An Efficient Hybrid Job Scheduling Algorithm for Computational Grids

G.K.Kamalam
Assistant Professor (Slec.Grade)/CSE
Kongu Engineering College
Perundurai, Erode

V.Murali Bhaskaran
Principal
Pavaai College of Engineering
Pachal, Namakkal

ABSTRACT
Scheduling of jobs is a challenging problem in grid. Efficient job scheduling is essential for the effective utilization of the resources. We propose a grid model as a collection of clusters. In this paper, we apply Divisible Load Theory (DLT) and Least Cost Method (LCM) to model the grid scheduling problem involving multiple worker nodes in each cluster. We propose a hybrid job scheduling algorithm that minimizes the overall processing cost of the job and divisible job scheduling algorithm that minimizes the overall processing time of the job in a grid system that consists of heterogeneous hosts. The results show that the proposed algorithms are feasible and improve the makespan and processing cost considerably.

Key words
Grid, Job Scheduling, Cluster, Coordinator Node, Worker Node, Heterogeneous Hosts.

1. INTRODUCTION
Wide-spread availability of low-cost, high number of powerful resources, high performance computing hardware and the fast expansion of the internet and advances in networking technology aims to integrate, virtualized, manage resources and coordinate problem solving in dynamic multi-institutional virtual organizations to provide inexpensive access to high-end computational capabilities [1,5]. Functionally, grid can be classified into data grid and computational grid. The computational grid is a collection of computational resources that helps in reducing the processing time and processing cost. The data grid is a collection of resources that helps in storing huge amount of data. Grid scheduling is a combination of job scheduling and resource allocation. The job scheduling involves the scheduling the jobs to improve the response time of the job. Resource allocation involves allocating the resource to a job to reduce the processing time of the job.

With its myriad of heterogeneous resources, an efficient scheduling of jobs across grid is essential for improving the performance of the system [6]. While considering the scheduling of jobs, many factors such as CPU utilization, throughput, turnaround time, waiting time, response time should be focused. The objective of the scheduling strategy is to reduce the makespan or the maximum completion time of

the job is the difference between the time when the job was submitted to the grid and the time it is completed [4].

A more recent approach is Divisible Load Theory (DLT), which is designed to solve the challenging problem of scheduling and allocating the available heterogeneous resources on a grid for independent jobs from large number of users [2,3]. A divisible job is one that can be arbitrarily partitioned into any number of fractions and can be processed independently on the available heterogeneous resources in a grid system. One of the primary objectives is to assign optimal fractions of the total job among the several available resources such that the entire job is processed in a minimal amount of time [7]. Divisible load theory optimizes the computing problems where the computational problem is partitioned into sub jobs.

The scheduling model is of three types, centralized scheduling model, decentralized scheduling model and hierarchical scheduling model. The centralized scheduling model is used for managing single or multiple resources located either in a single or multiple domains. It supports uniform policy and suits well for cluster management systems [12]. In decentralized scheduling model, the schedulers interact among themselves in order to decide which resources should be allocated to the jobs being executed. This model appears to be highly scalable and fault-tolerant [11]. Finally, hierarchical scheduling model fits grid systems as it allows remote resource owners to enforce their own policy on external users. This model looks like a hybrid model, but appears more like a centralized model and therefore well suits for grid system.

In this paper, we propose the grid architecture model as a collection of clusters. Users are associated to various clusters. When a user submits the job, the coordinator node then schedules the job to the worker node of the cluster where the user is associated. This research paper is mainly focused on improving the performance of a grid environment by the use of effective scheduling algorithm. In this paper, we propose the hybrid job scheduling that apply Divisible Load Theory (DLT) and Least Cost Method (LCM) for grid scheduling to improve the makespan of the jobs.

The organization of the paper is as follows: The second section presents the literature review. The third section focuses on the grid model. The section four, in turn, discuses the various job scheduling algorithm. While section five compares the result of local job scheduling algorithm, divisible job scheduling algorithm and hybrid job scheduling algorithm and. Section six talks about conclusion and future work.

2. LITERATURE REVIEW

In LCM method, the jobs are allocated to the resource having minimum allocation cost [10, 13, 15].

The major objective in the research area of grid computing is to design an efficient scheduling algorithm that minimize the overall processing time of the jobs. This is achieved by dividing the jobs into sub jobs. This is achieved by Divisible Load Theory. In DLT, it is assumed that computation loads can be partitioned arbitrarily and can be executed in any order [9,10, 14].

