Simulation of MJ_CDT_{min} Based Scheduling Algorithm in Grid Environment

Deepti Malhotra
Research Scholar
Department of Computer
Science & IT
Jammu University,

Devanand
HOD
Department of Computer
Science & IT
Jammu University,

Anik Gupta
Engineer
Department of
Physics & Electronics
Jammu University

ABSTRACT

To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Unfortunately, scheduling algorithms in traditional parallel and distributed systems, which usually run on homogeneous and dedicated resources, e.g. computer clusters, cannot work well in the new circumstances. In this research paper, we introduce a New Grid Job Scheduling algorithm MJ_CDT_{min} (multiple jobs based on the minimum cumulative departure time). The MJ_CDT_{min} is based on the rule that the cumulative arrival time of the next job arriving at the processor is compared with the minimum cumulative departure time of the processor. The main aim of proposed scheduling algorithm is to increase the system efficiency and to satisfy the job requirements from the available resources. In this research work the proposed algorithm has been implemented and validated. To demonstrate the usability of proposed techniques, a Simulation test bench was implemented using the Turbo C platform and successful simulation was achieved. The experimental results showed a significant improvement in terms of a smaller makespan time as compared to the already existing FCFS scheduling algorithm.

Keyword

Grid Computing, Job Scheduling, Scheduler, makespan, MCDT

1. INTRODUCTION

According to Foster and Keselman in [1], grid computing is hardware and software infrastructure which offers a cheap, distributable, coordinated and reliable access to powerful computational capabilities. Since multiple applications may require numerous resources which often are not available for them so that in order to allocate resources to input jobs, having a scheduling system is essential. Because of the vastness and separation of resources in the computational grid, scheduling is one of the most important issues in grid environment [2]. Job scheduling is one of the most fundamental and important aspects of distributed and parallel computational systems. Vast investigations have been done in this scope, which have led to theories and practical results [3, 4, 5]. However new scheduling algorithms have been offered with emergence of grid computing. Objectives of scheduling algorithm are increasing system throughput [5], efficiency, and decreasing job completion time.

- i. Increasing computing performance, its aim is to minimize the execution time of each application that is considered in parallel processing.
- ii. Increasing overall throughput, its purpose is to schedule a set of independent tasks in such a way that

Scheduling systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Scheduling in Grid environments is significantly complicated by the heterogeneous and dynamics nature of Grids. Compared to traditional scheduling systems such as clustering computing, Grid scheduling systems have to take into account diverse characteristics of both various Grid applications and various Grid resources. The different performance goals also place great impacts on the design of scheduling systems. To overcome the heterogeneous and dynamic nature of Grids, the information service plays a highly important role. A successful scheduling system should accommodate all these issues.

The job of the grid scheduler is to automatically assign the suitable resources to the independent jobs to maximize the system utilization. It also reduces the average response time of the jobs. The efficient utilization of grid computing resources can improve the overall job-throughput due to load balancing of the tasks between the grid resources. A Grid scheduler is different from local scheduler in that a local scheduler only manages a single site or cluster and usually owns the resource. A Grid scheduler is in charge of resource discovery, Grid scheduling (resource allocation and task scheduling), and job execution management over multiple administrative domains.

The grid scheduling is divided into three phases [6]. These are Resource exploring, Machine selection and Executing. The first phase recovers all the available resources, the second phase deals with finding the best match between the set of jobs and available resources. The phase two is a NP-hard Problem [7]. The behaviour of computational grid is dynamic and unpredictable as it depends upon various factors:

(i) network connection (ii) availability & non-availability of resources at the required time (iii) the no. of resources joining & leaving the grid (iv) performance of grid resources can vary from time to time.

The jobs will take different execution time on different machines. So, job scheduling in grid environment is a problem to schedule a stream of applications from different users to a set of computing resources to minimize the total completion time. This scheduling requires the matching of different jobs with the machines that satisfy their resource requirement. There are two different goals for job scheduling:

it increases the processing capacity of the systems for long period of time.

