EVALUATING SCHEDULING ALGORITHM IN GRID COMPUTING

Mr Vivek Sukla,Tapesh Kumar Roshan,Dr Rohit Miri

ABSTRACT

This paper presents the results of a study of a heterogeneous computational grid using different scheduling algorithms. After a definition of several algorithms based on the concept of work completion, efficiency is discussed; a new algorithm in grid computing is presented. Two well known scheduling algorithms and our own algorithm are then compared against each other, and it is proved that our algorithm has the highest computing efficiency performer.

In a shared environment, Good schedules involve the integration of specific information. The application performance may suffer when some resources represent abnormal usage pattern during applications execution. We study a performance-prediction based task scheduling system, which provides scheduling based on system-level performance prediction. The efforts to construct a grid computing environment have brought unprecedented computing capacity. Exploiting this complex infrastructure requires efficient middleware to support the execution of a distributed application, composed of a set of subtasks, for best performance. This presents the challenge how to schedule these subtasks in shared heterogeneous systems. For, this we develop our own algorithm whose computational efficiency is very less than the existing algorithm.

Index Terms: grid computing, scheduling, heterogeneous systems, distributed systems.

I. INTRODUCTION

The major goal of distributed computing research was to give users an easy, simple and transparent method of access to a vast set of heterogeneous resources. This is generally Known as metacomputing. Metacomputing done on local area networks (LAN) are typically known as Cluster Computing Environments and those, which are done on wide area networks (WAN), are known as Grid Computing.

This paper explores the results of using several well-known scheduling algorithms to schedule work on a grid we present a algorithm for modeling computational grids and we developed to analyze different scheduling algorithms under a variety of workloads.

We then describe three scheduling algorithms and give the results of three experiments, the first to investigate the performance of the three algorithms compare to each other, and the second to investigate the effects of variation in work completion times on a specific scheduler.

A. What are Grids?

The definition of a *computational grid* is still a subject of some debate. What follows here is a short definition of a computational grid sufficient to give an adequate background for the rest of the paper; for a more in-depth definition, Let us begin by giving an intuitive definition of what we mean by a computational grid, a computational grid is a collection of *nodes*, each of which may be thought of as a system that can perform work and has access to a network. Many systems share this property, including computer clusters. However, a grid is unique in that nodes on the grid vary in capability, and that the grid may provision more nodes to do work, or release nodes from the grid at any time. In addition to these unique properties, the grid also has several properties stemming from its distributed nature, namely, that nodes may fail (become unable to perform work due to software or hardware problems) at any time, and communication efficiency between nodes can vary widely.In addition, nodes are typically .

A computational grid, then, can be seen as an adaptive system that provisions extra computational capacity as demand requires or as machines fail, and assigns work to nodes where the work can be done most effectively.As promising as such a description of a full-fledged grid sounds , a number of obstacles remain before systems that have all the above properties can even be constructed , much less utilized effectively. For our purposes, we will study computational grids that are one step removed from their cluster counterparts: our grids will not expand or contract over time, nor will nodes fail, nor will there be significant communication delays between nodes; however, our grids will be made of nodes of heterogeneous capability. By studying grids of this type, we hope to enable an evolutionary approach to studying more complex grids.

Costs and Benefits

No need to buy large six figure SMP servers for applications that can be split up and farmed out to smaller commodity type servers. Results can then be concatenated and analyzed upon job(s) completion.

Much more efficient use of idle resources. Jobs can be farmed out to idle servers or even idle desktops. Many of these resources sit idle, especially during off business hours. Policies can be in place that allow jobs to only go to servers that are lightly loaded or have the appropriate amount of memory/CPU characteristics for the particular application.

Grid environments are much more modular and don't have single points of failure. If one of the servers/desktops within the grid fail, there are plenty of other resources able to pick the load. Jobs can automatically restart if a failure occurs.

Policies can be managed by the grid software. The software is really the brains behind the grid. A client will reside on each server which send information back to the master telling it what type of availability or resources it has to complete incoming jobs.

This model scales very well. Need more compute resources? Just plug them in by installing grid client on additional desktops or servers. They can be removed just as easily on the fly. This modular environment really scales well.

Upgrading can be done on the fly without scheduling downtime. Since there are so many resources some can be taken offline while leaving enough for work to continue. This way upgrades can be cascaded as to not affect ongoing projects.

Jobs can be executed in parallel speeding performance. Grid environments are extremely well suited to run jobs that can be split into smaller chunks and run concurrently on many nodes. Using things like MPI will allow message passing to occur among compute resources.

LIMITATIONS

For memory hungry applications that can't take advantage of MPI you may be forced to run on a large SMP.

