An Empirical Comparison of Allocation Policies

L. Versluis*, A. Losup*, A. Uta*,
L. Hakkesteegt†, E. Kultorp†, F. Zweet†, T.J. Kurk†, S.O. Swanborn†
‡ Contact information can be found below in respective order

Abstract*— This paper documents an empirical comparison of three different allocation algorithms for distributed systems. This is achieved with the use of the DGSim simulator developed by the Vrije Universiteit, where we vary cluster size and workload across experiments for each allocation algorithm. From the experiments we obtain the metrics of average makespan, task throughput and completion time. After obtaining the results of our experiments, we interpret these metrics to formulate specific observations which we explain in light of the nature of the algorithms and the input we fed them. With this material we are able recommend particular algorithms for different circumstances, answering our research question: *How do different allocation policies compare when using different workloads with varying cluster sizes?

I. Introduction

Many new schedulers and policies (algorithms) appear each year, but few are well-understood or adopted in practice [3]. Research into why this is so has been done by, among others, our supervisor and course instructor [1]. For our smaller-scale project, we select three allocation policies published in academic papers and benchmark them with the DGSim simulator under varying workloads and clustersizes.

This report will therefore answer the following question: ***How do different allocation policies compare when using different workloads with varying cluster sizes?***

A. General requirements

The general requirements our project must adhere to are described here, this paper aims to convince the reader that we satisfied them fully.

- At least three implemented policies.
- Processes real-world workloads of workflows.
- The policies must be compared in different settings.
- At least three different settings must be used.
- In-depth analysis of the results.

B. Specifications

To be able to select policies and test them with certain settings, a few assumptions about our system need to be made. The policies will be analyzed *in-silico* using the DGSim simulator.

- The system will rely on global static allocation policies.
- The system will rely on homogeneous independent nodes without communication costs.

[1]This research has been sponsored by WantScheduler BV.
*Experiment supervisor, Vrije Universiteit Amsterdam
†Authors and students, Vrije Universiteit Amsterdam
‡rens23@hotmail.com, ‡e.kultorp@student.vu.nl
‡frankzweet@live.nl, ‡thijmen.j.kurk@gmail.com
‡stanswanborn@gmail.com

C. Previous work

We have built our understanding of the designed system and its requirements from previous scientific papers. We read into a previous task allocation survey [5]. This paper from TU Delft gave a global, easily understood and fast comprehension of the various aspects that need to be addressed by allocation policies in general distributed systems. We continued with a more in-depth and specific look on static heuristic policies [4].

D. Further structure

Through this document we will first present some background on the DGSim simulator and each of the algorithms, measures, and variables. We then give some technical details on how we set up our experiment before we present its‘ results. From there we proceed to present the raw results, before interpreting them and formulating observations which we analyze. We then attempt to explain these findings based on theory, and finally we make a recommendation on which policy to use in different situations.

II. Background on application

DGSim is a closed-source distributed cluster simulator developed internally by Vrije Universiteit Amsterdam. We use this simulator to simulate the behavior of our selected policies in a distributed system. The system features a simulated CPU clock, allowing reproducibility of results independently of the host hardware. It has a multitude of configurable settings, some of which we vary in our experiments. Further it comes with a number of workloads which can be used as input for experiments. DGSim also includes various scheduling policy implementations. This allows for easy adaption to integrate new implementations. These features makes it ideal for our use-case, where we want to see how algorithms compare under different circumstances.

DGSim is set up to adhere to the specifications given in section I-B.

The chosen policies and the specific design of the implementation will be described further in section III.

A. Tool requirements

The simulation tool we use is subjected to a few requirements given by the assignment description. The term ”tool” encapsulates the software package we used, including DGSim and our own implemented policies. This paper has the goal to convince the reader in the coming chapters that our tool is able to adhere to the following requirements.

– Does the analysis tool compare state-of-the-art or commonly used allocation policies?
– Does the analysis tool report back performance statistics?
– Does the analysis tool use real-world workloads of workflows?
– Does the analysis tool compare policies in different settings?

