Delft University of Technology

IN4391- Distributed Computing Systems

Distributed Virtual Grid Scheduler

Final Report

Authors:

Nikola STAVREVSKI
N.Stavrevski@student.tudelft.nl

Anıl ŞAHİN
A.Sahin-2@student.tudelft.nl

Course Instructor:

Dr. Ir. A. IOSUP
A.Iosup@tudelft.nl

Lab Assistant:

Ir. A. S. ILYUSHKIN
A.S.Ilyushkin@tudelft.nl

Abstract

Utilizing the advantages provided by distributed systems and tackling the challenges in this domain has been a hot topic for the past few decades. Simulation has been a major tool in order to understand the behavior of distributed systems as they are very resource consuming to actually build and experiment upon. This report proposes a solution to the requirements provided by WantDS BV in simulating a Distributed Virtual Grid Scheduler, building upon a previous non distributed version of a Virtual Grid Scheduler. We propose a distributed system which runs on 100 AWS t2.micro instances with simulated jobs, that is scalable, fault tolerant, and consistent and experiment to see how the system achieves these features. It can be observed from the experiments that the distributed system is able to achieve scalability for various workloads, balance loads successfully, can tolerate certain forms of failure and thus yields better performance than a non distributed virtual grid scheduler.

1.Introduction

Being able to enhance the power of multiple heterogeneous computing resources to work together smoothly and coherently as one, to address the diverse needs of multiple users in large scale has been a continuous challenge for the past few decades. However, the study of distributed systems in itself comes with many challenges as it is usually quite difficult to obtain the hardware, the user base and real workloads in order to test and experiment on actual distributed systems. At this point simulation comes as an important tool for understanding the dynamics of designing a distributed system and the characteristics of how it will work due to expenses that actually building and testing on a real system bring.

The system introduced in this report is built in order to demonstrate behavior of a distributed virtual grid scheduler deployed on 100 AWS t2.micro instances with simulated jobs. We build upon a previous structure of Virtual Grid Scheduler, which contains a single Grid Scheduler node that handles coordination of all jobs among clusters. Having a single grid scheduler node for managing all jobs provides some limitations regarding scalability, fault tolerance and performance. Therefore, we propose a system with multiple grid scheduler nodes that aims to provide a scalable, fault tolerant, consistent system that can balance real life workloads and complete given jobs enhancing the advantages that distributed systems enable.

The rest of the report is structured as follows, in section 2 we present a background on the application describing a what a distributed virtual grid scheduler is and detail the system requirements. In section 3 we articulate on the specifics of the system and elaborate on how it is designed. In section 4 the experiments and test conducted with the system are explained and results are presented. Section 5 discusses the results of experiments with regards to tradeoffs related to the system design and is followed by a conclusion in section 6.

2.Background on Application

2.1 Distributed Virtual Grid Scheduler

Distributed Virtual Grid Scheduler (DVGS) demonstrates the behavior of a grid computing system which allows multiple users with diverse needs to submit independent jobs to multiple clusters, which perform job requests in a given period of time. Each cluster in the system has a fixed set of nodes which are responsible for execution of jobs and a resource manager which allocates jobs to available nodes in the cluster. When a resource manager is not able to allocate a request into one of the nodes in its cluster it sends it to a grid scheduler node which balances the load of the jobs across the clusters in the system. The system is composed of multiple grid scheduler nodes who amongst themselves coordinate and find the most suitable cluster for the execution of the jobs.

2.2 System Requirements

2.2.1 Scalability

The system is expected to be able to scale to various amount of jobs that are requested in a certain time frame. The system should continue to operate as desired with a high load of jobs either spread out in a certain period or requested from the system within a short period of time. The system should be able to handle the complexities that arise in increasing the number instances such as Grid Scheduler nodes, Clusters, and number of processing nodes in clusters in order to handle increasing number of job requests.

2.2.2 Fault Tolerance

The system should be able to operate as desired in the case of events such as a cluster that is already working on various jobs goes down. Then the system should be able to restart those clusters and enable that they continue on the jobs where they left off. Also the system should be able to operate in the case of a grid scheduler crash. Other grid schedulers should be able to take the load of the crashed grid scheduler and the system should continue normal operation. When the crashed grid scheduler node is restarted it should be able to take back the load that it handled before. It is assumed that nodes in the clusters will never fail therefore they are not taken into consideration for requirements regarding fault tolerance.

2.2.3 Consistency

The system is expected to log all activities undertaken by clusters and the grid schedulers. It is important that logs kept in multiple monitoring nodes are synchronized and consistent. In case of a failure, the consistency of the logs is vital in process of recovery. For this we apply a causal consistency model as the logical order of logs are more important than their temporal order.

