TU Delft: Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science (EEMCS)

DISTRIBUTED COMPUTING SYSTEMS
(IN4391)

LARGE LAB EXERCISE A
The Virtual Grid System:
Distributed Simulation of Distributed Systems

Authors
Filipe Fernandes, João Serra, Vasco Conde
{F.M.FonsecaMeirinhosFernandes, J.P.Serra, V.BarrosMendesNazareConde }@student.tudelft.nl

Support Cast
Dr. ir. Alexandru Iosup {A.Iosup@tudelft.nl}
ir. Yong Guo {Yong.Guo@tudelft.nl}

1. Abstract

Computing grids have been used in the past 20 years to provide computational resources to a heterogeneous public that want to realize simulations with big computational requirements. During the past years grids have evolved into multi-cluster distributed systems that provide a better solution by being able to distribute the workload among independent clusters by means of a dedicated scheduling entities. Our work consists in designing and implementing a distributed, replicated and fault-tolerant distributed simulator of a multi-cluster system, the Virtual Grid Scheduler. Within our findings we can point out that the performance of such a distributed multi-cluster simulator is better than a non-distributed one.

2. Introduction

The previous structure of the Virtual Grid System contains only a single grid scheduler node which is responsible for managing all the jobs. This architecture has limitations regarding scalability, fault-tolerance and performance due to the fact that the scheduler is composed only by a single node. A proposed solution to address these issues is a distributed scheduling system which consists of multiples grid scheduler nodes that communicate among each other to reach a solution for the presented problem, scheduling jobs from several clusters.

Our system comprises some of the ideas of past work and tries to solve in a simple way the problem of having a centralized scheduling. We propose a grid scheduler which is composed by several nodes, each responsible for a subset of the jobs. This approach tries to mitigate some of the limitations present in the non-distributed version.

Through this paper we present in section 3 the Background of our application with emphasis in the requirements. Further ahead, in section 4, we explain our System Design. Then the Experimental Results are present in section 5 followed by a Discussion in section 6. We then conclude in section 7, and in section 9 the time spent in this assignment is shown in different categories.

3. Background

VGS, Virtual Grid System, is a distributed simulator of a multi-cluster system that allows the user to, in a distributed way, deploy the principal nodes that are part of a multi-cluster architecture and run simulations on them. Such nodes are Grid Schedulers and Clusters. Grid Schedulers enable load sharing among the Clusters present in the system and Clusters, as a node of the simulation, are composed by a Resource Manager and a set of Computing Nodes. The Computing Nodes are responsible for the execution of the Jobs whilst the Resource Manager manages the jobs that are being sent to the Cluster.

Our system maintains a list of actions that were carried out by either Grid Scheduler or Cluster nodes. As far as Clusters are concerned there is no need for synchronization between logs. Although, this is a key issue in Grid Schedulers because they must know how the system is performing at a given point in time. We apply a form of causal consistency, with the help of vectorial clocks, to keep the logs consistent.

Scalability is important and was considered as a key aspect in our design. One of the requirements was to be able to add a considerable amount of computing nodes, meaning Clusters, in the system as a way of improving performance. At some extent the system can also thrive from having more grid schedulers. We will get into more detail about this subject in the experimental results section.

Losing all the data of an important simulation just because some nodes crash is not desirable and so fault-tolerance arises as one of the most important aspects in a distributed systems and more specifically in our application. We make sure that a temporary crash of a computing node doesn’t affect the outcome of the simulation in terms of data. Grid Schedulers crashes are also taken into account and the application is prepared to deal with multiple GS crashes.

Having a distributed scheduler should lead to an improve in performance since we have more components that can work in different sets of information. This aspect is tightly related with scalability since the system will suffer if we have high number of Clusters compared to the number of GS’s.

4. System Design

4.1 System Overview

The design of our system is based on two main components, Grid Scheduler and Cluster nodes. Grid Schedulers are an essential part of our system in a sense that without them we don't have the distributed environment. Even if this is not a requirement, a cluster can run and perform their actions without contacting the grid schedulers.

Now we are going to describe how the different nodes are initialized and from there we go into the flow of the application.

When launching a GS there is the option to provide a GS to which the new one should connect to. The intent with this is to create a self-sufficient network of GS's without the need of having a central point controlling the entities connected to the system. Information about new GS's connecting to the systems is then relayed to others that might be connected. GS's are, individually, responsible for managing their own list of GS's afterwards.

