the empirical analysis reported in Section 6.1, we tried to build a meta-scheduler, that is, a scheduler that evaluates



(a) Reduced topology



(b) Full topology

Figure 6.4: Makespan measurements using split Facebook workload.

the incoming workload (i.e. jobs that have been submitted but have not been scheduled before) and current conditions of the geo-distributed data centers, and then uses a scheduler from a pre-configured set of known schedulers to place the jobs in the current batch. So we define that our meta-scheduler has two main components: a decision making engine and a set of batch-based schedulers that it can select from. In our current implementation our meta-scheduler will use either GeoDis or SWAG but it can be extended to include additional batch-based schedulers. The next step is building the decision mechanism that will allow our meta-scheduler to select the most appropriate scheduler.

The decision making engine

For our decision making engine we will rely on the properties of the workload and data center current conditions and resources that seem to be most closely related to differences in performance between the observed schedulers: number of tasks per job, duration of tasks, and ratio of requested cores compared to total capacity. As part of future work, we plan to inform the meta-scheduler's decision making engine with additional information about jobs have been recently completed.

Our current decision engine uses statistics of the properties of the workload and data center current conditions and a sensitivity parameter that defines a threshold. According to the relationship between the value of the statistics and the sensitivity parameter p, our meta-scheduler will use either SWAG or GeoDis. The sensitivity parameter p can also be adjusted to fit better with different workloads or data centers, but this adjustment must be done ahead of use. Next we describe meta-schedulers we created based on these ideas.

6.2.1 TASK_NUM and TASK_DURATION

For the first attempt, we named the meta-scheduler TASK_NUM and used the number of tasks per job as the random variable. For the second attempt we named it TASK_DURATION and considered the duration of tasks. In these two meta-schedulers, the statistic models a relationship between the mean ($\bar{x}$) of the observed value and its variance (σ^2): if $\frac{\sigma^2}{\bar{x}} < p$ then we schedule the current batch with SWAG, otherwise with GeoDis.

6.2.2 CORE_RATIO

Finally, for the third attempt we named it CORE_RATIO and considered the relationship between workload resource utilization and data center resource capacity. We actually built three meta-scheduler (CORE_RATIO1, 2 and 3) based on this idea. For all the CORE_RATIO meta-schedulers, we define a ratio between a measure of utilization (b) and one of capacity (t): we use SWAG if $\frac{b}{t} < p$, otherwise we use GeoDis. The first of these schedulers, CORE_RATIO1 compares busy cores in the data centers (b) to total existing cores (t), available or busy. The variation CORE_RATIO2 uses the number of CPU cores that will be required by the jobs in the current batch for the value b. The variation CORE_-

RATIO3 also uses the number of CPU cores that will be required for b and uses. number of available cores for t.

6.2.3 Results

Figure 6.5 shows the distributions of makespans when using the described meta-schedulers with the best performing values of their parameters on the Facebook workload with the topology defined on Tables 5.1 and 5.2, with SWAG and GeoDis included for comparison. TASK_NUM, TASK_DURATION and CORE_RATIO1 generated results closer to GeoDis, while CORE_RATIO2 and CORE_RATIO3 were less efficient in this test.