3. GRID MODEL

Let m be the number of clusters, o be the number of users, k be the number of jobs, n be the number of worker nodes. By $C = \{C1, C2, \ldots, Cm\}$ we denote the set of clusters forming the grid. Each user Ui owns a cluster Ci. The set $U = \{U1, U2, \ldots, Uo\}$denotes the set of users. The set of all jobs produced by user U of cluster C is denoted by J. The set of jobs is denoted by $J = \{J1, J2, \ldots, Jk\}$.Each job is split into sub jobs $Ji = \{SJi1, SJi2, \ldots, SJil\}$. Each cluster has a coordinator node denoted as CN. Each cluster has a set of worker nodes denoted as $WN = \{WNi1, WNi2, \ldots, WNin\}$ [8].

In a schedule, the jobs Jk of cluster Ci is partitioned into sub jobs $SJki$ of maximum five partitions and assigned to the worker node of the same originating cluster $WNin$ with minimum processing time.

4. JOB SCHEDULING ALGORITHMS (JSA)

4.1 Cost Local Job Scheduling Algorithm (CLJSA)

The local job scheduling algorithm allocates the job to the worker node of the cluster where the job has been originated. The worker node is selected based on the minimum completion time and processing cost of the job. The detailed steps are as follows:

Step1: Assign each user to a cluster.

Step2: The user submits the job to the coordinator node of the cluster where the user is associated.

Step3: The coordinator node then schedules the job to the worker node of the associated cluster with minimum ((job length / processing power (worker node) + waiting time)*processing cost).

Step4: If no more jobs to be scheduled, the processing time is calculated as follows:

$$makespan = clusterlist[0].WNList[0].uptime$$

$$for\ each\ cluster\ Ci, 1 \leq i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$if\ (clusterlist[i].WNList[j].uptime > makespan)$$

$$makespan = clusterlist[i].WNList[j].uptime$$

Step5: If no more jobs to be scheduled, then find the total processing cost of completing the job. The total processing cost is calculated as:

$$for\ each\ cluster\ Ci, 1 \leq i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$totalcost += clusterlist[i].WNList[j].uptime * cluterlist[i].WNList[j].cost$$

Step6: If more jobs are still to be scheduled go to step 3.

Step7: END.

4.2 Makespan Local Job Scheduling Algorithm (MLJSA)

The local job scheduling algorithm allocates the job to the worker node of the cluster where the job has been originated. The worker node is selected based on the minimum completion time of the job. The detailed steps are as follows:

Step1: Assign each user to a cluster.

Step2: The user submits the job to the coordinator node of the cluster where the user is associated.

Step3: The coordinator node then schedules the job to the worker node of the associated cluster with minimum (job length / processing power (worker node) + waiting time).

Step4: If no more jobs to be scheduled, the processing time is calculated as follows:

$$makespan = clusterlist[0].WNList[0].uptime$$

$$for\ each\ cluster\ Ci, 1 \leq i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$if\ (clusterlist[i].WNList[j].uptime > makespan)$$

$$makespan = clusterlist[i].WNList[j].uptime$$

Step5: If no more jobs to be scheduled, then find the total processing cost of completing the job. The total processing cost is calculated as:

$$for\ each\ cluster\ Ci, 1 \leq i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$totalcost += clusterlist[i].WNList[j].uptime * cluterlist[i].WNList[j].cost$$

Step6: If more jobs are still to be scheduled go to step 3.

Step7: END.

4.3Divisible Job Scheduling Algorithm (DJSA)

The divisible job scheduling algorithm divides the jobs into sub jobs of equal size to the maximum of five partitions. The sub jobs are allocated to the worker node with minimum completion time. The steps of divisible job scheduling algorithm are as follows:

Step1: Assign each user to a cluster.

Step2: Upon receiving the jobs from the user, the coordinator node partitions the jobs into sub jobs of equal size to the maximum of five partitions and adds it to the job set J.

Step3: The coordinator node then schedules the sub jobs to the worker node of the same cluster where the user submits the job. The worker node is selected with minimum completion time

Step4: After completing all the jobs and its sub jobs to be scheduled, the processing time and the total processing cost of completing is calculated.

Step5: The processing time is:

$$makespan = clusterlist[0].WNList[0].uptime$$

$$for\ each\ cluster\ Ci, 1 \leq\ i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$if\ (clusterlist[i].WNList[j].uptime\ >\ makespan)$$

$$makespan =\ clusterlist[i].WNList[j].uptime$$

Step6: Calculate the total processing cost.

$$for\ each\ cluster\ Ci, 1 \leq\ i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$totalcost\ +=\ clusterlist[i].WNList[j].uptime *$$

$$cluterlist[i].WNList[j].cost$$

Step7: If more jobs to be scheduled go to step 3, otherwise go to step 8.

Step8: END.

