Performance Evaluation of Load Balancing in Hierarchical Architecture for Grid Computing Service Middleware

Abderezak Touzene, Al-Yahai Sultan, Hussien Almuqbali, Abdelmadjid Bouabdallah, Yacine Challal

Performance Evaluation of Load Balancing in Hierarchical Architecture for Grid Computing Service Middleware

Abderezak Touzene[1], Sultan Al-Yahai[1], Hussien AlMuqbali[1], Abdelmadjid Bouabdallah[2], Yacine Challal[2]

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

[2]Royallieu Research Center
University of Technology Compiegne
P.O. Box 20529, Compiegne, France
{bouabdal, ychallal@hds.utc.fr}

Abstract: - In this paper we propose a new architecture for grid computing service which allows grid users with limited resources to do any kind of computation using grid shared hardware and/or software resources. The term limited resouces includes disk or diskless workstrations, Palmtops or any mobile devices. The proposed grid computing service takes into account both hardware and software requierements of the user computing task, along with some quality of service. On other hand our grid system needs to maximize the overall system throughput, minimize the user responce time, and allows a good grid resources utilization. On this aspect we propose an adaptive task allocation and load balancing algorithm to achieve the desired goals. We have developed a simulation model using NS2 to evaluate the performance of our grid system. We have also conducted some experiments on our a test-bed prototype. The performance evaluation measures confirm the good quality of our proposed architecture and load balancing algorithm.

Key-Words: - Grid computing, Web-based applications, Load balancing, Performance evaluation.

1 Introduction

Computer users, professionals or none professionals, spend their time browsing the Internet or doing any of their daily routine office work (non computing tasks). Many computers are most of the time idle or underutilized. On the other hand there are an increasing number of computing applications that need a huge computing power, which cannot be afforded by a single institution. However, grid computing is a method based on collecting the power of many computers, in order to solve large-scale problems, the parallel processing aspect. In other hand, it offers to independent grid computing services users to share hardware and software grid resources. The grid-computing infrastructure will provide users with almost unlimited computing power on demand along with a desired quality of service.

Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. Traditional distributed systems can use Grid technologies to achieve resource sharing across institutional boundaries. Grid technologies complement rather than compete with existing distributed computing technologies [1]. Grid computing is intended to provide on-demand access to computing, data and services. In the near future Grid systems will offer computing services as simply as browsing web pages nowadays [2].

In this paper we propose a new Grid infrastructure focusing on the Grid Computing Services (GCS). In our approach we take into consideration as much as possible all the required design issues and features of Grid systems as defined in the literature. The objective of our work is to build a Grid computing service system that allows users to submit their computing tasks simply by having access to our Grid Computing Service Web Site (GCSWS). The Grid user may specify along with the submitted task the necessary hardware/software resources needed by the task and any desired quality of service, such as response time or other performance parameters. The minimum requirement to use our grid computing service is only having access to Internet. Indeed the "plumbing" for grid computing is essentially in place: we already have large-scale networks of distributed computers, connected by reliable networks using data communication protocols TCP/IP that is the standard and widely used protocol. The challenges in grid computing therefore lie in developing the software to drive the grid [2].

Another objective of our grid computing service is to save users to buy powerful computers or expensive software (compilers or other tools); all these resources are available in our Grid system. Further more our system will enable users with low memory devices like palmtops or mobile phones to do computation on our behalf provided they have access to Internet. On other hand our grid system needs to maximize the overall system throughput, minimize the user responce time, and allow good grid resources utilization. On this aspect we propose a scheduling algorithm which allow task allocation and an adaptive load balancing to achieve the desired goals.

This paper is organized as follows: Section 2 presents the background and related work. Section 3 presents the layered structure of our grid system and the fault tolerance issues in each layer. In section 4, we present our analytical model and the load-balancing algorithm. In section 5, we present both a simulation model some experimental results on our test-bed GCS. Section 6 concludes the paper.

2 Backgrounds and Related Works

From the definition of grid computing [3], we can see the following keywords, which summarize Grid computing: distributed resources, resource sharing, transparent remote access, infinite storage, and computing power. There are many research problems in the grid. In Condor [4][5], Grid computing service is based on cycle-scavenging strategy that uses the idle workstation model. Condor migrate the tasks when the owner of the machine starts using it. In our GCS, task allocation is based on the current load of the workstations participating in the grid. Once a new workstation joins the grid it fixes its share of CPU utilization to be allocated to Grid computing service. There is no need to do task migration, Grid tasks and workstation owner tasks run in pseudo parallel using the CPU scheduler of the workstation. This solution reduces the cost and implementation complexity of task migration mechanism. 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 focus on load balancing, resources management and fault tolerance problems.

Load balancing: more difficult to achieve in Grid systems than in traditional distributed computing environment 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 [6] [7]. On the other hand, some of those proposed algorithms are extensions of load balancing algorithms for traditional distributed systems. All of them suffer from significant deficiencies, such as scalability problems when we talk about the centralized approaches in [8]. A triggering policy based on the endurance of a node reflected by its current queue length is used in [8]. The authors tried to include the communication latency between two nodes during the triggering processes on their model, but lacks including the cost of the actual task transfer. In our model we also consider the node load or saturation level and we do consider the communication task transfer cost. We propose an adaptive load-balancing algorithm, which takes into consideration both computing, and network heterogeneity to deliver a maximum throughput for our Grid system.

