Implementation & Analysis of Different Allocation Policies

Distributed Systems – Lab exercise
Course Instructor: prof. dr. ir. A. Iosup
Lab supervisor: ir. L. Versluis

Jan Haenen
j.p.haenen@student.vu.nl
1822306

Kim van Putten
k.e.van.putten@student.vu.nl
2525287

Ruben Stam
r.d.stam@student.vu.nl
2597714

Thomas de Zeeuw
t.m.de.zeeuw@student.vu.nl
2599441

December 19th 2018

With the increasing demand and deployment of cloud computing, scheduling computing jobs on heterogeneous cloud environments while providing a high Quality-of-Service (QoS) for costumers is an important and difficult challenge. In this paper we present four allocation policies from literature which we implemented in the OpenDC datacentre simulator, and perform an analysis comparing the performance of these policies. We perform experiments with two different workloads and two different environment setups. For the analysis we looked at the wait time and makespan metrics, and from the results we find that the simpler allocation policies appear to outperform the more complex ones.

1. Introduction

In recent years cloud-computing has made rapid developments and large- scale cluster computing has rapidly become more commonly adapted. Scheduling computing tasks efficiently has become increasingly more important as the demand for high performance on distributed systems rises. A common problem in datacentres is how to schedule tasks to available resources efficiently, such that available resources are optimally used and computation jobs are executed in a reasonable time frame for the users without having the task scheduling process take up too much resources. For example in Google's datacentres the scheduler alone accounts for more than 5% of all datacentre cycles [11].

We were tasked by WantScheduler BV to implement a number of allocation policies from literature and analyse there performance through experimentation. We implemented these allocation policies in the OpenDC simulation environment [7].

1.1. Related work

Many different task scheduling algorithms have already been proposed. In this section we give a brief overview of some policies of different natures.

Matchmaking policies such as Gangmatching focus on finding compatible pairs between jobs and machines [12]. The Condor Project implements a variety of matchmaking policies for matching resources to jobs [14][8].

Several task scheduling policies implement a guided random search in an attempt to find the optimal scheduling of tasks. An example of such policies are Multiple Priority Queueing Genetic Algorithm (MPQGA) [18], which combines a genetic algorithm approach to assign priorities to tasks with the heuristic based HEFT [16] policy to map tasks to machines. Another example is the Ant Colony System (ACS) policy which applies ant colony optimization to task scheduling based on various quality of service (QoS) parameters [3].

Many heuristic allocation policies such as HEFT

[16], LBA, Min-min and Max-min [2] base their scheduling decisions on the expected runtimes of tasks. Ilyushkin et al. did research into scheduling workflows with unknown task runtimes [6].

Performing experiments to analyze the performance of scheduling policies on actual distributed systems can be time consuming and expensive. Research is being done to create simulating environments to provide researchers a tool to perform experiments with distributed systems of various sizes and setups without the need to reserve the actual resources. Iosup et al. introduced the OpenDC datacentre simulator, which we used to run our experiments [7], which was also used by Andreadis et al. in [1].

1.2. Implemented policies

For this assignment we decide only to implement allocation policies that fit well within the OpenDC framework. We decided against modifying the framework itself to ensured that it would continue operate correctly. This did mean we were limited in the algorithms that could be implemented. The following allocation policies were implemented and analyzed.

1. Heterogeneous Earliest Finish Time (HEFT), [16]
2. Critical-Path-on-a-Processor (CPOP), [15]
3. Priority Impact Scheduling Algorithm (PISA), [17] and
4. Round Robin. [13]

1.3. Report structure

In the following section 2 we describe OpenDC in some more detail and describe the requirements. The allocation policies are described in greater detail in section 3. In section 4 the experimental setup is described, followed by the results in section 5. The article ends with the conclusion in section 7 and the discussion, including future work, in section 6.

In this report we'll use the following terms: workloads, jobs and tasks. Workload consists of one or more jobs, where jobs could be Workflows, Bag of Tasks (BoT) or any other collection of tasks. Tasks are the smallest unit of computation and is the thing actually scheduled and run on machines. Tasks can have dependencies on other tasks, but only within the same job.

2. Background

The allocation policies were implemented in the OpenDC framework. OpenDC is framework that allows simulation of datacentres, including allocation policies, and is freely available at https://opendc.org [7]. Because OpenDC is still in early development it currently imposes a number of limitations on the allocation policies.

