Analytical Model for Performance Evaluation of Load Balancing Algorithm for Grid Computing

Abderezak Touzene, Hussein Al Maqbali

Department of Computer Science Sultan Qaboos University
P.O. Box 36, Al-Khod 123, Sultanate of Oman
{touzene, m010537@squ.edu.om}

***Abstract* — In this paper we propose a new load balancing algorithm for the grid computing service. The proposed load balancing is based on the CPU speed of the workers in the grid system. We developed an analytical model and a simulation model using NS2 to evaluate the performance of our load balancing algorithm. Our simulation results validate our anlytical model and in the same time it shows asymptotically optimal behaviour of our load balancing algorithms.**

***Index Terms* — Grid computing, Load balancing, Resource management, Performance evaluation, Queuing theory, Simulation models.**

I. Introduction

A definition for grid computing is provided in [1][2][3] where the following keywords are used summarizing the key features of grid computing: distributed resources, resource sharing, transparent remote access, and infinite storage and computing power. There are many research problems in grid computing. In Condor [4][5], Grid Computing Service (GCS) is based on cycle-scavenging strategy that uses the idle workstation model. In Condor the tasks migrate when the owner of the machine starts using it. In our proposed GCS, task allocation is based on the processing capacities of the workstations participating in the grid. In Condor there is no load balancing. Tasks distribution is based on a Matchmaker module: Each resource advertises its properties and each task advertises its requirement and then the Matchmaker performs the matching and ranks them. The resource with the highest rank is selected. In this paper we present a dynamic, distributed load balancing algorithm in a Grid environment. Our algorithm can also be classified as system-level scheme, i.e. [6], tends to maximize tasks throughput or the overall utilization rate of the grid machines. We focus on steady-state mode, where the number of tasks submitted to the grid is sufficiently large and the arrival rate of tasks does not exceed the grid overall computing power capacity. As in [7], steady-state mode will help us deriving optimality for our proposed load balancing algorithm. The class of problems we address is: computation-intensive and totally independent tasks with no communication between them. Our analytical model is based on queuing theory and we are interested to compute the average response time of grid tasks. This paper is organized as follows: Section II discusses related works in the literature. Section III presents our grid computing architecture. Section IV presents the load balancing algorithm and the analytical queuing model. In section V, we present our performance evaluation using NS2 simulator for the grid computing service. Section VI concludes the paper.

II. Related Works

Load balancing has been discussed in traditional distributed systems literature for more than two decades [8] [9] [10]. It is more difficult to achieve load balancing in Grid systems than in traditional distributed computing environments because of the heterogeneity and the dynamic nature of the grid. Many papers have been published recently to address this problem. Most of the studies present only centralized schemes [11] [12]. Some of the proposed algorithms are extensions of load balancing algorithms for traditional distributed systems. Many proposed methods suffer from significant deficiencies, such as lack of scalability in the centralized approaches proposed in [13]. A triggering policy based on the endurance of a node reflected by its current queue length is used in [13]. The authors tried to include the communication latency between two nodes during the triggering processes on their model, but did not include the cost of task transfer. In our previous work [14], we proposed a ring topology for the Grid Agent Managers GAM (responsible for managing a dynamic pool of workers) and the load balancing algorithm was based on the real load of the workers. In this paper we consider a hierarchical structure of GAMs rather than a ring topology to improve the scalability of the grid system. The proposed adaptive task allocation algorithm automatically regulates the steady-state flow of tasks directed to a given Grid Agent Manager. In [15], the authors consider hierarchical tree structure for grid computing services similar to ours. However, they did not provide any task allocation procedure. Their resource management strategy is based on a periodic collection of resource information by a central entity, which might be communication consuming. In our algorithm, resource information is done only when it is needed, as an example a new workstation joins the grid system, it publishes its status (CPU speed) on the closest grid manager.