You may need to have a fast interconnect between compute resources (gigabit ethernet at a minimum). Infiband for MPI intense applications

Some applications may need to be tweaked to take full advantage of the new model.

Licensing across many servers may make it prohibitive for some apps. Vendors are starting to be more flexible with environment like this.

Grid environments include many smaller servers across various administrative domains. Better tools for managing change and keeping configurations in sync with each other can be challenging in large environments.

Political challenges associated with sharing resources (especially across different admin domains). Many groups are reluctant with sharing resources even if it benefits everyone involved. The benefits for all groups need to be clearly articulated and policies developed that keeps everyone happy.

PROBLEM DESCRIPTION

Our problem is to develop an algorithm which in comparison to other existing algorithms have less waiting time and the system operating on that algorithm have higher computing efficiency. The Object grid computing is the integration of heterogeneous computing systems and data resources with the aim of providing a global computing space such that all the systems attached to that grid get benefited simultaneously . The efficient use of resources from different destinations spread over a large area globally.

PROBLEM FORMULATION

We are assigning some jobs to the systems following the different algorithm and comparing the waiting time obtained from the different system.

The goal of grid computing, which gets its name from its gridlock architecture, is to link surplus computing power and other spare IT resources with clients who have periodic needs beyond the capacity of their machines.

PRACTICAL IMPLEMENTATIONS

TKR ALGORITHM

In this algorithm, we implement a special type of technique to improve processor efficiency. We consider three different nodes and each node have two processor (or system). Firstly, all the processor is assigned with a process. Before executing the process, the deadline time of the remaining process is checked with the existing process's deadline time. If the existing deadline time is more than the remaining deadline time, the remaining one will execute first and the existing process will wait for the processor to get free. So, this way all the remaining process will be compared and assigned to the processor.

If a processor executes his process within deadline time then other waiting process will start to execute without waiting to complete his deadline. So computation efficiency is increased.

ALGORITHM:-

Step 1: initialize process, their execution time and their deadline, waiting time and status of processor.
For i=1 to i=10

```
{
enter ex[i];
enter dead[i];
wait[i]=0;
}
For j=1 to j=6
p[i] =0;
```

Step 2: assign first six processes into processors.

Step 3: for remaining process

```
for k=7 to k=10
{
if (p[1]==0 && dead[1]>dead[k])
{
wait[1]+=ex[k];
p[1]=1;
}

else if (p[2]==0 && dead[2]>dead[k])
{
wait[2]+=ex[k];
p[2]=1;
}

else if (p[3]==0 && dead[3]>dead[k])
{
wait[3]+=ex[k];
p[3]=1;
}

else if (p[3]==0 && dead[3]>dead[k])
{
wait[3]+=ex[k];
p[3]=1;
}

else if (p[4]==0 && dead[4]>dead[k])
{
wait[4]+=ex[k];
p[4]=1;
}

else if (p[5]==0 && dead[5]>dead[k])
{
wait[5]+=ex[k];
p[5]=1;
}

else if (p[6]==0 && dead[6]>dead[k])
{
wait[6]+=ex[k];
p[6]=1;
}

else
{
choose minimum execution time of process
small=minimum(execution of 6 processes)
then
wait[k]+=small;
}
}
```

Step 4: calculate waiting time of each process
calculate turnaround time of each process
calculate total turnaround time of each process

Step 5: Exit

Scheduling

The grid scheduler assigns applications to nodes. Each node can run one application at a time, and must run that application to completion. (By splitting multi cpu machines into a set of single cpu nodes, powerful machines may in fact run more than one job at a time.) The scheduler maintains a queue of work for each node, and may re-order the queue at any time. An application that is being run but whose previous application has not finished will wait until the previous application has finished before starting. (This is a simplifying assumption: in reality, some applications may begin before their previous application has finished. For instance , a sequence of applications that are streaming data to each other has this property.) The scheduler is invoked when *mapping events* occur. During a mapping event, the scheduler may reorder each node's work queue (thereby *mapping* applications to nodes), and assigns new jobs that have arrived to various nodes. In general, mapping events
occur whenever

- a new job arrives
- a node enters or exits the grid
- an application finishes or is aborted
- acceptable robustness/performance changes

However, for the purposes of this simulation, mapping events are only fired when new jobs arrive, and when applications finish. In general, a robust scheduler will need to take robustness into account when making mapping decisions. It is interesting to ask, however, how well scheduling algorithms which are not aware of robustness will perform. We have selected the following three scheduling algorithms to study.