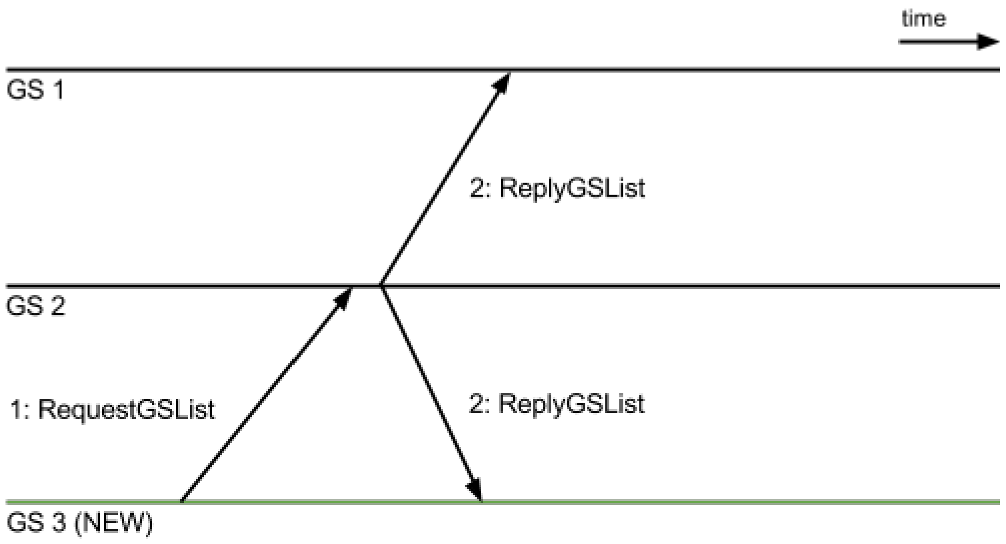

Figure 1. GS's exchange the list of GS's upon startup

As far as Cluster nodes are concerned the need to contact a GS upon initialization is required, being the objective the creation of a distributed simulator. Thus, a Cluster starts by creating it's virtual RM and a set of Computing Nodes, trying afterwards to connect with the given GS.

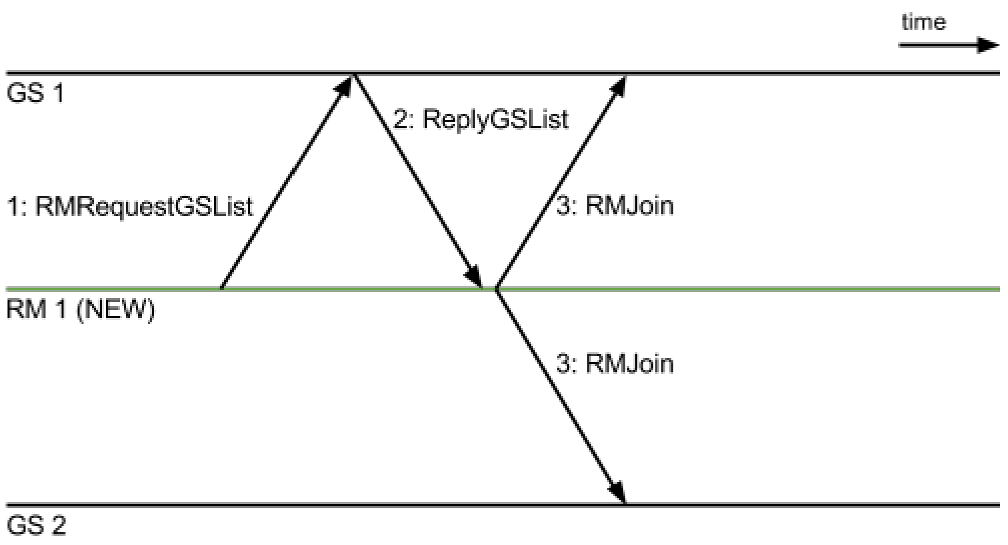

Figure 2. RM joining protocol

When the Cluster receives the list of GS's it tries to establish a connection with the GS's present on this list. In this way GS's are aware about which Clusters are connected to the system and thus available for use, meaning, job delegation.

Following this exchange of information a Cluster enters it's normal mode in which it will start consuming the Jobs available and scheduling them either to it's nodes or to an available GS. In our system as the GS is trading information with Clusters at a periodic rate we have an idea of how much loaded a Cluster is. With that information a GS tries always to delegate the job to the least loaded one, keeping the Clusters load at an uniform level.