We focused on the first goal and proposed a new job scheduling algorithm for grid computing that provides efficient utilization of resources. This grid scheduling algorithm tries to minimize

the total service time of the jobs. The resources in grid environment are not only dedicated to grid applications, as they have to handle their own local jobs also. So, the grid jobs need to compete for the resources according to their resource requirement.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 provides our grid scheduling model. Section 4 discusses the details of the proposed scheduling algorithm i.e. MJ_CDT_{min} (multiple jobs based on the minimum cumulative departure time). Section 4 also provides the results of the experiment carried out using our own grid simulated environment given in table 1. Section 5 gives the comparison results of MJ_CDT_{min} based scheduling with the commonly used FCFS (first come first serve) algorithm. Finally Section 6 concludes the paper by summarizing our contributions and future works.

2. RELATED WORK

The different phases of Grid Scheduling Process have been discussed in [8], [9]. In a grid, the abilities of the computing nodes vary, and tasks often arrive dynamically. Because of these, scheduling methods of parallel computing [10] may not be applicable in a grid. It is important to properly assign and schedule tasks to computing nodes [11]. Through good scheduling methods, the system can get better performance, as well as applications can avoid unnecessary delays. Grid scheduling algorithms in different occasions are discussed in paper [12], [13].

Various algorithms have been proposed which in recent years each one has particular features and capabilities. In this section we review several scheduling algorithms which have been proposed in grid environment. In [14] a scheduling algorithm which is based on HQ-GTSM is presented. This algorithm not only takes into account the input jobs but also considers the resource migration time in deciding on the scheduling. One of the most important features of this algorithm is that it guarantees the grid quality of service

At the present time, job scheduling on grid computing is not only aims to find an optimal resource to improve the overall system performance but also to utilize the existing resources more efficiently. Recently, many researchers have been studied several works on job scheduling on grid environment. Some of those are the popular heuristic algorithms, which have been developed, are UDA [15], OLB [15], min-min [15], the fast greedy [15], GA [15], SA [15] and tabu search [15] and an Ant System [16]. The heuristic algorithms proposed for job scheduling in [15] and [16] depend on static environment of system load and the expected value of execution times. In paper [17], the job will be moved from one machine to another machine. So the traffic in the grid system will be automatically increased. In paper [18] they considered communication cost and different ant agents.

DIANA [19] and BLBD [20] are scheduling algorithms that have been proposed in grid computing environment too. Based on these methods, scheduler chooses the best resource for job allocation by considering the system load and cost of resource allocation. It is interesting to know that BLBD is an adaptive scheduling which is more focused on the guarantee of quality of service. Weighted meta-scheduling is presented in [21]. This algorithm takes into account both the system load and workload in order to enhancing the resource allocation.

3. GRID SCHEDULING MODEL

Fig1: Job Scheduling Model

- 1) **Client:** Each client submits job to scheduler. Then each job can be scheduled to any of the nodes and further dispatched to any of the processors of that node [8].
- 2) **Node:** Node is a set of cumulative resources (CPUs, memory, storage space, list of specializations) with limited capacities. We have assumed that the capacity (load, ACR) of the machine can be a general value for the machine. Each Grid node comprises a number of computational resources (processors, denoted by $P_{ni} \{ \sum_{n=1}^x n = \text{node no.} \mid \sum_{i=1}^4 i = \text{processors no. in node} \}$) in Figure 1. The resources on the grid are usually accessed via an executing "job". A resource is a basic device where jobs are scheduled/processed/assigned. Each resource has a limited capacity (e.g., number of CPUs, amount of memory). Each resource also has a speed and a load. In our model we have assumed that each node consist of 4 processors. The specification of each processor is Intel(R) Core(TM)2 Duo CPU with different computing power.
- 3) **Client Job:** A job (task, activity) is a basic entity which is scheduled over the resources. A job has specific requirements on the amounts and types of resources (including machines). In our model Job j has the following variables (properties):
 - A **release date** (r_j) is the time when a job is available to start processing including i/o operations).
 - The **start time** (S_j) is a time when the job actually begins its processing ($S_j \geq r_j$).
 - The **completion time** (C_j) is the time when a job completes its processing.