III. System design

In this section we will give a general overview of our overall designed system by providing a description of:

– The selected policies and their inner workings.
– The evaluation metrics.
– The varying settings.
– The experimental setup.

A. Policies

We have selected three allocation policies:

– Hu‘s Algorithm (HU) [2]
  – Hu‘s Algorithm for In-Trees is a simple model which prioritizes tasks based on the depth of their dependency trees - it chooses the N tasks with the deepest depth D, where N is the number of available cores. If there are more than N tasks with depth D, the tasks are chosen arbitrarily from the set of tasks with depth D. Thus Hu‘s algorithm assumes all tasks are equal except for their position in the dependency graph. Hu‘s algorithm can therefore be characterized as naive - it makes assumptions that are not necessarily true. Thus, if Hu‘s algorithm compares well, this can indicate that the shape of a dependency graph is an important factor to consider when developing scheduling policies.
– MCP Algorithm (MCP) [6]
  – The Modified Critical Path (MCP) algorithm uses the As-Late-As-Possible (ALAP) attribute of a node. The ALAP is a measure of how far the node‘s start-time can be delayed without increasing the schedule length. The MCP algorithm first computes the ALAPs of all the nodes, then constructs a list of nodes in an ascending order of ALAP times. Ties are broken by considering the ALAP times of the children of a node. The MCP algorithm then schedules the nodes on the list one by one such that a node is scheduled to a processor that allows the earliest start-time using the insertion approach. The definition of the MCP algorithm was cited from [4].
– Shortest task first (STF)
  – This allocation policy checks which task is the shortest one to complete and schedules it. This procedure repeats itself until all tasks are scheduled. This algorithm sounds nice in theory, however, it is hard to know the execution time of a task before executing it. In our case we use the expected execution times to schedule the tasks.

B. Metrics

We will evaluate the results of the different policies based on three metrics that can be extracted from a successful run in DGSim:

– Makespan
  – Difference in time between the start and finish of tasks. The ”Average Makespan” is the average makespan over all tasks.
– Throughput
  – Throughput, a term to describe the rate at which tasks can be processed (i.e, the average number of processed tasks per time unit).
– Completion time
  – Time required to complete one particular task (includes the time that a task is in the queue).

C. Varying settings

In this section we describe the settings with which the experiments are constructed and each policy is tested. The two workloads are selected because they come from different industries and have different features. Each workload is reduced in size to allow for quicker experiments.

1) Workloads: We will test each policy with two different workloads: Chronos and Askalon. We give a general description of these two workloads.

a) Chronos workload: The Chronos workload is an industrial workload of workflows and is available within the source of the project. It contains a trace of Shell‘s Chronos environment that handles IoT sensor data. The traces are based on Poisson arrivals of tasks, every flow contains one task and there are no dependencies between tasks. The image shows a small set of the 1255 nodes illustrating this structure. The tasks have varying values for “RunTime”, so they are not identical. They do however all require exactly one core.

Fig. 1: Subset of the Chronos workload

b) Askalon workload: The Askalon workload is also available in the source of the project and contains an engineering workload of workflows. Askalon is characterized by the peak in tasks at the beginning of each workflow and its degree of parallelism. The workflows consists of a number of disconnected trees of various depths. As with the Chronos workload, the nodes have varying “RunTime” values and all require exactly one core.

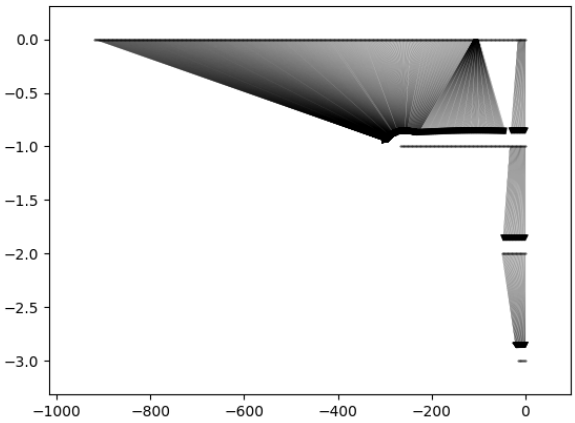

Fig. 2: Nodes in the Askalon workload

2) Clustersizes: On top of testing all the policies with varying workloads, we will vary the clustersize for each policy and workload. We will vary these between three values: 10, 50 and 100. These values are arbitrarily chosen. Each of the clusters will be setup in DGSim to be homogeneous.