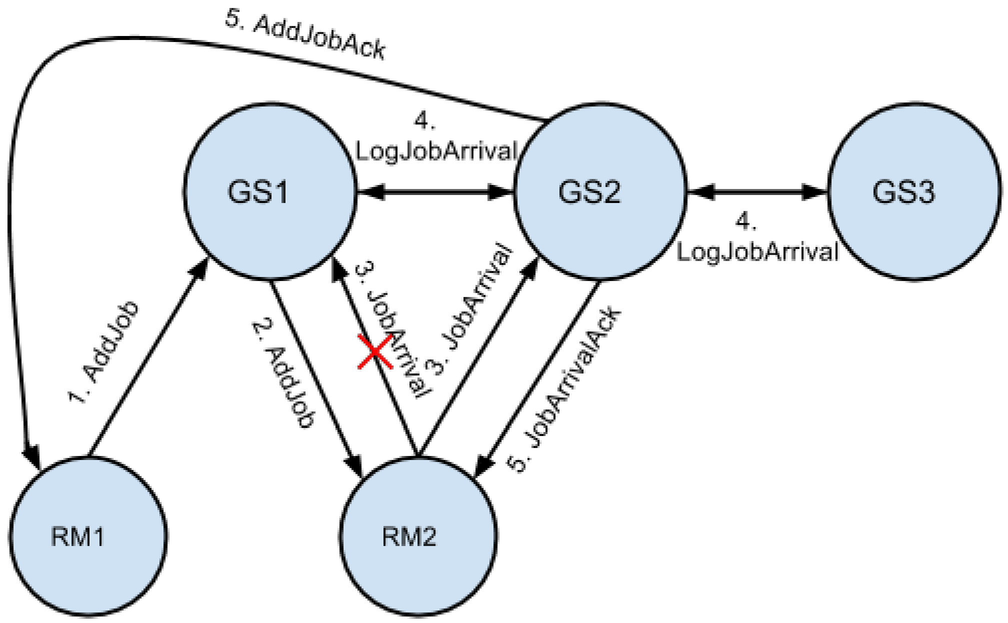

Figure 3. Image showing how job delegation works.

If a certain Cluster doesn't receive an acknowledge message during a predefined timeout it will add the job again to the queue to later delegation. To deal with the AddJob message, which is asynchronous due to the fact that the response might come from a different server, we used timers to control if the ack message was received or not. In the case of the JobArrivalAck we use the socket timeout and if it reaches the limit we relay the event to another GS.

4.1.1 Fault-tolerance

Each entity in the system is responsible to recognize when any other entity fails. We keep track of failures using a mapping between entities, unique address, and the number of failures that they incurred in. An entity is removed from the list after a predefined number of consequent failures.

As Clusters are the ones that connect to Grid Schedulers, and not the other way around, they have no idea when a new Grid Scheduler connects to the system. To mitigate the time that a Clusters spends without knowing about a connected GS the information of GS's present in the system is shared with the Clusters each time that an exchange of load information is carried out between Clusters and GS's.

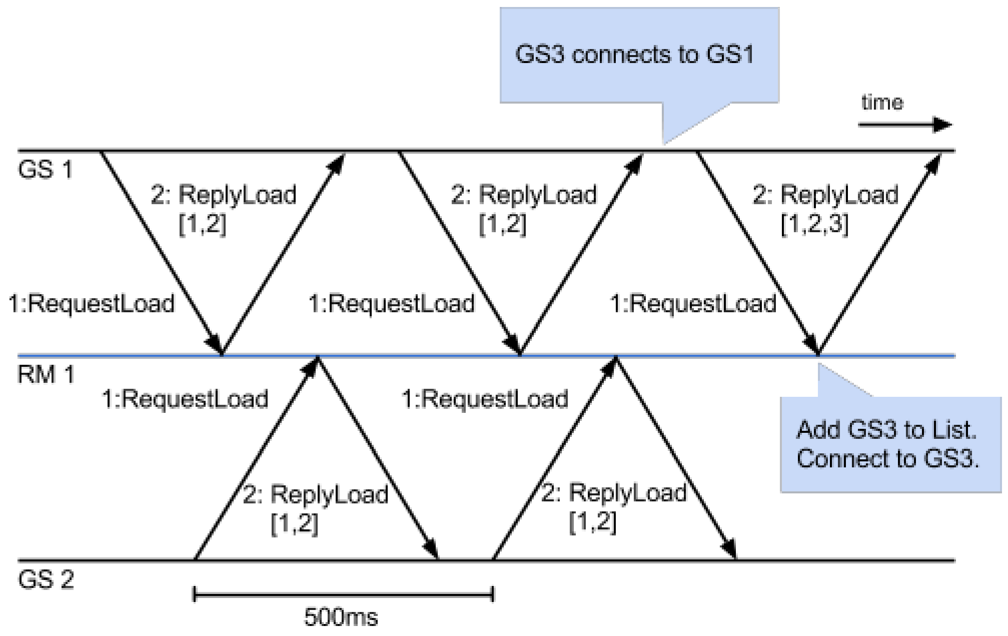

Figure 4. A GS sends the list of connected GSs with the periodic RequestLoad message

Above we explained two of the mechanisms that allow entities to know when some other entity fails or connects to the system. Now we are going to refer to how these entities recover themselves from such failures. Some assumptions were made regarding the failures that can occur without compromising the execution of the system in terms of output. Such assumptions are:

- One of the Grid Schedulers must work throughout all the simulation without failing.
- At a given point in time there is always the need for at least two Grid Schedulers to be running properly.
- Grid Schedulers may crash and not recover if the two first assumptions are not violated.
- A Cluster may crash but its recovery is mandatory to the simulation to end.

4.1.1.1 Cluster recovery

A Cluster is responsible for executing all the jobs that were assigned to him before he failed and so it needs to know which of the jobs it still needs to run. For this the log is used and the jobs that were started but are not logged as completed are put in the queue to future execution. After the Cluster starts it's normal execution it used the information that was recovered from the log file.