It is important to mention that r_j , C_j and S_j are often the wall-clock times.
- 4) **Queue:** A *queue* is an abstract structure that accepts, schedules, and/or sends jobs that have been submitted by users to resources that belong to the queue. The queue often

has a set of jobs it has accepted that are waiting for scheduling. The length of the queue indicates the load of resources that belong to the queue or/and the number of jobs waiting for processing or scheduling in this queue. Here we assume that at one time only five jobs can reside in a queue.

Now consider two basic types of queues that are included in the proposed model shown in figure 1.

- **A global job queue (GQ)** schedules jobs to the selected node based on the proposed scheduling algorithm given in figure 2 and sends the jobs to a local job queue (LQ) belonging to the selected node. The parameters of the GQ indicate which local queues (LQs) may be used for scheduling.
- **A local job queue (LQ)** schedules jobs to resources that belong to it based on the proposed scheduling algorithm given in figure 2 and distributes the jobs to smaller set of resources (Processors of the selected node) than those available to the GQ.

A queue schedules jobs on the resources that belong to it. In the proposed model, we have limited the number of clients (client jobs) that can be in GQ and LQ belonging to the GQ to five. The LQ_n schedules jobs on $LQ_n.R = P_{n1} \cup P_{n2} \cup P_{n3} \cup P_{n4}$ where $P_{ni} = i^{th}$ machine/processor that belongs to the Queue Q_n . The GQ schedules jobs on the resources of all LQ_n that belong to the GQ , $GQ.R = \cup_n LQ_n.R$, where $\sum_{n=1}^x n = \text{node no.}$

5) Scheduler: It is the core of the system and can be classified into two types: global scheduler (GS) and local scheduler (LS).

- **Global scheduler.** In the system, a client submits its jobs to the global GS. Once a GS receives a job, it immediately makes a decision on which Grid node the job should be assigned to, according to NSA (node selection algorithm). It may use the global information, such as the load of each Grid node, as input to its decisions. Each client is associated with a global scheduler, while each global scheduler is associated with all of the Grid nodes to assign jobs and acquire global information.
- **Local scheduler.** When a job is delivered to a Grid node, it is managed by the local scheduler of that node. The LS determines how to dispatch the job to one processor according to PSA (processor selection algorithm) at the local scheduler. It only uses the local information, such as the ACR (Available CPU Resources) of each local processor for its decisions.

4. PROPOSED GRID SCHEDULING ALGORITHM (MJ_CDT_{min})

In this section, we describe the routing strategies involved in the Grid model.

4.1 Objective Function

Our proposed algorithm has adopted an application-centric scheduling objective function aim to optimize the performance of each individual application. *Makespan/Execution time*, which is the time spent from the beginning of the first task in a job to the end of the last task of the job.

Makespan is the one of the most popular measurements of scheduling

4.2 Algorithm Description

In step 1 of Algorithm given in figure2, the user's request is processed and split into individual job requests. Step 2, consists of list of all nodes available in the grid. The actual scheduling process starts from step 3 that is repeated for each job request. In step 4, the scheduler discovers the available resources by contacting GRD (GridResource Database). Here resources are evaluated according to the requirements in the job request and only the appropriate resources are kept for further processing. Step5 gives the number of job request. Step8 randomly take the values for the arrival times (*at*) of the jobs by calling the random function and store them in an array (*at[j]*). Step9 calculate the cumulative arrival time (*cat*) for the jobs. Step10 take the random values for the services time (*st*) of the jobs and store them in an array (*st[j]*). Step 11, 12, 13 check the processors with minimum cumulative departure time. Then the Processor with minimum value of cumulative departure time (min CDT) is stored in the Step14. The processor with min CDT is selected for the allocation of the next job arriving. Step 15 calculates the cumulative departure time. Step16 compare the cumulative arrival time of the next job arriving with the min CDT store in step 14. If the value of cumulative arrival time is greater than the min CDT then the idle time for the processor comes i.e. *idt[nd]* is the amount of idle time processor spends while waiting for the job to arrive. Otherwise wait time for the job comes i.e. *wt[nd]* is the amount of time which job has to wait in the queue to be serviced by the processor. As a result of which queue length is incremented by one. In Step17 job is assigned to the processor. Step18 calculates the total execution time for executing the jobs.