The allocation scheduling is split in two parts, 1) sorting queued tasks and 2) selecting a machine to run the highest priority task on, where the highest priority task is defined by the sorting policing in step 1.

Unfortunately two (fairly) recent allocation policies, Quincy [8] and Firmament [4], did not fit this structure. Both policies needed complete information about all machines and all queued and running tasks. At the time of implementation OpenDC did offer this functionality, which made it impossible to correctly implemented these policies without modifying the framework itself, something we decide against to ensure correct simulation and testing of the policies.

3. Allocation policies

We implemented four allocation policies in the OpenDC framework. Two of these policies implement both task sorting and machine selection, the two stages of scheduling in OpenDC, and the other two only one of the stages.

The policies were implemented such that experiments were repeatable and the outcome of simulations always result in the same output. The source code of the implemented allocation policies is made available open-source at [9]. The policies are also in the process of being upstreamed the the OpenDC framework.

3.1. HEFT

Heterogeneous Earliest Finish Time (HEFT) is a greedy policy that prioritizes the tasks based on the earliest possible finish time of a task. For each task a upward rank is calculated, which is the longest path from the task to the exit node (task in a job) based on the average execution cost and average communication cost of the tasks that lie on the path. Tasks are assigned to a machine that minimizes the earliest execution finish time of the task, i.e. the machine that would complete the task the earliest. [16]

Note however that OpenDC the communication time between machines is not simulated, since this functionality is not (yet) available.

3.2. CPOP

Critical-Path-on-a-Processor (CPOP) is similar to HEFT in that it also uses the upward rank of a task. The full algorithm as described by Topcuoglu et al. [15] determines for each job what the critical path is and assigns all tasks on the critical path to a single machine that minimizes the execution time of the entire critical path.

Unfortunately OpenDC does not support this implementation as the current framework does not allow access to job information during the execution of a sorting policy. We therefore implemented a simplified version of the policy that sorts tasks based on the upwards rank + downward rank, where the downward rank is essentially the upward rank of a tasks predecessors. This way the policy will still give priority to tasks that lie on the critical path of a job as they share the highest rank within a job. The downside of this simplified implementation is that tasks on a critical path may not necessarily be scheduled to the same machine, and so the communication time between tasks on the critical path may be higher because data may have to be transferred between machines. However the version of OpenDC used at the time of implementation did not simulate transferring of data between machines, something that will effect the results.

3.3. PISA

Priority Impact Scheduling Algorithm (PISA) is only a task sorting policy that sorts tasks based on their priority, which may be assigned by the user or by the compute provider. The policy prevents starvation of low-priority tasks by keeping track how often a task was not scheduled because higher priority task was scheduled instead. Once a threshold has been reached a low priority task will be given maximum priority so that it will be scheduled next even if higher priority tasks are present in the queue. Because Wu et al. [17] do not present a recommended value for the threshold we arbitrarily decided on a threshold of 100 for our experiments. Since PISA is only a task sorting algorithm we used it in combination with a machine selection policy in our experiments, see section 4 Experimental Setup for more.

Since PISA is a sorting only policy it will be matched up with a number of machine selection policies; Round Robin (see below), Best Fit, First Fit, Worst Fit.

3.4. Round Robin

Round Robin is a machine selection policy. It simply iterates through the available machines and schedules each task on the next machine that has enough resources available for the task [13]. Similarly to PISA, this policy was used in combination with a sorting policy, see section 4 Experimental Setup for more.

Since Round Robin is a selection only policy it will matched up with a number of sorting policies. Shortest Remaining Time First (SRTF), First in First Out (FIFO), Random sorting and PISA.

4. Experimental Setup

To evaluate the performance of the different allocation policies we used two workloads and two environmental setups, which are described in the sections below.

4.1. Workloads

Property	Workload 1	Workload 2
Mean task runtime	34.12 s	33.64s
Median task runtime	3 s	3 s
SD of task runtime	65.35 s	87.25s
Mean job execution time	2367 s	2334 s
Median job execution time	1292 s	1954 s
SD of job execution time	3909 s	1222 s
Total task runtime	473,429 s	466,848 s
Mean workflow size	69 tasks	
Median workflow size	35 tasks	
SD of workflow size	98 tasks	
Total task number	13,876 tasks	

Table 1: Workload properties adapted from [5].