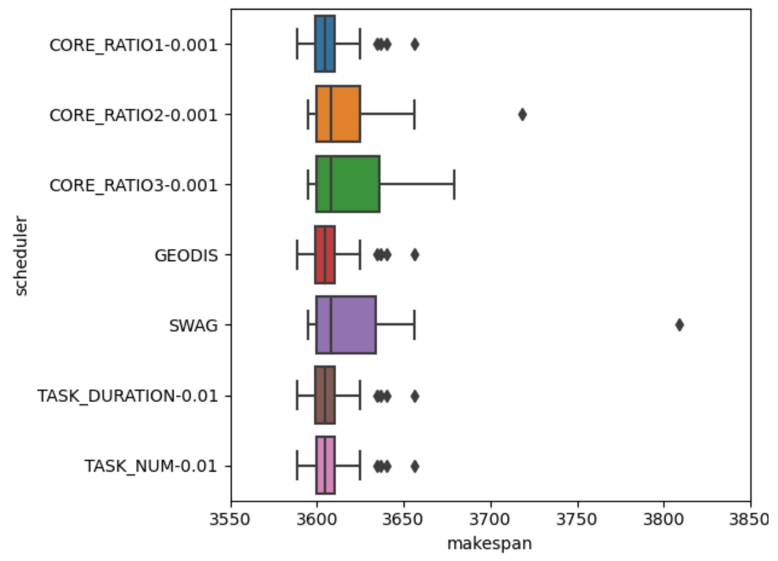


Figure 6.5: Makespan distribution for meta-schedulers on Facebook workload

Continuing this investigation, we also apply these meta-heuristics to the Google workload traces we have and we find the results in Figure 6.6. Now we find that TASK_NUM, TASK_DURATION and CORE_RATIO3 perform close to GeoDis, but with smaller outliers, while CORE_RATIO1 and CORE_RATIO2 were less efficient.

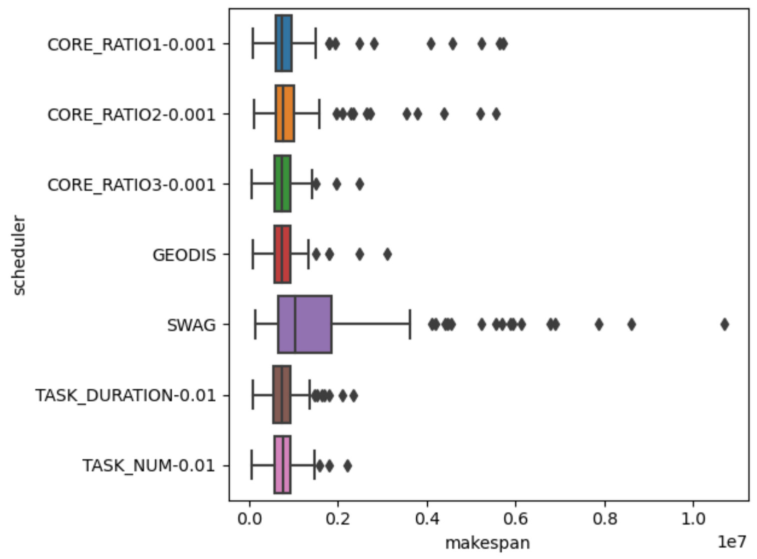


Figure 6.6: Makespan distribution for meta-schedulers on Google workload

These results are not conclusive to demonstrate the use of this particular meta-scheduler as a new general use scheduler, but they illustrate how one can fine-tune a scheduler for their specific needs as long as they have enough representative data. For the traces that we examined, TASK_NUM and TASK_DURATION seemed more consistent about maintaining the performance of the standard GeoDis job scheduler or providing a slight improvement over the standard GeoDis job scheduler.

This type of analysis benefits from the lower costs to run simulations, which let us compare more results in a short time interval. The same analysis would take considerable more time in a test bed or a real data center and it shows the benefits of having a lighter evaluation framework for developing geo-distributed job schedulers.

6.3 Summary

In this chapter we discussed how the features of the scenario (jobs to be scheduled and the topology) can favor different schedulers, and how we can leverage these features to build meta-schedulers that select the most appropriate scheduler from a list. We demonstrated the influence of the scenario first by creating synthetic workloads and small topologies that illustrate the possibility, then by examining how these correspond to real variation in workload traces' features.

Next we created multiple meta-scheduler proposals using simple features of the scenario that showed relevance in our comparisons. We used our evaluation framework to test these proposals under different configurations so that we can select the best performing, which show desirable features such as lower overall makespan even in the worst case. This work could be further extended in future work by considering more basic job-schedulers, or different methods to build the meta-schedulers, such as machine learning.

Chapter 7

Conclusion

This work examined the field of geo-distributed job scheduling and particularly how we can compare different approaches. We looked through some of the existing methods for job scheduling in geo-distributed data centers and their differences, and also the difficulties that researchers have when comparing them in terms of resulting performance. Then we presented a framework for analysis and showed how it can be used to test new proposals for job scheduling. The contributions of this work are as follow:

- introduction of a mechanism to generate synthetic workload traces that inherit characteristics from different existing traces, to allow for the exploration of more evaluation scenarios;

- introduction of an event-based simulator for evaluation of geo-distributed job schedulers in a reproducible and low cost manner;

- demonstration of how to evaluate a new job scheduling proposal using the introduced tools.