FCFS : First come, first serve. Maintains a queue of applications,in the order they arrive at the grid, and assigns them to the nodes in the order the nodes become available. When an application finishes running, if there is another application to run to complete that application's associated job, then that application will be added to the end of the incoming work queue. Thus, this is a FCFS algorithm on applications, not jobs.

SJF : Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run next. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst.The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.The SJF algorithm favors short jobs (or processors) at the expense of longer

ones.The obvious problem with SJF scheme is that it requires precise knowledge of how long a job or process will run, and this information is not usually available.

TKR ALGORITHM

In this algorithm, we implement a special type of technique to improve processor efficiency. We consider three different nodes and each node have two processor (or system). Firstly, all the processor is assigned with a process. Before executing the process, the deadline time of the remaining process is checked with the existing process's deadline time. If the existing deadline time is more than the remaining deadline time, the remaining one will execute first and the existing process will wait for the processor to get free. So, this way all the remaining process will be compared and assigned to the processor.

If a processor executes his process within deadline time then other waiting process will start to execute without waiting to complete his deadline. So computation efficiency is increased.

LITERATURE REVIEW

The concept of grid has emerged as a new approach to high performance distributed computing infrastructure. In general, Grids represent a new way of managing and organizing computer networks and mainly their deeper resource sharing. In this project we are trying to develop a new algorithm which will be more better than the existing one. We are supposing three nodes which will consist of two processor each.Therefore we have six processor in total.We are providing ten processes in total to the six processers. Now we are trying to execute those proceses with the help of all the six processors such that the average waiting time recorded will be less than the waiting time of the existing algorithm.We are working on the algorithm which will be use to execute the processes more effectively with minimum waiting time and high computing efficiency.The existing algorithms have their own waiting time and thus the system following that algorithm have their particular computing efficiency. We are just comparing the existing algorithm with our algorithm and thus note down the difference in the respected waiting time as well as the computing efficiency.

