Dynamic Job Scheduling in Multilayer Grid Networks

Cihan Varol, *Member, IEEE*

***Abstract*— Although grid structures were built to increase the performance of the system and process the job in less time, most of the individual grid groups (hives) only perform one task at a time which can cause hotspots. However, introducing multi-purpose hives eliminates big amount of possible hot spots that can be forecasted. In this study, a matrix is developed to identify which grid groups can perform the same task. Moreover, a grid scheduling algorithm is provided for multi-purpose hives. A simulation of the proposed system shows a substantial decrease in the hotspots, compared to the original setup.**

***Index Terms*—Grid Computing, Grid Scheduling, Job Scheduling, Multi-Purpose Grids**

I. Introduction

Technology is quickly developing in computational devices. This trend should continue, considering the new requirements by all fields of science for capable computers. Solving problems in almost all of the scientific fields such as astronomy, physics, mathematics, bioengineering, and bioinformatics requires more computational, storage and visualization resources and devices then ever before. These trends will always demand much more then the existing technology. To overcome these difficulties, parallel computing techniques or the use of more than one CPU are being used. In addition to these solutions the concept of grid computing was presented in order to solve massive computational problems. Grid computing adds value by making use of the unused resources of large numbers of nodes, often desktops or servers, by using the nodes' cpu cycles, and/or disk storage. The main difference between a grid and traditional distributed computing is that the grid computing focuses on the ability to support computation across large (administrative) domain sets.

The advances in these fields also lend themselves favorably to the industrial and commercial tasks that drive a business. As the needs of the business partner continually emerge in computational devices, the need to better utilize existing hardware to a higher capacity through innovative ideas will remain in the forefront. As the number of nodes or frequency of jobs of the grid increase, the task scheduling and management environment become more difficult. Managing the grid nodes based on the performance of the system is a challenging job.

Manuscript received July 23, 2012; revised August 9, 2012.

C. Varol is an Assistant Professor in the Department of Computer Science, Sam Houston State University, Huntsville, TX 77341 USA (phone: (936) 294-3930, e-mail: cvarol@shsu.edu).

The system itself changes quickly and there is a need to evaluate the effects of the changes in the environment before the changes are introduced to the system. Moreover, there is also a need for an automated decision tool to help respond to performance issues. Before applying a model to the system, simulating the behavior helps to ensure the success of the applied model. Therefore, this paper aims to provide better task scheduling between groups of nodes by using a model and simulating the results.

II. Background

In current grid computing systems the availability of computational resources is unpredictable and jobs are allocated on a first come first served basis. However, it is known that occasionally the first-come-first-served systems fail to provide the results in a timely manner. Therefore, a number of attempts have been developed to facilitate economy based scheduling systems [1, 2, 3, 4, 5, 6, and 7]. Besides the listed, Nimrod-G, Condor-G, and GRaDS are other well known techniques. Nimrod-G is part of the Grid Architecture for Computational Economy (the GRACE project) and as such it is an economy based scheduler which allows requesting resources of more than one machine for a single job. It may perform load balancing of workload across multiple systems. Each system would then have its own local scheduler to determine how its job queue is processed which requires advance reservation capability of local schedulers. Nimrod-G also supports quality of service based scheduling [1]. Condor-G is task broker designed to front end a computational grid. It acts as an entry point to the grid dispatching jobs to run on the various nodes available [2]. On the other hand, GRaDS provides software execution environment for code to be run on a computational grid. The GRaDS attempts to adapt the application according to changes in the available resources while attempting to maintain as high performance as possible. Feedback is an important part of this algorithm, because it updates its own behavior with the current status [3]. These three scheduling algorithms assume one entry point into the grid, control the scheduling policies for all the nodes and the nodes are closely linked to the grid. However in real world, there might be more than one entrance points to the grid, the resource allocation would be done on the basis of market trading, which determines the allocation of resources to nodes and all nodes will have different, unknown job execution policies.

Our simulation involves real time data collected over a period of time from grid architecture. The purpose of the grid

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

is to provide services for the organizations business functions. In an effort to find a system solution to minimize the number of hotspots, a simulation must be constructed to prove and validate an efficient modeling of the new system. A new method of handling the processes has been devised.