Resource Managements: Different approaches have been proposed in the literature [9]. Our approach is based on multiple resource manager agents, each one is responsible to track and collect information on its pool of workers in the Grid system. The resource manager agents may cooperate in order to achieve performance. Our GCS system is heterogeneous in its nature from all point of views: different machines speed; different

operating systems; different network technologies and speed.

Fault Tolerance: Grid systems are by nature large in the scale, faults may occur any where any time. The purpose of fault tolerance is to increase the reliability, availability and dependability of a system. Solutions for fault tolerance exist for the traditional distributed systems. We propose simple grid adapted solutions to deal with faults, taking into consideration the "when/where" faults occurrence. We thus tackle the faults at different layer in our architecture.

3 GCS Architecture

GCS is a Grid computing service that allows users to submit their computing tasks along with indication of the required hardware or software resources. They can also submit any desired Quality of Service (QoS). The GCS system allocates tasks to the available resources and then executes the tasks. After task execution, GCS will reply to the user and send back the results. We identify four main steps in a GCS:

Task Submission Mechanism: The task submission process needs to be as simple as possible and it should be accessible to the maximum number of clients. The best approach for task submission is through web site, using their favorite web browsers. This submission mechanism is also suitable for mobile devices like laptops and mobile phones.

Task Allocation Mechanism: GCS needs to allocate the computation task to one of the available resources. This may involve task transfer between different components of the GCS system. It may also involve message exchange (such as load balancing information).

Task Execution Mechanism: After GCS allocates the task to suitable resources, then the task needs to be executed. The resource will perform the required execution (compilation may be needed) and then it will prepare the result file to be sent back to the client.

Results Return Mechanism: After the execution of the tasks, clients need to be notified whether their tasks have been executed correctly or there were some problems. If the tasks were executed correctly the results will be sent back to the clients. The best approach to display the results is using the Internet browser itself. We propose the following a layered architecture as shown in Figure 1.

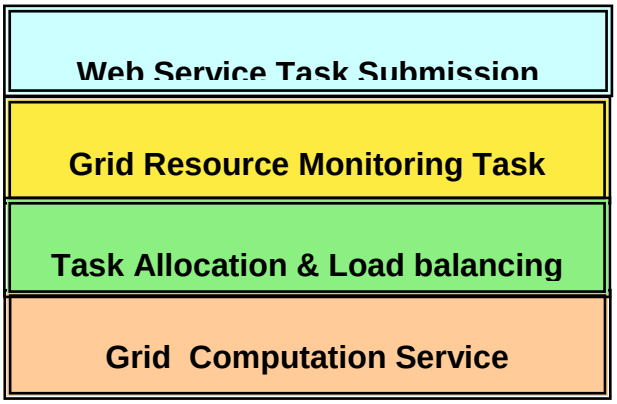

Fig. 1 GCS layered architecture

Web Service Task Submission Layer: Grid users submit their tasks to the Grid system through the GCS web site using their favorite web browser. Our aim is to make access to the Grid like browsing internet. The only requirement for the user to access the Grid is to have internet access and a web browser. In this layer, we deal with user tasks submission and their requirements (resources and quality of service information).

Our Grid system has many access points where users can submit their tasks. To minimize the user's response time, tasks might be directed to an appropriate Grid entry using mirror web site mechanisms. Each Grid entry point is called a Grid Agent Manager (GAM). A GAM is responsible of a dynamic pool of workers. The Web information service layer offers to the clients some information about statistics, and some information about the expected response time based on the actual overall load of the grid system. Information about load and availability can grid be collected from the below layer. The Web service layer may decide which GAM to direct the client task based on the grid load and the desired user quality of service.

For the fault tolerant aspects: Our grid system has many access points, where users can submit their tasks, then faults can occur in any one of these access points. This kind of failures could be tolerated by redirecting the user request to any other access point. Fault discovery at this level is implemented using a connection failure timeout mechanism. Another kind of failure the user may experience is sending a task for computation and getting no reply to the request. This layer deals also

with task resubmission mechanism by example in this case resend the task to another access point.

Grid Resource Monitoring Layer: Our grid system is composed of many hardware and software resources. It is by nature heterogeneous from all aspects: network technologies in different computing sites and different operating systems for the computing workstations even within the same site. To fully utilize the resources, we need to monitor those resources to know which one is underutilized and which one is overloaded. Monitoring functions are provided by the Grid resource monitoring layer. The GAMs are the building blocks of this resource monitoring layer. Resource tracking and monitoring is based on distributed mechanisms where workers in a given pool reports to their GAM the status of their resources if any significant change has been noticed since the last report. For example if there is an important change in the CPU utilization or other hardware resources, a change of status is reported to the parent GAM. In our implementation we are using Network Weather Service NWS (for the Unix based resources) [10], and we developed our own tracker tool for the windows-based resources. The parent GAM publishes all information about the resources of its pool at the web service layer. GAMs in the Grid system may interact and exchange information to achieve load balancing operations.