RESULTS AND CONCLUSIONS

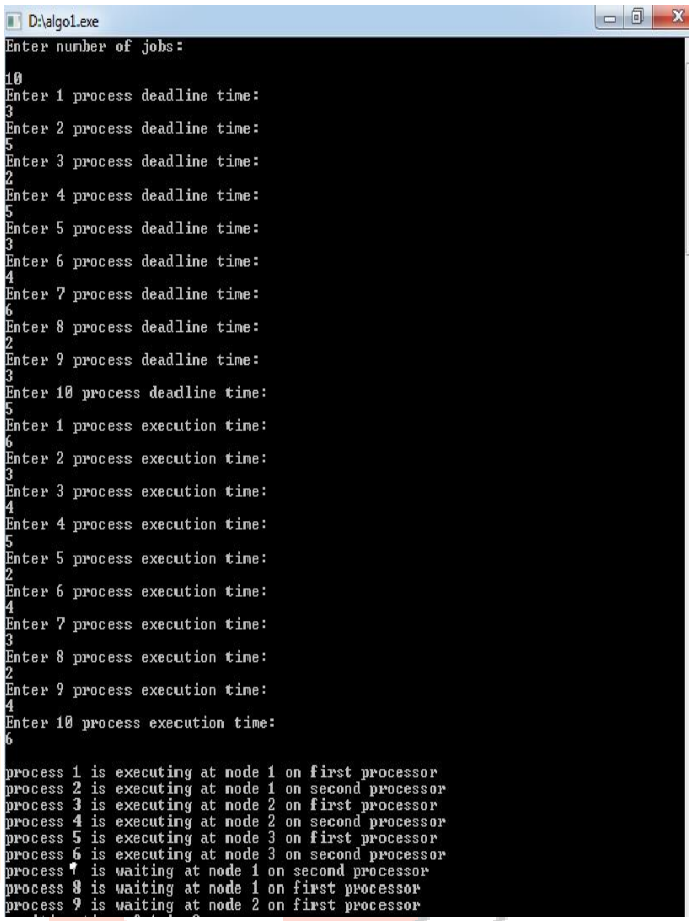


Fig1.Result from TKR algorithm

Fig 2. Result from TKR algorithm

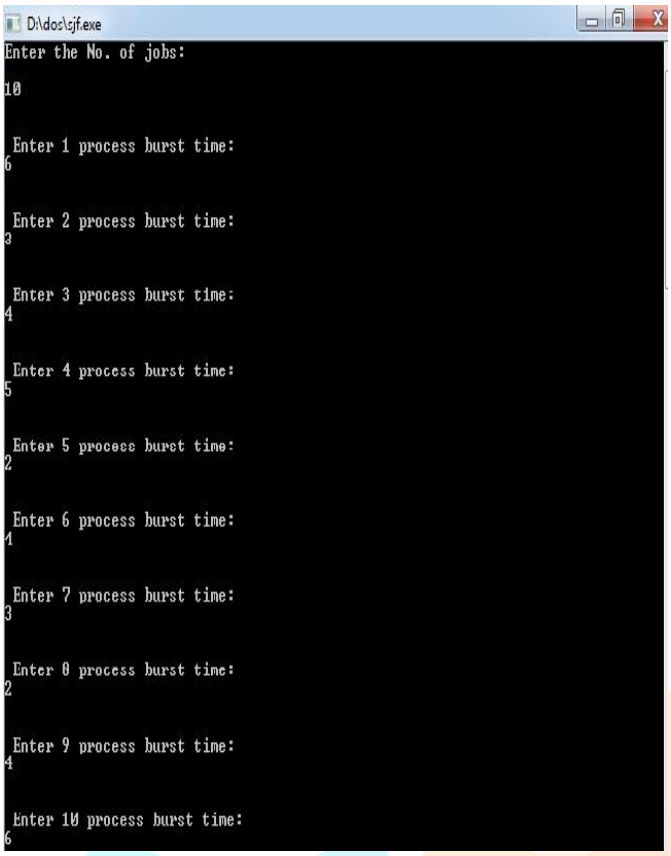


Fig. 3 : Result of Shortest Job First

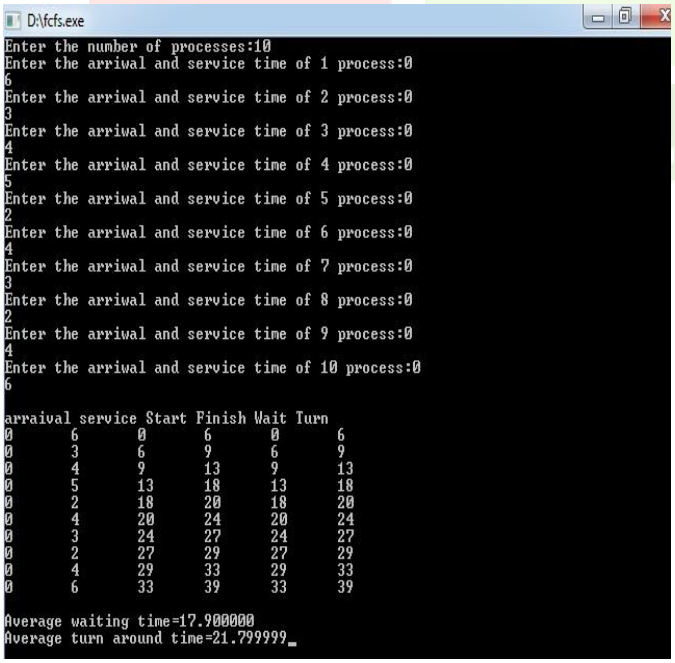
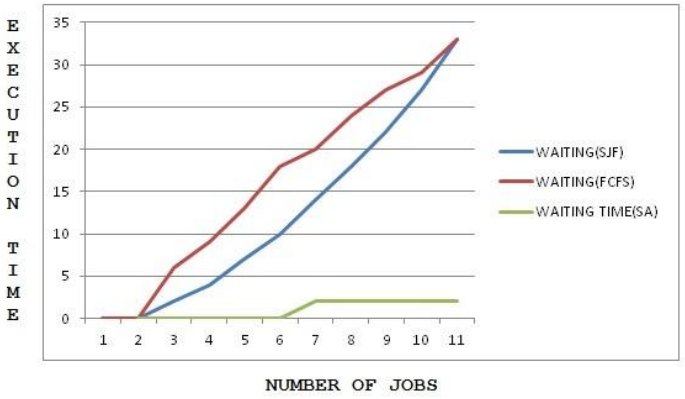