Based on the hypothetical business grid environment, the main goal is to apply existing performance models and workload characterization techniques to the system to reveal various factors that affect the system performance, to realize the potential performance problems (hotspots) before they arise in the system. The layout and language used throughout the paper is listed below. The computers for the business have groups of computers dedicated to special jobs and moreover each individual computer in the within a group can perform a specialized function. Each group of computers will be referred to as a hive. Each hive has a primary function and no two hives share the same task. For instance one hive of computers may perform only email operations while another handles server requests. Each hive only does its job and doesn't share it with the others. When hives work in conjunction with each other, the hive with no hotspots is the hobby hive. All of the hives together compose the grid. All jobs entering the system that have e-mail operations will go to a hive. Once the job is sent to a hive, other decisions are made to determine which individual computer within the hive will actually perform the job. Each individual computer within a hive is called a node. The nodes can be further stratified to break up the job as needed. For instance, once a job reaches a hive, the nodes may be split into alphabetical categories with each node assigned specific letters to handle according to the last name of a customer. Or, on the other hand it can be first come first serve wherein all nodes will perform the same type of work. Occasionally, some hives work load may exponentially increase beyond the hives capacity to accommodate them. When this happens, a hot spot occurs. A hot spot is when the hive is so busy completing other tasks that it can't continue to process its incoming job. In the occurrence of a hotspot, human intervention is used to manually configure other hives to temporarily handle the workload until the jobs take on a more steady arrival time. An example of an instance that may cause a hot spot are banks that send the processing of there accounts at approximately the same time and overload portions of the system. Just as some hives incur frequent hot spots, others may exhibit none. The determination of which hives have common hotspots are found by examining the history of the jobs entering the system and monitoring the load of the system. This history is stored in a database. The goal is to eliminate the hotspots. After looking at historical data, one can

1. Find problematic hives that have frequent hot spots
2. Identify low volume hives that only use a small amount of its capabilities
3. Configure the low volume hive to also do the work of the hotspot laden hive.

The hive that is found to work the best with a hot spot laden hive will only serve as the hobby to one hive. This hobby hive will be configured to perform the same operations as the original hive. A simulation will have scenarios of both hives having a full load and the job transfer mechanism. We can subsequently take the hotspots from the system or at the very least drastically minimize them. In this study we have provided an algorithm that determines which hives are best to run together. This solution will increase throughput, energy, and possibly increase processing capabilities by translating into less configuration changes in the event of a hot spot and better reliability for customers.

III. METHODOLOGY

A node in a grid may be a single CPU. Equally, it may be a vast super computer or a private array of workstations. It is generally estimated that each node in the grid will have a single scheduling policy and single high level scheduler. Instructions are not only dependent from one client, but also from the individual machines. Widely used scheduling techniques are:

- Determining the tasks that the user wants to complete first.
- Evaluating the current load on the machine against the loading requirements given by the application
- Splitting a processor into time slices and allocation all jobs to it equally or an a first come first served basis

However, the GRaDS project [3] first verified that the applications made there scheduling decisions based on conditions of the system when competing applications are executing. It then temporarily stopped long running and resource consuming jobs in order to run the shorter ones. This facilitated new applications to execute faster by stopping certain competing applications and thereby minimizing the impact of new applications on already running applications. However, the system is avoiding from killing jobs, since dropped or skipped tasks may cause some unexpected results and delays to the system performance.

As the first task, a Java program code written was to determine which hive can be used as a hobby hive. The java code connects to a database that has two different information in it. One contains information about the activity in the hive and the other about the nodes. They correlate with each other. Starting at the first time point, the system evaluates the incoming data, then a point system is assigned and the job is run against its hive and all other hives. The output is a matrix that has numbers that indicate if the two hives acted as hobby hive, this is the resulting number of times a job would have been killed because the job couldn't be completed. So, this translates into both hives being very busy and working to capacity. The purpose of this java program is to simply use historical data to determine which hives are best to run together.

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

As a second task an algorithm is developed for scheduling the jobs between the nodes of the original and hobby hives. For the proof of concept, only one original and one hobby hive are selected for the scheduling mechanism. Presuming we know the node in the original hive that performs the task, a node needs to be selected in the hobby hive that will be the best match to do the hobby job of the original node. The algorithm is dynamic. That is every time a job needs to be sent of from the original hive to the hobby hive, the algorithm is executed to find the best match at that given time. This means that if two random nodes were linked once, there is no guarantee that they will be linked together again. The accepted assumption is any node in the hobby hive has to be able to perform every job that is performed in the original hive (Figure 1).