2.2.4 Performance

Due to having multiple grid scheduler nodes and thus potentially a higher number of clusters the system is expected to perform more efficiently and handle execution of jobs faster than that of a system with a single grid scheduler node.

3. System Design

Design of the system is explained in this section. We provide an overview of the distributed system and elaborate on the components and their relations. We detail the implementation of the system features and mechanisms; messaging, scalability, fault tolerance, load balancing and consistency.

3.1 System Overview

The distributed system is designed as to be composed of clusters with multiple fixed number of processor nodes and a resource manager. On top of the clusters are multiple grid scheduler nodes to coordinate and delegate loads to the clusters.

When a user makes a job request it arrives in a Resource Manager of a cluster. The job will either be executed on the same cluster if there are resources, or it will be sent to the Grid Scheduler for execution on a different cluster. Each cluster has a unique identifier and a fixed number of nodes. Jobs also have unique identifiers and fixed duration for the purpose of this lab. The multiple Grid Schedulers communicate with each other and share information about their available resources which are the clusters and their nodes. The response they get is the total number of free slots in each grid scheduler, meaning the sum of the free nodes and places in each of the resource managers' queues. Each Grid Scheduler node, can coordinate to send a job to any cluster via another Grid Scheduler node.

The system is implemented using Java and deployed on AWS t2.micro instances where each entity in the system instantiates a separate AWS instance. With this approach, we did not only simulate a distributed system, but rather put the emphasis on making a fully working and distributed system. A start up script initiates the given image in an instance in order to run as either a Grid Scheduler, Resource Manager or processing Node.

3.2 System Workflow

The client creates a job which has a fixed duration submits the job to the system as a RMI message. Each cluster has a separate interface, so when a job is submitted via that interface, it is expected to execute on that cluster. After the submission, the job is sent to the Resource Manager of that cluster. If there are free nodes, it will immediately start executing the job. Otherwise it will be put in the queue. If the queue is full, then the Resource Manager passes the job to the Grid Scheduler node. The Grid Scheduler node also has a queue of pending jobs. The Grid Scheduler then communicates with other Grid Schedulers and finds the Grid Scheduler with the freest slots for jobs (total free nodes + places in the queue). When it passes the job to the least loaded Grid Scheduler, that Grid Scheduler

sends the job to the corresponding Resource Manager, which will send the job to a free node for execution or put it in the queue and leave the scheduling up to the Resource Manager. All these steps are logged in a Logger instance which keep a general log of the whole system. This Logger is used for retracing steps for reconstruction after a failure and also has a replica in case the actual Logger fails. This workflow is illustrated in Figure 1.