4.1.1.2 Grid Scheduler recovery

When a Grid Scheduler recovers there is no synchronization so it just starts working normally after the initial information exchange with a GS already running. From this point on it is just a matter of time until a Cluster starts to query again this GS to delegate their Jobs since the information about GS's present in the system is forwarded to the Clusters by the GS's that were already working before.

4.1.2 Scalability

To ensure that GS's are not flooded with requests from Clusters, the Clusters choose only one GS to which they will try to delegate the job. After that the GS will try to delegate the job to the Cluster that most fits the conditions, in our test case, the least loaded one. With this approach we avoid unnecessary complexity in the job delegation flow and the number of messages traded is kept at a low value for each job delegation.

4.2 Additional Features

The additional features that our system comprises are:

- Advanced fault-tolerance: As explained above, sections 4.1.1.1 and 4.1.1.2, in the fault-tolerance section our Clusters and Grid Schedulers can recover from a failure, continuing their work.
- Benchmarking: The system is ready for benchmarking since we can run as much instances as we want and also define the amount of jobs that we want to run in each Cluster. This feature was used to carry out our experiments.

5. Experimental Results

5.1 Setup

For carrying out our experiments we used one of the Clusters of the DAS-4, The Distributed ASCI Supercomputer 4. We used time, sar and top to monitor execution time, CPU and memory usage respectively. To launch the application through the several nodes we used human input after opening the several ssh connections with a script. It is important to notice that the several instances of Clusters are initialized at different points in time due to the human input.

5.2 Experiments

In order to test our system we tried to approach the experiments from different viewpoints. We tested our system taking into account ratios between the number of Jobs created in the Clusters and also different numbers of Clusters and GS's in execution.

The maximum number of Clusters and GS's used was 20 and 5 respectively. Each Cluster has 100 internal nodes and generates jobs at a rate of 20 milliseconds.

All the experiments ran with a total of 10.000 jobs each in order to have a point of comparison between them. We decided to use this amount of total jobs to effectively test our scheduling algorithm since the number of jobs in at least one of the Clusters was always superior to the number of internal nodes.

We divided experiments in three categories, the first in which we change the initial distribution of jobs among all the Clusters, a second where we vary the number of Clusters or GS's running and a third where we tested some failures..

Below we will describe the different experiments from each type and report the results obtained followed by an analysis section.

5.2.1 Job distribution experiments

In this section of experiments the following setup was used: 10.000 jobs, 5 GS and 20 Clusters. The difference between them is the distribution of the initial jobs among all the clusters. We decided to have one Cluster with a higher number of jobs than the remaining ones. With this in mind we will use the concept of ratio that will give the relation between the initial jobs in the most loaded Cluster and the others.

5.2.1.1 Experiment 1

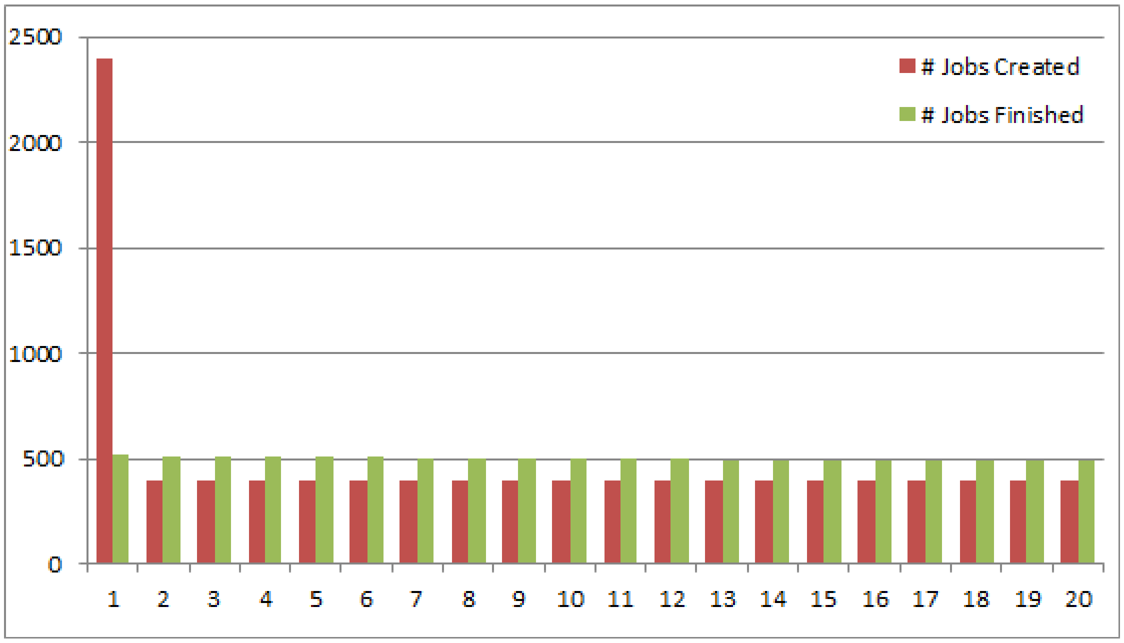

Figure 5. Ratio of 6:1 with the distribution of the jobs being 2400 to 400 between the cluster with more and any of the others. The average execution time was 72 seconds.