4.3 Algorithm

Step 1:	cList= list of all individual requests by validating the client specification(s);
Step2:	nodeList =list of all nodes. Each Grid node comprises a number of computational resources (processors, denoted by
	$P_{ni} \{ \sum_{n=1}^x; n = \text{node no.} \mid \sum_{i=1}^4 i = \text{processors no. in node} \};$
Step 3:	For each job do the following steps;
Step 4:	Filter out the resources that do not fulfill the job requirements. Contact GRD (GridResource Database) to obtain a list
	of available resources;
Step 5:	k = no of jobs
Step6:	if (k == 0)
	{
	go to step 20;

```

    }
    /* checking Jobs*/
    Step 7:   For j = 1 to k; {
    Step 8:   at[j] = random(100); /*randomly take the values of arrival time and store them in array*/
    Step 9:   cat = cat + at[j]; /* cumulative arrival time */
    Step10:  st[j] = random(100); /* service time*/
    /*checking..... Processor with Min CDT (cumulative departure time)*/
    Step11:  minCDT = cdt[1];
    Step12:  nd = 1;
    Step13:  For i = 1 to 4;
    {
        If (cdt[i] < minCDT);
        {
            minCDT = cdt[i];
            nd = i;
        } /*End If */
    } /*End For */
    Step14:  x[nd] = cdt[nd]; /*Processor with Min CDT*/
    Step15:  cdt[nd] = cat + st[j];
    /* comparing the arrival time of the next job with the min CDT store in step 14*/
    Step16:  If (cat >= x[nd])
    {
        idt[nd] = cat - x[nd]; /* idle time for the processor*/
        x[nd] = cdt[nd];
        wt[nd] = 0; /* wait time for the job*/
        cdt[nd] = cdt[nd] + wt[nd];
    } /*End If */
    Else
    {
        idt[nd] = 0; /* idle time for the processor*/
        wt[nd] = x[nd] - cat;
        q[nd] = q[nd] + 1; /* increment the queue length*/
        cdt[nd] = cdt[nd] + wt[nd];
    } /*End Else */
    Step17:  Processor Allocated = nd;
    /* calculate the total Execution time for all the jobs*/
    Step18:  if (j == 1)
    {
        et = cdt[1]; /* execution time*/
    }
    if (cdt[nd] > et)
    {
        et = cdt[nd];
    }
    Step19:  } /*End For */
    Step20:  exit

```

Fig2: Scheduling Algorithm**4.3.1 Assumptions**

- AT_k (Arrival Time) = Time gap between the arrival of (k-1) job and the k job.
- CAT_k (Cumulative arrival time)
 $CAT_k = CAT_{k-1} + AT_k$
 $CAT_1 = AT_1 = 0$
- ST_k (Service Time) = Service time of the k^{th} job
- IDT_k (Idle time) = Idle time of the processor awaiting the arrival of the k^{th} job.
- WT_k (Wait Time) = Wait time denote the waiting time of the k^{th} job in the queue.
- CDT (Cumulative Departure time)
 $CDT_k = CAT + ST_k + WT_k$
- QL_k (Queue Length) = number of jobs waiting for processing scheduling in this queue. If the wait time

for the job comes then the queue length is incremented by 1.

4.4 Experimental Setup For (MJ_CDT_{min})

We carried out a simulation-based study. We created our own Grid simulation environment in Department of Computer Science & IT in University of Jammu. The simulated grid environment is carried out by using Turbo C. We simulate the grid computing environment by using, multiple jobs with random arrival time and service time arriving at single node. Each node consists of 4 processors. The specification of each processor is Intel(R) Core(TM)2 Duo CPU with different computing power. Table1 gives the results of the experiment carried out using our own grid simulated environment.

