RESEARCH ARTICLE OPEN ACCESS

Deadline-Based Scheduling Algorithm for Divisible-Load in Grid Computing

Prashant Ku. Chandan[1], Yashwant Singh Patel[1], Moumita Ghosh[1], Sarita Das[2], Manoj Ku. Mishra[1]
[1](School of Computer Engineering, KIIT University, Bhubaneswar-24, Odisha, India)
[2](Department of Computer Science & Engineering, EAST, Bhubaneswar-24, Odisha, India)

ABSTRACT
The *divisible load model* is motivated by divisible load theory, where both communication and computation can be arbitrarily divisible into as many independent partitions as required and facilitates a good approximation for many real-world application systems such as those arising in large physics experiments. Scheduling an application with divisible load in data grid is significantly important because of its dynamic nature. Therefore, this paper presents DBSA scheduling model to provide deterministic QoS to arbitrarily divisible applications executing in a grid environment. In addition, the simulation uses a more realistic platform and provides an analysis of the algorithm for homogeneous platforms and presents the comparison results with multi-round algorithm known as UMR (Uniform Multi Round) based on two factors cost and makespan. The simulation result shows that the proposed DBSA minimizes the makespan, cost and balance the load more efficiently.

Keywords- Divisible Load Theory (DLT), Quality of Service (QoS), UMR (Uniform Multi Round).

I. INTRODUCTION

Initially, the grid computing provided vast computational platform to the applications requiring high performance computations. Day by day, the use of grid has modified its own definition. Now a day, besides computational resources, data services are also provided by the grid computing to various kinds of applications ranging from vast scientific and research applications to short term on-demand applications. Grid computing creates a virtual platform where dynamic, geographically dispersed, heterogeneous computer resources connected over high latency networks are combined together to achieve a common goal. The emergence of grid includes; (i) use of remote, (ii) underutilized resources and (iii) need to execute large applications with less cost. But, to find an idle system and to allocate resources properly to the appropriate applications is a challenging task in grid environment.

Grid scheduling is defined as the process of mapping applications over multiple numbers of administrative domains. This process contains searching of multiple administrative domains for the resources to execute jobs at a single machine or schedule a single job to multiple resources at a single site or multiple sites [1]. Job is described as to be anything that requires resources and the phrase 'resource' means anything that can be scheduled, it can be a machine, disk space, a QoS network and so forth. Grid scheduling involves three major phases- Phase1: Resource Discovery: In this phase resources are gathered for further processing, Phase2: Resource Information Gathering: Detailed information of available resources is accumulated, Phase3: Job Execution: Finally jobs are executed with their assigned resources [2]. A job scheduler is a software application that maintains information about the current utilization of machines to determine idle systems and assigns jobs to them for computation. According to ref. [3] Divisible Load scheduling in grid computing has emerged to take the prevalence of parallel computing where cumulative data load is distributed among lower hierarchical level nodes and processors.

The remainder of this paper is organized as follows. In section II, a few divisible load scheduling models are briefly discussed along with their shortcomings. In section III, A "Deadline-Based Scheduling Algorithm for Divisible-Load in Grid Computing Environment" has been proposed. The simulation result is shown in section IV. The final section concludes the paper with discussion and analysis of results.

II. RELATED WORK

The Divisible Load Theory (DLT) is a powerful model for modeling data intensive grid problem where both communication and computation load can be partitioned arbitrarily among a number of processors and links, respectively. Many models are

proposed based on DLT. Some of them are presented here.

Dantong Yu et al. [4] proposed a model for grid computing platform using divisible load scheduling theory. They use the divisible load (data) scheduling theory for grid scheduling and predict the performance. They discussed an example regarding the STAR experiment in the RHIC project. A simple divisible calculation using typical grid infrastructure numbers is also presented to illustrate the suitability of divisible load theory for grid problems.

In the paper [5], architecture of decoupled scheduling for data intensive applications is proposed. They proposed a model of Task Data Present (TDP). The experimental results have shown that data transfer is minimum, when the job is scheduled on that site which contains the data. In this case, when there is no replication of data the response time suffers. It happens because only few sites those are hosting the data are completely overloaded. The drawback of this model is it maps tasks only to those sites which contain the required data. In this model only communication time is considered but not for dividing the load.