Task Allocation and Load Balancing Layer: In this layer we consider two levels of load balancing, and we propose a load balancing algorithm which works similarly for both levels. The lower level of load balancing consists of the GAM, which distributes the users tasks (load) received from the above layer to the receptor workers of its pool. A worker is declared as a receptor if its CPU utilization is below a given threshold. The load balance strategy is to distribute uniformly the load on the receptor workers. The higher level load balancing is performed at the GAMs level. Whenever a GAM sees that its workers have reached their saturation level (overloaded) and the incoming arrival rate is high, the GAM may decide to direct the over-flow incoming tasks rate to another GAM on the Grid system. In fact the GAMs exchange information about processing availability of their pools. The extra-load at any GAM will be distributed uniformly to other receptor GAMs. A receptor GAM is a GAM with receptor workers. The load balancing algorithm and the fault tolerance issues are discussed in more detailed in section 4.

Grid Service Computation Layer: This layer is the lowest layer and it is mainly responsible to perform tasks computation. It consists also in updating the status of the hardware and software resources at a given computing unit. In this layer, some statistics on the usage of the resources are computed as an example: the average task size, average task execution time are continuously calculated and updated at the GAM levels. Information such as average task execution time is an important parameter in our model, it can be use to determine the computing capacity (number of task per unit of time) at a given computing node and thus the computing availability in a given pool of workers.

Failures can occur in this layer, a worker may fail or simply the owner of the worker machine shuts it down. Each GAM monitors the tasks that have been sent for execution. It set a timeout parameter for each of them. After passing the timeout, it will resend them to other workers.

4 Load Balancing

In this paper we present an adaptive, distributed and sender initiator load balancing algorithm in a Grid environment. Our algorithm takes into account the processing capacity of the nodes and the communication cost during the load balancing operation. The class of problem we address is: computation-intensive and totally independent tasks with no communication between them. A task is a user source program written in any programming language. The user program needs to be compiled first then executed.

Load Balancing Model:

We start by giving some definitions and concepts useful for our load balancing model. We define a pool of worker as a group of computing nodes willing to participate in the grid system. The pool is dynamically configured, in which some nodes may join or leave the group at any time. As mentioned earlier the pool is managed by a Grid Agent Manager. Grouping the nodes might be based on the type of nodes such as nodes of a cluster of computers, supercomputer machine, or simply just nodes interconnected via the same physical network or sub-network. We define the following parameters of our model:

- **Task:** A task is defined as a source code written in any programming language (or in bytecode). In our model we consider that a task consists of a number of instructions (TNI). In general it is not possible to know

a priori the exact task execution time until it is executed on a specific worker (it is machine dependant). We define the Task Size (TS) as the storage size in byte of the task file.

- **Processing Capacity (PC):** Number of tasks per second (similar to the measure used in [11]) the worker can process at full load. This can be calculated using the CPU speed (Instruction per second) and assuming an average task (TNI).

- **Load (L):** CPU utilization of the node (given by NWS [10] and our Window-based tool).
- **Actual Processing Capacity (APC):** Actual processing capacity of the system, $APC = PC \times L$
- **Grid Processing Capacity (GPC):** the maximum processing capacity (tasks/seconds) at "Grid threshold" utilization. We assume that the CPU is shared between the node owner tasks and the grid tasks. In our model the node owner is the one who decides what will be the share of CPU (percentage) he/she delegates to the GCS. This share is what we call "Grid threshold" utilization.
- **Available Grid Processing Capacity (AVGPC):** The number of tasks/seconds the node can perform until it reaches its maximum allowed grid processing capacity GPC. In other words, the additional load which can be offered for grid computations. $AVGPC = GPC - APC$

In fact this formula is conservative because the actual load L in APC is a shared load between the owner tasks and the grid tasks. This will always ensure the agreed upon node's share utilization.

Example: $PC = 500t/s$, $L = 30\%$. If the grid threshold is 80%, then:

$APC = 500x0.30 = 150t/s$, $GPC = 500x0.80 = 400t/s$, and the available processing capacity :

$AVGPC = 400 - 150 = 250t/s$.

Workers Level Load Balancing

Each GAM is managing one pool of workers. The GAM receives the submitted tasks and stores them in a queue. It checks the current status of its pool and distributes the tasks between the workers according to their loads. Under utilized workers (only) report their load status to their managers waiting for new tasks to be executed on them. This will reduce the number of messages to be exchanged between the manager and its workers. The manager keeps the workers load status on a list.

The tracking of the resources is event driven and not periodical to minimize message exchange between the GAMs and their workers. In the pool, only underutilized workers will report their available processing capacity *AVGPC* and only when they notice a significant change in their values. Those workers will be called receptors.