D. Experiments

The above mentioned settings and policies have been combined to form three experiments. Each experiment is ran for each policy with a varying workload. This means 2 workloads x 3 policies = 6 runs per experiment. There are 3 experiments so 6 * 3 = 18 different experiment results that will be visualized and analyzed in section IV.

TABLE I: Experiment 1

Attribute	Content
Workloads	Chronos, Askalon
Allocation Policies	STF, HU, MCP
Clustersize	10

TABLE II: Experiment 2

Attribute	Content
Workloads	Chronos, Askalon
Allocation Policies	STF, HU, MCP
Clustersize	50

TABLE III: Experiment 3

Attribute	Content
Workloads	Chronos, Askalon
Allocation Policies	STF, HU, MCP
Clustersize	100

a) System setup: The DGSim simulator is written in Python 2.7. The experiments are ran on a Macbook Pro 13 inch 2018 with an 2,5 GHz Intel Core i7. The results are analyzed and visualized in R.

IV. Experimental results

A. Expected outcomes

We expect for the Chronos workload to see that the policies do not differ much in our metrics. This is because Chronos has no dependency between tasks and therefore scheduling is less impactful.

On the opposite, we expect that the Askalon workload will result in higher differentiating metrics between the policies since this workload has a high degree of dependency between tasks. Scheduling will therefore have a significant impact on the metrics.

