Lab Assignment

Roy van der Heide (2538699), Khalil Mayyasi (2644550)
Enrique Tejera González (2594477)

November 2018

Abstract

This paper presents a comparison of three different resource allocation policies. The goal of this paper was to implement and compare three different allocation policies. The policies that are implemented and compared in this paper are the worst fit policy, the fairness policy and the dominant resource fairness policy. The experimental setup is used to compare the policies where different machine specifications are used. Additional cores will not

1 Introduction

In this report, three allocation policies are implemented and experimented with using multiple experimental setups. The policies are implemented into a simulation tool called OpenDC. The implemented policies are the worst fit policy, fairness policy and the dominant resource fairness policy. We first explain the details of the system and give a brief overview of the simulation tool OpenDC. Second, we explain the three different allocation policies in depth. Third, we define the experimental setup and test the implemented policies according to this setup. Finally, we explain and conclude the results from the experimental setup for each of the implemented policies and compare the aformentioned policies using the results. ,

2 System design

We have researched and implemented three allocation policies according to literature. These policies include the worst fit policy, the fairness policy and the dominant resource fairness policy. These policies are implemented in Java and tested using the simulation tool OpenDC. OpenDC is a data center simulation that allows us to create experiments and tests for self implemented allocation policies. Furthermore, OpenDC enables us to analyze the experiments and tests. However, since OpenDC is a simulation tool, the experimental results will most likely deviate from an actual data center. Therefore, it should be clearly noted

that the results do not represent actual data centers. The description and features of each of the worst fit policy, fairness policy and the dominant resource fairness policy can be found in subsections 2.1, 2.2 and 2.3 respectively.

2.1 Worst Fit Policy

The worst fit policy[2] takes all the workload provided and assigns every task to the largest resource available to execute it. The policy reduces the production of small gaps but as a downside if a process that requires large resource arrives later in the workload then won't be able to be executed and it will be split into smaller task.

2.2 Fairness Policy

The fairness policy[1] distributes and allocates the resources equally to each user, in our case that is, each task. This policy provide better fairness for users but not a good system throughput and also does not have user satisfaction. The fairness metric of this policy is measured to be fair if the throughput of all users with the policy are either all increasing or decreasing.

2.3 Dominant Resource Fairness Policy (DRF)

The Dominant Resource Fairness policy[3], DRF from Now on, is a generalization of max-min fairness to multiple resource types. The algorithm that we implemented works as follows, it tracks the total resources allocated in the system, in our case the cores available. If the arriving task demand can be satisfied, i.e., there are enough resources available in the system, the tasks is launched. The process continues until it is no longer possible to run new tasks, this is, as soon as the machines CPUs has been saturated. Then new tasks are allocated by their size, measured by the time they will take to be finished, executing first those tasks that will take less time. DRF provides provides multiple multiple-resource fairness in the presence of heterogeneous demand. First generalization of max-min fairness to multiple-resource. DRF also satisfies all of the desired properties of fair allocation except for resource monotonicity.

3 Experimental Setup

We ran all our experiments in the **OpenDC** working environment[4]. OpenDC (Open Data Center) can be used as a data center simulation where you can identify how many servers and how many cores in each server you want. It provides the ability to run and test either available policies or run and test your custom policies. The workload we used had many requests and was provided to us by one of team member that created OpenDC, which we provided with the code on github.[**?**] In implementing the policies we used the variable and function of the classes given from the OpenDC on github [5] as well as used

many functions of the List class. We ran three types of data center setups to test and compare our policies. The details of the three setups can be found in the subsections below.

3.1 Setup 1

For the first experimental setup, we used a data center containing two machines. Machine 1 has 16 racks each containing 2 CPUs, which totals to 32 CPUs. Machine 2 has 16 racks each containing 1 CPU.

3.2 Setup 2

For the second experimental setup, we used a data center containing two machines. Both machines have 16 racks each containing 2 CPUs.

3.3 Setup 3

For the third experimental setup, We used a data center with two machines. Both machines have 16 racks each and contains 1 CPU.

