Memoir: A History based Prediction for Job Scheduling in Grid Computing

Swarna M
Department of I.T.,
MVGR College of Engineering,
Vizianagaram.

P. S. Sitharama Raju
Department of I.T.,
MVGR College of Engineering,
Vizianagaram.

Nagesh Vadaparthi
Department of I.T.,
MVGR College of Engineering,
Vizianagaram.

ABSTRACT
Computational grid is an emerging trend in the area that allows the management of heterogeneous, geographically distributed and dynamically available resources in an effective way by extending the boundaries of what is being perceived as distributed computing. The most crucial problem in any grid environment is job scheduling which is observed to be as NP-Complete problem. Thus there is no possible best solution for scheduling the repeatedly submitted jobs in particular to the jobs that have long duration for execution, I/O intensive and resource requirements which vary at different times during the task execution. It is also essential to analyze the variation in resource requirements based on the past history of the same job executed earlier and use the gathered information in the decision making process. Hence, in this paper we propose and develop a novel approach for job scheduling based on past history.

Keyword
Scheduling, I/O intensive, prediction, NP-Complete

1. INTRODUCTION
A computational grid is a hardware and software infrastructure that provides dependable, consistent pervasive and inexpensive access to high-end computational capabilities[1]. The functions of grid areto co-ordinate the resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The workload is broken and submitted as pieces to the idle server cycles which are being handled by the job schedulers. A scheduler plays a vital role by setting rules and priorities for routing jobs on a grid-based infrastructure.

Most of the grid systems include some sort of job scheduling software which is capable of locating the machine on which to run a grid job that has been submitted by the user. In general, it may usually assign the jobs in a round-robin fashion to the next machine which matcheswith the resource requirements. However, there are advantages of using a more advanced scheduler.

Some schedulers implement a job priority system which is sometimes done by utilizing several job queues, each with a different priority. When the grid systems are available, jobs that are in the highest priority queues are taken first to execute the jobs. Various kinds of policies are also implemented using schedulers. Policies can include various kinds of constraints on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

Schedulers usually react according to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower level scheduler rather than to an individual machine.

The advanced schedulers have a capability of monitoring the progress of scheduled jobs managing the overall work-flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for re-submission and some of which are not.

Reserving resources on the grid in advance is accomplished with a "reservation system." It is more than a scheduler. It is first a calendar based system for reserving resources for specific time periods and preventing any others from reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached. Similarly, backfilling is another technique which enables lower-priority jobs to use blocked resources which does not influence the resource reservation. Also it is possible to allocate a specific job to execute on the designated machine or resource whenever it is submitted.

But, none of the schedulers bother about the I/O intensive jobs which are submitted frequently but at irregular times. It is observed that when such jobs are submitted in batch, the CPU remains idle for longer time and the memory utilization is increased drastically leading to longer execution time of the jobs and in some times, job remains in infinite loop due to unavailability of resources (memory) leading to large number of Swap-ins and Swap-outs which is called as Thrashing [2]. Hence, in this paper we have implemented a novel and effective approach to avoid thrashing by considering the past history of the job when it was submitted earlier and the amount of memory consumed by the job at specific intervals of time when submitted earlier. The proposed technique is implemented on Sun Grid Engine (SGE)[3]has been discussed in section-3. The rest of the paper is organized as follows: Setion-1.2 discusses about the related work and in section-2 addresses the features of SGE. The pitfalls present in the SGE experimentally are demonstrated in section-3. Section-4 deals with the algorithm of the proposed approach; the experimental results are discussed in detail in section-5 and finally in section-6 the paper is concluded.

RELATED WORK
Condor [4] is a resource management system which supports high-throughput computations by allocating the resources which are found idle in the network to the application tasks. It is designed to solve the problem of wait-while idle problem by utilizing the idle machines effectively.Condor implements a centralized scheduling process by using a dedicated machine (Central Manager) to which every workstation submits the jobs in the local queue to the central scheduler responsible for

allocating the suitable resources for execution of the job. It supports the pre-emption of jobs when a machine decides to withdraw and is scheduled on another machine. Condor also has check-pointing mechanism introduced which offers fault tolerance and recovery.

Globus[5] toolkit is a collection of various tools which provide the basic services that are required for grid computing such as security resource management, information services and many more. It adopts a decentralized scheduling model which is done by the application level schedulers and the resource brokers namely AppleS, Nimrod-G etc.

Legion[6] is an object-based OS which offers an infrastructure for grid computing. It provides a framework for scheduling that accommodates different placement strategies for different classes of applications. Legion scheduler has a hierarchical structure. Scheduling for running jobs is invoked by either the users or active objects.The scheduling decision is taken based on the information regarding security and resource usage policies for using class object attributes specified by the resources owners.Co-allocation of resources is not supported.Legion also supports resource reservation and object persistence.

Punch [7] is a demand-based grid computing system that allows end users to transparently access and use globally distributed hardware and software resources. The resource management system in PUNCH has a pipelined architecture. With individual components in the pipeline replicated and geographical distributed for scalability and reliability. PUNCH employs a non-preemptive, decentralized, sender-initiated resource management framework.Scheduling is performed in a decentralized manner by resource and pool managers. A query manager parses the resource request, transforms it into an internal representation and forwards it to a resource manager based on some criteria. Individual Resource managers try to map the requests to a pool manager from the list of pool managers stored its local database. If a request cannot be mapped to any of the pools, a new pool is created. When a new pool manager is created, it queries a resource information database for information of all the resources which satisfy the pool's criteria.Scheduling policy is extensible. Each pool manager has one or more scheduling processes associated with it. The function of these processes is to sort the machines in its pool cache according to some specified criteria (average load, available memory) and to process queries sent by resource managers. Pool managers can be configured to utilize different scheduling policies.

Europeandatagrid [8] was designed to provide distributed scientific communities access to large sets of distributed computational and data resources. The three main application areas of EU datagrid are the High Energy Physics (HEP) where geographical distributed researcher analyze the same data generated by single source but replicated distributed data stores, Earth Observation data are collected at distributed stations and also maintained in distributed databases, while in Molecular Biology research large number of independent datasets are used which need to be integrated into one logical system. The data grid project develops data grid services and depends on the Globus toolkit for core middleware services like security.The workload management package consists of a user interface, resource broker, job submission service, book keeping and logging service. A job request from user is expressed in a Job Description Language based on the Classified Ads of Condor. The resource broker (RB) given a job description tries to find the best match between the job requirements and available resources on the grid, considering also the current distribution of load on the grid.

Nimrod-G [9] is grid-enabled resource management and scheduling system based on the concept of computational economy. It was designed to run parametric applications on computational grid. It uses the middleware services provided by Globus Toolkit but can also be extended to other middleware services.Nimrod-G uses the MDS services for resource discovery and GRAM APIs to dispatch jobs over grid resources. The users can specify deadline by which the results of these experiments are needed. Nimrod-G broker tries to find the ***cheapest*** resources available that can do the job and meet the deadline. Nimrod uses both static cost model (stored in a file in the information database) and dynamic cost model (negotiates cost with the resource owner) for resource access cost trade-off with the deadline.

GRACE [10] provides middleware services needed by the resource brokers in dynamically trading resources access costs with the resource owners. It co-exists with other middle-ware systems like Globus. The main components of the GRACE infrastructure are a Trade Manager (TS), trading protocols and Trade Server (TS). TM is the GRACE client in the Nimrod-G resource broker that uses the trading protocols to interact with trade servers and negotiate for access to resources at low cost. Trade Server is the resource owner agent that negotiates with the resource users and sells access to the resources. TS use pricing algorithms as defined by the resource owner that may be driven by the demand and supply. It also interacts with the accounting system for recording resource usage. It has an extensible application-oriented scheduling policy and scheduler uses theoretical and history based predictive techniques for state estimation. Scheduler organization is decentralized and the namespace is hierarchical.

2. SUN GRID ENGINE

The current SGE has enormous features among the existing grid engines. Usually, the load schedulers are not aware of the required resources for a particular job prior to its execution. But the SGE6.1 provides a feature of advanced resource reservation for the jobs which will be submitted periodically. In this process, the administrator can configure a job to be executing periodically and when the job is about to submit, the master host reserves the required amount of resources in one of the execution host before the job is being submitted. Also an advanced feature introduced in SGE6.1 is that the administrator can reserve a specific host for execution of a job every time it is submitted. This process leads to preventing the periodically executing and prioritized jobs from waiting in the queue for availability of resources.