In the experiments we use two workloads from the SPEC RG Cloud Group [5]. These workloads have different task runtimes and are identical in all other properties, namely arrival times, number of tasks per job and task dependencies. Properties of the tasks and jobs from both workloads are shown in table 1. The difference in the distribution of the runtimes can also be seen in figure 1 which shows that arriving load on the system is a lot more bursty for workload 1 compared to workload 2.

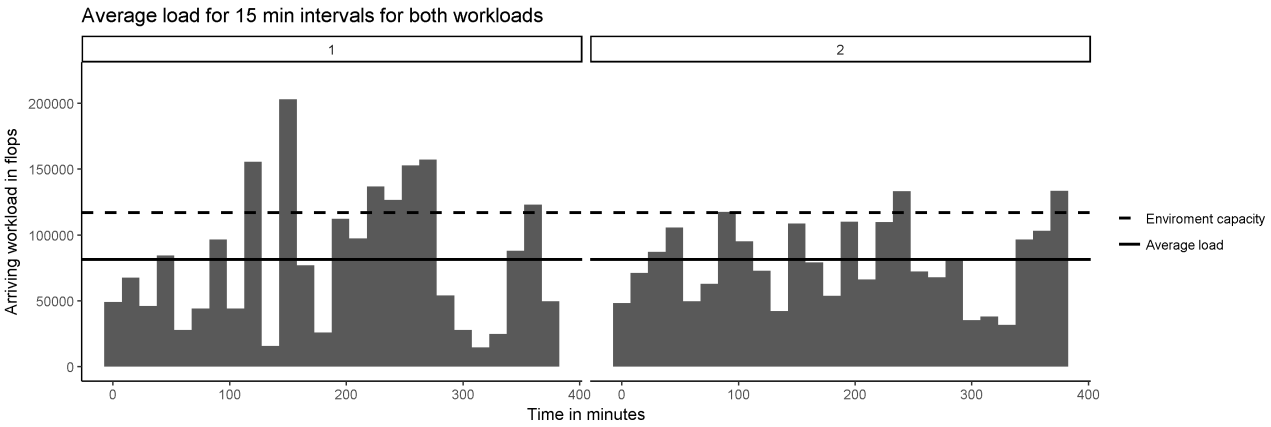

Figure 1: Average load, in 15 minute intervals, for workloads 1 (left) and 2 (right)

The allocation policies HEFT and CPOP use communication cost and PISA uses priority. These require information not present in the source workloads, therefore the required data was generated. In order to compute communication cost input and output sizes are generated that are normally distributed with mean 1000 MB and standard deviation 250 MB. Sizes with generated negative values are set to 0. Priorities are set at a job level. Job priorities are generated using a discrete uniform distribution with three levels.

4.2. Environment Setup

In the experiments we want to compare the performance of the different allocation policies for the above described workloads. To achieve this we need an environment setup that matches the workloads, e.g. if we over provision no significant scheduling queues will form. However given the stochastic nature of the arrival process and runtimes, and that task have dependencies, an utilization of 100% is unattainable. A resource utilization of 70% is more realistic. [10] Therefore we designed the environments in such a way that the average load on the system is equal to roughly 70% of the systems capacity, as can be seen in table 2.

Workload	Setup	Avg. load
w1	s1	0.696
w1	s2	0.709
w2	s1	0.686
w2	s2	0.699

Table 2: Average load for each setup

Two of the policies, HEFT and CPOP, take into account the heterogeneity of an environment, whereas the other policies do not. Therefore we wanted to compare the performance of the policies in a heterogeneous and homogeneous environment. We designed two datacentre setups, shown in table 3, where the i7 and i5 cores are defined in the OpenDC framework (i7 is roughly 17% faster then i5). As the table show setup 1 is heterogeneous and setup 2 is homogeneous.

Setup	i7 cores	i5 cores	Machines
s1	20 (5x4)	10 (5x2)	10
s2	28 (7x4)	0	7

Table 3: Datacentre setups

5. Results

For the analysis we look at the following metrics: the wait time and the makespan. First starting with the overall wait time with respect to the different policies and second the average wait time in 15 minute intervals. Both will show the wait time at the task and the job level. Finally, the makespan metric results will be discussed.

5.1. Wait time

The wait time metric measure shows how long it takes for a task to be run from the moment it was added to the scheduling queue. Figure 2 shows the measured wait time of the tasks. We observe an increase in wait times for the second setup, both for workloads 1 and 2, this is due to setup 2 having less compute power, as discussed in 4.2.