5.2.1.2 Experiment 2

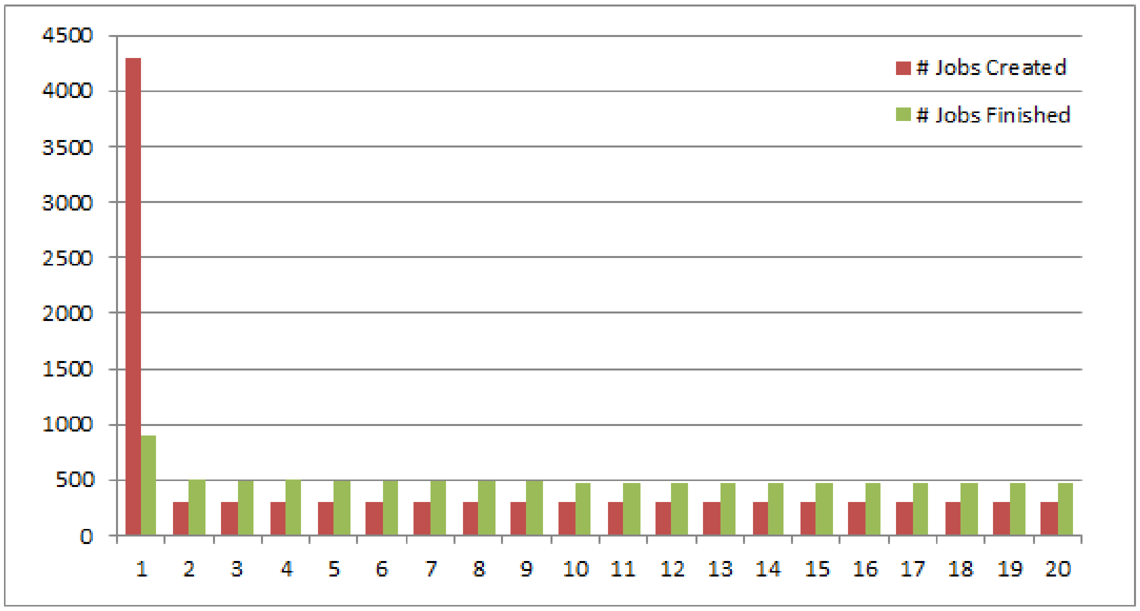

Figure 6. Ratio of 14:1 with the distribution of the jobs being 4300 to 300 between the cluster with more and any of the others. The average execution time was 114 seconds.

5.2.1.3 Experiment 3

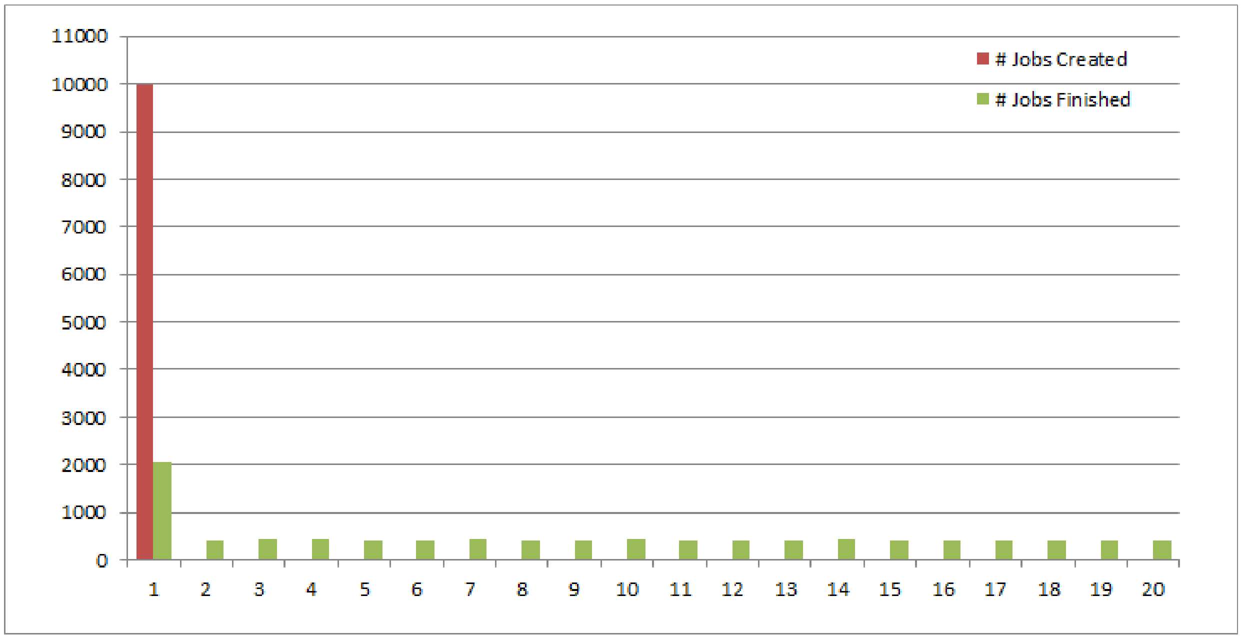

Figure 7. Infinite ratio with the distribution of the jobs being 10000 to 0 between the cluster with more and any of the others. The average execution time was 261 seconds.