The GAM distributes the received tasks between the receptors according to their reported *AVGPC* to maximize the throughput of the group. If N is the number of received tasks at a given GAM, we define the following parameters:

Total Processing Capacity *(TPC)* of the pool: is the summation of the processing capacities of the pool's members.

$TPC = \sum_{i=1}^{n} PC(i)$.

Total Available Processing Capacity *(TAPC)* of the pool: is the summation of the available processing capacities of the receptors in the group. $TAPC = \sum_{r=1}^{n} AVGPC(r)$.

Receptor Share *($share_{r(i)}$)*: is the number of tasks to be given to receptor *i*.

$$share_{r(i)} = \frac{AVGPC_{r(i)}}{TAPC}.N$$

Example: If manager GAM1 received 200 tasks and it has three underutilized (receptors) workers r_1, r_5 and r_7 with *AVGPC* of 250, 300, 140 tasks/sec respectively, then:

$TAPC = 250+300+140 = 690 \ tasks/\sec$

$$share_{r(1)} = \frac{250}{690}.200 = 72tasks$$

$$share_{r(5)} = \frac{300}{690}.200 = 87tasks$$

$$share_{r(7)} = \frac{140}{690}.200 = 41tasks$$

GAMs Level Load Balancing

Let us now focus on the GAMs interconnection structure and explain how the interaction between the GAMs of the grid helps to maximize the overall throughput or what we call GAMs level load balancing. We propose to arrange the GAMs of the Grid according to a logical ring (backbone) to help each others as shown in Figure 2. Logical ring structure has been selected because it is the most popular backbone and many distributed algorithms (synchronization, election, communication) have been proven to be efficient and simple to implement using the ring structure.

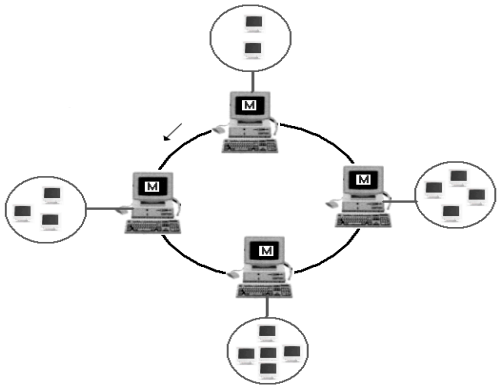

Fig. 2: GAMs arrangement on a ring

Ring structure and precisely the token ring mutual exclusion algorithm is used to ensure that only one load balance operation at a given GAM can be initiated a any time. Having more than one load balance operation at a time may induce information inconsistency and then wrong load balance operations. Exchanging information between the GAMs of the Grid uses a token message (privilege) to be circulated on the ring. The token message contains the global view of the Grid system. This token message contains the following information about each GAM:

- **Manager ID:** the communication address of the manager.
- Total Available Processing Capacity *TAPC* of the GAM.
- **Status: t**he status of the pool, which can be one of the following:
  - **Neutral (N):** Pool under normal load.
  - **Receiver (R):** Available *TAPC* is high (pool is under-utilized), ready to receive new tasks and thus increase the throughput.
  - **Sender (S):** Small *TAPC* and the pool is overloaded. Need to transfer some load to other pools to help.

Example: A sample of the token message could be the following:

GAM	M1	M2	M3	M4
Status	R	N	S	R
TAPC	500	0	400	250

From this token message, we can see that GAM M1 is Receiver. The total available processing capacity of the pool is 500 tasks/sec. On the other hand GAM M3 is a Sender. It has an overload of 400 tasks/sec.

Load Balancing Operation:

Some GAMs may receive much more requests than others. When a GAM keeps receiving tasks when its pool of workers is overloaded, the GAM queues up the requests and waits for the token message in order to check if any GAMs are ready to help (status Receiver). When an overloaded GAM (Sender) receives the token message it performs the following steps:

1. **Calculates the share:** (number of tasks) to be delegated to other pools depending on the Total Available Processing Capacity of that pool and on the speed of the link connecting this GAM with the other GAMs. In our model we take into account the communication cost to send tasks form GAM to GAM. We will discuss this matter in more details in the next section.
2. **Modify the token message:** It will modify both the status and *TAPC* of the Receptor GAMs as well as its own values. Since the token message is seen and modified by only one manager, then we ensure the consistency of the information inside the message. In fact, the token received at any GAM gives the privilege (mutual exclusion) to the token holder if it is sender to initiate the load balancing operation. The update of the *TAPC* and the Status of the receivers GAM within the token message is necessary to reflect the load changes after the load balance operation is performed.
3. **Send that share**: Amount of tasks to be transferred to the Receptors managers. Receptor Manager deals with the tasks as if they are external tasks.
4. **Pass the token:** to the next GAM in the ring.

Calculating the GAMs share: Since each GAM is connected to another GAM using probably different type of network (different network speed) and each GAM may has a different total available processing capacity *TAPC*, then when a GAM manager of an overloaded pool need to distribute the extra tasks to the other GAM, it needs to take into consideration both factors Receptor *TAPC* and also link speed from the sender GAM and the Receptor GAMs. Pools with high processing capacities and fast network connection should get more tasks than pools with low processing capacities and slow network connection. We express the network speed or capacity in terms of number of task transferred per second (tasks/sec) [13].