We see that the wait times for the HEFT policy have a large number of outliers. We believe this is due the greedy nature of the algorithm, as previously discussed in section 3.1. HEFT sacrifices long-term benefits, e.g. overall average wait

time, for short-term benefits, e.g. reducing the time until a next task is completed. This is one of the known short-coming of HEFT. SRTF-Round Robin shows similar results. Again we believe that the greedy nature of the SRTF algorithm causes these results.

We do find it interesting that CPOP doesn't have a large amount of outliers despite the algorithm being so similar to HEFT. The only difference with the HEFT policy is the calculation of the downward rank of a task, this seems to improve the algorithm significantly compared to HEFT.

Figure 3 shows the wait time of jobs. The job wait time metric measure shows how long it takes for the first task of this job to be run from the moment it was added to the scheduling queue. We see that for CPOP, FIFO-Round Robin and HEFT the wait time is much higher compare to the other algorithms.

For FIFO this is expected as tasks are always placed at the end of the queue, meaning that a long job will block all other jobs and increase there wait time. For HEFT we have the same argument as we had for the wait time of a task, we believe this is caused by the greedy nature of the algorithm. CPOP seems to have roughly the same results as HEFT, meaning that although the addition of the downward helps the wait time of individual tasks, it doesn't seem to be the case of the wait time of jobs.

5.2. Average wait time in intervals

In the previous section we looked at the wait times in absolute numbers, in this section we'll look at the them in intervals of 15 minutes. This is interesting because it might be acceptable, but at least expected, that the wait times increase after an increase in workload.

Figure 4 shows the average wait time of tasks. An increase in wait time is clearly visible after the 200 minute mark for workload 1, which corresponds with the increase in workload around the same time mark, see 1.

SRTF-Round Robin is a clear winner here, especially during the bursts of increased workload. Apparently simple is better, at least for this metric and these workloads.

Figure 5 shows the average wait time in intervals of 15 minutes for jobs. Much like we saw in section 5.1 CPOP, HEFT, FIFO-Round Robin preform badly, compared to the other algorithms.

5.3. Makespan

The makespan is the time elapsed from the moment the first task of a job is added to the scheduling queue, to the completion of the last task of the job [1]. Since the wait time for the jobs for the policies CPOP, FIFO-Round Robin and HEFT is higher than the others we expect that the makespans will also be higher.

Figure 6 shows the makespans for all jobs for the different allocation policies. We observe that HEFT indeed has noticeable more jobs with longer makespans than the other policies, but CPOP and FIFO-Round Robin do not. Again here the calculation of the downward rank seems to improve the CPOP algorithm compared to HEFT.

We believe that because HEFT only prioritises based on the upward rank, the lasts tasks of a job are more likely to be blocked by tasks from newly arriving jobs compared to CPOP which also considers the downward rank. This means that during times with peak arrival rates, the last couple of tasks of nearly completed job may not be scheduled until the newly arriving jobs are at least partially completed. This may explain why HEFT shows a considerably longer makespan for some tasks.

Like HEFT, we also see that SRTF-Round Robin has many outliers with a high makespan while the median remains low. We believe this may be because with both HEFT and SRTF-Round Robin it might occur more often that newly arriving tasks get higher priority than tasks already in the queue. In the case of SRTF-Round Robin tasks with a long completion time are blocked by shorter tasks. In the case of HEFT the last tasks of a job are given a low priority and thus the tasks from newly arriving jobs are scheduled first.

Figure 7 shows the makespan for jobs in 15 minute intervals. For workload 1 we see the same increase in makespan around the 200 minute mark, as we saw for the wait times previously.

But in this graph HEFT is the only outlier, while the other policies are closer together. We already determine that this was the case for HEFT, but it's interesting to see that CPOP and FIFO-Round Robin are closer to the other policies. From figure 3 we saw that for CPOP and FIFO-Round Robin the wait time of the first task may sometimes be higher, possibly caused by contention during peak arrival times, but here we see that the average completion time per job are generally lower.