Similarly, the scheduler verifies various parameters before allocating the job to the host, the parameters to be considered are limited to CPU availability; time to execute the job, etc. The scheduler daemon is not aware of how much memory is required by the job during the execution of the job. This may lead to serious problem because of the dynamism in memory utilization of jobs, i.e., the running jobs may not need the same amount of memory throughout its life cycle. This process is done by taking the snap shots of job status. Some Grid Engines maintain the information called as checkpoints. These checkpoints are used to re – schedule the task only when a job is terminated under abnormal conditions. In SGE also, the scheduler considers the system load of execution hosts and migrates the jobs when any one node is unavailable in middle of execution of jobs and re-schedules the job based on check points.

Though the SGE provides various features, the most promising issue is load balancing. The efficiency of any grid

engine lies in its load balancing process. In the present grid engine tools even though the dynamism in load balancing is attained, the major issue concerning is the procedure adopted when a job which is already executed earlier made to execute once again. The current SGE allocates the job based on existing scheduling policies(viz., Function policy, urgency based policy, override policy)without considering the past history of the job. Thus when submitting the jobs repeatedly which takes increased memory consumption from time to time, it is observed that the number of swap-ins and swap-outs increases drastically at a particular time. Finally this condition leads to a situation called **thrashing**which is discussed in the next section**.** To avoid this situation, we propose new approach.

3. EXPERIMENTAL RESULTS OF THRASHING IN SGE

When a new job is submitted to Sun Grid Engine, scheduler checks the current load in terms of CPU utilization of all nodes. If the CPU utilization is low on any node, and the requesting profile matches with the node then the job is submitted. But the scheduler does not consider the current and future memory utilization.

The following figures depict the situation of Thrashing when 3 jobs submitted to the grid, where each job approximately requires 600MB and to execute each job, the execution host takes approximation of 20 minutes. But when such 3 jobs are submitted takes longer than 60 minutes.

Figure-1 shows CPU, Memory and N/w utilization by the system after immediate submission of 3 such above said jobs where the CPU utilization is 100%. Grid Engine system has the load_avg parameter which will be checked by the scheduler at the time of allocating jobs to execution hosts. And the default value is >=1.75 per core. Thus, at a time a single node can execute 3 jobs which is a peak load. If load_avg default value is exceeded, the processes/jobs which ever are submitted to queue will be in waiting state. But the scheduler does not verify whether the memory is available to execute the job.

Figure-2 demonstrates the situation above as Figure-1. This was taken after 1 minute of submission. Figure-3 depicts the situation of increased swap memory in 1 minute of submission and the number of swapped out pages are 1686250.