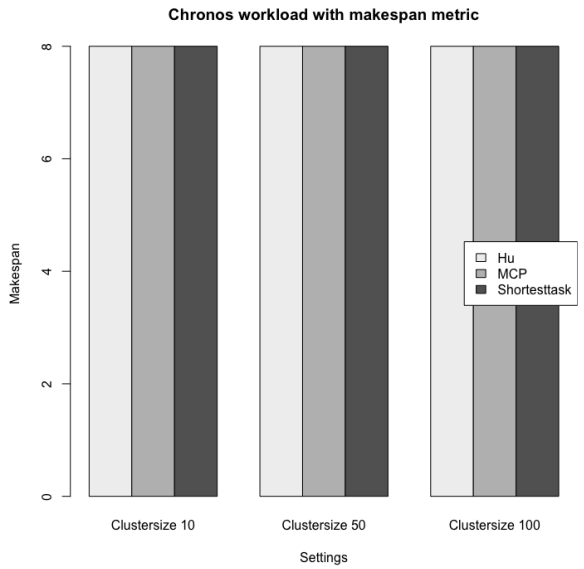

Fig. 3: Average Makespan for the Chronos dataset

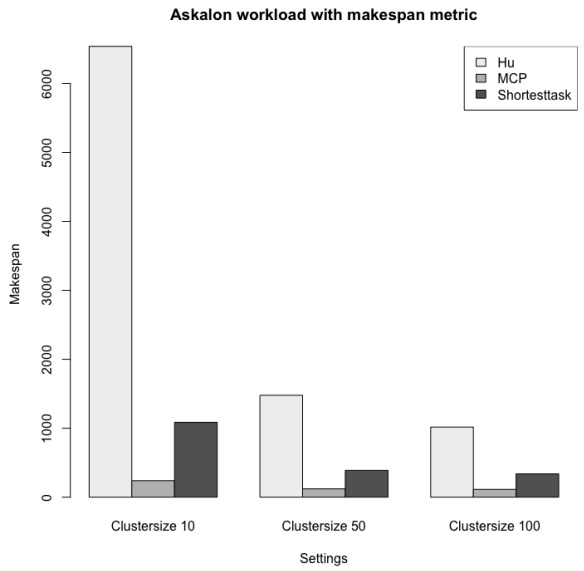

Fig. 4: Average Makespan for the Askalon dataset

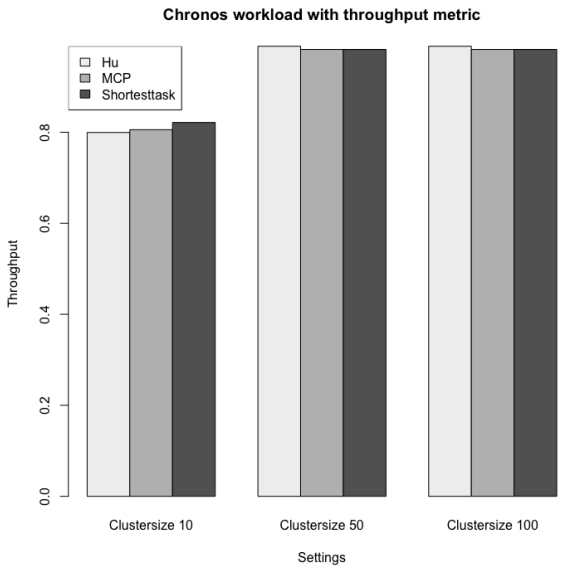

Fig. 5: Average throughput for the Chronos dataset

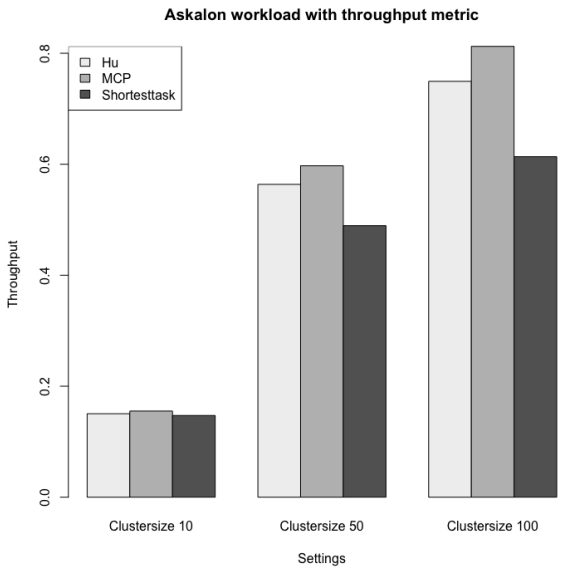

Fig. 6: Average throughput for the Askalon dataset

1) Makespan: First the metric ”Average Makespan” is investigated. In figure 3 and 4 the results of the various simulations is shown. It is immediately evident that the makespan does not change for the Chronos workload. This is because there are no dependencies in the workload so the scheduler can run all of them in parallel. The Askalon workload is more interesting to look at. Because of the nature of the Hu‘s algorithm it will not perform great on a small cluster when a lot of different workflows with a lot of dependencies are submitted. It will try to finish critical tasks first so the algorithm constantly shifts to other workflows. The MCP algorithm performs best, because it gives priority to the workflow it is working on.

2) Throughput: The other measure which is analyzed is the average throughput per time unit. The comparisons are shown in figure 5 and 6. Looking at Chronos workload data it seems that for clustersize 10, Hu is the slowest and then MCP and then Shortesttask. However this changes around for the two bigger clustersizes. However, the difference is so small that we disregard this finding as insignificant.