4.3 Hybrid Job Scheduling Algorithm (HJSA)

The hybrid job scheduling algorithm divides the job into sub jobs of equal size to the maximum of five partitions and adds it to the job set J. The sub jobs are allocated to the worker node of the cluster where the job is originated using the Least Cost Method. The detailed steps of the hybrid job scheduling algorithm are described as:

Step1: Assign each user to a particular cluster.

Step2: The user submits the job to the associated cluster and the scheduler then partitions the job into sub jobs of equal size to the maximum of five partitions and adds it to the job set J.

Step3: If the job set J is empty go to step8.

Step 4: The scheduler then schedules the sub jobs to the worker node of the originating cluster with waiting time + processing time * processing cost is minimum. Remove the sub job from the job set J.

Step5: Continue step 4 until the job set J is empty.

Step6: If the user is still there to submit the jobs go to step 2, otherwise go to step8.

Step 6: The processing time is

$$makespan = clusterlist[0].WNList[0].uptime$$

$$for\ each\ cluster\ Ci, 1 \leq\ i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$if\ (clusterlist[i].WNList[j].uptime\ >\ makespan)$$

$$makespan =\ clusterlist[i].WNList[j].uptime$$

Step7: The total processing cost of completing the job is

$$for\ each\ cluster\ Ci, 1 \leq\ i \leq noofclusters$$

$$for\ each\ worker\ node\ WNj, 1 \leq j \leq workernodes$$

$$totalcost\ +=\ clusterlist[i].WNList[j].uptime *$$

$$cluterlist[i].WNList[j].cost$$

Step8: END.

5. COMPUTATIONAL RESULTS

In this paper, we compare the analysis of our proposed hybrid job scheduling algorithm and divisible job scheduling algorithm with local job scheduling algorithm.

We consider simulation parameters taken from [4]. In this scenario, the grid system consists of five users and the parameters are listed in Table 1.

Table 1. Simulation Parameters

Parameters	Value
No. of Clusters	10
No. of worker nodes per cluster	10
Processing power of worker node	500 – 5000 MIPS

Job length	2,50,000-6,50,000 MI
Cost	1-5 G$ unit
No. of users	5
No. of Jobs	50 - 500

The performance of job scheduling algorithms is based on the three metrics: Total processing time, Total processing cost and Number of jobs. The performance of the three job scheduling algorithms is compared by varying the number of submitted jobs.

Table 2 and Table 3 shows the processing time (makespan) and processing cost obtained by the job scheduling algorithms.

Graphical representation of Table 2 in Figure 1 shows that the DJSA provides a better makespan than the CLJSA, MLJSA, and HJSA.

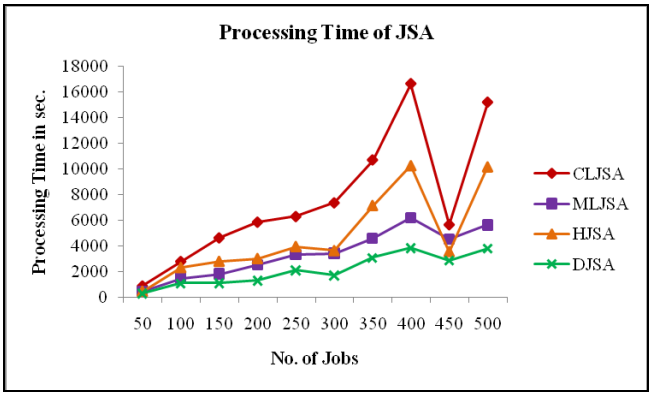
Graphical representation of Table 3 in Figure 2 shows that the HJSA provides a minimum processing cost than the CLJSA, MLJSA, and DJSA.

Table 2. Processing Time of JSA

No. of Jobs	JSA			
	CLJSA	MLJSA	HJSA	DJSA
50	907	502	486	313
100	2811	1508	2333	1102
150	4644	1829	2816	1116
200	5863	2546	3037	1324
250	6300	3340	3973	2114
300	7362	3432	3651	1710
350	10695	4581	7173	3083
400	16615	6168	10272	3844
450	5665	4519	3578	2866
500	15177	5618	10179	3803

Table 3. Processing Cost of JSA

No. of Jobs	JSA			
	MLJSA	CLJSA	DJSA	HJSA
50	10732	7676	5534	4036
100	30390	28942	23745	22855
150	49015	46261	29796	28363
200	70534	55112	35635	27608
250	91251	58829	59209	38084
300	89777	69827	44456	34643
350	125437	104897	83984	70545
400	186485	163821	114402	99863
450	124744	111332	77631	68897
500	177196	148306	117963	99139