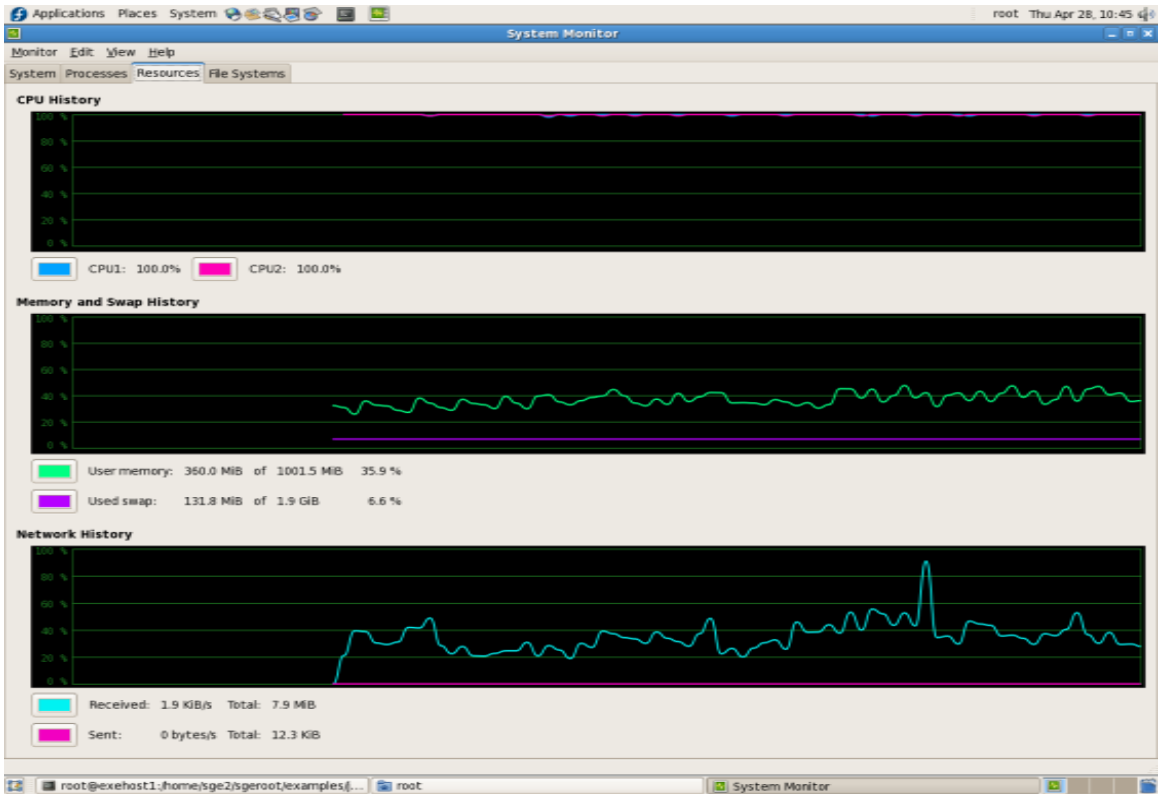


Figure-1: Immediate submissions of the jobs

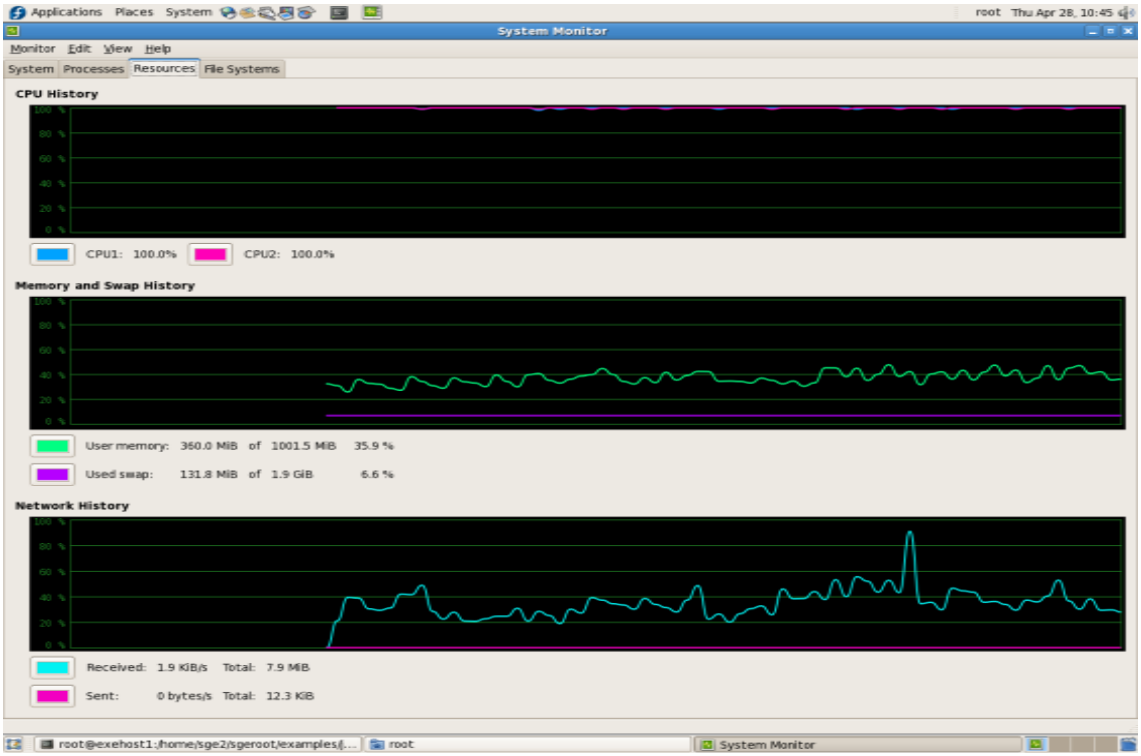


Figure-2: After 1 minute

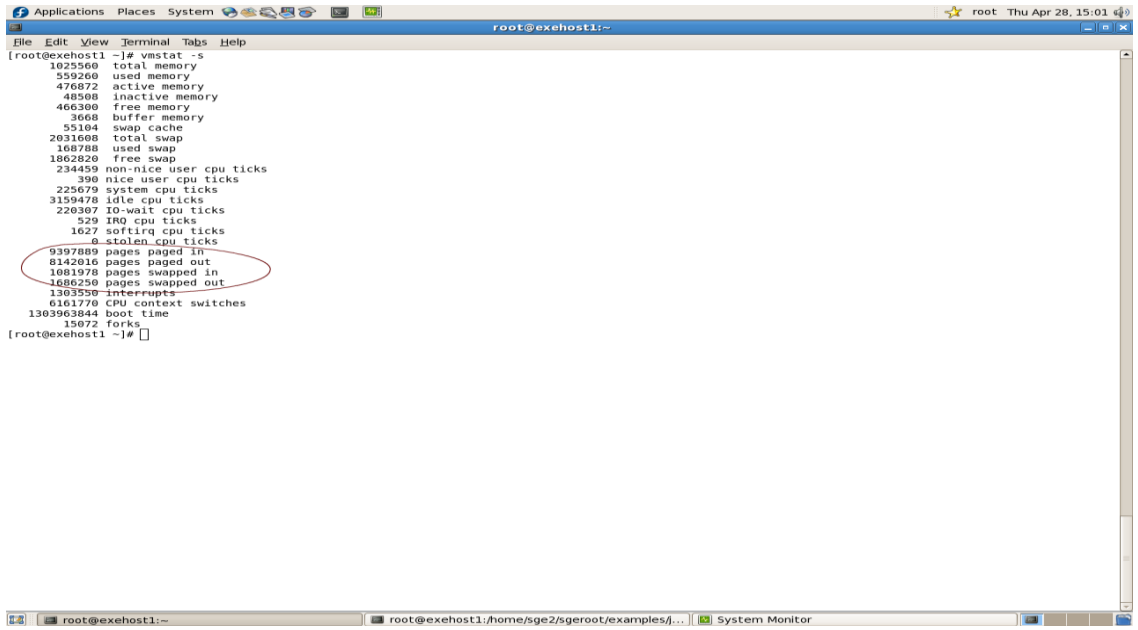


Figure-3: Number of swap-ins and swap-outs after 1minute

Figure-4 depicts the situation of increased swap memory, where the CPU utilization is zero. Within 5 minutes swap memory is increased to 7.4% from 6.6% of total swap memory which is of 1.9GB. And the total memory consumption is almost 85.1%. Figure-5 depicts the above situation numerically.

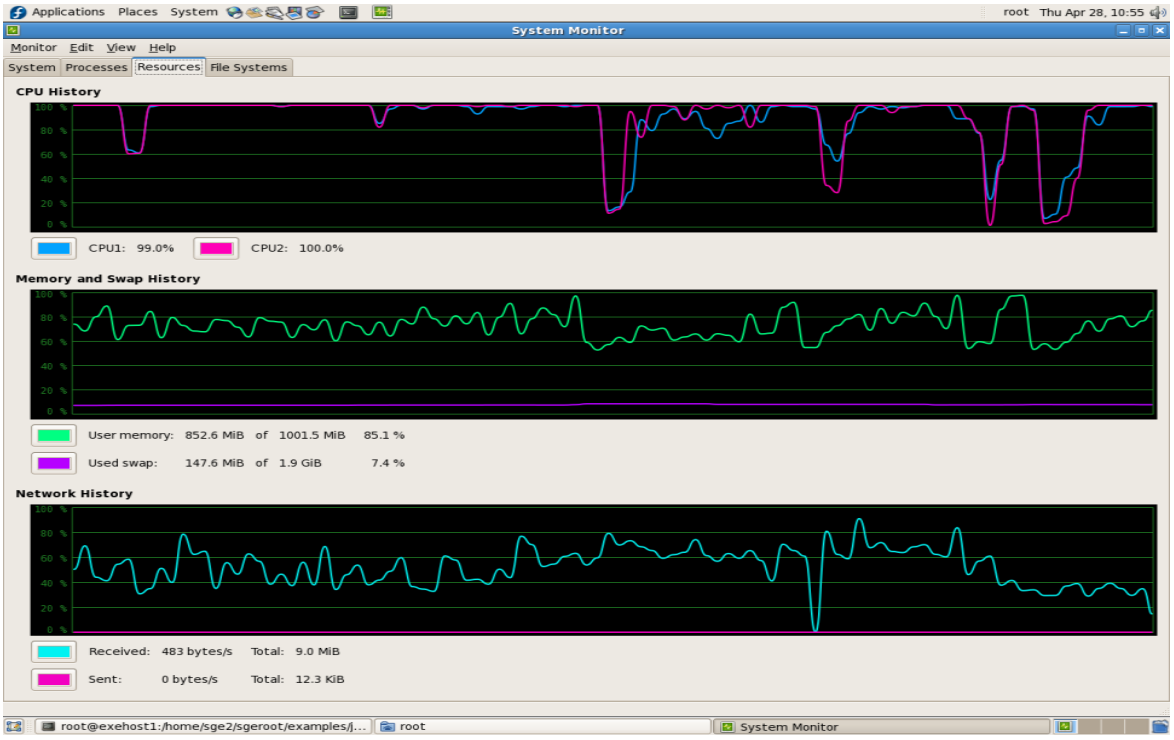


Figure-4: After 5 minutes (CPU is idle, increased swap-in, swap-out)

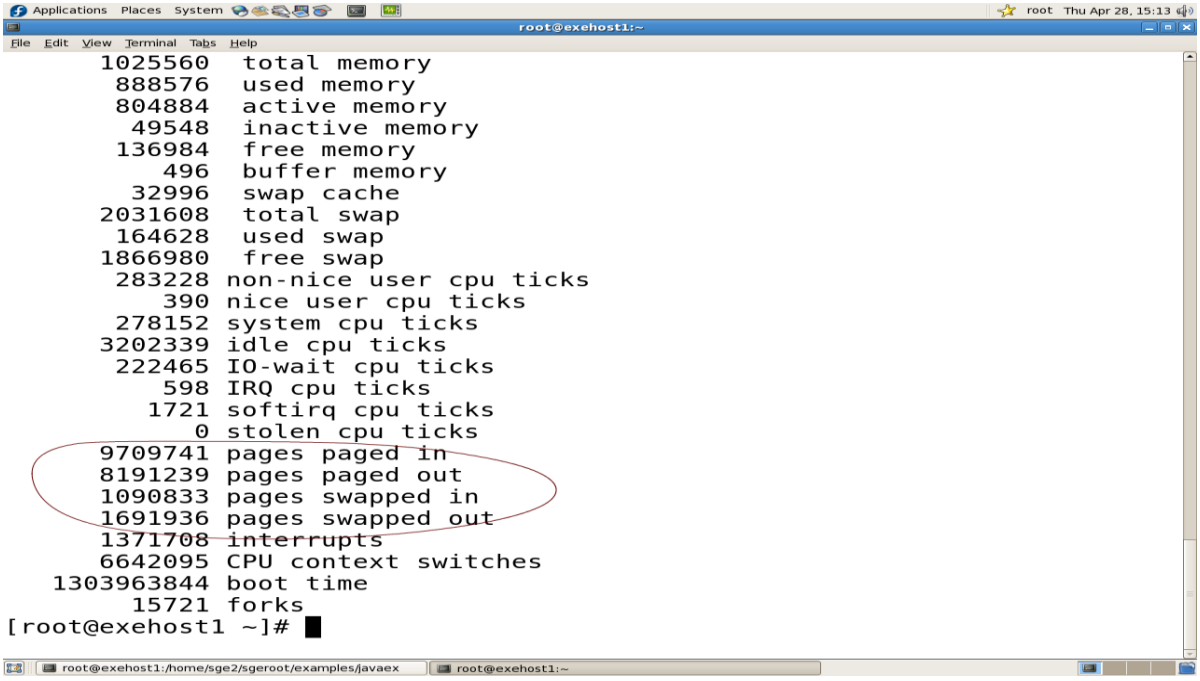


Figure-5: Number of swap-ins and swap-outs after 5 minutes

After 10 minutes CPU is almost idle in situation. Memory utilization is 85% of 1GB. Swap memory used is 22%(i,e.,. No. of pages swapped in and swapped out are also high).This situation is demonstrated by Figure-6.