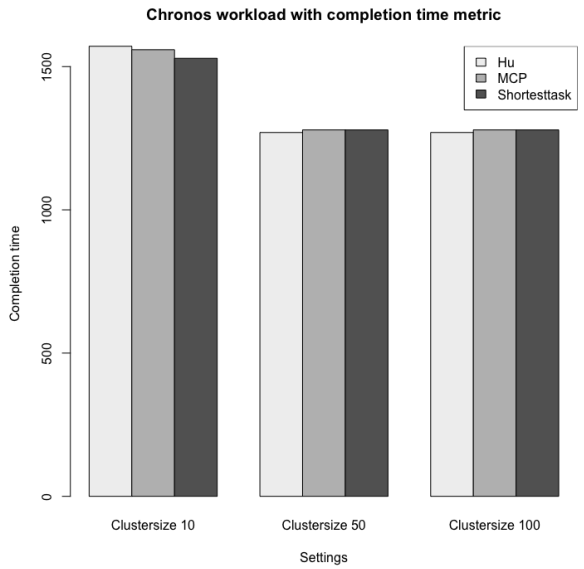

Fig. 7: Completion time for the Chronos dataset

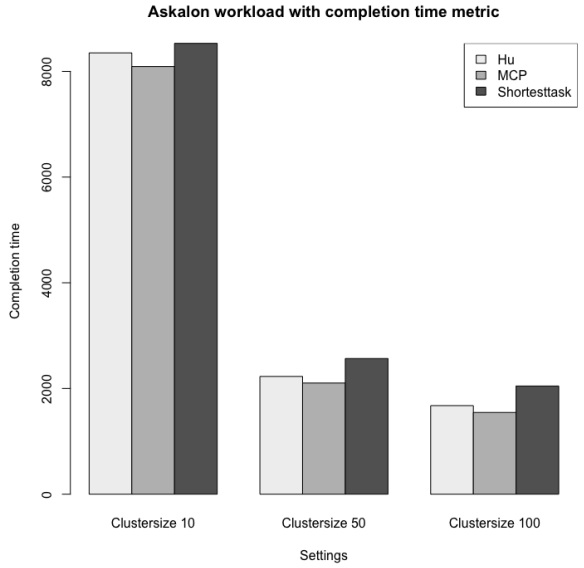

Fig. 8: Completion time for the Askalon dataset

3) Completion time: Finally the completion time is compared. In figure 8 and 7 the various completion times are shown. In the Chronos dataset we can see the same result as for the throughput. The Hu algorithm is taking the longest (highest completion time, lowest throughput) for the clustersize 10, it changes again for the other two clustersizes. We disregard this result once more as insignificant. When looking at the Askalon figure it can be seen that shortest task first performs worst which makes sense since it does not take into account the structure of a workflow which is inefficient.

V. Conclusion / Discussion

We enumerate our observations before going on to interpret and explain them.

1) From figures 3, 5 and 7 we see that the makespan, throughput and completion time for the Chronos workflow has little-to-none variation across allocation policies.
2) From figure 4 we see that the difference in makespans between Hu‘s algorithm and the other policies is higher with lower cluster sizes for the Askalon workflow
3) From figures 4, 6 and 8 we see that the MCP algorithm consistently has the lowest makespan and completion time, and highest throughput.
4) From figure 6 we see that the differences in throughput become more prominent with higher cluster sizes

Observation 1 can be explained by the structure of the Chronos workflow: the Chronos workflow is completely flat, with no task depending on another task first completing. This means that the degree of parallelism is independent of the prioritization of tasks, unlike in a tree where more parallelization can be achieved by giving tasks that block many other tasks higher priority. Thus the variety in throughput is just insignificant. Observation 2 can be explained by the Hu‘s algorithm naively ignoring properties of the tasks themselves. Observation 3 tells us that the MCP algorithm performs the best for this kind of job. Observation 4 shows the importance of maximizing core utilization. Algorithms which are better able to maximize core utilization generally perform better, and the degree to which they out-compete other policies increases with the cluster size.

So we find that the preferred policy depends on the shape of the workflow, and that the importance of policy selection increases with the cluster size. This finding indicates the importance of considering the shape of a job before deciding on how to schedule its tasks. However Hu‘s algorithm schedules in-trees in an optimal way with respect to its‘ shape, but still it was bested by MCP. This is because Hu‘s algorithm does not consider the characteristics of the tasks itself. If all tasks were equal, we would expect Hu‘s algorithm to perform better than MCP.

A. Evaluation of expected outcomes

In the expected outcome, in section IV-A, we stated that we did not expect many differences in the metrics across the policies for the Chronos workload, and that we did expect so for the Askalon workload. This expected outcome turned out to be the case. The insight gathered from the experiments and this expected outcome leads us to the conclusion stated above. The insight is valuable since after this evaluation and reading this paper, a best fitting allocation policy can be chosen based on the workload structure.