. **Fig 1: Impact of JSA on Makespan**

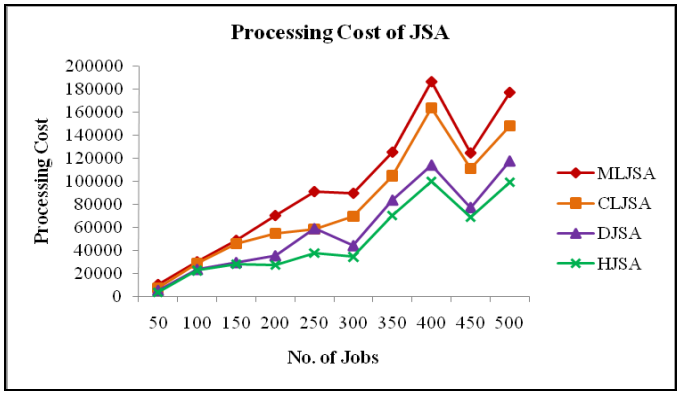


Fig 2: Impact of JSA on Processing Cost

6. CONCLUSION AND FUTURE WORK

In this paper, we presented an efficient job scheduling algorithm for allocating the jobs to the worker node of the originating cluster. The hybrid job scheduling algorithm produces minimum processing cost as compared to the local job scheduling algorithm and divisible job scheduling algorithm. The divisible job scheduling algorithm produces better makespan as compared to the local job scheduling algorithm and hybrid job scheduling algorithm. The hybrid job scheduling algorithm and divisible job scheduling algorithm is a potential solution for scheduling of jobs in a computational grid environment.

7. REFERENCES

[1] Tracy D. Braun, Howard Jay Siegel, and Noah Beck, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, Vol.61, pp. 810 – 837, 2001.

[2] V.Bharadwaj, D.Ghose, and T.G.Robertazzi, "Diisible Laoad Theory: A New Paradigm for Load Scheduling in Distributed Systems", Cluster Computing on Divisible Load Scheduling, pp. 7 – 18, 2003.

[3] T.G.Robertazzi, "Ten Reasons to use Divisible Load Theory", Computer, Vol.36, No.5,pp. 63 – 68.

[4] Monir Abdullah, Mohamed Othman, Hamidah Ibrahim, and Shamala Subramaniam, "Load Allocation Model for Scheduling Divisible Data Grid Applications", Journal of Computer Science 5(10), pp. 760 – 763, 2009.

[5] I.Foster, C.Kesselman, and S.Tuecke, " The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal on High Performance Computing Applications, Vol.15, No.3, pp. 200 – 222, 2001.

[6] Ruchir Shah, Bhardwaj Veeravalli, Senior Member, IEEE, and Manoj Misra, Member, IEEE, " On the Design of Adaptive and Decetralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments.

[7] Wim Depoorter, Ruben Van Den Bossche, Kurt Vanmechelen, and Jan Broeckhove, " Evaluating the Divisible Load Assumption in the Context of Economic Grid Scheduling with Deadline Based QoS Guarantees", 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 452 – 459, 2009.

[8] Syed Nasir Mehmood Shah, Ahmad Kamil Bin Mahmood, and Alan Oxley 2010, "Hybrid Resource Allocation for Grid Computing", in Proceedings of the IEEE Second International Conference on Computer Research and Development, 426 – 431.

[9] D.Yu, T.G.Robertazzi 2003, "Divisible Load Scheduling for Grid Computing ", in Proceedings of the International Conference on Parallel and Distributed Computing Systems.

[10] G.Murugesan, and C.Chellappan 2009, "An Economical Model for Optimal Distribution of Loads for Grid Applications ", in Internal Joural of Computer and Network Security,Vol.1,No.1.

[11] Jaehwan Lee, Pete Keleher, Alan Sussman, "Decentralized dynamic scheduling across heterogeneous multi-core desktop grids", in 2010 IEEE International Symposium on Parallel&Distributed Processing, Workshops and Phd Forum, pp.1-9.

[12] Po-Chi Shih, Kuo-Chan Huang, Che-Rung Lee, I-Hsin Chung, Yeh-Ching Chung, "A Performance Goal Oriented Processor Allocation Technique for Centralized Heterogeneous Multi-cluster Environments," ccgrid, pp.614-615, 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2011

[13] Amril Nazir, Hao Liu, Søren-Aksel Sørensen, "A Rental-Based Approach in a Cluster or a Grid Environment," cit, pp.2501-2508, 2010 10th IEEE International Conference on Computer and Information Technology, 2010

[14] Anwar Mamat, Ying Lu, Jitender Deogun, Steve Goddard, "An Efficient Algorithm for Real-Time Divisible Load Scheduling," rtas, pp.323-332, 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, 2010

[15] Syed Nasir Mehmood Shah, Ahmad Kamil Bin Mahmood, Alan Oxley, "Modified Least Cost Method for Grid Resource Allocation," cyberc, pp.218-225, 2010 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2010