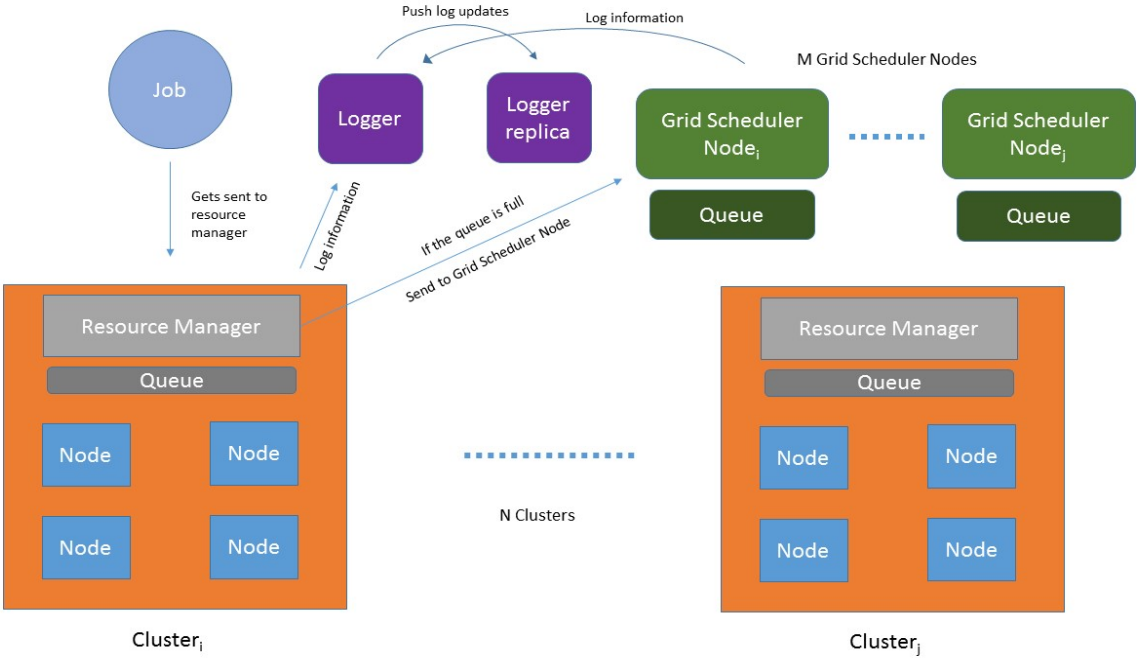

Figure 1: Workflow diagram of the Distributed Virtual Grid Scheduler

3.3 Communication and Messaging

The communication of the system is implemented through Java RMI which provides an effective interface for communication between different entities of the system, which are deployed in different AWS instances. In order to minimize communication overhead clusters only send jobs to one designated grid scheduler and the grid scheduler takes care of sending the job to other available clusters or grid schedulers. The grid scheduler nodes periodically poll the clusters that are connected to them and log the retrieved state. The Resource Managers in a similar fashion poll the processing nodes to get their status (Idle, Busy or Down if they don't respond) and that way we keep track of the load in the system. Since every entity in the system is run on different EC2 instances, we only use the public IP of every instance to uniquely identify it.

3.4 Scalability

The system is designed to handle various amount of workloads with different setup of instances. By increasing the number of instances that is the number of Grid Scheduler nodes, Clusters and nodes in clusters, it is possible handle multiple workloads up to a certain extent. The system is designed to handle at least 10000 job requests with 5 Grid Scheduler nodes and 20 Clusters with 100 processing nodes each. When we tested the system, because of the limits on our free AWS account, we were only allowed to run 100 instances at one time, so the maximum we tested was 4 Grid Scheduler nodes, 15 clusters, and each cluster containing 5 processing nodes, so the experiments are run on this configuration.

3.5 Fault Tolerance

The system is expected to recognize and recover in the cases where a cluster or a grid scheduler might crash at any point during system operation. It is assumed that nodes in the clusters do not fail. We also assume that there are at least two grid scheduler nodes running for normal operation, and that at least one grid scheduler node is alive

during a recovery process and that if no grid scheduler is alive the system will not be able to converge to normal operation. We further explain below how failure in Grid Scheduler nodes and Clusters is handled.

3.5.1 Grid Scheduler Failure

The crash of a grid scheduler is handled in a way that, the systems tries to ensure that there are at least two grid scheduler nodes running during the operation of the system, and that both of these grid schedulers receive activity logs of the jobs in each of the clusters from the Logger instances. In case that one of the grid scheduler nodes fail, then the other grid scheduler node can continue system operation normally and the system tries to restart the grid scheduler node that had failed. Also the grid schedulers communicate among themselves, the clusters which they coordinate therefore, the running grid scheduler node takes over the coordination the clusters that were previously supervised by the grid scheduler node that crashed. When the failed grid scheduler node is restarted a load is passed back through use of the logs, from the only grid scheduler node that was left and they both continue to coordinate system activities.

3.5.2 Cluster Failure

Once a job is assigned to a Node in a Cluster it is assumed that it will be completed by that given Node. Therefore, in the case that a Cluster fails, it is expected that it will restart and continue with the jobs at hand. All the activities of the Clusters are continuously logged, therefore the recovery of job execution by the failed Cluster is enabled through retrieving the job information from these logs from the Logger instance.

3.6 Load Balancing

The system is designed so that it should be able to handle increasing loads, keep the load of each cluster similar and be able to always take new requests. In cases that certain clusters receive significantly more requests than others, the system is expected to balance those requests towards other clusters.

The implementation of the load balancing algorithm is the following. Each resource manager has a queue with a length of 32. When jobs arrive in the resource manager, it tries to find an idle node to submit the job. If it doesn't find one, it puts in the queue. The queue is then polled periodically to offload a job to a node that has become idle. If a job arrives to a resource manager, and the queue is full, then the job gets sent to the resource manager. In the resource manager there is a hash map which contains all the resource managers connected to that grid scheduler, and the corresponding free slots they have. It then sorts them and sends the job to the resource manager with the freest slots. If the queue of the grid scheduler is full or there is not a resource manager under its' management with a free slot, the grid scheduler offloads the job to another grid scheduler, again in the same fashion, it sorts the free slots of the other grid schedulers and sends it to the one with the freest slots. With this mechanism we achieve load balancing in the system, with some caveats that we analyze in the experiments section.

3.7 Consistency

All system activities are logged on the Logger instance by all entities in the system. In case of failure of instances be it grid scheduler or cluster, as explained in section 3.5 the information from these logs are used in order to recover the system from these failures. We also provide a replica of the logging node to protect against crashes of the logging node. We therefore apply a causal consistency model for the logging of system activities.