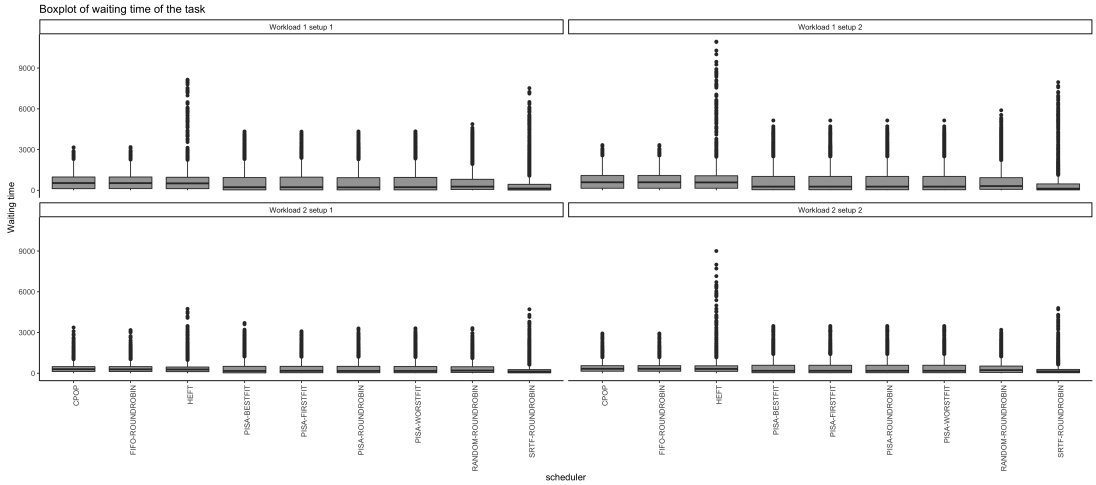

Figure 2: Wait time for tasks in workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

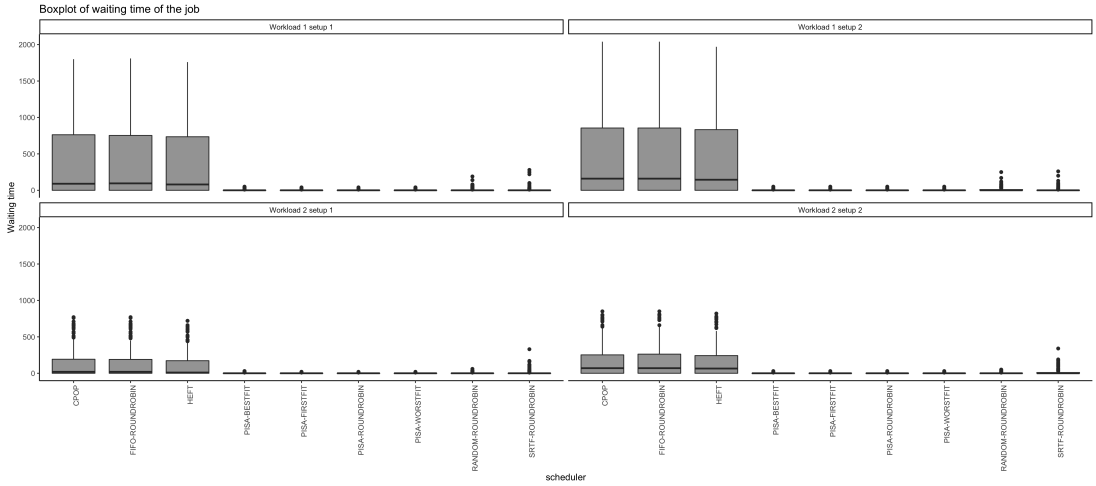

Figure 3: Wait time for jobs in workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