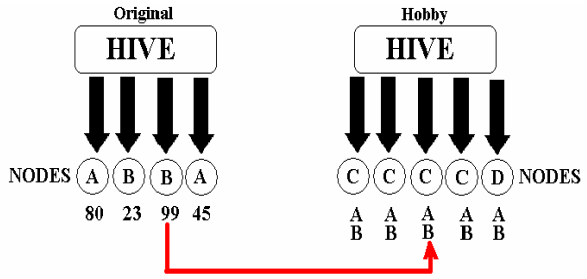


Figure 1. Structure of the Original and Hobby Hives

For the purposes of the algorithm, an input file which consists of Job Name, required CPU to perform the particular task, job process time, and job arrival time based on the data obtained from the database (Table 1) was created. With this input information, we are able to discern which node of the hobby hive is currently in use, or the current CPU usage at a given node. Moreover, we are able to present the busy "tied up" nodes and the kind of the job is being performed.

Table 1: Input Data for the Simulation

	K1	K2	K3	K4	K5	K6	…	Km
State	Up	Down	Down	Down	Up	Up	…	Up
CPU	24	3	27	95	23	76	…	49
Hobby	1	0	0	1	1	1		1

The algorithm uses a simple selection of the node that is up and has the least amount of the CPU used. However, a threshold value F that allows predicting if there is going to be a possibility of the hot spot needed to be set. The tool accepts any value as the threshold. In the simulation 80% has been selected as the determining point.

Typically task reallocation can occur when a given node needs to execute some other process more urgently than the one on which it is currently executing. Moreover, the algorithm takes place when the scheduler feels that a particular application would benefit by moving it to a lightly loaded CPU. One of the possible solutions is to look at the average CPU usage at each hive (Figure 2). If we have job A coming into the system, we are looking at the average CPU of the nodes that are capable of processing this job at the original node.

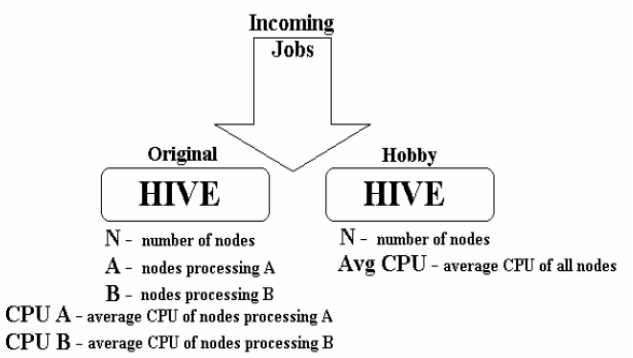


Figure 2. Average CPU of Nodes

If the average CPU of the nodes performing job A on the original node is low, that means more than half of the nodes are using small amounts of CPU. On the other hand, if it is high, then they use a big amount of CPU. The average CPU usage of Hobby Hive (H) is useful for the following reasons: for instance we have H=76%, and the average CPU for job A=60%; in this case we can predict that most likely, if another job will be send to the hive, it will create a hot spot. At the same time, if the average CPU of the nodes in the hobby hive is low, we can simply route incoming job there, without fear of creating a hot spot.

This approach recognizes the priority of a task which not only evaluates the current workload, but also splits the job to the nodes that have the lowest CPU usage at the current moment. Moreover, the determination of the node is done by prediction based on the history. Lets say for any given node we look at the N past jobs and calculate how much time they spent in the system (how long did it take any given node to process N past jobs). Then, we select the node with the smallest number T, where T is:

```
N=5;
for( j=0; j<num_nodes_given_job; j++ )
{
      time=0;
      for( i=0; i<N; i++ )
      {
         time += (end_time – start_time);
      }
      T = time/N;
}
```

This approach tells us that current node had to perform the least amount of tasks in a given time period. It suggests that the tasks it gets are small in processing. So, when new tasks are to be processed they are not likely to create the hot spot on this node.

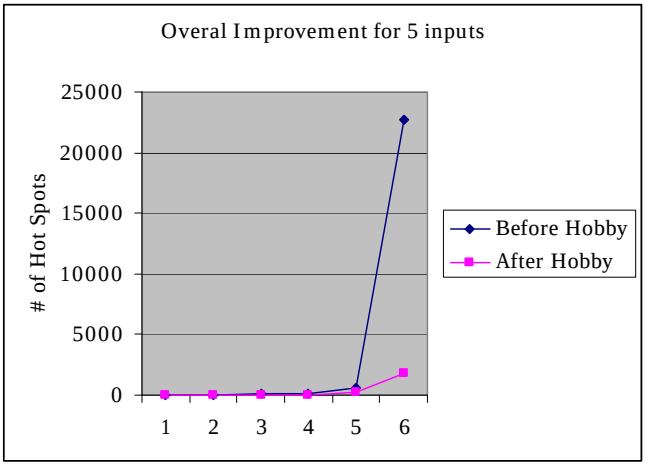
ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

After running some experiments which have different values for the required CPU and process time, it can be said that the solution that is proposed provides acceptable results for the problem. Based on the two simulation cases, we can see the decline in the number of hot spots. Looking at the results in the Table 2 and Figure 3, even though the numbers of hot spots at the hobby hive slightly increased for the proposed solution, the overall number of hot spots dramatically decreases. For example, for input of 100 at the hobby hive, the number of hot spots at the hobby hive increased by 12%. However, the decrease of hot spots in original hive is around 95%.