4.Experiments

Experiments are carried out in order to test how the system behaves in certain situations and to assess whether it complies to the requirements provided by WantDS BV. Experiments are conducted for the scalability, fault tolerance, load balancing and consistency features of the system.

4.1 Setup

The experiments were conducted by making use of instances deployed from AWS. Every mentioned instance in the experiments which are the Grid Scheduler Nodes, Clusters with 1 resource manager and 5 processing nodes and 2 activity logger nodes are actual AWS t2.micro instances. Throughout our experiments we have had access to 100 free instances, from which we actually exceeded our limit of free hours, in which we deployed our system.

Therefore, our implementation and testing is also closer to a real system except that we have deployed simulated jobs which each take 20 seconds to run. Through most of out experiments we have used a batch of jobs we send at once to one or more resource manager of a cluster. We have one experiment where we send jobs with random intervals which we discuss further.

4.2 Scalability and Performance

For scalability we want to measure how the system performs with various loads and different computational setups. For this purpose, we test the system with different number of Grid Scheduler nodes and different number of clusters, issuing 1000, 5000, and 10000 jobs respectively for each setup. Our experiments have shown that our system is able to handle deployment of at least 10000 jobs and that the results for 1000, 5000 and 10000 jobs show similar tendencies therefore, we will mostly discuss in detail the experiments with 5000 jobs due to the relatively longer time that it takes in order to execute experiments with 10000 jobs.

We first test the system with a setup of 4 Grid Scheduler Nodes, 15 Clusters with 5 processing nodes each as shown in Figure 2. For our second experiment we reduce the number of Grid Schedulers to 3 with same number of Clusters and processing nodes in order to be able to observe the variance of performance with change of number of Grid Scheduler Nodes as shown in Figure 3. Lastly we experiment with 4 Grid Scheduler Nodes and 10 Clusters with 5 processing nodes, in order to see how the number of Clusters will affect system performance as shown in Figure 4.

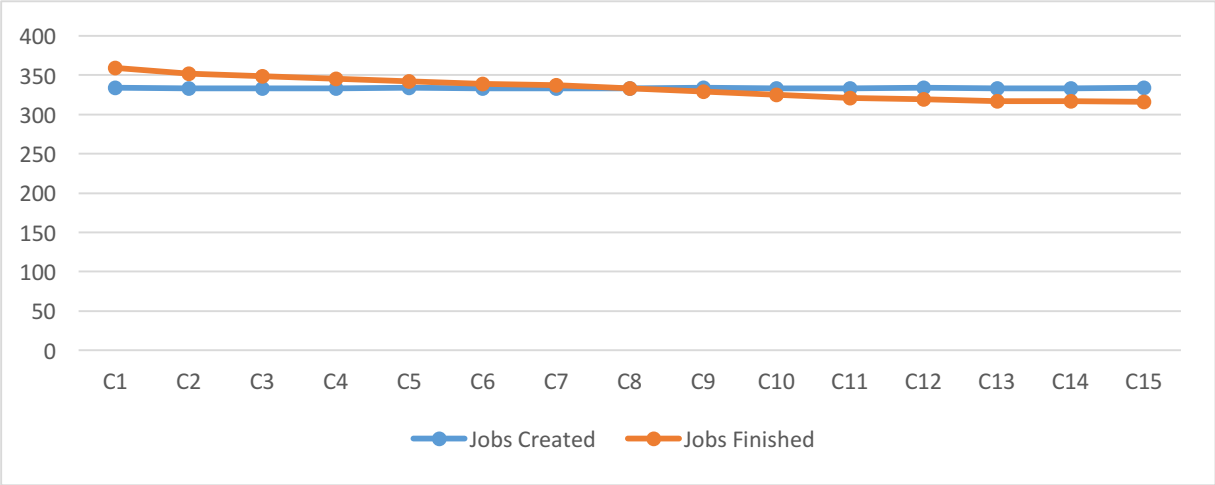

Figure 2: Number of jobs sent to and completed by each cluster with a setup of 4 Grid Scheduler nodes, 15 Clusters. This execution was completed in1391 seconds.