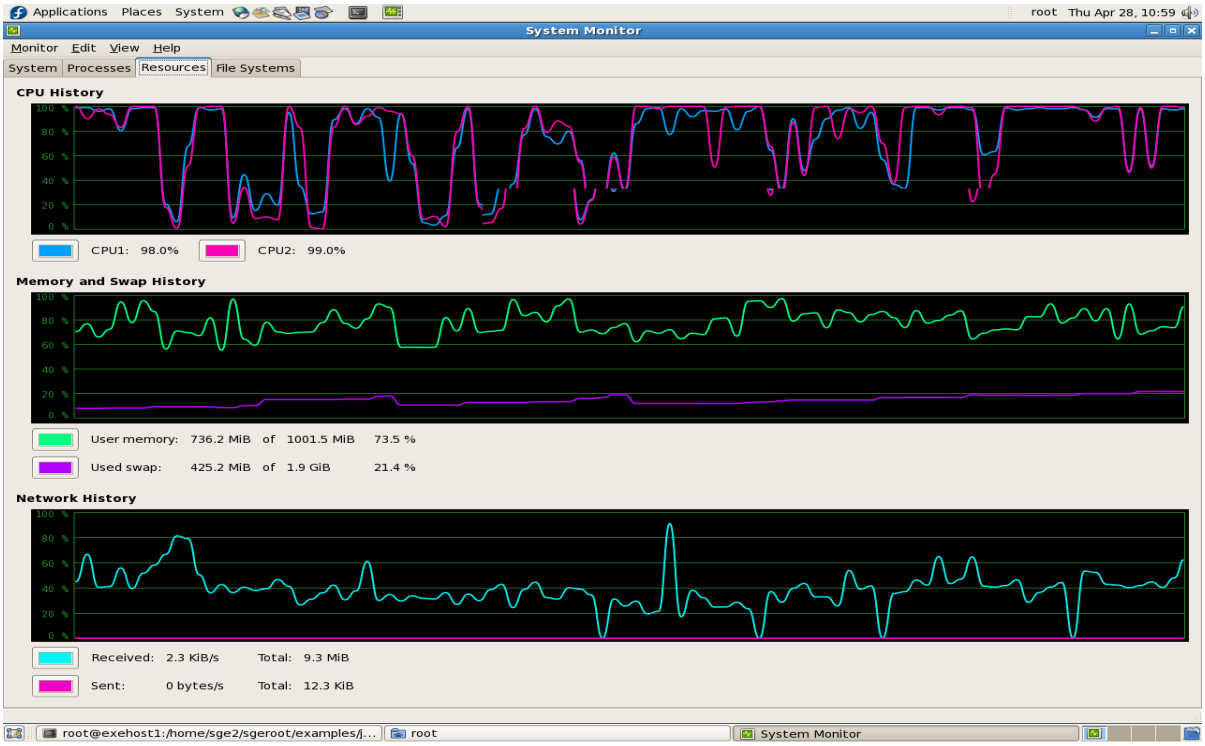


Figure-6: After 10 minutes

When 2 such jobs are killed manually, on a sudden memory consumption was decreased and reached to 20%. And swap memory consumption was decreased to 10%. This situation leads to a serious problem called Thrashing. And this can be demonstrated by Figure-7. Figure-8 demonstrates a system generated bug report, when the system was in thrashing situation.

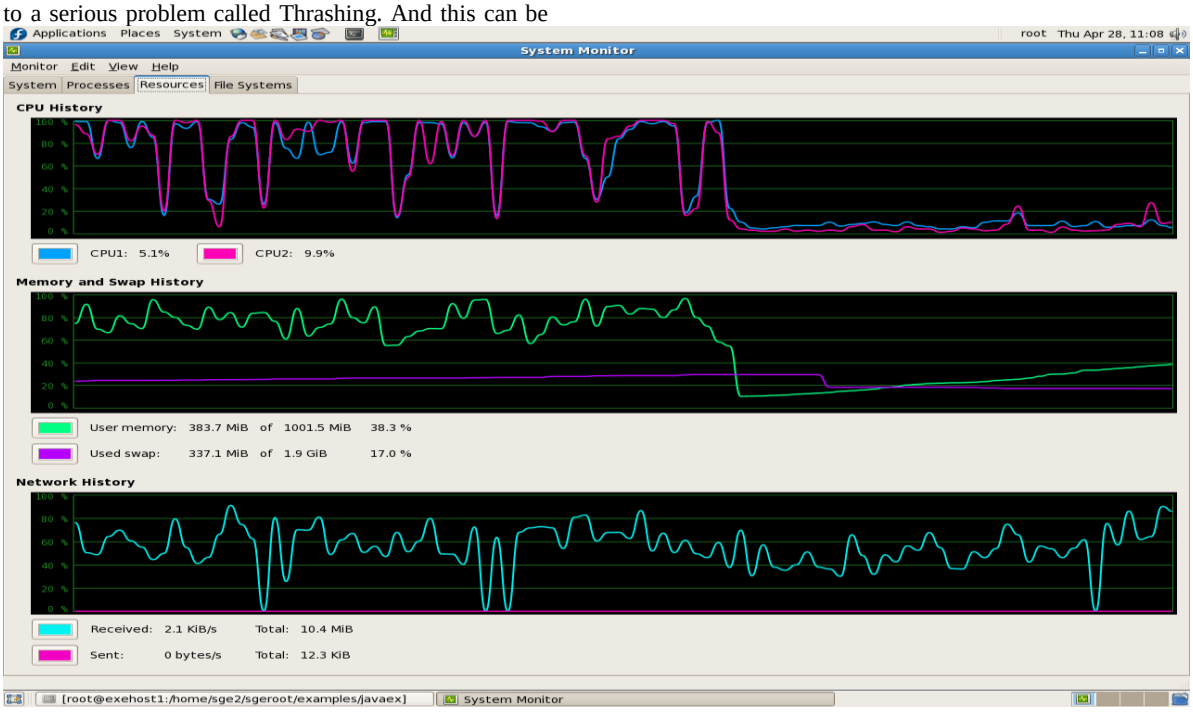


Figure-7: After killing 2 jobs

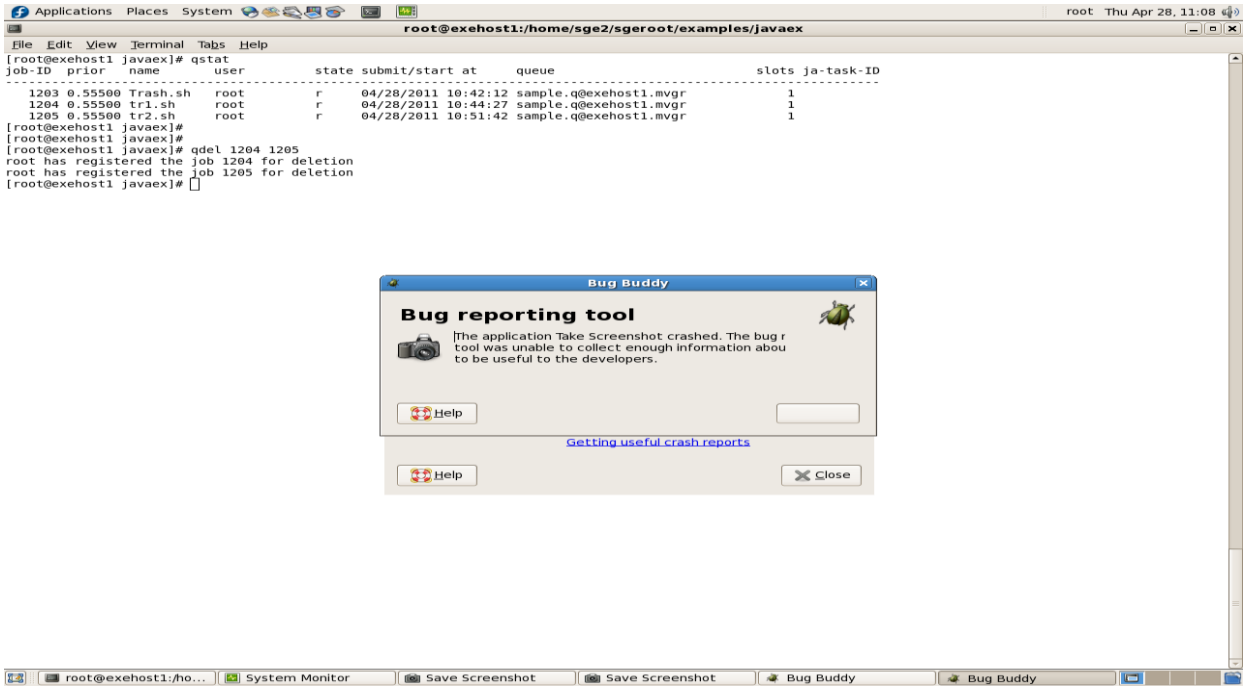
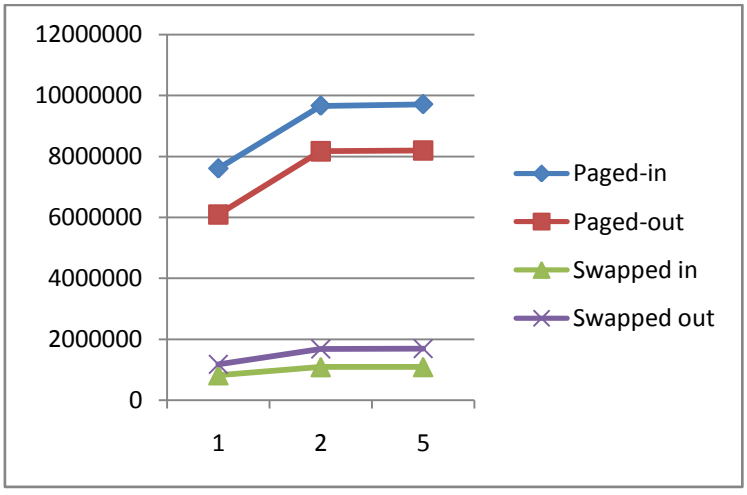