- demonstration of how to adjust the parameters of a scheduler according to the properties of the environment using the introduced tools.

In summary, this work demonstrated how we can conduct first-step evaluation of geo-distributed job schedulers with simulation and synthetic traces. This can also be used to better adjust schedulers to the particular needs of the data centers in which they will operate.

This work was limited by the lack of resources to evaluate the considered job schedulers in more realistic conditions, so future work would benefit from this comparison. It would also benefit from the expansion of the simulation tool, both with the addition of more schedulers and expansion of functionality, such as dependencies. In particular, the network module could be extended to integrate with external networking simulators and thus provide more accurate simulations. The workload generator could also be further studied in terms of the diversity of the generated patterns and how to best use the generator in terms of obtain diversity measurements that correspond to those observed in real traces.

Bibliography

[1] T. Adhikary, A. K. Das, M. A. Razzaque, and A. M. J. Sarkar. Energy-efficient scheduling algorithms for data center resources in cloud computing. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1715–1720, 2013.

[2] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 2, USA, 2010. USENIX Association.

[3] Daniel Alves, Katia Obraczka, and Rick Lindberg. Telemetry triaging in data centers using change point detection. Unpublished, Submitted to Performance 2020, 2020.

[4] Daniel S.F. Alves and Katia Obraczka. An empirical characterization of internet round-trip times. In *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '17, page 23–30, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Nikhil Bansal and Mor Harchol-Balter. Analysis of srpt scheduling: Investigating unfairness. *SIGMETRICS Perform. Eval. Rev.*, 29(1):279–290, June 2001.

[6] Mutaz Barika, Saurabh Garg, Albert Y. Zomaya, Lizhe Wang, Aad Van Moorsel, and Rajiv Ranjan. Orchestrating big data analysis workflows in the cloud: Research challenges, survey, and future directions. *ACM Comput. Surv.*, 52(5), September 2019.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[8] L. Chen, S. Liu, B. Li, and B. Li. Scheduling jobs across geo-distributed datacenters with max-min fairness. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[9] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 390–399, 2011.

[10] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proc. VLDB Endow.*, 5(12):1802–1813, August 2012.

[11] Moïse W. Convolbo, Jerry Chou, Ching-Hsien Hsu, and Yeh Ching Chung. Geodis: Towards the optimization of data locality-aware job scheduling in geo-distributed data centers. *Computing*, 100(1):21–46, January 2018.

[12] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer. A survey on geographically distributed big-data processing using mapreduce. *IEEE Transactions on Big Data*, 5(1):60–80, 2019.

[13] Jyoti V Gautam, Harshadkumar B Prajapati, Vipul K Dabhi, and Sanjay Chaudhary. A survey on job scheduling algorithms in big data processing. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–11, 2015.

[14] F. Giroire, N. Huin, A. Tomassilli, and S. Pérennes. When network matters: Data center scheduling with network tasks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2278–2286, 2019.

[15] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 629–647, Boston, MA, March 2017. USENIX Association.

[17] Z. Hu, B. Li, and J. Luo. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[18] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, page 111–124, New York, NY, USA, 2015. Association for Computing Machinery.

[19] Y. Jin, Y. Gao, Z. Qian, M. Zhai, H. Peng, and S. Lu. Workload-aware scheduling across geo-distributed data centers. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1455–1462, 2016.

[20] Dalibor Klusácek and Hana Rudová. Alea 2: job scheduling simulator. In *SimuTools*, 2010.

[21] Dalibor Klusáček, Šimon Tóth, and Gabriela Podolníková. Complex job scheduling simulations with alea 4. In *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, SIMUTOOLS'16, page 124–129, Brussels, BEL, 2016. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[22] Joanna Kołodziej and Samee Ullah Khan. Data scheduling in data grids and data centers: A short taxonomy of problems and intelligent resolution techniques. In Ngoc-Thanh Nguyen, Joanna Kołodziej, Tadeusz Burczyński, and Marenglen Biba, editors, *Transactions on Computational Collective Intelligence X*, pages 103–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[23] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[24] Patrick Kugler, Philipp Nordhus, and Bjoern Eskofier. Shimmer, cooja and contiki: A new toolset for the simulation of on-node signal processing algorithms. In *2013 IEEE International Conference on Body Sensor Networks*, pages 1–6, 2013.