Table 1.Analysis of MJ_CDT_{min}

Job No	AT (sec)	CAT (sec)	ST (sec)	min CDT Processor		CDT (sec)	IDT (sec)	WT (sec)	QL	PA
				Processor no	MCDT (sec)					
1	40	40	46	1	0	86	40	0	0	1
2	62	102	32	2	0	134	102	0	0	2
3	83	185	19	3	0	204	185	0	0	3
4	04	189	6	4	0	195	189	0	0	4
5	27	216	92	1	86	308	130	0	0	1
6	48	264	79	2	134	343	130	0	0	2
7	69	333	66	4	195	399	138	0	0	4
8	91	424	52	3	204	476	220	0	0	3
9	12	436	39	1	308	475	128	0	0	1
10	34	470	25	2	343	495	127	0	0	2

TOTAL EXECUTION TIME FOR 10 JOBS IS 495sec

At=arrival time, CAT= cumulative arrival time, ST= service time, MCDT= minimum cumulative departure time, CDT= cumulative departure time, IDT=idle time, WT=wait time, QL=queue length, PA=Processor allocated. Experiment results of the MJ_CDT_{min} based scheduling is give in figure3.

Fig3: MJ_CDT_{min} based scheduling

5. COMPARISON BETWEEN MJ_CDT_{min} AND FCFS

We have developed our own simulation for evaluation of newly proposed MJ_CDT_{min} based scheduling algorithm in Grid environment. In the experimental testing we used heterogeneous machines and tested it for different number of tasks/jobs. We compared the results of our minimum cumulative departure time based scheduling algorithm with the commonly used FCFS algorithm. The results are shown in Table 2. It is observed that makespan/Execution time for the given number of jobs is shorter in case of MJ_CDT_{min} scheduling algorithm as compare to FCFS algorithm.

Table 2: Execution time (in secs) for MJ_CDT_{min} based algorithm and FCFS

No. of Jobs	MJ_CDT _{min}	FCFS
12	201	203
22	218	222
32	500	506
42	941	950
52	1078	1087
62	1241	1255
72	1011	1023
82	1008	1024
92	1276	1293

MJ_CDTmin and FCFS based Scheduling

Fig4: Execution time of MJ_CDT_{min} based algorithm and FCFS

6. CONCLUSION AND FUTURE WORK

Resource sharing in grid environment is an inevitable task and the scheduling concept is one of the most important issues in grid computing. In this paper a model and a job scheduling algorithm in grid environment have been presented in order to enhance the average Makespan of input jobs. In this paper we have proposed a scheduling algorithm that considers the resource requirement of the jobs in the Grid environment. This newly proposed scheduling algorithm achieves high efficiency in the Grid computing. A simulation system was developed to test the minimum cumulative departure time based scheduling algorithm in a simulated Grid environment. We used the makespan/Execution time of batch jobs as the comparison criteria. When Compared with first come first served scheduling algorithm, the experimental results show that MJ_CDT_{min} scheduling algorithm show a noticeable increase in performance than FCFS algorithm. As memory is an important resource, In future research can be done considering others factors such as memory as the resource requirement in task scheduling algorithm.