Figure-8: System generated Bug tool (due to thrashing)

Table – 1: Statistical Analyses

Job status	Paged-in	Paged-out	Swapped-in	Swapped-out
Immediate submission	7602185	6091160	823199	1180388
After 2minutes	9661133	8167407	1088774	1686586
After 5 minutes	9709741	8191239	1090833	1691936

The Table-1 and Graph-1 shown above demonstrates clearly that the number of page faults and number of swap-ins and swap-outs occurred during different time slots. Hence, in order to prevent the problem of thrashing, we propose a novel approach for load scheduling based on past history. The algorithm is demonstrated in the next section.



Graph-1: Page faults at different time slots

4. PREDICTIVE SCHEDULING

In this paper we propose a new approach for job scheduling based on past history. The below diagram explains the implementation of the wrapper scheduler proposed in this paper.

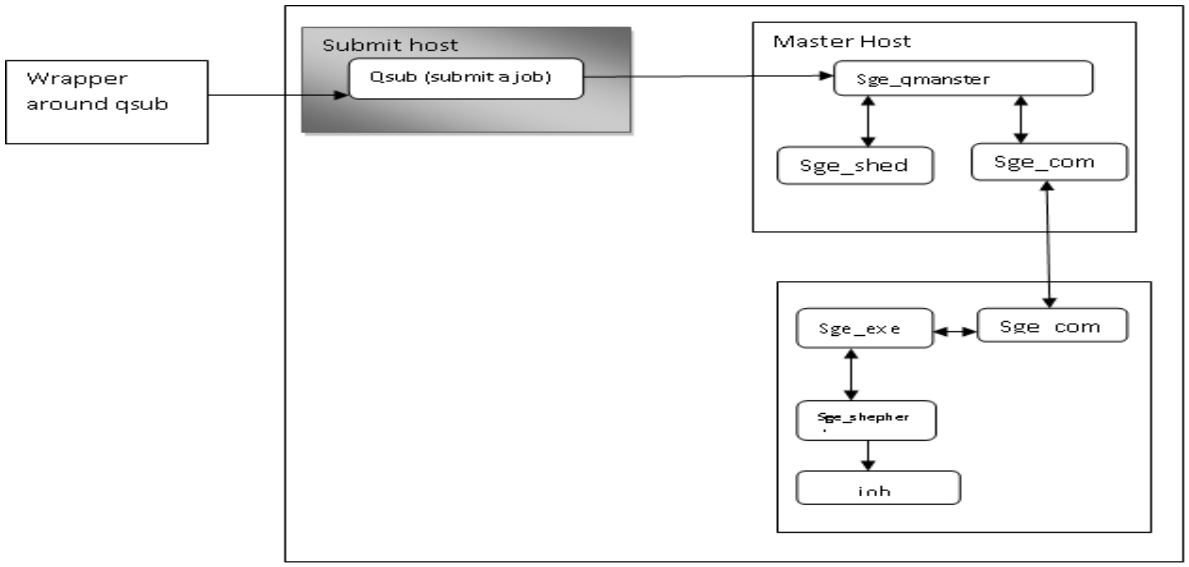


Figure-9: Wrapper based Grid Architecture

Algorithm

Pre-Requisites: Past history should be maintained for the jobs to be submitted

Step-1: When a new job arrive for job submission.
Check whether this job has submitted previously or not (by qacct -j job name)

Step-2: If it is submitted previously retrieve the related information, i.e., Load values

Step-3: Check the details of jobs which are already submitted to SGE and still in running state

Step-4: Retrieve the load values for every job which is in running state throughout their life cycle.

a. Retrieve the initial memory consumption of job to be submitted (Say m1).
b. Retrieve the memory consumption after n1 minutes, where n1= total time elapsed from job submission to now (Say, m2).
c. Retrieve the memory consumption after n1+2 minutes (Say, m3).
d. Retrieve the memory consumption of first 2 minutes of job to besubmitted (Say, m4).
e. Calculate the total memory available at present (Say, m6).

Step-5: Now compute the total memory consumed,
M5=m1+m2+m3+m4;

Step-6: if (m6>m5) then submit the job to the SGE and delete the job from queue.
Else make the job to wait for submission in queue

Step-7: Repeat the above steps from 2 to 6 for every 3 minutes.

5. RESULTS AND DISCUSSIONS

According to algorithm, when a job is submitted to the queue, each submitted job will be verified that the job can be submitted to the Grid Engine System based on the criteria of memory consumption. If the job can be submitted to the Grid Engine System, then the job will be submitted to the SGE Grid Engine along with its execution parameters and the submitted job will be deleted from the Queue. Otherwise job will be in the queue till the memory is available.
The Figure-10 depicts the situation of submitting jobs to the wrapper queue. The jobs named loadck.sh, loadck1.sh are submitted to the queue which is wrapper around SGE. These jobs are in turn submitted to the Grid Engine's queue (Say sample.q or all.q)

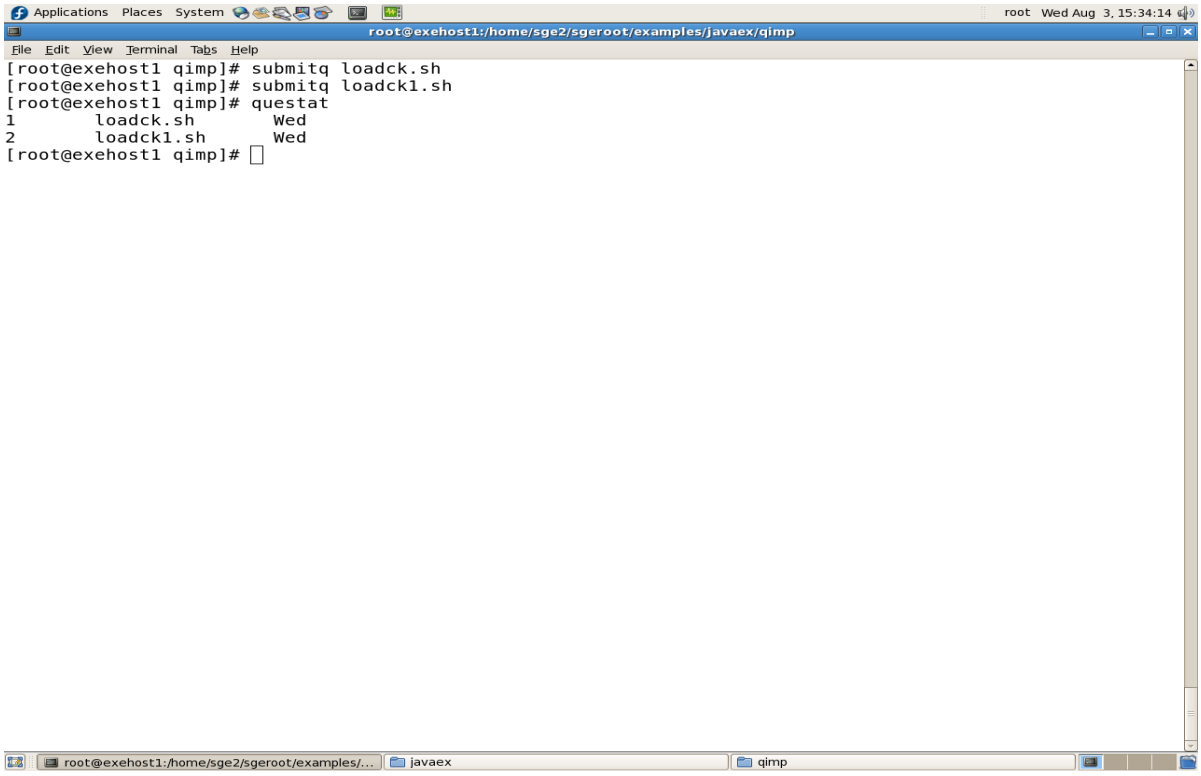


Figure-10: Submitting jobs to the wrapper

The Figures-11, Figure-12 and Figure-13 illustrates the situation of algorithmic results. Here none of among 2 is submitted to Grid Engine's queue. In Grid Engine queue Hello.sh job is in running state which was submitted earlier to the loadck.sh and loadck1.sh. According to the modified load balancing function, the loadck.sh is in first priority.