III. Grid Computing Service Architecture

A Grid Computing Service (GCS) system allows users to submit their computing tasks along with indication of the required hardware. After task execution, the GCS will reply to the user sending back the results. We identify four main steps in a GCS system:

1) Task Submission Mechanism

The task submission process needs to be as simple as possible and it should be accessible to a maximum number of clients. Task submission can be done through web sites using available web browsers.

2) Task Allocation Mechanism

A GCS needs to allocate computation tasks to available resources.

3) Task Execution Mechanism

After a GCS allocates a task to suitable resources (processing unit), then the task needs to be executed.

4) Results Collection Mechanism

After the execution of the tasks, clients need to be notified of the outcome of their tasks execution.

The Proposed Hierarchical GCS Architecture:
We consider a large-scale grid computing service based on a hierarchical geographical decomposition structure. The following is a bottom-up view of our proposed grid computing service architecture.

Worker Level: Any workstation, here called computing unit, which belongs to a private or public institution can join the grid system and offer its computing resources to the grid. Participating in the grid is as simple as installing a GCS client code available from the Grid web site. When the computing unit starts the GCS service it will report to a manager some information about its resources such as CPU speed.

Site Grid Manager (SGM) Level: Each newly joining computing unit registers itself within a Site Grid Manager which is responsible for managing a dynamic pool of workers. The role of the SGM is to collect information about active workers (CPU speed) in the same site such as a university or a private institution. SGM is also responsible of allocating the incoming tasks to its pool of workers according to a specified load balancing algorithm.

Metropolitan Grid Manager (MGM) level: A MGM collects information about active resources managed by Site Grid Managers in its geographical area. A MGM manages a pool of SGMs. As for workers, new SGMs can join the grid system by registering them selves at a given parent MGM. Within this hierarchical distributed architecture, adding or removing SGMs or workers becomes very flexible and serves both the openness and the scalability of our grid computing service. MGMs are also involved at a higher level in task allocation and load balancing in the grid. Besides to the described functionalities of the MGMs, they constitute the entry points of tasks to our grid system. An MGM acts as a web server for the grid system. Clients in any metropolitan area submit their computational tasks using their web browser to the associated MGM, and then the MGM will pass it to an appropriate SGM, which in its turn transfers it to a chosen worker for execution. MGMs all over the world may be interconnected using an ad-hoc high-speed network. Figure 1 shows an overview of the proposed hierarchical architecture of our grid computing service.

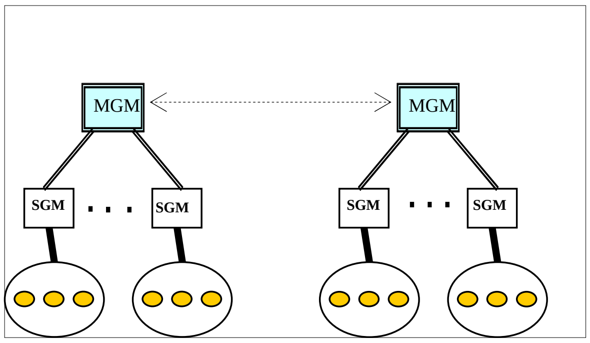

Fig.1. GCS Architecture.

Resource Management Issues: In our design we propose a distributed resource management schema. As mentioned earlier, if any workstation joins or leaves the grid system its status (CPU speed) information is collected at the parent SGM which in turn transmits it to its parent MGM in the hierarchy. All these resources status information are used for the load balancing operation.

IV. Load Balancing

Our algorithm takes into account the processing capacity and the steady-state load of the workers in the Grid. The class of problems we address is: computation-intensive and totally independent tasks with no communication between them.