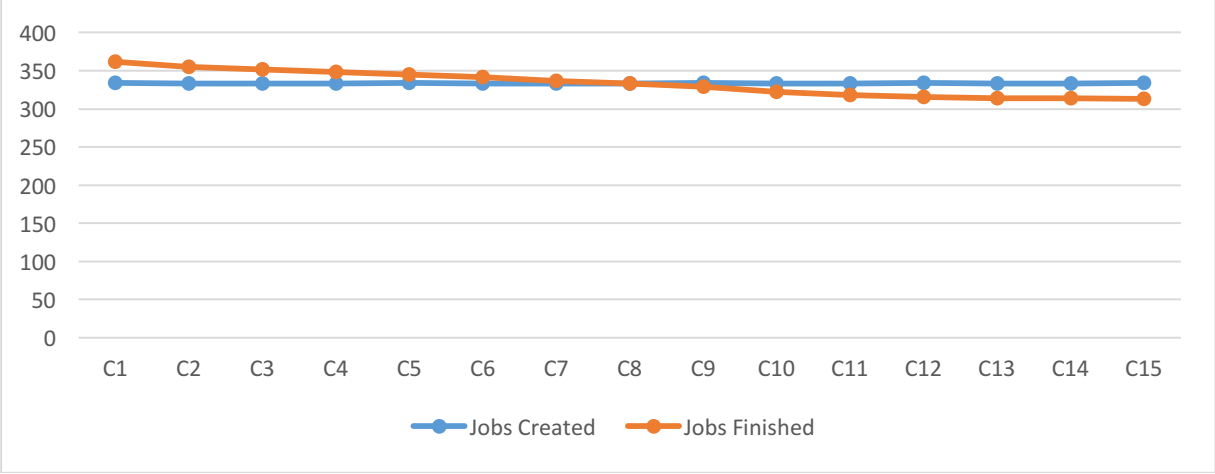

Figure 3: Number of jobs sent to and completed by each cluster with a setup of 3 Grid Scheduler nodes, 15 Clusters. This execution was completed in 1398 seconds.

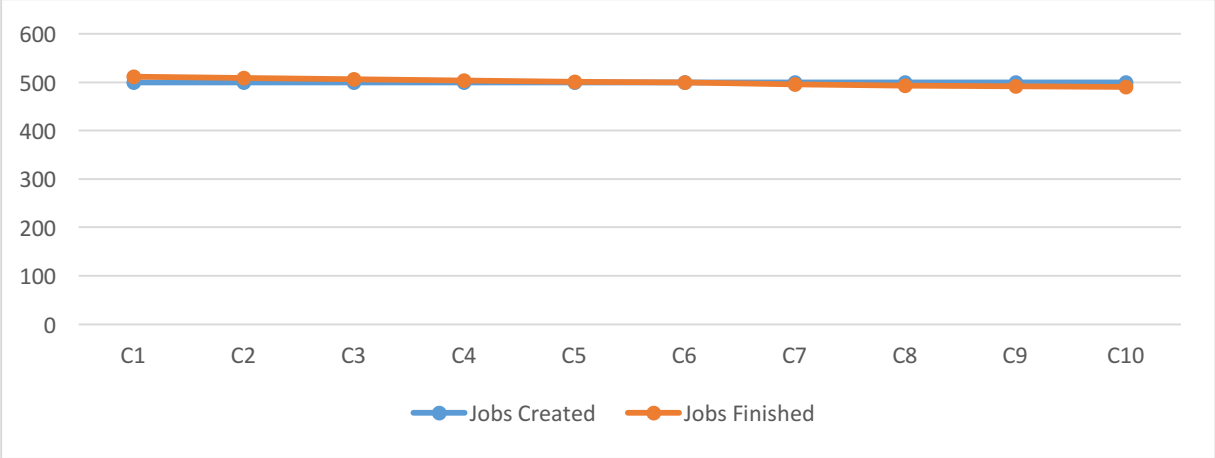

Figure 4: Number of jobs sent to and completed by each cluster with a setup of 5 Grid Scheduler nodes, 10 Clusters. This execution was completed in 1713 seconds.

4.3 Fault tolerance

Fault tolerance was tested through shutting down certain number of instances in the system and observing whether the system is able to complete all submitted jobs. All tests were conducted with a setup of 4 Grid Scheduler nodes and 15 Clusters with 5 processing nodes and with a submission of 10000 jobs.

It is observed that the system is able to recover when Grid Scheduler nodes are shut down one by one until there is only 1 Grid Scheduler node left. In this case even though the system is still able to operate the until the shut down Grid Schedulers are restarted there occurs a larger imbalance between the jobs that are handled by the clusters. When the other Grid Scheduler nodes are restarted the system is observed to converge back to a more load balanced state.

In the case where a Cluster is shut down then the jobs assigned to that Cluster are halted for the period until it restarts. When the cluster restarts it is observed that it can continue execution of the jobs it previously had through the activity logs that give the information about the jobs and their progress. However, if that given cluster does not restart then the designated jobs for that cluster are not able to be completed and therefore the system is not able to recover and successfully complete all jobs.