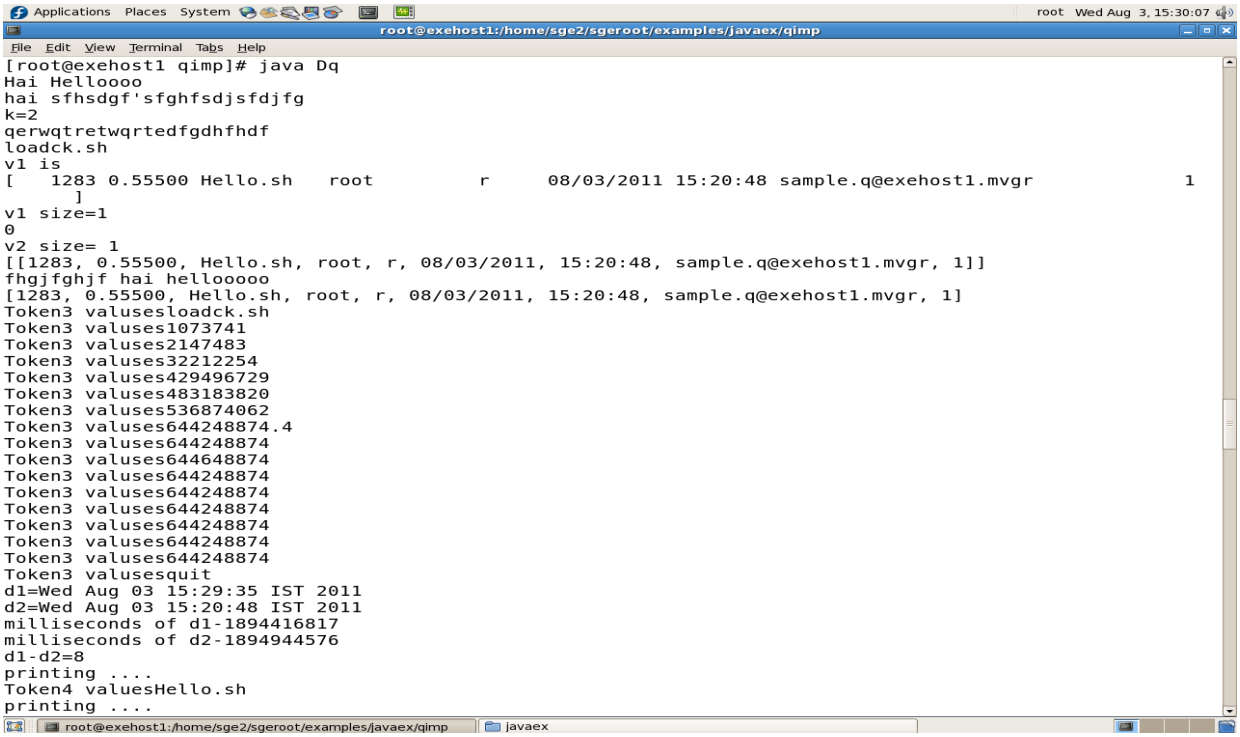


Figure-11: Algorithmic results

The past history of loadck.sh and Hello.sh will be taken from the file system and these values will be compared against the available memory. If the available memory is more than to be consummated memory then the job will be submitted by the wrapper queue to Grid Engine's queue. Otherwise the job remains in wrapper queue. If a job is submitted to Grid Engine's queue, that job is deleted from the wrapper queue.