5.2.1.4 Analysis

By analysing the above described experiences we can see that there is a relation between the initial job distribution ratio and the final execution job distribution among the clusters. What happens is that the more a Cluster creates jobs, the more it executes locally. This occurs because the cluster has a fixed, not instantaneous, generation rate. Basically if the Cluster is responsible for creating a higher amount of jobs it will have more opportunities to schedule jobs in it’s own queue since past jobs start to finish.

Moreover, we can also see that the execution time increases when a Cluster is responsible for producing more jobs since, as said before, this job creation takes time. This impact is only relevant when the ratio supasses a certain threshold that is set around the 6:1 mark. With lower ratios, the problem does not occur and average time should be similar.

With this experiments we can also see that the scheduling algorithm is working well, meaning that each time a cluster needs to delegate a job it actually happens in a distributed form among all the Clusters available.

5.2.2 Cluster and GS's instances running

Unlike the previous section of experiments in which the focus was the initial job distribution, in this section we focus on how well the system behaves when it misses some components, instances of Clusters or GS's. To perform this test we use always 10.000 jobs equally distributed per all the Clusters present. More about the setup will be described in each of the experiments.

5.2.2.1 Experiment 4

This experiment was carried out with 20 Clusters and 5 GS's. It will be used to compare with others where the number of instances running will be different. The average execution time was 74 seconds.

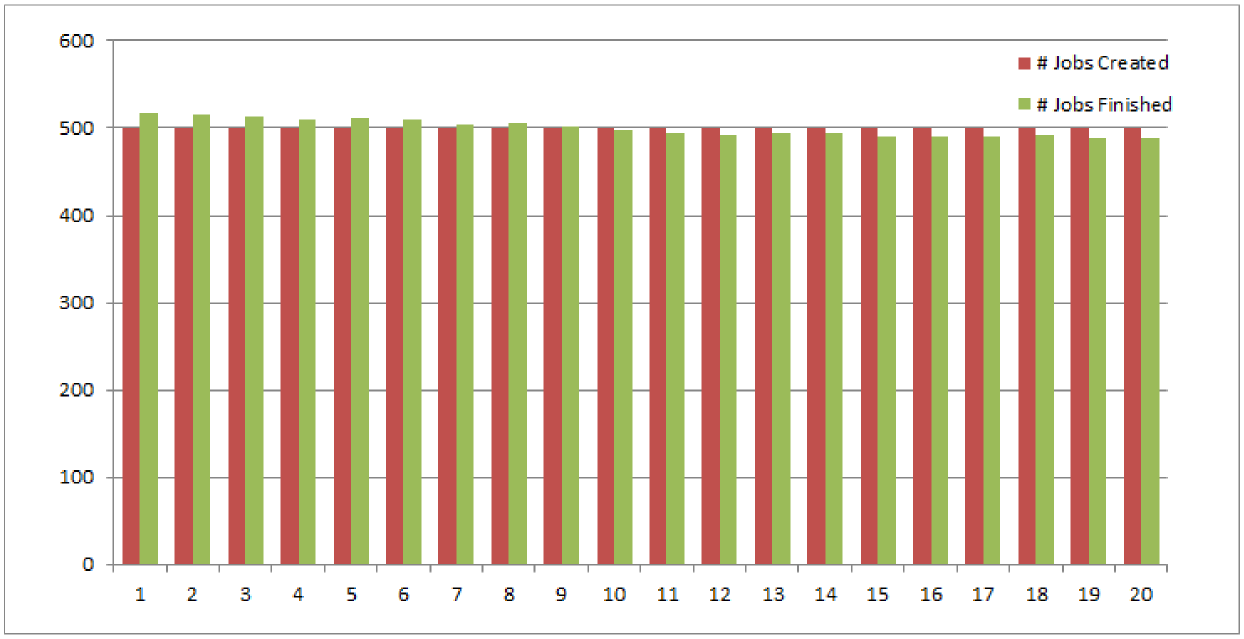

Figure 8. Graphic showing the relation between the average number of jobs created and executed at each Cluster.

5.2.2.2 Experiment 5

In this experiment 4 GS's were used instead of the 5 used in the previous one. The average execution time was 76 seconds.