To overcome this draw back Monir Abdullah et al. [6] proposed a new model namely Adaptive TDP (ATDP) model which reduces the makespan. They try to balance the load by considering the whole system (all sources), in other word, the node speed fraction was calculated together with communication time. Here both communication and computation time are considered. In addition to this, paper [7] proposes another model named as A^2DLT which considers both the communication time as well as computation time. These models are better than TDP because TDP model is proposed without considering input transfer time. But main problem with this model is that it transfers data from site to the working node without considering bandwidth and processing capability of the working node.

In divisible load theory, a single round scheduling strategy is addressed in [8]. In this strategy the load is completely divided and distributed among all the nodes for processing in a single round. In this strategy, the processing nodes that are waiting for data transmission will be affected by long idle times. To resolve this issue for data intensive applications, a multi-round strategy has been proposed in [9]. In this strategy application is divided and again subdivided into fractions and distributed in a repetitive sequence. The multi-round strategy uses pipelining and thus reduces the idle time.

Yang Yang et al. [10] proposed a multi-round algorithm to schedule parallel divisible workload applications named as UMR (Uniform Multi Round). This algorithm uses multiple rounds for overlapping communication and computation among a master and several workers. However rounds must be uniform and in each round master dispatches identical chunks to all the workers. So this results in an approximately optimal number of rounds.

The above algorithms are better for cluster computing environment where resources are more closely connected but may not be suitable in a grid environment where resources are more loosely connected leading to a significant communication cost. Therefore this paper proposes a new divisible load algorithm which computes the application within a given deadline and cost.

III. DEADLINE-BASED SCHEDULING MODEL FOR DIVISIBLE-LOAD IN GRID COMPUTING ENVIRONMENT

The proposed model maximize Divisible-load distribution among the processors and computes the application within the given deadline and cost in a homogeneous environment.

A. Task Model

Each divisible job *Ji* of an application is characterized by a 3-tuple (S_i, $Load_i$, D)**,** where $S_i \geq 0$ is the startup time of the working node, $Load_i > 0$ is the total load size of *Ji*, and $D > 0$ is the user defined relative deadline, indicating that it must complete execution by time-instant $S_i + D_i$.

B. System Model

The computing cluster used in DLT is comprised of a head node denoted by W, which is connected to *m* working nodes i.e. processing nodes denoted by p_1, p_2, $p_3 \ldots p_m$. All processing nodes have the same computational power and all the links from the head to the working nodes have the same bandwidth as shown in fig. 1.