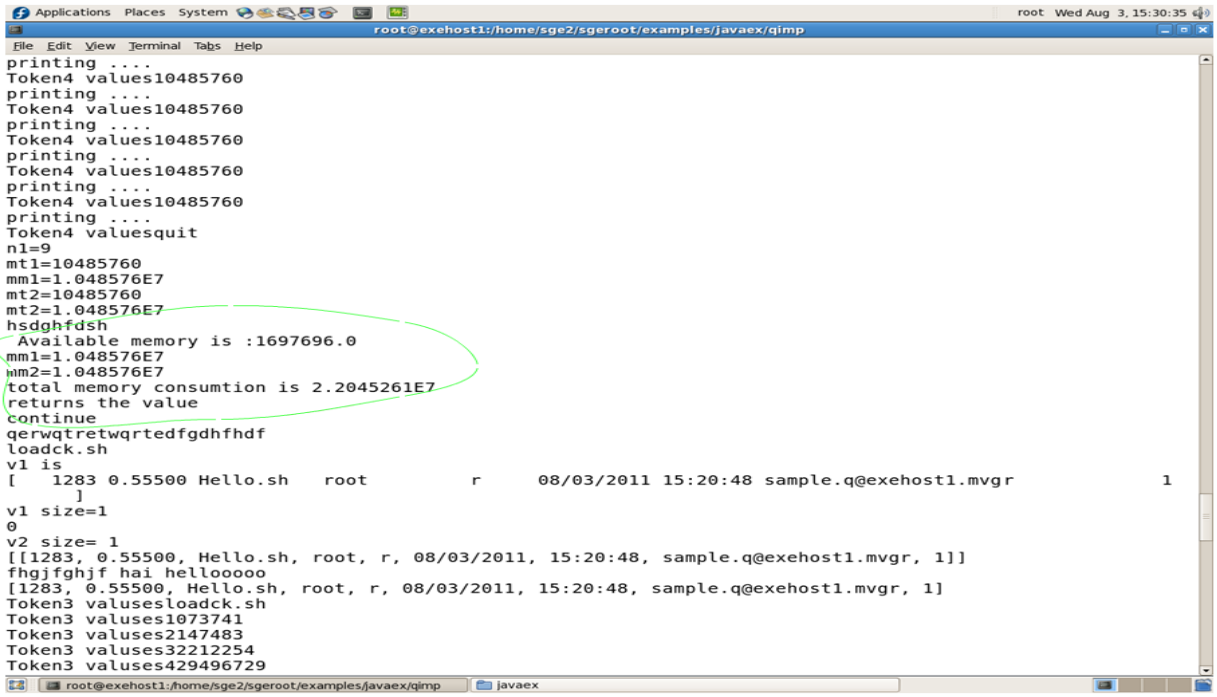


Figure-12: Algorithmic results

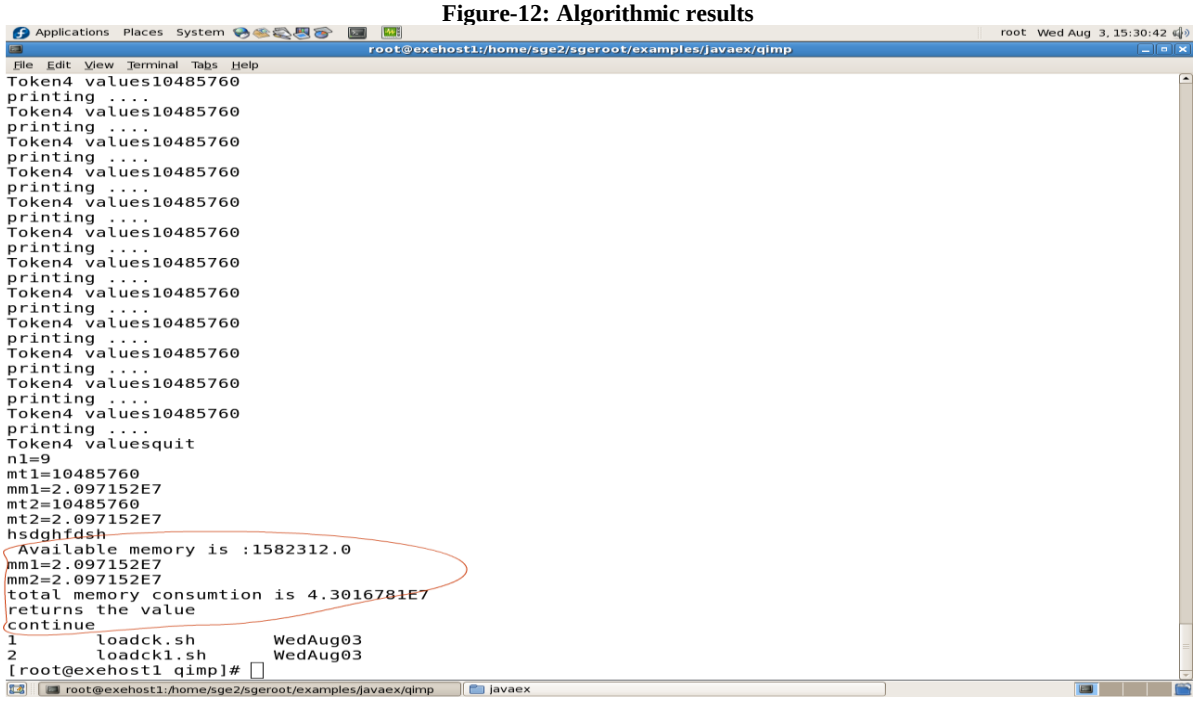


Figure-13: Result of algorithm

The Figure-14depicts the result of wrapper queue after submitting the jobs to the Sun Grid Engine System. Here the wrapper queue contains only one job loadck.sh, loadck1.sh which ware submitted earlier. None of the job is submitted to the Grid Engine System as the memory is insufficient to load the loadck.sh, loadck1.sh. And the Grid Engine remain contains Hello.sh batch job only init's queue. This situation is illustrated by the Figure-15. And this also shows the number of jobs that are running in Grid Engine System along with their job ids and submission time after the algorithm has been worked out.