REFERENCES

- [1] I. Foster.2002.What is the Grid? Daily News And Information For The Global Grid Community, Vol.1, No.6.
- [2] Rajkumar Buyya.2002.Economic-based Distributed Resource Management and Scheduling for grid computing. PhD thesis, Monash university, Melborn, Australia.
- [3] K. Al-Saqabi, S. Sarwar, and K. Saleh.1997.Distributed gang scheduling in networks of heterogeneous workstations.Computer Communications Journal, pp.338-348.
- [4] Maheswaran M, Ali S, Siegel H J, et al.1999.Dynamic mapping of a class of independent tasks on to heterogeneous computing systems. In the 8th IEEE Heterogeneous Computing Workshop (HCW '99),San Juan, Puerto Rico, pp.30-44.
- [5] XiaoShan He, XianHe Sun, and Gregor von Laszewski.2003.QoS Guided Min-Min Heuristic for Grid Task Scheduling.Computer Science and Technology, 18(4):442-451.
- [6] J. Krallmann,U. Schwiegelshohn, and R. Yahyapour.1999.On the design and evaluation of job scheduling algorithms. In 5th Workshop on Job Scheduling Strategies for Parallel Processing, volume LNCS 1659, pages 17–42.
- [7] V. Hamscher, U. Schwiegelshohn,A. Streit, R.Yahyapour.2000.Evaluation of job-scheduling strategies for grid computing.In Proceedings of First IEEE/ACM International Workshop on Grid Computing, Lecture Notes in Computer Science, vol. 1971, Springer, Berlin,pp. 191–202.
- [8] Deepti Malhotra, Devanand.2012.Framework for Job scheduling in grid environment. International journal of computer applications (0975-8887),vol 38-No.7,pages42-48.
- [9] Malarvizhi.N, Rhymend Uthariaraj. 2008. A Broker-Based Approach to Resource Discovery and Selection in Grid Environments. In IEEE International Conference on Computer and Electrical Engineering, pp. 322-326.
- [10] Feitelson, D.G.Rudolph R. 1995. Parallel Job Scheduling: Issues and Approaches, Springer Lecture Notes in Computer Science, vol 949, pp. 1- 18.
- [11] M. Harchol-Balter, T. Leighton, D. Lewin.1999.Resource Discovery in Distributed Networks.In 18th ACM Symposium on Principles of Distributed Computing.
- [12] Anne Benoit, Murray Cole, Stephen Gilmore and Jane Hillston.2005. Enhancing the effective utilization of Grid clusters by exploiting on-line performability analysis.In IEEE International symposium on Cluster Computing and the Grid, pp.317-324.
- [13] Shun-Li Ding, Jing-Bo Yuan and Ji-U-Bin Ju.2004.An algorithm for agent based task scheduling in grid environments, proceedings of International Conference on Machine Learning and Cybernetics, p2809- 2814.
- [14] Huyn zhang, chanle wu, Q.xiong, and L.Wu,G. Ye. 2006. Research on an Effective Mechanism of Task Scheduling in Grid Environment.In IEEE, Fifth International Conference on Grid and Cooperative Computing (GCC'06).
- [15] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. 2001.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(6):Pages 810-837.
- [16] G. Ritchie and J. Levine.A.2003.A fast, effective local search for scheduling independent jobs in heterogeneous computing environments.Technical report,Centre for Intelligent Systems and their Applications, School of Informatics,University of Edinburgh
- [17] Li Liu, Yi Yang, Lian Li and Wanbin Shi.2006.Using Ant Optimization for super scheduling in Computational Grid. In IEEE proceedings of the Asia-pacific Conference on Services Computing (APSCC' 06).
- [18] HUI YAN, XUE-QIN SHEN, XING LI, MING-HU MU.2005.An Improved ant algorithm for Job Scheduling in Grid Computing. In fourth IEEE International Conference on Machine Learning and Cybernetics, Guangzhou.
- [19] A.Anjum, R. McClatchey, H. Stockinger, A. Ali, I. Willers, M. Thomas, M. Sagheer, K. Hasham, and O.Alvi.2006. DIANA Scheduling Hierarchies for Optimizing Bulk Job Scheduling. In Proc. Second IEEE International Conference On e-Science and grid computing (e- Science'06).
- [20] T.Wang, X. zhou, Q. Liu, Z. Yang, and Y. Wang.2006. An adaptive Resource Scheduling Algorithm for Computational Grid.In IEEE Asiapacific Conference on Services Computing (APSCC'06).
- [21] Jie Song, Chee-Kian Koh, Simon See, and Gay Kheng.2005. Performance Investigation of Weighted Meta-scheduling Algorithm for Scientific Grid.The 4th International Conference on Grid and Cooperative Computing (GCC 2005) LNCS 3795,pp.1021-1030.