A Model for Parallel Job Scheduling on Dynamic Computer Grids

Alfredo Goldman Carlos Queiroz
Department of Computer Science - University of São Paulo
{gold,carlosq}@ime.usp.br

Abstract

This work presents a model that allows the execution of parallel applications in a grid environment. Our main focus is on how to share the idle cycles of clusters and computers to execute parallel applications. We introduce a new model with the notions of locality and adaptability. The locality is used for job allocation, and for job migration. The adaptability provides a simple mechanism to allow clusters to join or leave a grid. We also propose a middleware architecture to implement the model.

1 Introduction

A great effort has been done in the construction of huge grids trough the interconnection of several smaller clusters [1]. The main advantage for cluster owners is to have access to much more processing power rather than using only their clusters stand alone. However, the cluster is no longer exclusive. There should be special restrictions in order to guarantee priorities, on each participating cluster, for jobs submitted by the cluster administrator. That is, for each cluster owner, the grid environment should use only their idle cycles. Similar ideas have been developed for networks of personal computers (Condor [2], InteGrade [3]).

There is a tradeoff when participating in a grid environment: in one side the access to more computational resources, and in the other side there is no exclusive access for the owners of each cluster. In a grid scenario we can imagine two main types of job submission: from inside by the local administrator, or from outside by the grid scheduler. One of the main difficulties in this scenario is how to balance both types in a transparent way. Given some simple hypothesis, we derive a new model for this scenario.

In this paper we show some very simple ways to share resources. The main idea is to focus on the effort on how to provide access to idle cycles. There are many related works on grid computing, however we are mainly interested on doing parallel processing on grids. We focus on two aspects: network heterogeneity (as in [4]) and adaptability (as in [5]).

The text is organized as follows. First, we present the adopted model and the basic algorithms. Then, we show how it can be easily implemented. We conclude with some ideas for future work. In our work we intend to use the advantages of a multi-cluster environment with some constraints to create a adaptative environment. In this environment, the cluster administrators should be able to dynamically leave or join the grid.

2 The model

The grid is represented by a graph; the vertices correspond to the processing nodes, which can be clusters of computers, parallel machines, or even single computers. To characterize the different number of processing units on each cluster to each vertex i we associate its cardinality n_i. For stand alone computers we have $n_i = 1$. To simplify the analysis we assume that all computers are equal. We also denote single computers as clusters of size one.

There are edges interconnecting the neighboring vertices if there is a fast link between them. Here, fast means similar latency and bandwidth as inside a dedicated cluster. However, as it is not a dedicated link, we suppose that it has worse performance than a link in a cluster. The motivation for this is to consider that not all clusters are in the same domain.

For each vertex in the graph there is a local scheduler which is responsible for the submission of local jobs. This local scheduler periodically informs the current load of the cluster to the grid scheduler, and it is also responsible for allocating grid jobs. In our model, the load is simply the number of available processors, so this number is an integer greater or equal to zero.

There are two different job priorities, one for the local jobs and other for the grid jobs. The jobs submitted by the grid scheduler have smaller priority, and their tasks can be preempted. The jobs submitted by the local administrator should not be movable.

We also assume the existence of a mechanism to migrate a parallel job, or part of it, from a cluster to another one, this can be done using languages like Charm [6], or in the case of a parallel job with communicating tasks by leaving in the original nodes a *proxy* which informs the new task location. If a task from a grid job finds no new host, the whole job is suspended. There can be a waiting time to start new local jobs, as the tasks from the grid cannot be moved while they are communicating. If a programming model like BSP [7] with small super steps is used, this waiting time should not be large. To simplify the problem we assume a fixed cost proportional to the number of tasks to be transfered. We also provide a distributed mechanism for migration.

3 Algorithm

The clusters are interconnected and the grid scheduler takes care of some jobs. The jobs submitted by the grid scheduler can be executed in several different sites. However, the different sites have to be adjacent, two-by-two, and an overhead has to be considered. We consider only a fixed overhead for multi-site execution, even if there are more than two sites involved. This is reasonable on communication patterns like complete exchange where the slowest link should be the origin of the larger overhead [8]. With a more complex allocation scheme this is no longer true. However, to find a good embedding of a parallel application is a difficult problem.

As the given graph can have a large number of vertices, a centralized control and monitoring of all local clusters activities is difficult. So, we propose the following mechanism on each cluster:

1) Each cluster scheduler maintains its own queue of submitted jobs. Each job has its processing requirements. Associated to each job we have also its estimated processing time;
2) For each local scheduler, the first job in the queue is analyzed. If this job can be executed on the cluster, the scheduler allocates the job. Otherwise, there are three possibilities: (1) the job can be executed within time Δ_t, i. e., one or more of the jobs in execution will finish soon, (2) a grid job running on the cluster can migrate freeing the necessary processors, and (3) the job is sent to the grid scheduler;
3) To migrate a grid job, all of its tasks have to be suspended. Then, the local scheduler will check the availability of processors on its neighbors (the communication is done without the participation of the grid scheduler). If there is a neighbor with enough free processors, the job tasks are transfered. Otherwise the grid job is stopped and sent back to the grid scheduler;
4) A grid job or a regular job can be stopped if its current execution time is significantly larger than the predicted time;
5) Periodically, the scheduler sends its status to the grid scheduler.