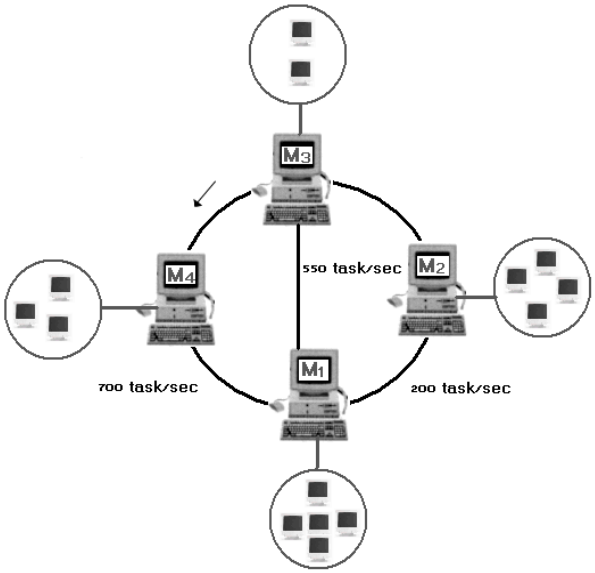

Fig. 3: Network cost example

For each Receptor GAM we need to calculate the load share that it will receive from the Sender GAM depending on its reported *APC* and its network connection speed (network speed between the sender and the receiver). The load that can be received at a receiver GAM is the minimum between its reported *APC* and the network link capacity.

$$APC\_offered = \min(TAPC, NW\_Capacity).$$

Then we can define, the Grid Total Available Processing Capacity *(GTAPC)* as the sum of the *APC_offered* for all the Receiver GAMs:

$$GTAPC = \sum_{rec} APC\_offered(rec)$$

and then we can calculate the share for each receiver GAM as,

$$share_{rec(i)} = \frac{APC\_offered(i)}{GTAPC}.N \quad tasks$$

Example: If GAM M1 in Figure 3 has N=1000 unprocessed tasks, and the following token information, then it will distribute the load as follows:

Manager	GAM1	GAM2	GAM3	GAM4
Status	S	R	R	R
TAPC	1000	700	550	600

- M2 can offer up to 700 tasks/sec, but since we can transfer a maximum of 200 tasks/sec over the link connecting M1 and M2, then M2 can only supply 200 tasks/sec.
- M3 can offer up to 550 tasks/sec, and since the link can offer that amount then M3 can supply the 550 tasks/sec.
- M4 can offer up to 600 tasks/sec, and since the link can offer that amount then M4 can supply the 600 tasks/sec.

$$GTAPC = 200 + 550 + 600 = 1350 tasks/\sec$$

$$share_{gam(2)} = \frac{200}{1350}.1000 = 149 tasks$$

$$share_{gam(3)} = \frac{550}{1350}.1000 = 407 tasks$$

$$share_{gam(4)} = \frac{600}{1350}.1000 = 444 tasks$$

After updating the token message by GAM1, the updated message which will be passed to GAM2 is:

Manager	GAM1	GAM2	GAM3	GAM4
Status	N	R	N	R
TAPC	0	500	0	350

Fault Tolerance at Resource Management and Load Balancing Layer

The main role of the managers GAMs in our architecture is to receive the tasks and distribute them among the available workers in the pool according to the proposed load balancing algorithm. A GAM node could be a powerful machine in terms of processor speed, memory and hard disk. On the other hand, there is no special requirement on the GAM node. In case of failure any node in the pool can take this temporary role until the original manager is repaired. Faults in this layer can be detected by:

- Low utilized workers: In case of failure, when any of the low utilized workers try to report their utilization status to their manager, one of those workers will detect the manager failure.
- GAM Predecessor in the ring: When it sends the token message to the next GAM, it can detect the failure by getting a timeout connection error.

- Overloaded GAM: Whenever it tries to send some of its load to that manager, it can detect the fault.

Whenever one of the workers detects the failure, it starts an election to elect a new manager. The election is based on the classical Bully algorithm, and then the node with the lowest utilization wins the election and it will send a victory message for other nodes in the pool. Sending this message is not costly operation since the pool is in the same LAN.

Since our architecture is based on a logical ring structure, then the problem of lost token exists in case of failure. To overcome this problem, whenever new manager is elected it should regenerate a new token message to replace the old one.

Regenerating new message will solve the lost token problem but will introduce another problem which is duplicated tokens in the ring. This problem can happen when the old token is not yet lost and a new token has been regenerated. Duplicate tokens can cause information inconsistency if one sender GAM is using the old token and another sender manager starts using the new token. But as soon as the manager using the old token try to send to the failing manager and there is time out reply, then it should understand that there is a failing manager and it should discard this token and wait for the new token which will soon come.