Load Balancing Model: For each worker participating in the grid we can define the following parameters, which will be used later on the tasks allocation operation. We define a pool of workers as a group of computing units registered in the grid system. The pool is dynamically configured, i.e. workers may join or leave the pool at any time. We define the following parameters for our model:

1) Task: A task is represented by an Id task, a number of instructions NIT, and a task size TS in bytes.
2) Worker Processing Capacity (WPC): Number of tasks per second (similar to the measure used in [16]) the worker can process at full load (100% utilization). This can be calculated using the CPU speed and assuming an Average Number of Instructions per task ANIT.
3) Site Processing Capacity (SPC): Number of tasks per second the site can process. This can be calculated as the sum of the WPCs of all the workers of that site.
4) Metropolitan Processing Capacity (MPC): Number of tasks per second that can be processed under the responsibility of an MGM. This can be calculated as the sum of all the SPCs managed by the MGM.

A Multi-Level Load Balancing Algorithm**:** Our load balancing algorithm is a distributed multi-level algorithm. In what follows we describe the proposed algorithm at each level of the grid architecture:

A) Worker level load balancing (SGM):

Let us consider one Site Grid Manager and its pool of workers. The SGM maintains information about the resources in its pool. In our load balancing algorithm we consider only the CPU resources and their corresponding Worker Processing Capacity *WPC*. The total processing capacity at this site is given by *SPC*. If we denote by *NS* the number of tasks arrived at the SGM at steady-state, the number of tasks to be allocated to each worker of the pool will be proportional to the processing capacity of the worker in order to maximize the throughput and have a good utilization of all the workers in the pool. We define the *worker share* of a worker *i* in the pool by:

$$WShare_i = NS \cdot \frac{WPC_i}{SPC} \qquad (1)$$

Example 1: If we assume that *NS* = 100 *tasks/second* arrive at an SGM manager with three workers having respectively the following worker processing capacities: WPC_1 = 10 *tasks/second*, WPC_2 = 20 *tasks/second*, WPC_3 = 40 *tasks/second*. The load (or the share) of each of the three workers is given by:

$Wshare_1$= 100. $\frac{10}{70}$ = 14.28 *tasks/second*; $Wshare_2$= 100. $\frac{20}{70}$ = 28.57 *tasks/second*; $Wshare_3$= 100. $\frac{40}{70}$ = 57.14 *tasks/second*. This corresponds to allocating more work to faster workers.

B) Site level load balancing (MGM):

Let us consider one Metropolitan Grid Manager MGM which is responsible of a group of site grid managers SGMs. Since the MGM maintains information about all its SGMs in terms of processing capacity, the same load balancing strategy as for the workers will be applied for the SGMs. The total processing capacity managed by this MGM is *MPC* which is the sum of all the SPCs in this group. If we denote by *NM* the number of tasks arrived at the MGM, the number of tasks to be allocated to each Site Grid Manager of the group is also proportional to the total processing capacity of the workers of the site. We define the share of each SGM in the group by:

$$Sshare_i = NM \cdot \frac{SPC_i}{MPC} \qquad (2)$$

Example 2: If we assume that *NM* =1000 *tasks/second* arrive at an MGM manager with three SGMs having respectively the following processing capacities: *SPC_1=70 tasks/second, SPC_2=220 tasks/second, WPC_3=410 tasks/second.* The load or the share of each SGM is: $Sshare_1$ = 1000. $\frac{70}{700}$ = 100 *tasks/second*; $Sshare_2$= 1000. $\frac{220}{700}$ = 314.28 *tasks/second*; $Sshare_3$ = 1000. $\frac{410}{700}$ = 585.71 *tasks/second*. Note that the share of 100 *tasks/second* of the first site manager will then be allocated to the workers managed by this site manager as in example 1.

V. Analytical Queuing Model

In order to come up with an analytical model to predict the tasks response time we simplified our grid system and we will focus on only the time spent by a task in the workers.

A) Analytical Model

Worker Level Model: We model each worker in the pool by an M/M/1 queue. The arrival process to the worker can be modeled by a Poisson process with arrival rate λ_{worker} tasks/second. The tasks are completely independent. The service time at the worker is assumed to be of exponential distribution with service rate μ_{worker} tasks/second which represents the worker's processing capacity *WPC* in our load balancing algorithm.