[25] Matthias Maiterth, Gregory Koenig, Kevin Pedretti, Siddhartha Jana, Natalie Bates, Andrea Borghesi, Dave Montoya, Andrea Bartolini, and Milos Puzovic. Energy and power aware job scheduling and resource management: Global survey — initial analysis. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 685–693, 2018.

[26] T Y J Naga Malleswari, S. Ushasukhanya, A. Nithyakalyani, and S Girija. A technical survey on optimization of processing geo distributed data. *Journal of Physics: Conference Series*, 1000:012140, apr 2018.

[27] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288, New York, NY, USA, 2019. Association for Computing Machinery.

[28] A. Mohan, M. Ebrahimi, S. Lu, and A. Kotov. A nosql data model for scalable big data workflow execution. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 52–59, 2016.

[29] Zhiping Peng, Delong Cui, Jinglong Zuo, Qirui Li, Bo Xu, and Weiwei Lin. Random task scheduling scheme based on reinforcement learning in cloud computing. *Cluster Computing*, 18(4):1595–1607, December 2015.

[30] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. *SIGCOMM Comput. Commun. Rev.*, 45(4):421–434, August 2015.

[31] Imran Qureshi. Cpu scheduling algorithms: a survey. *International Journal of Advanced Networking and Applications*, 5(4):1968, 2014.

[32] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, 2012. Association for Computing Machinery.

[33] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, October 2012.

[34] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2014-11-17 for version 2.1. Posted at `https://github.com/google/cluster-data`.

[35] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, page 351–364, New York, NY, USA, 2013. Association for Computing Machinery.

[36] Garima Sharma and Anita Ganpati. Performance evaluation of fair and capacity scheduling in hadoop yarn. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 904–906, 2015.

[37] Mohan Sharma and Ritu Garg. Higa: Harmony-inspired genetic algorithm for rack-aware energy-efficient task scheduling in cloud data centers. *Engineering Science and Technology, an International Journal*, 23(1):211 – 224, 2020.

[38] S. Shen, V. v. Beek, and A. Iosup. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 465–474, 2015.

[39] D. Sun and R. Huang. A stable online scheduling strategy for real-time stream computing over fluctuating big data streams. *IEEE Access*, 4:8593–8607, 2016.

[40] Dawei Sun, Hongbin Yan, Shang Gao, Xunyun Liu, and Rajkumar Buyya. Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams. *J. Supercomput.*, 74(2):615–636, February 2018.

[41] El-Ghazali Talbi, Matthieu Basseur, Antonio J. Nebro, and Enrique Alba. Multi-objective optimization using metaheuristics: non-standard algorithms. *International Transactions in Operational Research*, 19(1-2):283–305, 2012.

[42] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2012.

[43] S. Vincent, J. Montavont, and N. Montavont. Implementation of an ipv6 stack for ns-3. In *VALUETOOLS*, 2008.

[44] K. Wang, Q. Zhou, S. Guo, and J. Luo. Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey. *IEEE Communications Surveys Tutorials*, 20(4):3560–3580, 2018.

[45] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[46] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Gener. Comput. Syst.*, 27(5):430–439, May 2011.

[47] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu. Load balancing in data center networks: A survey. *IEEE Communications Surveys Tutorials*, 20(3):2324–2352, 2018.

[48] Qing Zhao, Congcong Xiong, Ce Yu, Chuanlei Zhang, and Xi Zhao. A new energy-aware task scheduling method for data-intensive applications in the cloud. *J. Netw. Comput. Appl.*, 59(C):14–27, January 2016.

[49] Y. Zhao, R. N. Calheiros, G. Gange, K. Ramamohanarao, and R. Buyya. Sla-based resource scheduling for big data analytics as a service in cloud computing environments. In *2015 44th International Conference on Parallel Processing*, pages 510–519, 2015.

[50] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Transactions on Parallel and Distributed Systems*, 27(12):3501–3517, 2016.