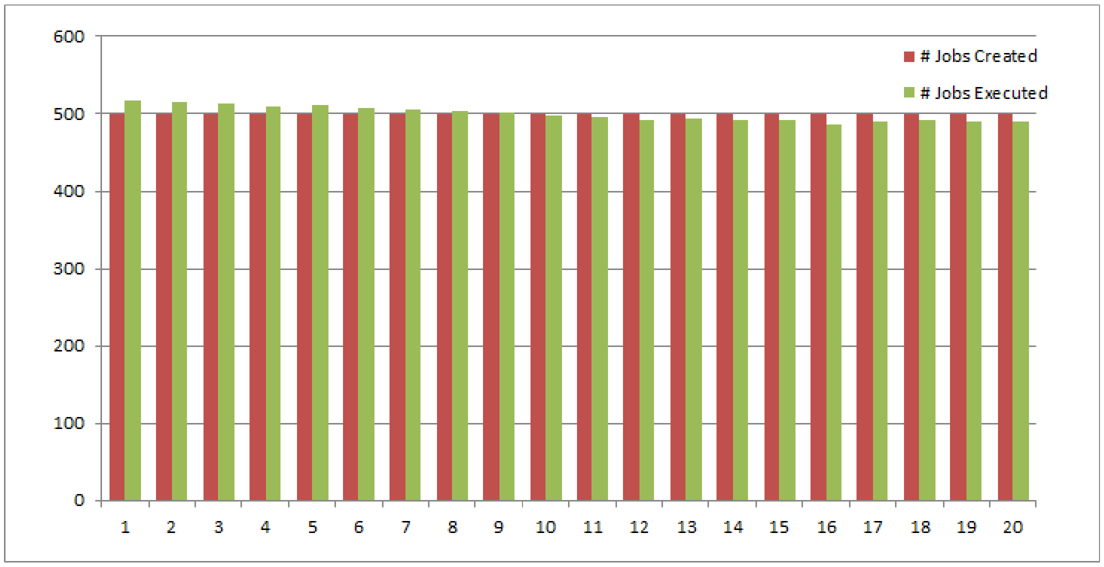

Figure 9. Graphic showing the relation between the average number of jobs created and executed at each Cluster.

We tried also running the system with less GS's for the same number of Clusters, 20, but it reaches a point where the GS's can't handle all the requests coming from the Clusters. What we can point out is that there is the need of maintaining a certain ratio between GS's and Clusters since they are tightly related.

5.2.2.3 Experiment 6

In this experiment the number of Clusters was reduced to 10, half of the ones used in the first experiment of this section. The objective was to see if Cluster reduction had an impact in the performance of the system and if so in which way. The average execution time was 118 seconds.

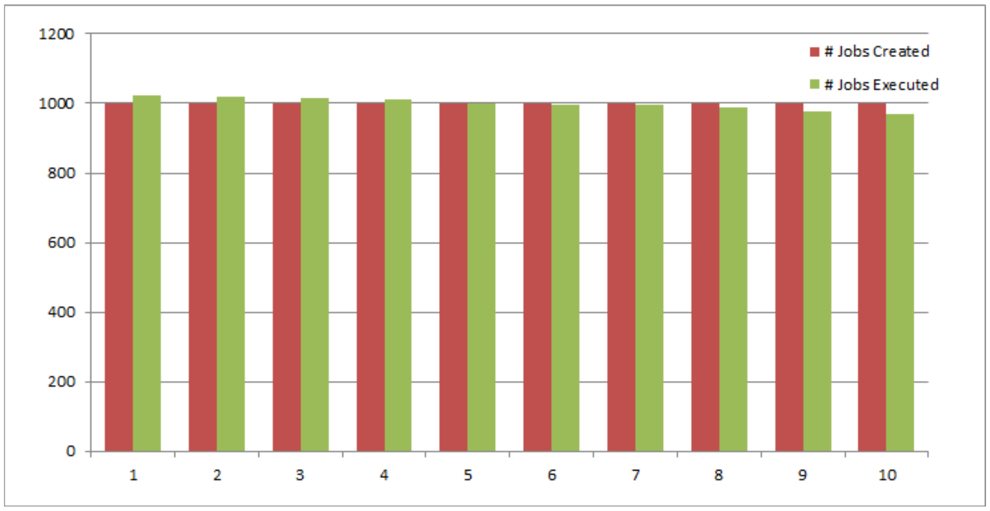

Figure10. Graphic showing the relation between the average number of jobs created and executed at each Cluster.

5.2.2.4 Analysis

As far as the scheduling is concerned we can say that changes in the number of instances running at a certain point in time, either GS or Cluster, have no impact.

Regarding the performance, that we measure by means of execution time, we can see in Figure 11 that the lack of a GS has no impact but the lack of Clusters does. This makes sense because Clusters are responsible for the execution of the jobs which execution times are orders of magnitude higher than the generation and delegation.