5 Performance Evaluation of our GCS

In this section we present some experimental results conducted on our GCS prototype, which has been implemented using Java RMI system. We also provide a simulation model , which helped us to study the behavior of our GCS under different system parameters: varying the workers CPU speed and varying the network bandwidth between the GAMs.

5.1 GCS Experimental Results

We have implemented our grid system using Java RMI technology. The choice for Java RMI technology to implement grid computing services is the object of another paper. We just summarize our findings by the fact that all the grid services defined as a standard in [2], either they are supported in Java RMI technology or might be implemented simply. Our prototype Grid system is composed of 8 GAM single-processor Leo Presario Workstations. Each node (GAM) has a single 900 MHZ Pentium III processor with 128 MP RAM and 40 GB IDE Disk. These nodes are connected by an 8-port Myrinet switch. On the other hand, these nodes are connected with Ethernet LAN network to their pool of workers. In our prototype we use 24 workstations (workers) running under RedHat9 Linux operating system, with kernel version 2.4.20-8.

Test Task: During these experiments, we have used a task that performs 25x25 matrix multiplication. The task starts by creating two 25x25 matrices, and then performs the multiplication. Our test task is written using java. Each worker needs to compile and then execute the task.

Worker Level's Load Balancing Evaluation

Our objective in this set of experiments is to measure the quality of our load balancing algorithm GAM worker (worker level). During this set, we will measure the user response time and the system throughput.

- **Uniform Load Distribution Experiment:** We use one GAM with 12 workers in the pool with no load balancing algorithm.
- **Load Distribution Experiment:** Same configuration as in the previous but with load balancing.
- **Double the number of Workers Experiment**: We use one GAM with 24 workers in the pool. The manager is using the proposed load balancing algorithm to distribute the tasks.

Figure 4 represents the user response time for the different experiments for evaluating the load balancing at worker level. It shows clearly the benefit of having our load balancing algorithm compared to the version without load balancing (NOLB). In other hand, we can also see the good performance of our algorithm when we double the number of workers(1M_24W_LB), the response time is reduced to almost half.