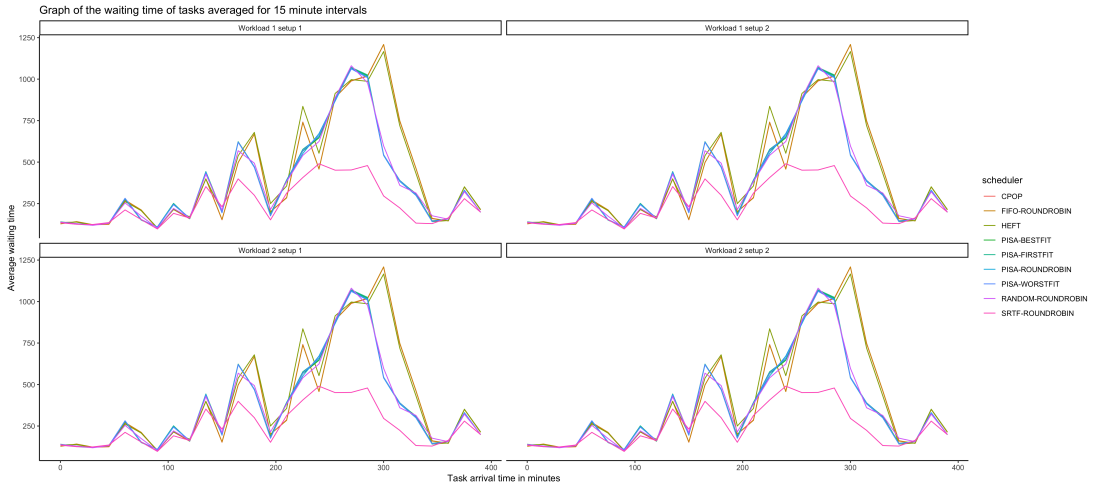

Figure 4: Wait time for tasks in intervals of 15 minutes in workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

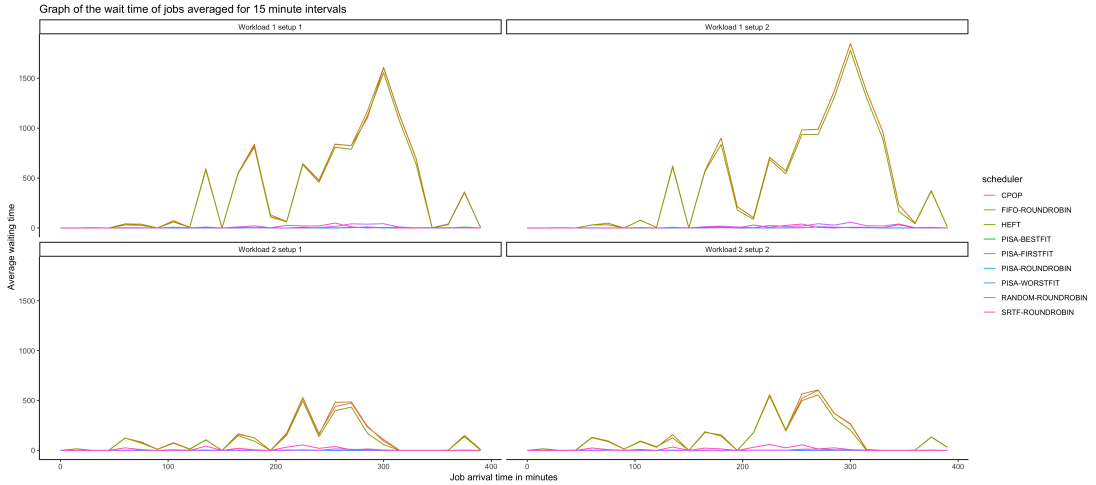

Figure 5: Wait time for jobs in intervals of 15 minutes in workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

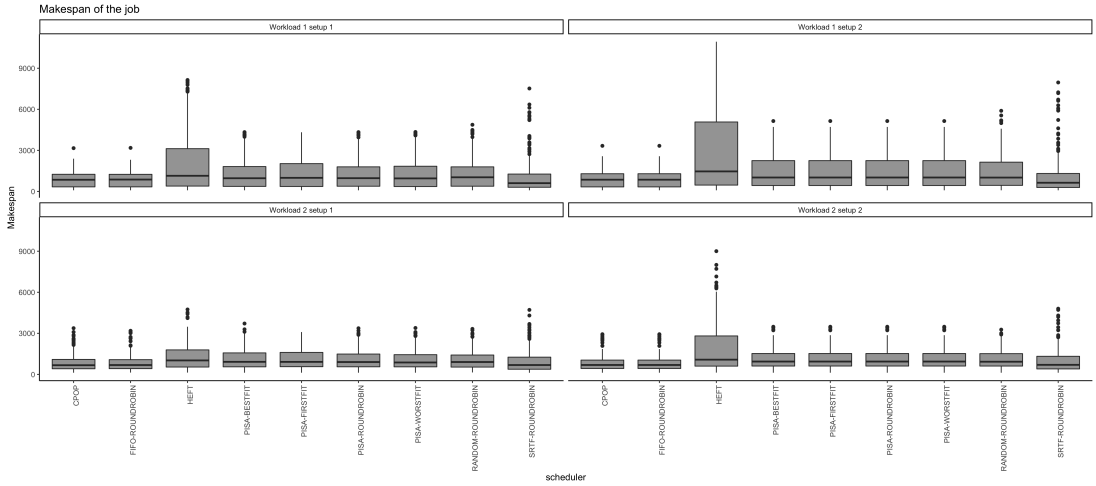

Figure 6: Makespan for jobs for workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