The grid scheduler has two priority queues, one for new submissions (NJ), and another for submissions in progress, CJ (that were stopped by migration).

1) If the submission in progress queue (CJ) is not empty, the grid scheduler finds (if it exists) a set of adjacent clusters to continue the execution of the first job in the queue. Otherwise, the grid scheduler finds a candidate job on the new submissions queue (NJ);
2) If all the chosen processors on the clusters are available, the job is sent to them. Otherwise, the previous item is repeated until a candidate is found;
3) Periodically, the grid scheduler receives the status from local schedulers.

4 An Implementation Proposal

The idea is to propose a simple add-on middleware to provide grid facilities to interconnect independent clusters. For each cluster, we only need a small daemon which communicates with the grid scheduler, which should be able to handle several requests at the same time. This daemon contains three components: neighborhood information (NI), a monitor, and a scheduler (see Figure 1). The daemons should inform periodically the availability of processors in each cluster, moreover, the local schedulers can also send jobs to be processed by the grid scheduler. On the other side, the grid scheduler can also check the current availability of a group of clusters (by the grid conditions component) and send tasks to the local schedulers (by the allocator, according to the availability). Finally, there should be a mechanism which allows the tasks of a parallel application to be migrated (this is another line of research in Integrade [3]).

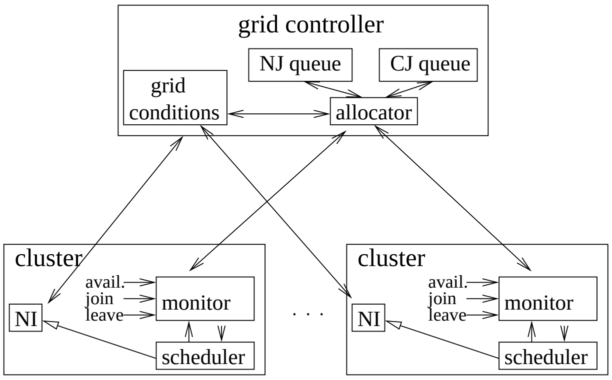

Figure 1: Proposal for the Grid Architecture.

The grid scheduler is more complex. It has to maintain the list of available processors in neighboring clusters; this can be done using some preprocessing techniques in order to find the cliques (complete sub-graphs) on the graph formed by all clusters.

Another characteristic of the model is the possibility to join and leave the grid dynamically. For joining the grid, it is only necessary to run the daemon, inform the processors availability, and provide which are the adjacent clusters. On the other side, to leave the grid, the cluster has to send a signal to the monitor which sends signals to the grid applications running on it, and wait for their migration. For the migration, the neighborhood information (NI) is consulted in order to find available processors on the adjacent clusters. If the number of processors is insufficient, the grid job is stopped and sent back to the grid allocator. After the migration, only a lightweight daemon acting as a proxy will stay alive until the end of the grid application.

For a less flexible, but still dynamic grid environment, a lease policy can be used. In this case, the cluster monitor is responsible for the lease policy. Instead of allowing clusters to leave the grid at any moment, when joining the grid the cluster administrator can also provide its time availability. This time availability can be renewed if the cluster can stay in the grid after the end of the given time period. A proposal of an architecture can be seen on Figure 1.

5 Conclusion

In this paper we studied the proposal of a new model to allow the execution of parallel applications in a dynamic grid environment. The model provides parameters to measure the overhead time to share a job among different clusters, and to consider parallel tasks migration cost. We also propose an architecture for the implementation.

To continue this work we intend to explore the different possibilities of the model through more elaborated simulations. Initial simulations results can be found on the extended version of this document in [3]. We also intend to study more mechanisms for migrating tasks of a parallel job, and to implement the proposed architecture in our computing environment.

References

[1] *The grid forum.* `www.gridforum.org/`.

[2] *Condor.* `www.cs.wisc.edu/condor/`.

[3] *Integrade.* `gsd.ime.usp.br/integrade/`.

[4] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids. In *6th PASA*, pages 169–179. VDE-Verlag, 2002.

[5] L. V. Kal, S. Kumar, and J. DeSouza. A malleable-job system for timeshared parallel machines. In *2nd CCGRID*, pages 230–237, May 2002.

[6] *Charm.* `charm.cs.uiuc.edu/`.

[7] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[8] A. Goldman. Scalable algorithms for complete exchange on multi-cluster networks. In *2nd CCGRID*, pages 286–287, May 2002.