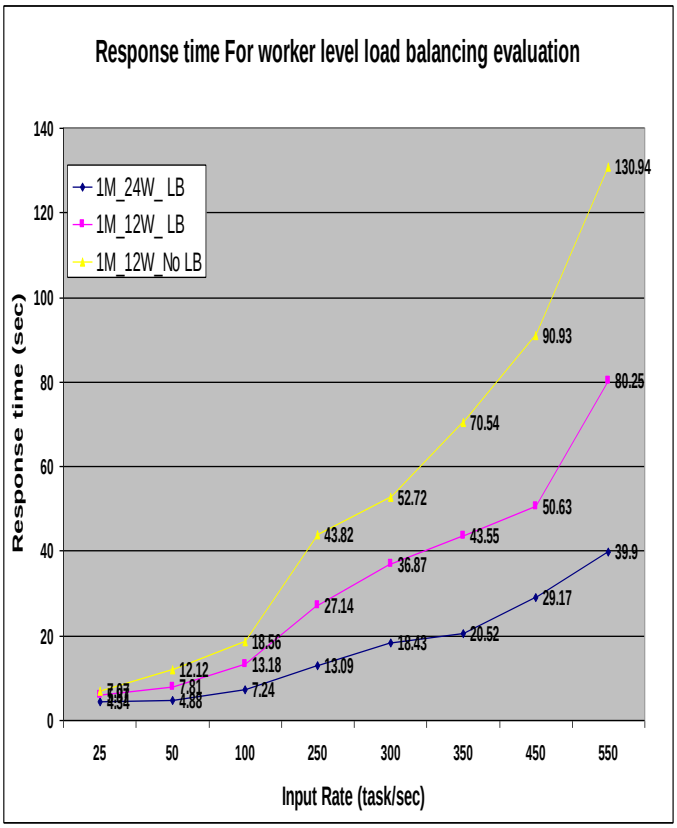

Fig. 4: Worker level load balancing

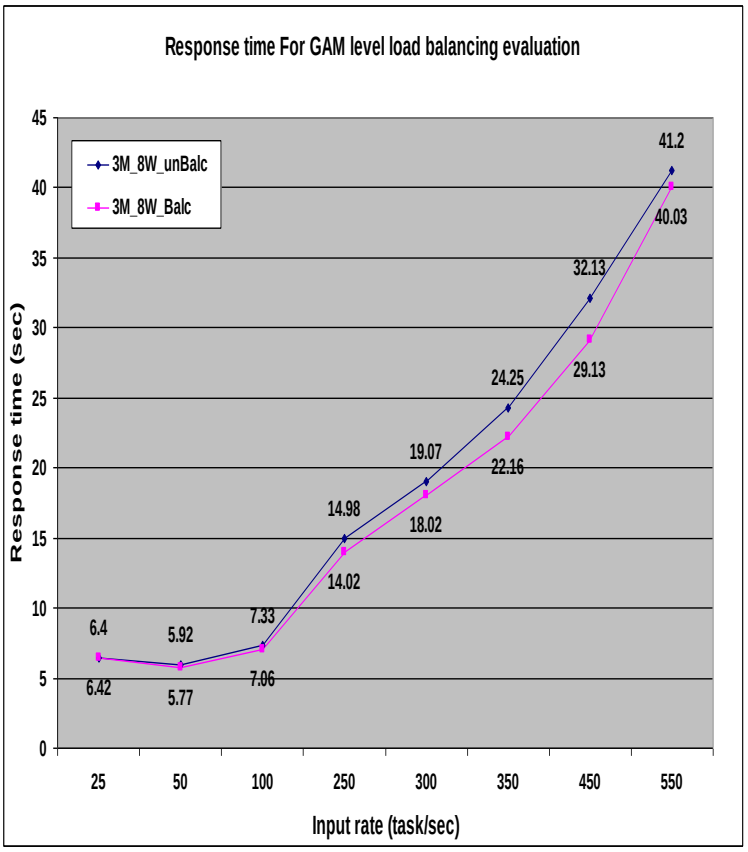

Fig. 5: GAMs level load balancing

GAMs Level's Load Balancing Evaluation

- **Unbalanced Load Traffic:** In this experiment, we use three GAMs. Each GAM manages a pool of eight workers. In this experiment, the load is directed to only one GAM (unbalanced load traffic).
- **Balanced Load Traffic:** In this experiment we, use three managers. Each GAM manages a pool of 8 workers. In this experiment, we consider different load rates directed (independently) to three GAMs (balanced load traffic).

Figure 5 shows that directing the load independently to three GAMs gives better results than directing the load to one GAM only. These results are expected because it will reduce the communication cost due to tasks transfer between GAMs. We can see clearly that after the rate 250 tasks/sec the two curves are separated and this correspond exactly to the saturation rate for the target GAM. Beyond this saturation rate the GAM starts to direct the overflow rate to other GAMs and hence the communication cost effect starts. Note also that the difference between these two curves is small because communication between GAMs uses the full duplex Myrinet switch that provides 2+2 Gigabit/sec.

5.2 Simulation Results Using NS2

The objective of the simulation is to study the performance of our GCS and the load balancing algorithm under different workers speed and different network bandwidth between GAMs. We carried out simulations using NS2. We considered a topology with 3 GAMs, where each GAM manages a pool of 8 workers. Figure 6 illustrates this topology.

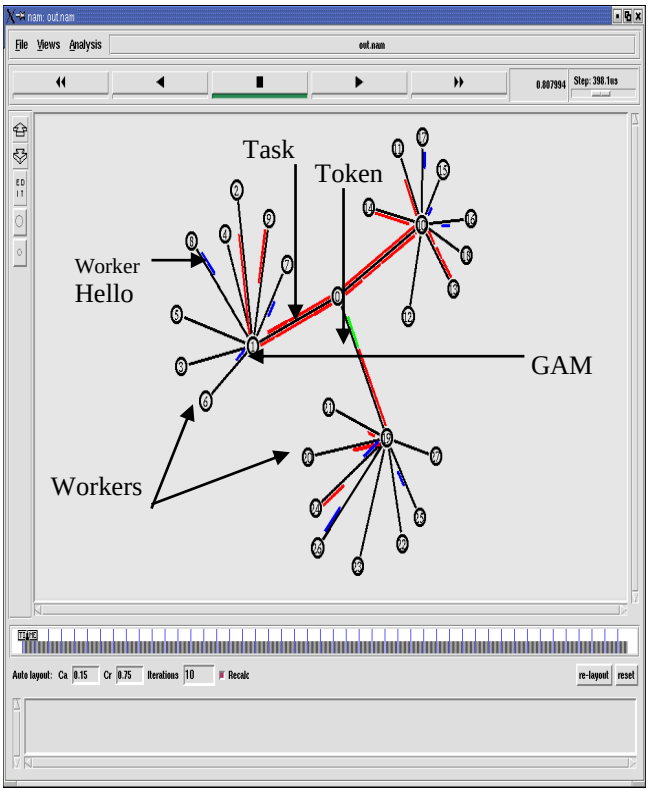

Fig. 6: NS2 scenario

We assumed that tasks submission to our Grid Computing Service follows a Poisson law with parameter λ which corresponds to the arrival rate of the tasks to the system. We supposed also that the number of instructions in a task, and hence the task's execution duration follows an exponential distribution with parameter μ, which corresponds to the average execution duration of a task.

Impact of the input rate:

In a first stage, we were interested in evaluating the impact of the input rate (tasks per second) on the performance of GCS. We considered two cases: in a first case we submitted the tasks to the same GAM (unbalanced arrival of tasks into the system). In the second case, we distributed the arrival of the tasks uniformly over the three GAMs of the system. Figure 7 illustrates the response time of the system with respect to the input rate. We notice that the system behaves better when the tasks arrival is uniform over the three GAMs. Indeed, this minimizes the communication delays due to load balancing at the GAM level.