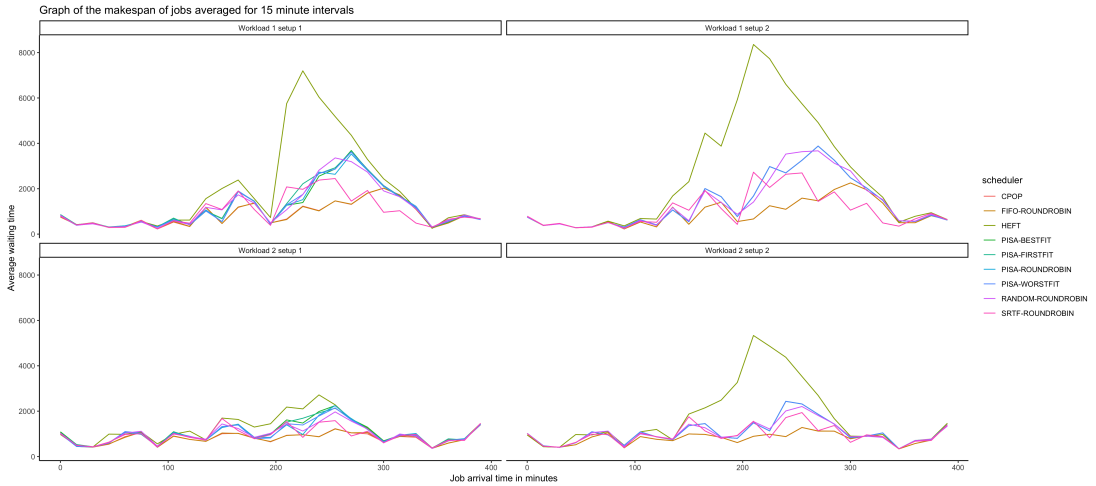

Figure 7: Makespan for jobs in intervals of 15 minutes for workload 1 (top) and workload 2 (bottom), using setup 1 (left) and setup 2 (right)

6. Discussion

It is interest to see that appears to CPOP improve on HEFT in a number of metrics by adding the calculation of the downward rank of tasks. But still both CPOP and HEFT are often found at the bottom of the results for our tests. The other algorithms often do better, even though they are very simple in design.

One reason could be that OpenDC doesn't simulate take communication time and does not take it into account when measuring, where both CPOP and HEFT do scheduling. If communication was taking into account we could see different results. As it currently stands, we may be seeing inaccurate results because HEFT and CPOP may choose to schedule tasks to less optimal machines in order to decrease communication times.

It is hard to declare a winner here, we can however say that for the makespan metric FIFO-Round Robin and SRTF-Round Robin have a slight edge over the other allocation policies. For the wait time metrics, both for tasks and jobs, the SRTF-Round Robin has a slight edge.

We would also like to discuss the OpenDC framework. The concept of simulation a datacentre is quite nice and, as far as we can tell, it is done very well. However, partly due to it's age, it far from complete. A number of elements are missing from the simulator, some that we would like to mention:

- Communication between machines not taking into account, e.g. when moving input data for a task to a machine.
- No metrics about the runtime of the scheduling policy. This one is much harder to get then the virtual runtime of tasks on machine, but still very important. As mentioned previously in Google's datacentres the scheduler alone accounts for more than 5% of all cycles [11], so choosing a simpler scheduling over a more complex one could be beneficial in practice.
- Evaluating policies using makespan can have bias towards larger jobs. It would be better to evaluate using job slowdown, which is the makespan divided by the makespan of that job in an empty system with exclusive access to all the processors [6]. An upper bound estimate of this optimal makespan could be made by simulating jobs individually in OpenDC using the corresponding environment setup and multiple scheduling policies. Then use the minimum makespan found as an estimate for the optimal makespan for that job and environment setup. Due to time constraints we were not able to implement this.

7. Conclusion

In this report we have presented a performance analysis comparing four allocation policies for heterogeneous cloud environments. This analysis was doing using the OpenDC datacentre simulator. Our analysis shows that in many cases the simpler scheduling policies outperform the more complex polices, however due to OpenDCs current limitations we feel that these results may be inaccurate and put the HEFT and CPOP policies at a disadvantage.