4.4 Load Balancing

For testing the load balancing behavior of the distributed system we try to create various imbalance situations and observe the distribution of job completion amongst each cluster in the aftermath of the imbalance situation. We test the system with 4 Grid Scheduler nodes and 15 Clusters with 5 processing nodes each and submit 5000 jobs. We also compare the performance of the system with the imbalance situations.

In the initial experiment we create an imbalance situation where we submit 5 times the amount of jobs to 1 cluster than all other 14 clusters as shown in Figure 5.

We then increase the imbalance situation to a ratio of 15 to 1 as shown in Figure 6.

Finally, we send all jobs to only one cluster and observe how the system balances this edge situation to other clusters, this can be seen in Figure 7.

We also try to measure how the imbalance situations affect performance through completion times in seconds of all jobs.

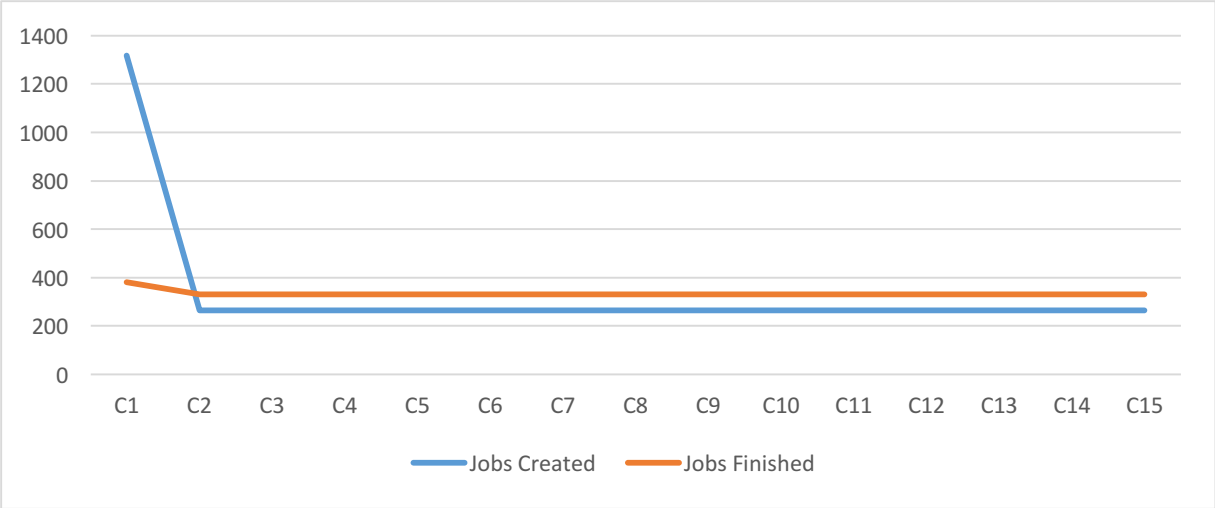

Figure 5: Imbalance situation with the ratio of 5:1 where 1318 jobs are sent to one cluster and 263 jobs are sent to all other clusters. The completion time of all jobs were 1402 seconds.

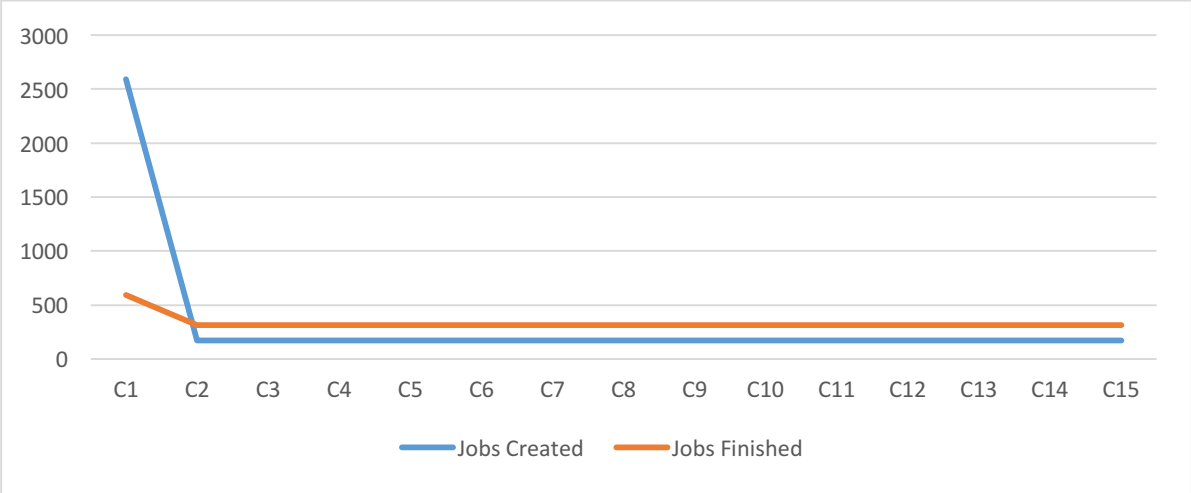

Figure 6: Imbalance situation with the ratio of 15:1 where 2592 jobs are sent to one cluster and 172 jobs are sent to all other clusters. The completion time of all jobs were 1477 seconds