4 Experimental Results

In this section, the results for the worst fit policy, fairness policy and the dominant resource fairness policy are compared. The results for the different implemented policies are compared using the experimental setup which is explained in section 3.

4.1 Experiment 1

We ran our workload first with setup 1, which is explained in subsection 3.1. The result of this setup, for the fairness selection policy, is that the execution times are higher compared with another selection policy which is the worst fit. The execution times for the first three tasks of the fairness policy are 8, 10 and 5 milliseconds. Yet, the execution times for the worst fit policy are 6, 8 and 4 milliseconds. This is logical since the fairness doesn't give each task the amount of cores it needs to execute, but it gives a fair amount to all tasks which makes the tasks take longer to execute fully.

The result of running DRF sorting policy we found out that the highest turnaround time is 1156 ms and the largest waiting time is 1140 ms while compared to another sorting policy which is FIFO we find the largest turnaround time is 736 ms and the largest waiting time is 720 ms, which is also logical since the tasks are sorted by their arrival time which allows the tasks to be allocated faster.

4.2 Experiment 2

We ran the same workload but with setup 2, this setup is explained in subsection 3.2. The result for the same two selection policies we get the same three execution times for the fairness as in experiment but the worst fit execution times have now increased to 8,10,5 too. The reason is that fairness didn't get affected since the same fair amount of cores are distributed among the tasks. But the worst fit is affected because now some tasks are taking large amount of cores which they don't need which can stop other tasks from executing.
Now the result of running the sorting policies is that DRF's highest turnaround time has increased now to 1886 ms and the highest waiting time has increased to 1870 ms, also the FIFO's highest turnaround time has increased to 1516 ms and the waiting time to 1500 ms. That happens because both policies will have to wait for a big job that already have entered and taken larger size than in experiment 1.

4.3 Experiment 3

We ran the same workload again but lastly with Setup 3, which is explained in section 3.3. The result for the same selection policies we see that the execution time for fairness is now decreased to 6,8,4 to the same as the worst fit in experiment 1, while the worst fit has not increased or decreased compared to experiment 1. The reason now is because now all core are equal to 1 which means that fairness between each machine has now decreased. The worst fit hasn't changed is because the execution for these three tasks hasn't been affected by the setup change of cores.
Now the result of running the sorting policies is that DRF's highest turnaround time has decreased to 843 ms and the highest waiting time has decreased to 830 ms, also FIFO's highest turnaround time has decreased to 383 ms and the highest waiting time has decreased to 370. The reason is because now large size tasks doesn't take larger cores than in experiment 2.

5 Conclusion

In conclusion, the addition of cores to a machine might not always be a good idea for some of the policies as we saw in experiment 2 the larger the amount of CPUs we have in a fixed amount of machines the worst turnaround and waiting time and this depends on the policy and its features. For example, the fairness policy execution time wasn't affected by the increase of CPUs as the fairness measure that is distributed between tasks wasn't affected by much, and we might also find a policy where keeping the same amount of machines and increasing the CPUs would actually help it perform better.

6 Discussion

7 Appendix A: Timesheet

	Time
total-time	67 hours
think-time	6 hours
dev-time	24 hours
xp-time	3 hours
analysis-time	2 hours
write-time	8 hours
wasted-time	24 hours

References

[1] Eunji Hwang, Suntae Kim, Tae-kyung Yoo, Jik-Soo Kim, Soonwook Hwang, and Young-ri Choi. Resource Allocation Policies for Loosely Coupled Applications in Heterogeneous Computing Systems.

DESCRIPTION

[2] `https://www.tutorialspoint.com/operating_system/os_memory_allocation_qa2.htm`

[3] Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. `https://cs.stanford.edu/~matei/papers/2011/nsdi_drf.pdf`

[4] OpenDC Webpage `https://opendc.org/`

[5] OpenDC-Simulator GitHub `https://github.com/atlarge-research/opendc-simulator`

[6] Our Code, Setups, and workload we used in this project on github. `https://github.com/KhalilMay/DS_AllocationPolicies`