We have identified and suggested some points for improvement for the OpenDC simulator which, when implemented, may provide more insight into the performance of different allocation policies. But this is a subject for future work.

References

[1] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. A reference architecture for datacenter scheduling: design, validation, and experiments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, pages 37:1–37:15. IEEE / ACM, 2018.

[2] Robert Armstrong, Debra A. Hensgen, and Taylor Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Seventh Heterogeneous Computing Workshop, HCW 1998, Orlando, Florida, USA, March 30, 1998*, page 79, 1998.

[3] Wei-neng Chen and Jun Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 39(1):29–43, 2009.

[4] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. Firmament: Fast, centralized cluster scheduling at scale. In Kimberly Keeton and Tim-

othy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 99–115. USENIX Association, 2016.

[5] Alexey Ilyushkin, Ahmed Ali-Eldin, Nikolas Herbst, Alessandro Vittorio Papadopoulos, Bogdan Ghit, Dick H. J. Epema, and Alexandru Iosup. An experimental performance evaluation of autoscaling policies for complex workflows. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*, pages 75–86, 2017.

[6] Alexey Ilyushkin, Bogdan Ghit, and Dick H. J. Epema. Scheduling workloads of workflows with unknown task runtimes. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, pages 606–616, 2015.

[7] Alexandru Iosup, Georgios Andreadis, Vincent van Beek, Matthijs Bijman, Erwin Van Eyk, Mihai Neacsu, Leon Overweel, Sacheendra Talluri, Laurens Versluis, and Maaike Visser. The opendc vision: Towards collaborative datacenter simulation and exploration for everybody. In Radu Prodan, Florin Pop, and Ralf-Peter Mundani, editors, *16th International Symposium on Parallel and Distributed Computing, ISPDC 2017, Innsbruck, Austria, July 3-6, 2017*, pages 85–94. IEEE, 2017.

[8] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew V. Goldberg. Quincy: fair scheduling for distributed computing clusters. In Jeanna Neefe Matthews and Thomas E. Anderson, editors, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, pages 261–276. ACM, 2009.

[9] Ruben Stam Thomas de Zeeuw Jan Haenen, Kim van Putten. Open-source code of implemented allocation policies, 2018.

[10] James Patton Jones and Bill Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'99 Workshop, JSSPP'99, San Juan, Puerto Rico, April 16, 1999, Proceedings*, pages 1–16, 1999.

[11] Svilen Kanev, Juan Pablo Darago, Kim M. Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David M. Brooks. Profiling a warehouse-scale computer. In Deborah T. Marr and David H. Albonesi, editors, *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, pages 158–169. ACM, 2015.

[12] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA*, pages 80–89, 2003.

[13] Philip J. Rasch. A queueing theory study of round-robin scheduling of time-shared computer systems. *J. ACM*, 17(1):131–145, 1970.

[14] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[15] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *8th Heterogeneous Computing Workshop, HCW 1999, San Juan, Puerto Rico, April12, 1999*, pages 3–14. IEEE Computer Society, 1999.

[16] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[17] Hu Wu, Zhuo Tang, and Renfa Li. A priority constrained scheduling strategy of multiple workflows for cloud computing. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*, pages 1086–1089, Feb 2012.

[18] Yuming Xu, Kenli Li, Tung Truong Khac, and Meikang Qiu. A multiple priority queueing genetic algorithm for task scheduling on

heterogeneous computing systems. In *14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27, 2012*, pages 639–646, 2012.

Appendices

A. Time Sheet

Table 4 shows the documented time spend on each activity related to the assignment.

	Thomas	Ruben	Kim	Jan	Total
Think-time	6:30	10:30	10:45	12:30	40:15
Dev-time	17:55	4:00	15:30	11:30	48:55
Xp-time	1:30	1:30	1:30	1:00	5:30
Analysis-time	0:00	9:00	0:00	5:30	14:30
Write-time	12:30	12:00	12:55	8:30	47:55
Wasted-time	4:15	2:00	3:15	2:45	10:15
Total	42:40	39:00	43:55	41:45	167:20

Table 4: Time sheet

B. Source code

Our source code is available at `https://github.com/Thomasdezeeuw/distributed-systems-lab`. The policies we implemented for this report are also contributed back to OpenDC in the following pull requests: pr#34, pr#35, pr#36, pr#37 and pr#38.