Table 2: Results from First Experiment

	Sample Input					
	10	**50**	**75**	**100**	**200**	**300**
Original Problem Total	5	28	157	165	577	22675
Solution w/ Hobby Total	0	0	0	56	260	1849

Figure 3: Number of Hot Spots from First Experiment



Since, the second experiment involves high CPU usage and processing time, more hot spots were seen in the system. However, using a Hobby Hive shows its effect if there are more than 50 or 75 jobs that need to be processed as shown in Table 3 and Figure 4.

Table 3: Results from Second Experiment

		Sample Input					
		10	**50**	**75**	**100**	**200**	**300**
Original Problem	At Original	5	28	115	121	402	21545
	At Hobby	0	0	42	44	175	1130
Solution w/ Hobby	At Original	0	0	0	6	46	201
	At Hobby	0	0	0	50	214	1648

IV. Conclusion

In this study, as a first task, an algorithm is developed to determine which hives can collaborate together based on the recorded historical data. After determining the hives, a custom job scheduling algorithm is designed for the system to overcome the hot spot deficiency. It is reflected from the simulation of the proposed system that hotspots are eliminated substantially, compared to the original setup. However, although the algorithm provides acceptable results, more flexible hives and nodes will result in less problematic job processes.

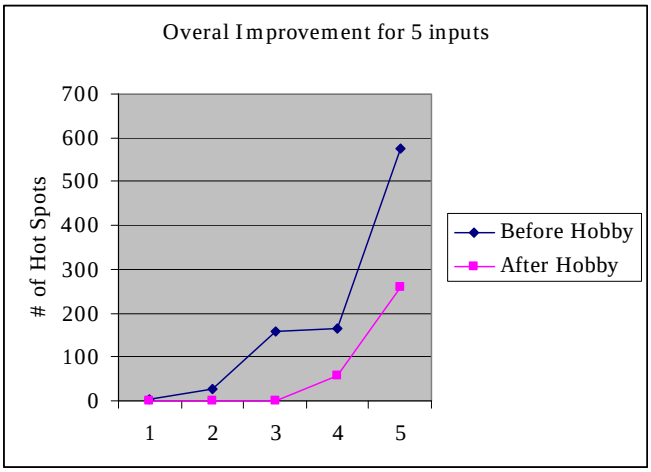


Figure 4: Number of Hot Spots from Second Experiment

References

[1] Buyya, R., Murshed, M, Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm, Software—Practice & Experience, v.35 n.5, p.491-512, 25 April 2005

[2] Baraglia, R., Ferrini, R, Ritrovato, P.: A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research Articles, Concurrency and Computation: Practice & Experience, v.17 n.13, p.1579-1605, November 2005

[3] Bidot, J.: A General Framework Integrating Techniques for Scheduling under Uncertainty. PhD thesis, Institut National Polytechnique de Toulouse, France (2005).

[4] Buyya, R., Murshed, M.: GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) 14, 1175-1220 (2002).

[5] Capannini, G., Baraglia, R., Puppin, D., Ricci, L., Pasquali, M: A job scheduling framework for large computing farms, Proceedings of the 2007 ACM/IEEE conference on Supercomputing, November 10-16, 2007, Reno, Nevada [doi>10.1145/1362622.1362695]

[6] Fibich, P., Matyska, L., Rudová, H.: Model of Grid Scheduling Problem, Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, Papers from the AAAI 2005 workshop. Technical Report WS-05-03, AAAI Press (2005).

[7] Klusáček, D., Matyska, L., Rudová, H.: Local Search for Deadline Driven Grid Scheduling. In: Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007), pp. 74-81 (2007).

[8] Abramson, D., Sosic, R., Giddy, J., Hall, B.: Nimrod: A tool for performing parameterized simulations using distributed workstations. In HPDC, pages 112–121, 1995.

[9] Condor-G (http://www.cs.wisc.edu/condor/)

[10] Vadhiyar, S. S., Dongar, J. J: A metascheduler for the grid, 2002. http://www.cs.utk.edu/~vss/publications/ vadhiyar-metascheduler.pdf.

ISBN: 978-988-19252-4-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)