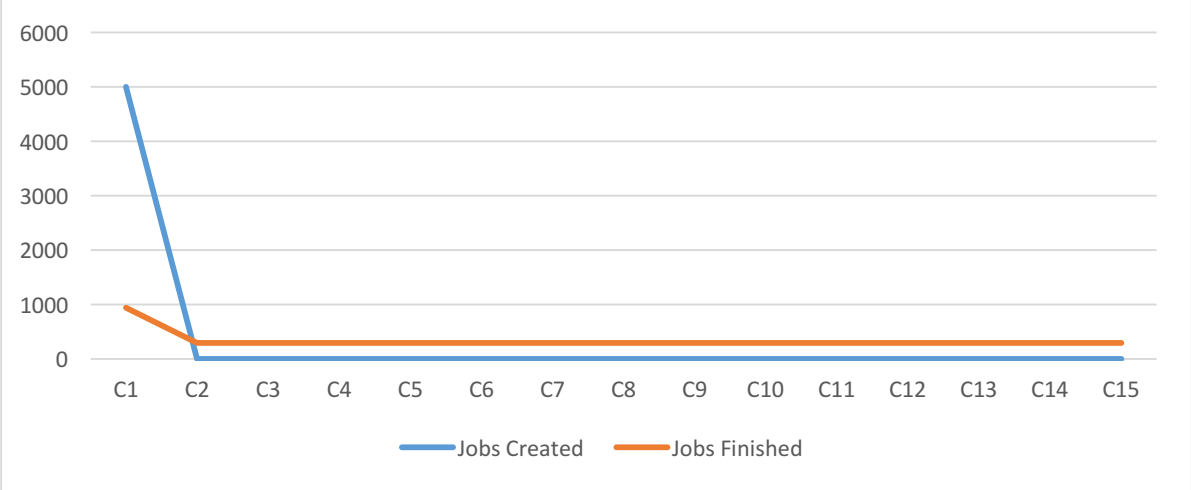

Figure 7: Imbalance situation with 5000 jobs where all jobs are sent to one cluster none to other clusters. The completion time of all jobs were 1501 seconds

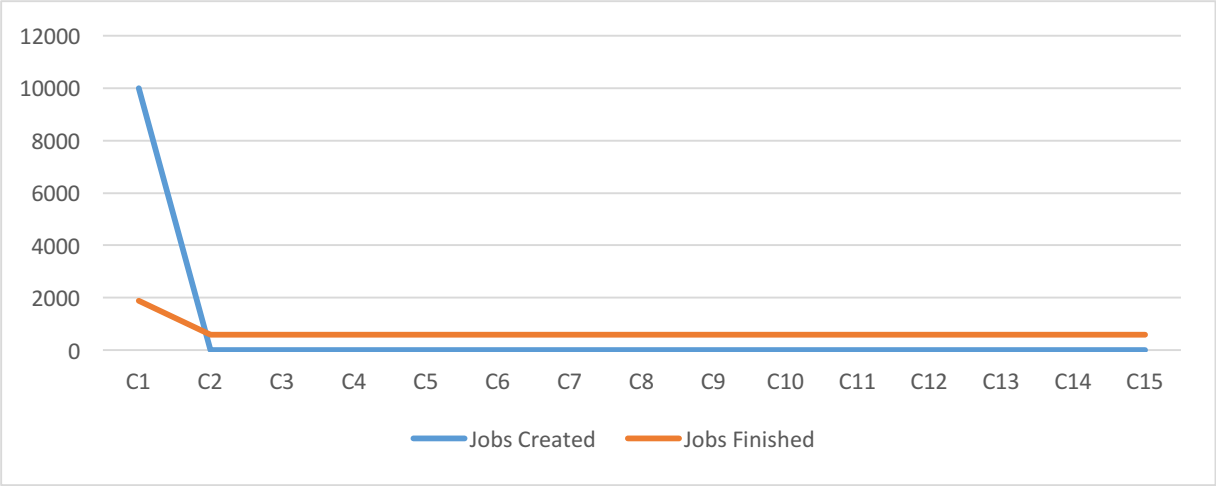

Figure 8: Imbalance situation with 10000 jobs where all jobs are sent to one cluster none to other clusters. The completion time of all jobs were 2963 seconds

4.5 Consistency

The consistency of the distributed system was observed through analyzing the separate activity logs to see if they showed the same casual state for the activities of the system. In runs without any failure of instances we were able to observe that the log data was causally consistent in each of the separate logs in the system.