Fig 4. Result from first come first serve

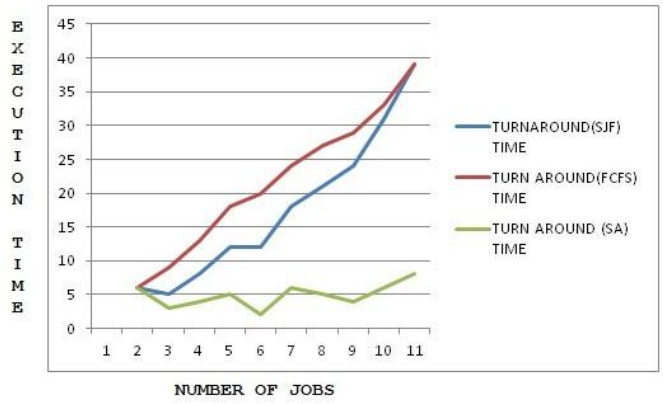
COMPARISION TABLE

				SHORTEST JOB FIRST		FIRST COME FIRST SERVE		**TKR ALGORITHM**	
Jobs	**DT**	**ET**	**BT**	**WT**	**TOT**	**WT**	**TOT**	**WT**	**TOT**
1	3	6	6	0	6	0	6	0	6
2	4	3	3	2	5	6	9	0	3
3	5	4	4	4	8	9	13	0	4
4	2	5	5	7	12	13	18	0	5
5	6	2	2	10	12	18	20	0	2
6	4	4	4	14	18	20	24	2	6
7	7	3	3	18	21	24	27	2	5
8	9	2	2	22	24	27	29	2	4
9	6	4	4	27	31	29	33	2	6
10	8	6	6	33	39	33	39	2	8
AVERAGE				**13.7**	**17.6**	**17.9**	**21.79**	**1.00**	**4.9**

COMPARISION GRAPH



COMPARISION GRAPH (WAITING TIME)



COMPARISION GRAPH (TURNAROUND TIME)

CONCLUSIONS

In conclusion, we believe that our project is of great use for many organizations and the systems on which they runs. We believe that the algorithm which we developed in this project help the task get executed with minimum waiting time. The algorithm is quite efficient in carrying out the task given to the system. It is very helpful in increasing the processor's efficiency and giving up the output in minimum time. It can work on multiple systems connected together in a network and thus carry out the number of tasks simultaneously. By implementing this we can get much faster response from the processors. By applying this algorithm we are getting major differences in waiting time, burst time, turnaround time etc.

Related Work

The concept of computational grids, and grid computing in general, is being studied by researchers in many fields, including high-performance computing, networking, distributed systems ,and web services is an extensive introduction to what a computational grid actually consists of, and what is required to implement it. The Globus Consortium (http://www.globus.org) is a consortium of dozens of companies, government agencies, and universities that is creating an open standard for grid development using web-services as an RPC mechanism. The modeling of computational grids with heterogeneous resources is just beginning to be explored. published in 2002,can point to no directly related work in the field. Scheduling tasks of unknown duration on distributed systems is investigated in The evaluation of scheduling algorithms focused on efficiency is explored in the construction of actual grids for industrial and scientific work has been undertaken by many companies and scientific groups. One particular success story is the Grid 2003 project (http://www.ivdgl.org/grid2003), which has developed a grid consisting of 2000 CPUs spread across the world.

.

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to Prof. Rohit Miri, HOD of Computer Science & Engineering, Dr C V Raman University, Kota Bilaspur, for his generous guidance, help and useful suggestions.

I also express my sincere gratitude to Mr. Vivek Sukla, Asst.Prof of Computer Science & Engineering Dept, Dr C V Raman University, Kota Bilaspur, for his stimulating guidance, continuous encouragement and supervision throughout the course of the present work.

I am extremely thankful to Prof. S. R. Tandan, Asst.Prof of Computer Science & Engineering Dept, Dr C V Raman University, Kota Bilaspur for providing me infrastructural facilities to work in, without which this work would not have been possible.

References

[1] C. Boeres et al. *A tool for the Design and Evaluation of Hybrid Schedulin g Algorithms for Computational Grids.* Proceedings of the 2nd workshop on Middleware for grid computing, ACM International Conference, October 2004.

[2] Harchol-Balter, Mor. *Task Assignment with Unknown Duration*. Journal of the ACM, Volume 49 Issue 2, March 2002.

[3] Ian Foster. *What is the Grid? A Three Point Checklist* Grid Today, volume1 number 6, July 22 2002.

[4] Ian Foster, Carl Kesselman. *Computational Grids*. Chapter 2 of *The Grid:Blueprint for a Future Computing Infrastructure*. Ian Foster and Carl Kesselman, Morgan Kaufman, 1998.

[5]Evaluating Scheduling Algorithms On Distributed Computational Grid

[6] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra A. Hensgen, Sahra Ali. *Task Execution Time Modeling for Heterogeneous Computing Systems*. Heterogeneous Computing Workshop, 2000: 185-199.

[7]Grid Computing Models: A Research by D Jankiram.

[8] Grid Computing: By Fram Berman

[9] International Journal of Grid and Utility Computing

[10]Future Generation Computer Systems

[11] A Game-Theoretic Analysis of Grid Job Scheduling Maria Grazia Buscemi · Ugo Montanari , Sonia Taneja