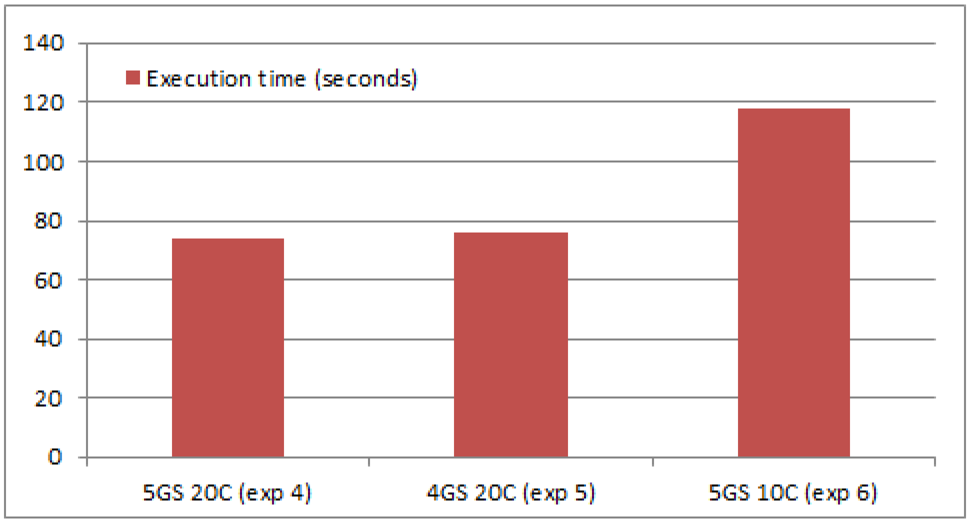

Figure 11. Graphic showing the average execution times of the experiments of this section.

Other thing that we can also see from this set of experiments is that even though the initial jobs are equally distributed the number of jobs executed varies, always decreasing from the first to the last Clusters. This occurs due to the fact that the Clusters were launched manually and in the order of 1 to 20 and thus we consider this difference irrelevant.

5.2.3 Fault-tolerance experiments

To test our system in the fault-tolerance topic we did several experiments where GS's and Clusters were stopped and then restarted to see what was the outcome, taking into account the assumptions described in section 4.1.1.

As expected both GS's and Cluster restarts were successfully carried out without affecting drastically the system, meaning by this that the performance was affected, which was expected, but the outcome in terms of jobs completed, one of our goals, was not. More about how GS and Clusters restarts are carried out was already discussed in section 4.

We have no metrics that are relevant to this set of experiments other than the total number of jobs executed, which was always the expected.

6. Discussion

The revised CAP theorem [2] clearly states that today the main tradeoff is between consistency and availability. In our work we had to deal with this by making specific decisions. In different occasions the role played by availability and consistency is different. In our project it is especially important to keep consistent some information but even though this is done, at some points focus on availability is given. When a Cluster delegates a job it needs to wait for the Grid Scheduler response that confirms the correct delegation of the job. In order to a Grid Scheduler to confirm the delegation it needs to sync with at least one other GS and this is obviously focused on consistency. Although we focus also on availability by allowing the GS to confirm the delegation right after logging with only one of the other GS's that might be working. With this approach we don't focus totally on either availability or consistency but in a compromise between both.

From our experience developing the distributed VGS we believe it to be a good solution for WantDS to implement their grid system. The non-distributed VGS has some constraints related to fault-tolerance, scalability and performance that the distributed version, to some extent, solves.

For a higher number of jobs the system should behave well since the workload put into the entities is the same. Since with more jobs the system will have more messages to be logged something should be done to improve the logging system that is currently residing in memory.

7. Conclusion

Concluded the study on the feasibility of the Distributed Grid Scheduler, we can say that we accomplished an overall good result that demonstrates how well this system can perform comparatively to a non-distributed version. As shown in the previous sections of this report, we have successfully implemented a distributed grid scheduler that is capable of handling a bigger workload in a shorter period of time by equally distributing the jobs among a group of distributed clusters. It is also needed to state that after completing the work, we faced some minor details that we missed during the design of our distributed system, one of the most importants being the shutdown condition of the whole system being based on a predetermined number of jobs given at the start up, which now makes less sense than when we decided to implement it, being the best solution to keep the system running and waiting for a manual shutdown, making it more realistic.

8. References

[1] Tanenbaum, Andrew S., and Maarten Van Steen. "Distributed Systems: Principles and Paradigms." (2006).

[2] Brewer, Eric. "CAP twelve years later: How the." Computer 45.2 (2012): 23-29.

9. Appendix

This report is public and is available with the source-code at the following GitHub repository: https://github.com/cybernabojr/DCS-VGS

We apply for the following bonuses:

- Advanced fault-tolerance (section 4.2);
- Benchmarking (section 4.2);
- Excellent report;
- Open-source source code and public report.

Time spent in the assignment:

- the **total-time** = 252 hours;
- the **think-time** = 25 hours;
- the **dev-time** = 150 hours;
- the **xp-time** = 30 hours;
- the **analysis-time** = 19 hours;
- the **write-time** = 18 hours;
- the **wasted-time** = 10 hours.