In case of failures such as shutting down of a Grid Scheduler Node or a Cluster, we were able to observe that the system was able to recover from the cases explained in more detail in section 4.3 through the use of the activity logs. In the case where there are only 2 Grid Scheduler nodes running the logs were still observed as consistent, and when the other Grid Schedulers were restarted they were also supplied with these logs and continue logging in a consistent manner. Even in the case when only 1 Grid Scheduler node was left in the system and run for some time in this setup, when another Grid Scheduler restarted the logs were again observed to be transferred successfully and continued to be kept consistently.

4.6 Non uniform job arrival

We have conducted one experiment not sending a batch of jobs instantly but sending 1000 jobs with a random interval between 0 to 5 seconds each job. This was an interesting experiment which actually took 2694 seconds (around 45 minutes) to complete all the jobs. In this experiment we observed that the system did not load balance because the 5 nodes and the 32 spaces in the queue of the resource manager of one cluster was enough for the way the system is implemented to complete the job. This was due to the fact that the grid scheduler never saw the given cluster as fully loaded so it did not send any job to any other cluster let alone grid scheduler. This created a bottleneck in the system where all the jobs were completed by the 5 processor nodes in one cluster.

5.Discussion

The results of our experiments show the aspects in which the proposed distributed system is able to achieve the features of scalability, fault tolerance, load balancing and consistency.

In terms of scalability, from observing that the system can handle workloads ranging from 1000 jobs to 10000 jobs we can say that it can scale up to a certain point and perform in a desired way. If it was desired to test the system with a job number greater than 10000, such as with 10^5 or 10^6 jobs, than this would bring certain complexities especially in terms of communication. From the experiments with various changes to the number of instances in the system it can be seen even though the change in number of Grid Scheduler nodes may not have a significant effect in terms of coordination and in performance, in a scale much higher, the communication overhead is expected to lead to degradation of performance and limit coordination abilities of Grid Scheduler nodes. Also it can be observed from the experiments that reducing the number of Clusters can cause a significant performance reduction, which again in the case of higher magnitude of jobs the number of clusters would need to be increased creating more complexity. Increasing the number of processing nodes in Clusters is also solution however it also

has its limits regarding the coordination abilities of the resource managers and also in cost regarding obtaining these resources.

From experiments that load one cluster with a significantly higher number of nodes we can see that the system is still able to distribute this amount to all other clusters through making use of the coordination abilities of having multiple Grid Scheduler nodes. In a larger scale of jobs to be submitted with a diverse number of request to each cluster, this load balancing ability through multiple Grid Scheduler nodes will also contribute to the scalability of the Distributed Virtual Grid Scheduler. We also observe when comparing the experiment where all jobs are sent only to one cluster and the experiment where all jobs are distributed evenly across clusters that even though there is a slight loss of performance when all jobs are sent to one cluster, this does not cause a major overall degradation in system performance.

Utilization of multiple Grid Scheduler nodes has also enabled the distributed monitoring of systems activities and when this is achieved in a consistent way such as exemplified in our proposed system and observed in its experiments enables a fault tolerance mechanism that would not have been possible through only a single Grid Scheduler node.

With the advantages discussed here and showcased by our experiments we suggest that the Distributed Virtual Grid Scheduler requested by WantDS BV provides better performance, scalability, fault tolerance and load balancing capabilities compared to a non distributed Virtual Grid Scheduler though also bearing many complexities in implementing them.

6.Conclusion

Our proposal for a Distributed Virtual Grid Scheduler demonstrates the feasibility of such a system through a simulation environment. Through implementing and testing on a distributed system in which multiple Clusters coordinated by multiple Grid Schedulers can handle multiple independent job request we have shown that such a system is able to scale up to a certain extent, can tolerate failures of its instances, consistently monitor the activities of the system and balance the load of jobs evenly among its clusters.

We conclude, from the results obtained from this simulation that a distributed system for building a Virtual Grid Scheduler poses as a better alternative to a non distributed Virtual Grid Scheduler.

Appendix A: Time Allocation on Assignment

Total-time: 137 hours
Think-time: 12 hours
Dev-time: 74 hours
Xp-time: 28 hours
Analysis-time: 5 hours
Write-time: 11 hours
Wasted-Time: 7 hours

Appendix B: Github repository link

The code for the project is located at: https://github.com/nstavrevski/DistributedComputingLargeLab