B. Recommendation

Based on the interpretations in the conclusion, we recommend that WantScheduler BV deploys a scheduler which first considers the shape of the workflow and then employs the best-performing allocation policy (MCP) unless the workflow

is flat (no dependencies) and the tasks are equal. In which case the chosen allocation policy does not matter, and should be based on which one introduces the least amount of scheduling overhead into the system. Scheduling overhead is not taken into account in the DGSim simulator, because it needs to be deterministic, but one can reason that in a real-world situation it is of course of significant impact.

VI. Reflections

In this section we would like to add some reflection to our work. None of us had any previous experience with distributed systems, let alone the allocation policies used within. This let to us making some mistakes which we would like to elaborate on here.

A. Interpretation of static scheduling

We interpreted static scheduling to be a form of scheduling where all workflows (tasks) are known in advance and arrive at the same time. We therefore altered the workflows in the beginning of our experiments and changed the submit time to 0. This however changed the way the workflows worked and therefore did no longer represent a real-world workflow. We only realized this later when reading a different definition of static scheduling, describing it to be a form of scheduling where all tasks arrive before execution and not during execution. Which can be achieved with the submit times as they were. We therefore reverted the change and ran the experiments again. This took time, time we would have liked to have at the end of the deadline.

B. Wrong implementation of allocation policies

When discussing our results with our supervisor we realized that our runtime of the experiments were too long (we compared the runtimes for a workload, for us it was around 40 minutes and for our supervisor only a few minutes).

This made us wonder what could cause this significant decrease of performance. We quickly found out that all our policies contained an inefficient loop, it did not stop iterating when it could have stopped. We fixed the issue and could now run the full, unmodified workload in a reasonable amount of time. However, the results from the full workload are not included in this report because of time constraints.

VII. Open source

The report is made available open source, under the MIT license. And can be found in the following repository: `https://github.com/TeamBLTN/Allocation-Policy-Comperison`.

References

[1] Georgios Andreadis et al. "A reference architecture for datacenter scheduling: design, validation, and experiments". In: *A Reference Architecture for Datacenter Scheduling: Design, Validation, and Experiments*. IEEE. 2018,

[2] Alexey Chernigovskiy, Roman Tsarev, and Alexey Knyazkov. "Hu's algorithm application for task scheduling in N-version software for satellite communications control systems". In: *2015 International Siberian Conference on Control and Communications, SIBCON 2015 - Proceedings* (July 2015). DOI: 10.1109/SIBCON.2015.7147270.

[3] Dalibor Klusáček and Šimon Tóth. "On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations". In: *European Conference on Parallel Processing*. Springer. 2014, pp. 138–149.

[4] Yu-Kwong Kwok and Ishfaq Ahmad. "Static scheduling algorithms for allocating directed task graphs to multiprocessors". In: *ACM Computing Surveys (CSUR)* 31.4 (1999), pp. 406–471.

[5] Mahyar Shahsavari et al. "Task scheduling policies in general distributed systems: a survey and possibilities". In: (2004).

[6] Min-You Wu and Daniel Gajski. "Hypertool: a programming aid for message-passing systems. IEEE Trans Parallel Distrib Syst". In: *Parallel and Distributed Systems, IEEE Transactions on* 1 (Aug. 1990), pp. 330–343. DOI: 10.1109/71.80160.

Timetable

The table below shows the time spent to analyze the different allocation policies and writing this report. The total time is divided into subcategories to give more insights per category.

Category	Description	Hours
total-time	Total amount of time spent in completing the assignment.	62 hours
think-time	Total amount of time spent in thinking about how to solve the assignment	20 hours
dev-time	Total amount of time spent in developing the code needed to solve the assignment.	8 hours
xp-time	Total amount of time spent in experiments for the assignment.	8 hours
analysis-time	Total amount of time spent in analyzing the results of the experiments for the assignment.	4 hours
write-time	Total amount of time spent in writing this report.	12 hours
wasted-time	Total amount of time spent in activities unrelated to the assignment.	10 hours