Grid Level Model: The Metropolitan Grid Manager receives tasks according to Poisson process of intensity λ_{grid} (tasks/second) and then automatically distribute them to the grid workers with a routing probability $PW_i = \frac{WPC_i}{MPC}$, where i identifies the worker. Note that the sum of the routing probabilities for all workers in the Site is equal to 1. The site queuing model is illustrated in Figure 2.

B) Grid Task Average Response time

We are interested in studying the grid system at steady state. As described in the previous section, the grid system is modeled by an open network of M/M/1 queues. We want to compute the expected response time for grid tasks. We will use the Little's formula which

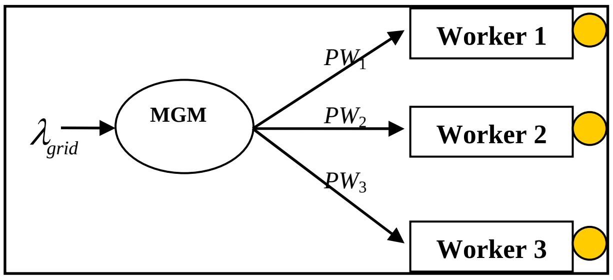

Fig.2. Grid Queuing Model

relates the expected time spent by a task in the Grid system $E[T_{grid}]$ to the arrival rate λ_{grid} and the average number of tasks in the system $E[N_{grid}]$ by the following formula: $E[N_{grid}] = \lambda_{grid} \, E[T_{grid}]$. We assume that there are no task is lost or blocked. First we need to compute $E[N_{grid}] = \sum_{worker} E[N_{worker}]$, the sum of the expected number of task in each worker queue. Since the worker queue is an M/M/1 queue, it is known that the expected number of task in a worker is given by

$E[N_{worker}] = \left(\frac{\sigma_{worker}}{1-\sigma_{worker}}\right)$, where

$\sigma_{worker} = \left(\frac{\lambda_{worker}}{\mu_{worker}}\right)$, $\mu_{worker} = WPC_{worker}$, and

$\lambda_{worker} = \lambda_{grid} \cdot \frac{WPC_{worker}}{MPC}$ according to our load balancing algorithm.

The expected tasks response time $E[Tgrid]$ =

$$\frac{1}{\lambda_{grid}} \sum_{worker} \left(\frac{\sigma_{worker}}{1-\sigma_{worker}}\right)$$

C) NS2 Simulation Model and Results

In this section we present a simulation model to measure the performance of our load balancing algorithm. We use NS2 network simulator to simulate the grid system. We use TCL to build the hierarchical topology of our grid system, which consists of one MGM and parameterized by a given number of SGM's, and a number of workers (CPU speed) managed by each SGM.

In our experimental scenario we generated a grid system with 1 MGM, 10 SGM's with 10 workers each. The CPU speeds of the workers have been generated randomly. We assumed a Poisson arrival process to the grid of parameter λ_{grid} (tasks/second). We compare our load balancing algorithm with a simple uniform load distribution algorithm:

Uniform Load Balancing Algorithm: In this algorithm we fix the routing probability from the MGM to the SGM to the value $\frac{1}{ns}$, where ns is the number of SGM's in the grid. We also fix the routing probability from any SGM to its workers to the value $\frac{1}{nws}$, where nws is the number of workers in the corresponding site.

Table I and II show close tasks response time (in seconds) between the once predicted by our analytical model and those given by the simulations. Table II, shows clearly the superiority of our load balancing algorithm over the uniform strategy. It shows also that our load balancing algorithm is asymptotically optimal because its saturation point $\left(\frac{\lambda_{grid}}{\mu_{grid}}\right) \approx 1$ is close to the saturation level of the grid, independently from any load balancing strategy.

