IN4391: Large lab exercise A: Distributed Virtual Grid Simulator

Ir. Yong Guo
Delft University of Technology
Yong.Guo@tudelft.nl

Dr. ir. Alexandru Iosup
Delft University of Technology
A.Iosup@tudelft.nl

Michael de Jong
Delft University of Technology
M.deJong-2@student.tudelft.nl
1314793

March 20, 2013

Abstract

Clusters or grids of computers allow their users to run massively parallel jobs. One interesting and active area of research is to explore the possibilities of joining multiple grids together for better reliability and throughput. These grids are expensive to build and operate. So in order to investigate how one can couple multiple grids together is best simulated in a smaller environment. For this purpose I present a system called Distributed Virtual Grid Simulator (*DVGS*) which simulates an environment where multiple grids are managed by a group of distributed schedulers.

DVGS is comprised of an arbitrary number of resource managers: nodes which represent a grid capable of running jobs of a certain duration, and which have a certain maximum capacity of simultaneous running jobs. *DVGS* employs at least one scheduler which is capable of redistributing jobs to other resource managers if a resource manager is fully utilized and cannot accept an offered job. Multiple schedulers can be run in tandem to provide fail-over in case the master scheduler goes down.

Simulations of *DVGS* show that it allows single grids to ‘accept‘ more jobs than they could do without *DVGS*. This is due to the fact that *DVGS* allows grids to offload their exceeding load to other grids with less utilization.

Introduction

1 Problem statement

Clusters and grids are comprised of many tens to thousands of computers which typically use a network to communicate with each other. These computers can execute many tasks or jobs of varying duration and load in parallel. Systems like this are particularly interesting when a problem can be reduced to many smaller problems which is each solvable by a single computer. Solving all required subproblems with a single machine might take very long so grids can be used to distribute these jobs across multiple machine. This allows their users to solve these large and complex problems in a relatively short timespan.

Typically a grid is managed by a single node called a resource manager. This node accepts incoming job offers and distributes them to idle nodes in the grid. Resource managers are a bottleneck for each grid and a single point of failures (*SPOF*). From a user's perspective this means that if the resource manager crashes, other nodes in the same the grid will not be able to process their jobs.

Another issue is utilization. Every grid has a certain amount of resources which allow it to run up to a certain number of simultaneous jobs. If all the resources of a grid are fully occupied with jobs, the resource manager is faced with a problem. It can either decline the offer, or store it in a queue. In the latter case the resource manager has a serious problem if it crashes because it will loose the jobs stored in the queue.

2 Solutions

There have been several attempts to link multiple grids together. But the most notable system known to me, is a system called Condor Flocking [1] is built on top of an existing grid middleware solution for single grids. In this system gateways impersonate a resource within a grid, and simply relay the jobs to another Condor grid. This was done to ensure that the original Condor middleware did not have to be modified.

DVGS attempts to provide a fully distributed, very fault-tolerant layer between the grids allowing them to offload excessive load to other less utilized grids in the network. More details on the design and implementation can be found in section **??**.

3 Structure of this article

The remainder of this article is structured as followed:

Background on Application

1 Previous work

The original Virtual Grid Scheduler (*VGS*) system on which *DVGS* is based, is a Java application which can simulate a grid handling jobs. Every offered job has a set duration. When a resource manager receives a job, it starts a timer with the duration of the job and decrements the available capacity with 1. When the timer has elapsed, it increments the

[1] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A Worldwide Flock of Condors: Load Sharing among Workstation Clusters, Future Generation Computer Systems, Vol. 12, pp. 53-65, 1996.

available capacity with 1, significy that the job has ended. This simple mechanism allows us to run very large simulations of running jobs on grids. *DVGS* employs the same mechanism to simulate grids.

2 Requirements

- **Operation** Jobs can be submitted to any resource manager in the distributed system. A resource manager can put submitted jobs in a local job queue or pass them on to a scheduler node. The scheduler node can in turn send it to a resource manager, which has to accept it and put the job in its local queue.
- **Fault-tolerance** The distributed system must be fault tolerant. This means that the system should be able to cope with a crashing node. To facilitate this, at least two schedulers must keep track of the state of the entire distributed system. In case one goes down the system can continue to operate.
- **Scalability** The system should be able to simulate a system consisting of 20 resource managers representing grids with a maximum capacity of 1,000 concurrent jobs, and 5 schedulers.

System Design

1 Messaging

One of the suggested means of messaging between nodes in the distributed system was Java's Remote Method Invocation (RMI). Each node can call a remote method with a Message object and a Address object, describing the sender. The sender can either wait for the method call to complete, which in essence signifies an acknowledgement, or ignore the result and continue onwards right after sending the Message.

2 Topology

DVGS is built with a high degree of fault-tolerance in mind. I wanted the system to continue working as long as their was at least one scheduler operational. Additionally I wanted the system to recognize new schedulers and resource managers trying to join the distributed system. In order to facilitate this, every node in the distributed system periodically scans a predefined range of IP addresses and ports. If it can establish a connection with an unknown node, it will send its currently known Topology. This object consists of all known nodes and their roles (resource manager / scheduler) in the distributed system. When an object receives a Topology object it will merge the data from that Topology with its own, and try to connect to nodes it previously did not know about. This simple mechanism allows all nodes to quickly discover each other and exchange information about what role they play in the network. Similarly if a node crashes, it will eventually (during the next scan) or immediately (if a Message could not be delivered to it) be dropped from the Topology object, which is then broadcast to the rest of the network.

3 Chain of command

Schedulers are responsible for redirecting jobs to under utilized grids, each managed by a resource manager. In *DVGS* schedulers are ordered by age. The oldest scheduler acts as the master scheduler and is responsible for assigning jobs to grids. In case the master scheduler crashes, it is removed from the Topology and the then oldest scheduler becomes the master scheduler. In this linear ordering each scheduler has a backup with the exception of the youngest scheduler. If a new scheduler is detected using the mechanism from the previous subsection, it is automatically added to the end of the scheduler ordering.

4 Replication & Fault-tolerance

Experimental Results

(recommended size: 1.5 pages) a. Experimental setup: describe the working environments (DAS, Amazon EC2, etc.), the general workload and monitoring tools and libraries, other tools and libraries you have used to implement and deploy your system, other tools and libraries used to conduct your experiments. b. Experiments: describe the experiments you have conducted to analyze each system feature, such as consistency, scalability, fault-tolerance, and performance. Analyze the results obtained for each system feature. Use one sub-section per experiment (or feature). In the analysis, also report: i. Service metrics of the experiment, such as runtime and response time of the service, etc. ii. (optional) Usage metrics and costs of the experiment.

Discussion

(recommended size: 1 page): summarize the main findings of your work and discuss the tradeoffs inherent in the design of the VGS system. Should WantDS use a distributed system to implement the VGS system? Try to extrapolate from the results reported in Section 6.b for system workloads that are orders of magnitude higher than what you have tried in real-world experiments.

Conclusion

Appendix A

Time sheets (see Section E)