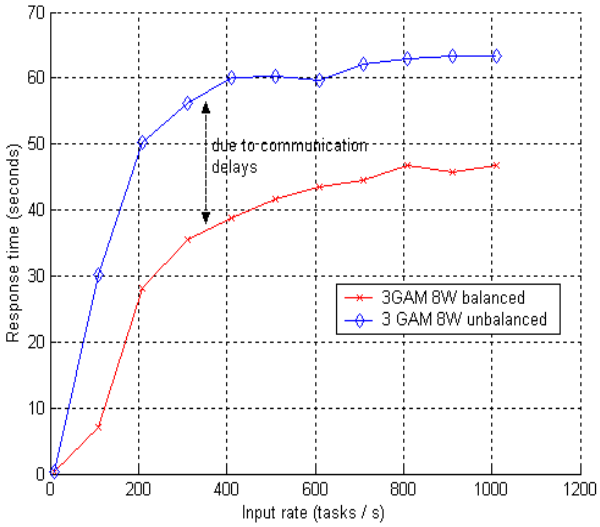

Fig.7: Average response time of GCS with respect to input rate.

Figure 8 illustrates the output rate of the system with respect to the input rate. We notice again the difference in output rates due to communication delays between GAMs. We remark that at 110 tasks/s the system reaches the saturation point and its output rate becomes constant.

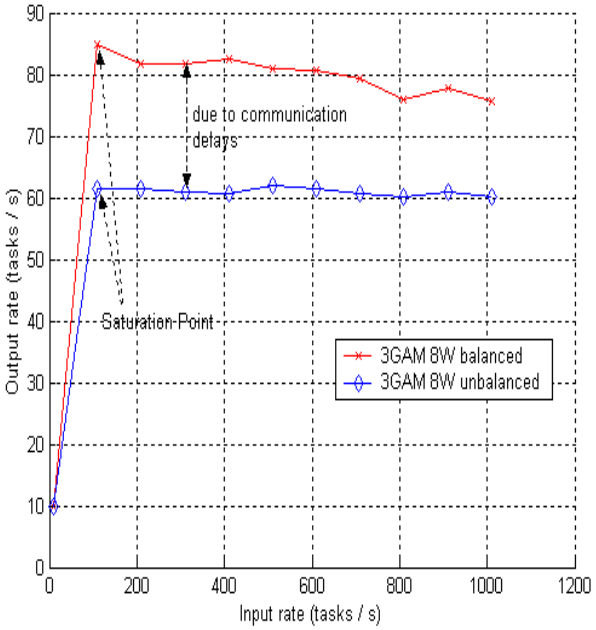

Fig. 8: Output rate (tasks/s) with respect to input rate.

Impact of the GAMs' links bandwidth:

In a second stage, we were interested in evaluating the impact of the bandwidth between GAMs. Figure 9 illustrates the average response time with respect to the available bandwidth between GAMs. We notice that, when the available bandwidth between

GAMs exceeds 1.5Gbps, the communication delays between GAMs become negligible in conformance with our experimental measures.

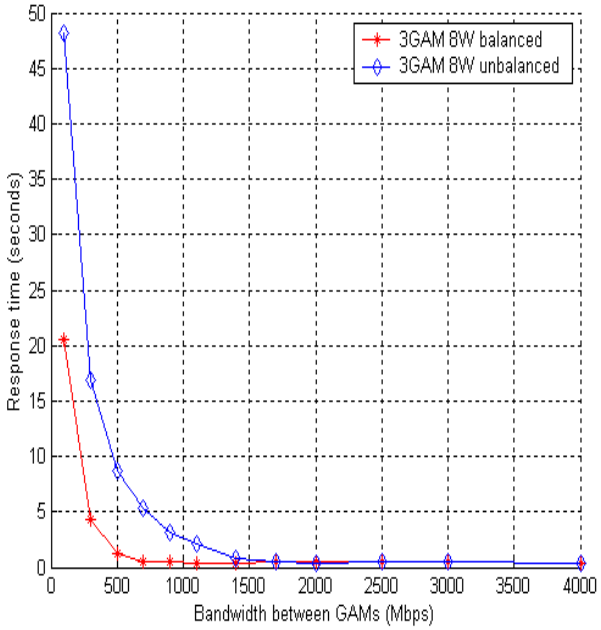

Fig.9: Response time with respect to bandwidth between GAMs

6. Conclusion

We presented a new layered architecture for implementing Grid computing services. We proposed an adaptive two level load balancing algorithm, which minimizes the overall tasks response time and maximize the grid system throughput. The experimental and the simulation results show the good efficiency of our load balancing algorithm on our prototype Grid system. In our future work we investigate the security aspects and an analytical model to measure performance of our GCS.

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 22, 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, 1999,131-155.

[4] Douglas Thain, Todd Tannenbaum, and Miron 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. ISBN: 0-470-85319-0

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

[6] 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.

[7] 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.

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

[9] T.K. Apostolopoulose, G. C.Oikonomou. A scalable, Extensible framework for grid management. *IASTED international conference Feb. 2004*, Austria.

[10] R. Wolski. Experiences with Predicting Resource Performance On-line in Computational Grid Settings. *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Number 4, pp 41-49, March, 2003.

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