Table I
Task response time (s) for uniform load balancing strategy

Load= $\left(\frac{\lambda_{grid}}{\mu_{grid}}\right)$	**Simulator**	**Analytical**
1.03611E-05	1.812562	1.836861806
3.10833E-05	2.196113	2.204342397
5.18054E-05	2.799685	2.767137924
0.000103611	7.850639	8.104159554
0.000124333	40.145146	40.1057248

Table II
Response time (s) for our load balancing strategy

Load= $\left(\frac{\lambda_{grid}}{\mu_{grid}}\right)$	**Simulator**	**Analytical**
0.777081283	0.004625	0.004648
0.828886702	0.006062	0.006055
0.88069212	0.008546	0.008684
0.984302958	0.05583	0.066007

VI. Conclusion

We presented a hierarchical architecture to design Grid Computing Services. We also proposed a two-level load balancing algorithm, which minimizes the overall tasks response time and maximizes the grid system utilization and throughput at steady-state. We developed an analytical model to predict the expected response time of a task in the grid system. We validate our analytical model by a simulation model using NS2 to evaluate the performance of our proposed load balancing algorithm. Results obtained our analytical model are close to those given by the simulation, and both by show the asymptotical optimal behavior of our load balancing strategy.

References

[1] I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International J. Supercomputer Applications, 15(3), 2001.

[2] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG,* Global Grid Forum, June 2002.

[3] T. DeFanti and R. S. Teleimmersion. *The Grid: blueprint for a New Computing Infrastructure,* In Foster, I. and Kesselman, C.eds. Morgan Kaufmann, pp:131-155, 1999.

[4] D. Thain, T. Tannenbaum, and M. Livny, *Condor and the Grid. In Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality,* John Wiley, 2003.

[5] J. Basney and M. Livny, *Managing Network Resources in Condor.* In Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania. pp: 298-299. August 2000.

[6] Y. Li and Z. Lan, *A Survey of Load Balancing in Grid Computing. Computational and Information Science,* First International Symposium, CIS 2004, Shanghai, China, December 2004

[7] O. Beaumont, A. Legrand, L. Marchal and Y. *Robert. Steady-State Scheduling on Heterogeneous Clusters.* Int. J. of Foundations of Computer Science, 2005.

[8] H.A.James and K.A.Hawick. *Scheduling Independent Tasks on Metacomputing Systems.* Proc. ISCA 12th Int. Conf. on Parallel and Distributed Computing Systems (PDCS-99). Fort Lauderdale, USA, March 1999.

[9] V. Subramani, R. Kettimuthu, S. Srinivasan and P.Sadayappan, *Distributed Job Scheduling on Computational Grid Using Multiple Simultaneous Requests.* Proc. of 11-th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002.

[10] M. Arora, S.K.Das and R. Biswas. *A De-Centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments.* ICPP Workshops pp: 499-505, 2002.

[11] T.L. Casavant and J.G. Khul. *A taxonomy of scheduling in general purpose distributed computing systems.* IEEE Transaction on Software Engineering, 14(2):141-153, 1994.

[12] C.Z. Xu and F.C.M. Lau. *Load Balancing in Parallel Computers: Theory and practice.* Kluwer, Boston, MA, 1997.

[13] M.J. Zaki, W. Li , and S. Parthasarathy. *Customized dynamic load balancing for network of workstations.* In Proc. of the 5th IEEE Int. Symp. HDPC: 282-291, 1996.

[14] A. Touzene, S. Al Yahia, K.Day, B. Arafeh, *Load Balancing Grid Computing Middleware,* IASTED International Conference on Web Technologies, Applications, and Services (WTAS 2005), July 2005, Calgary, Canada.

[15] B. Yagoubi and Y. Slimani. *Dynamic Load Balancing Strategy for Grid Computing.* Transactions on Engineering, Computing and Technology. Vol 13, p:260-265, 2006.

[16] G. Shao, F. Berman, R. Wolski. *Master/Slave Computing on the Grid.* In Proceedings of the 9th Heterogeneous Computing Workshop, pp:3-16, (Cancun, Mexico, 2000).