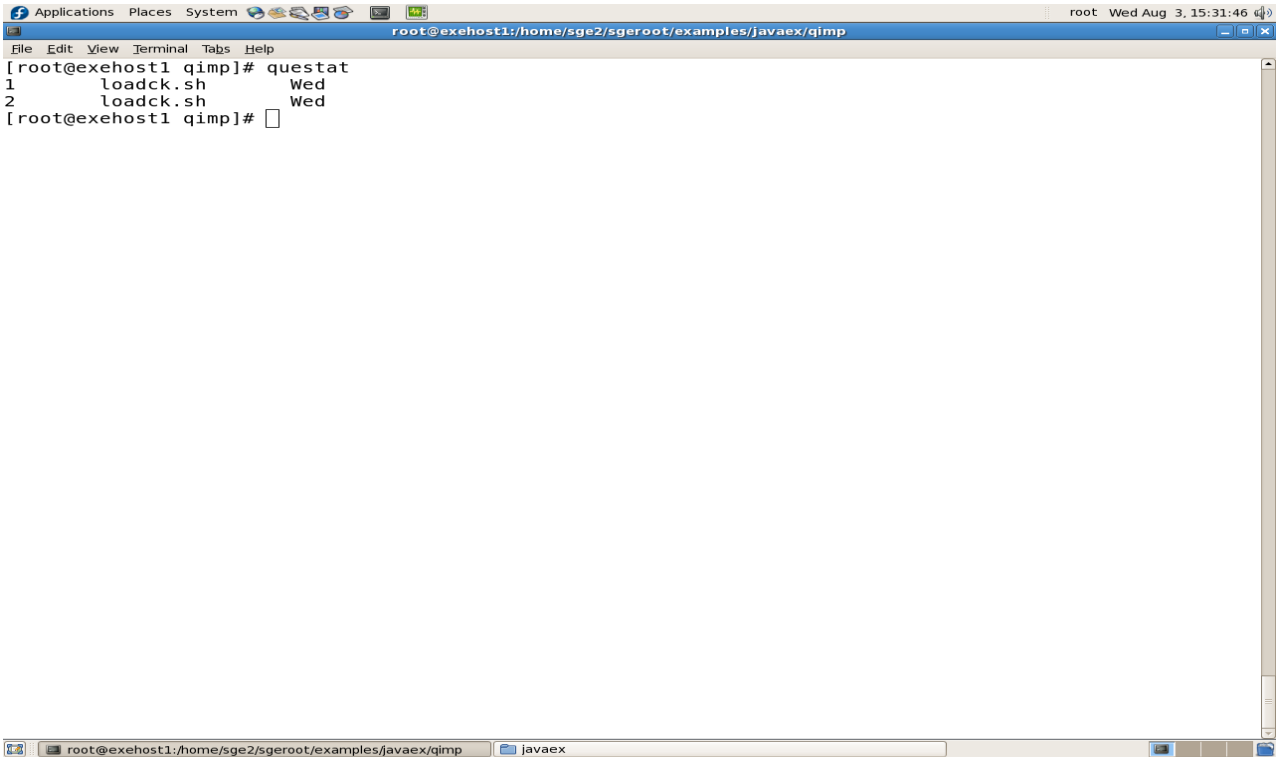


Figure-14: Result of Wrapper queue

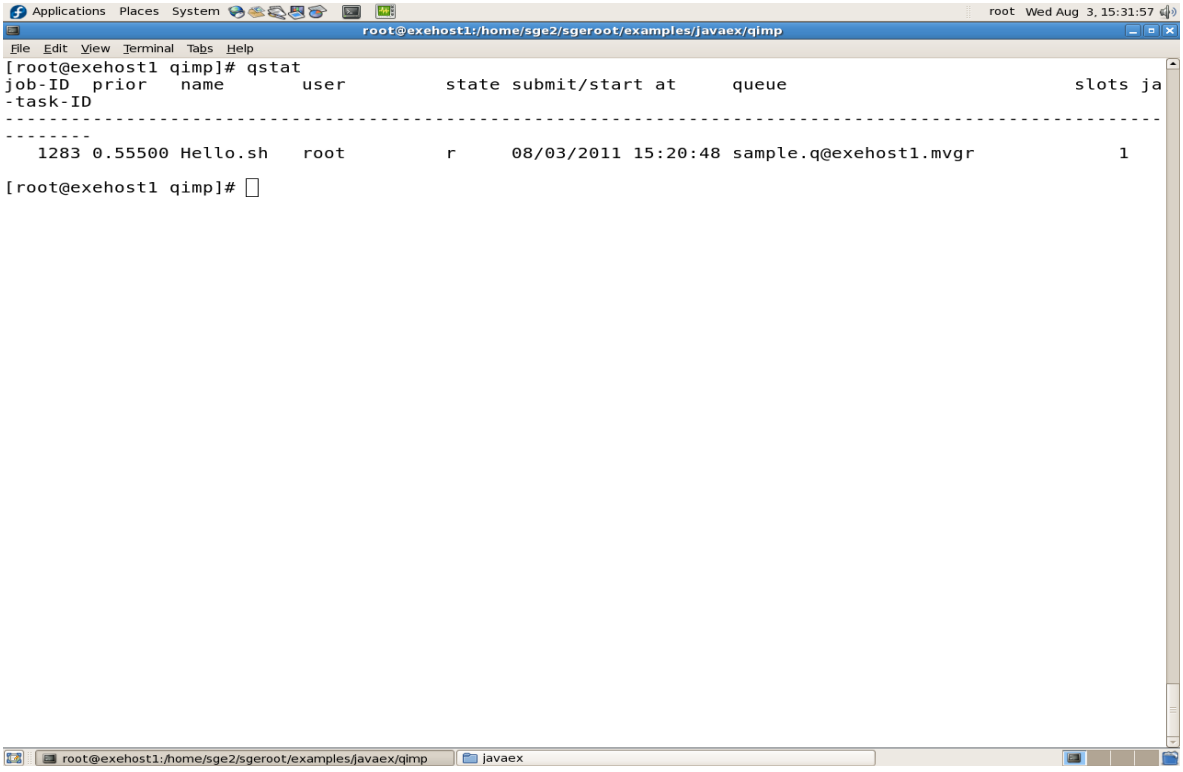


Figure-15: Result of qstat

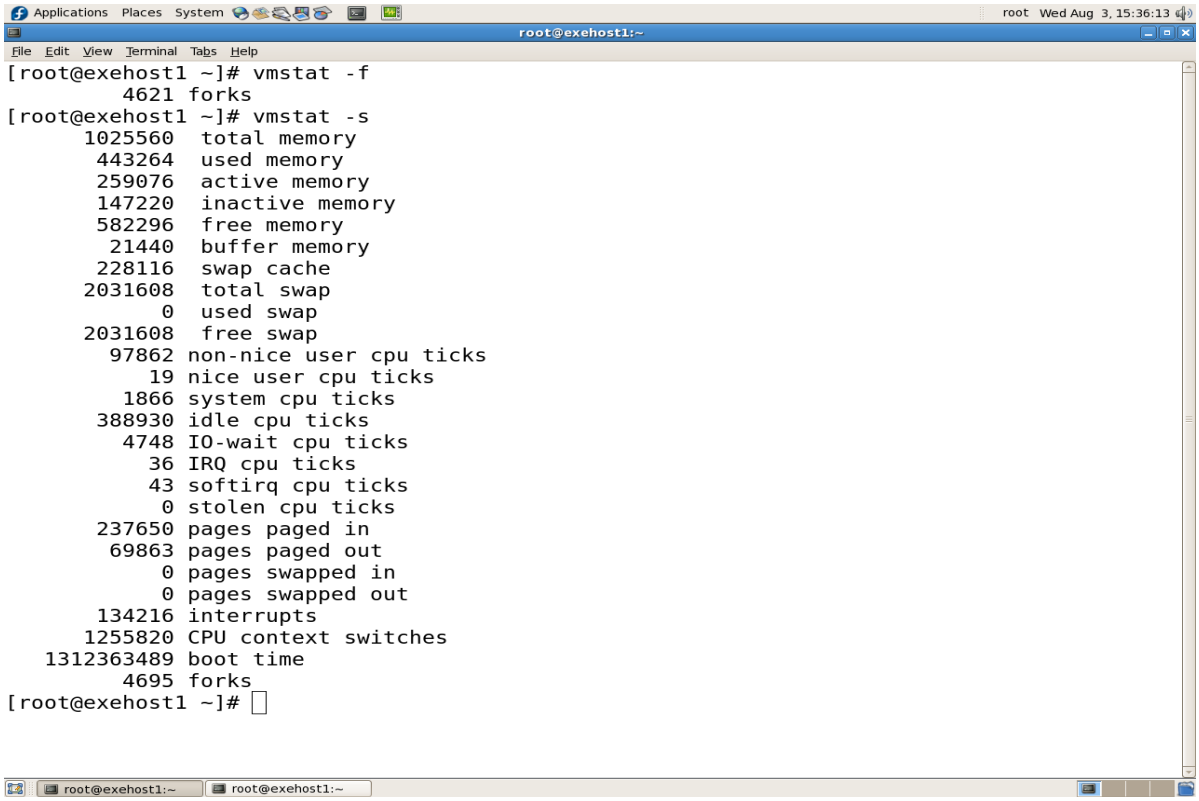


Figure-16: Numerical status of swap-ins and swap-outs after 16 minutes

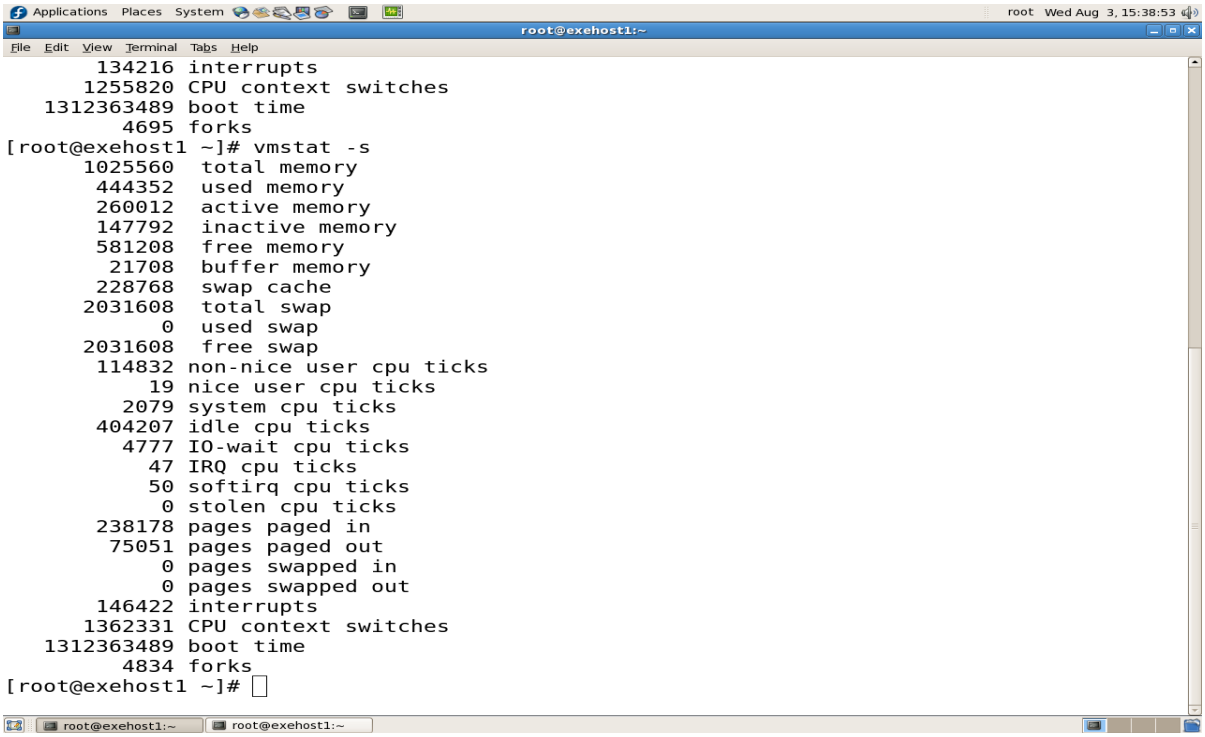


Figure-17: Numerical status after 20 minutes

Figure-16and Figure-17 illustrates the situation that number of pages swapped-in and swapped-out during the submission of jobs loadck.sh and loadck1.sh. The above results shows that number of pages swapped in swapped out are zero (0) as the memory is sufficient to load the batch job Hello.sh. Thus the modified load balancing function can prevent the situation of Thrashing by preventing the job being submitted to the Grid Engine when memory is not as sufficient as much to load the job.

6. CONCLUSION

In the recent years, the utilization of grid computing has increased extensively in various research organizations, companies and universities which force the improvement in the efficiency of grid environment. The job scheduling process has an important role to play in any grid computing

environment especially when the I/O intensivejobs are submitted repeatedly which is causes the problem of thrashing. Hence, the proposed algorithm effectively encounters this problem. At the outset, our approach outperforms the existing job scheduling algorithms.

7. REFERENCES

[1] Foster. I and Kesselman, C.: "The Grid2: Blueprint for a new computing infrastructure", Elsevier Inc., Second Edition,2004

[2] Swarna. M, Sitharama Raju. P and Nagesh V, "A Novel Approach for Load Balancing in Grid Computing", International Journal of Computer Science & Informatics, 1(2): 93-96, 2011

[3] SunGridEngine: http://www.sun.com/software/Gridware/

[4] Condor, http://www.cs.wisc.edu/condor/

[5] Foster.I and C.Kesselman," Globus: A metacomputing infrastructure toolkit",1997

[6] A. Natrajan, A. Nguyen-Tuong, M.A. Humphrey, M. Herrick, B.P. Clarke andA.S. Grimshaw, "The Legion Grid Portal", Grid Computing Environments, Concurrency and Computation: Practice and Experience, 14(13-15): 1365-1394, 2001.

[7] N.H. Kapadia, J.A.B. Fortes, PUNCH: an architecture for webenabled wide-area network-computing, Cluster Computing, The J. Networks, Software Tools Appl. 2 (2) (1999) 153–164.

[8] Eurogrid. http://www.eurogrid.org.

[9] Buyya,R, Abrason,D and Giddy(2000), "Nimrod-G: An architecture for a resource management and scheduling system in a global computational Grid", The 4th International conference on High performance Computing in Asia-Pacific Region(HPC Asia 2000), Beijing, China,2000.

[10] Buyya, R., Giddy, J. and Abramson, D. (2000) An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications, *TheSecond Workshop on Active Middleware Services (AMS 2000), In Conjunction with HPDC 2001*. Pittsburgh, USA: Kluwer Academic Press.

[11] RaihanurRasool, GuoQingping and Zhou Zhen, "Users-Grid: A Unique and Transparent Grid – Operating System", Ubiquitous Computing & Communication Journal, 1(1), 2006.

[12] L.Yang, Jschopf and I.Foster, "Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments", Proceedings of the 2003 ACM/IEEE conference on Supercomputing, pp:33-48, IEEE Computer Society, 2003

[13] Joshy Joseph, Craig Fellenstein, "Grid Computing", IBM Press, 4thEdn., 2008.

[14] Dr. D.I. George Amalarethinam, P. Muthulakshmi,"An Overview of the Scheduling Policies and Algorithms in Grid Computing", International Journal of Research and Reviews in Computer Science (IJRRCS), 2(2): 280-294, April 2011

[15] Clovis Chapman, Micro Musolesi, Wolfgang E and Cecilia Mascolo, "Predictive Resource Scheduling in Computational Grids", IEEE InternationalParallel and Distributed Processing Symposium, 2007. IPDPS 2007, pp.1-10.