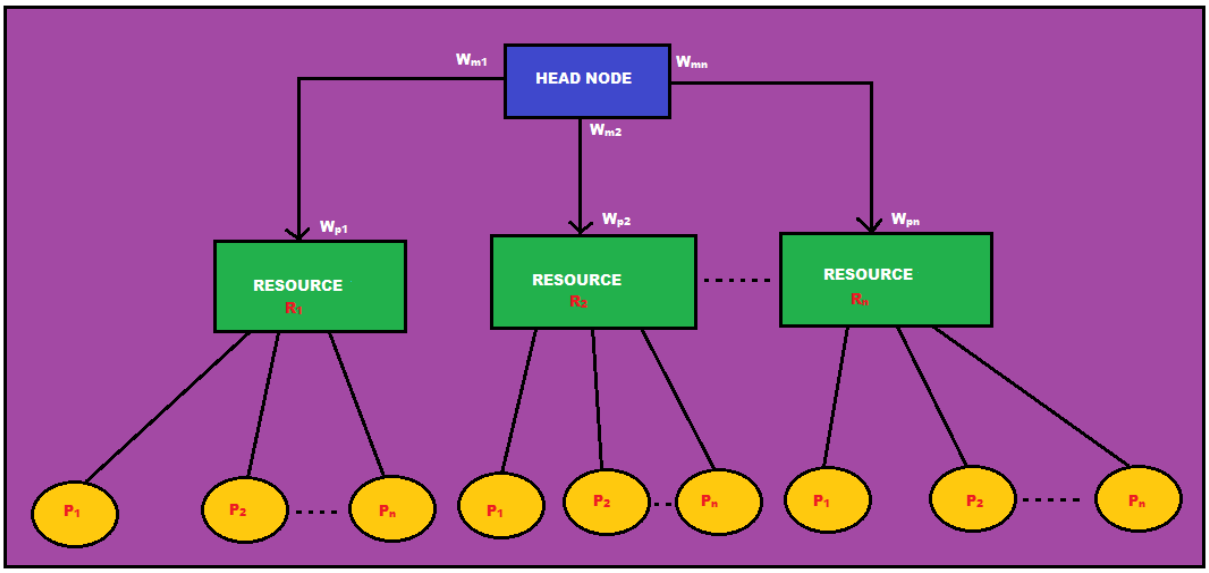

Fig. 1 System Model

C. Assumptions for the proposed model

1) The Grid is a collection of geographically distributed clusters.
2) The Load of the application is divisible i.e. where both communication and computation of the application can be partitioned arbitrarily among a number of processors and links respectively. Each divisible part is called job.
3) The head working node of the cluster does not participate in the computation – its role is to accept or reject incoming jobs, execute the scheduling algorithm, divide the received workload and distribute data chunks to the working nodes.
4) Data transmission does not occur in parallel i.e. at any time, the head working node may be sending data to at most one working node. And working node transfer data to processing node for parallel execution. And each processing node completes executing one job's chunk before moving on to the chunk of any next job that may also have been assigned to it.
5) The head working node and each processing node is non-preemptive. In other words, the head node completes the dividing and distribution of one job's workload before considering the next job.
6) Different jobs are assumed to be independent of one another; hence, there is no need for processing nodes to communicate with each other.
7) All the resources upon which a particular job will be distributed by the head node are available for that job over the entire time-interval between the instant that the head node initiates data-transfer to any of these nodes, and the instant that it completes execution upon all these nodes.
8) Each processors of a Cluster have same bandwidth and processing capability.

IV. SCHEDULING ALGORITHM

The following notations are used:
D = Deadline of the Client Application
W_{Pi} = Cost of processing time per unit workload of the resource i
W_{Mi} = Communication Cost per unit workload of the resource i
S_i = Execution start time of a resource i
Load = The size of workload of the application
LoadR = Remaining load
CHUNK = Total work load
$Chunk_j$ = Load for processor j
LC_{ti} = Local communication cost for unit load of a resource i
LP_{ci} = Local processing cost for unit load of a resource i
GC_{mi} = Global communication time of a resource i
$Makespan_G$ = Global makespan of a resource i

Resource capability can be calculated as follows

W_P = processor capability/number of processors availability in that resource……..(1)
Communication-to-Computation ratio = W_{Mi}/W_{Pi}……..(2)
$GC_{mi} = CHUNK \times W_{Mi}$ …….. ……..(3)
Makespan of jobs for a resource R_i = $Makespan_G = GC_m + D_i$……..(4)
Processing cost of the executing load in a resource = Makespan × Cost per unit time ……..(5)

DBS ALGORITHM

1. For i=1 to n
2. W_{pi} = LP_c of a processor in Ri \ number of processors in R_i \\ Resource capability

3. Sort the Communication-To-Computation ratio of resources ($W_{M[i]} / W_{P[i]}$) in decreasing order.
4. LoadR←Load
5. While (LoadR > 0)
6. Take resources one by one in decreasing order by their weight ratio ($W_{M[i]} / W_{P[i]}$)
7. D_i ←Set the deadline for the resource i
8. $S_i \leftarrow 0$
9. $Alloc_i \leftarrow 0$
10. j ←1 // Take the jth processor of resource Ri
11. While (j ≤ m)
12. Begin
13. If (LoadR ≤ 0) then
14. Break EndIf
15. If ($S_i > D_i$) then
16. Break EndIf
17. $Chunk_j = (D_i - S_i) / (LC_{ti} + LP_{ci})$
18. If ($Chunk_j$ > LoadR) then
19. {
20. $Chunk_j$ = LoadR
21. }
22. If ($Chunk_j$ > LoadR && j = 1)
23. {
24. $D_i = Chunk_j (LC_{ti} + LP_{ci})$
25. }
26. $Comp.Time_j = Chunk_j \times LP_{ci}$
27. $Comm.Time_j = Chunk_j \times LC_{ti}$
28. $S_i = S_i + Comm.Time_j$
29. $CHUNK_i = CHUNK_i + Chunk_j$
30. $Alloc_i = Alloc_i + Chunk_j$
31. LoadR = LoadR – $Alloc_i$
32. Else
33. j = j+1
34. Endwhile
35. $GC_{mi} = CHUNK_i \times W_{Mi}$
36. $Makespan_G = GC_{mi} + D_i$
37. Dispatch different chunks to their respective processors one after another in the order of their index.
38. If (LoadR > 0) then
39. Load = LoadR and Schedule in the next scheduling cycle with a next resource and new deadline.

V. EXPERIMENTS AND RESULTS

The proposed work is implemented in java and J2EE to simulate a grid environment for the experiment. Simulation is done by taking the user input. The application asks for number of working node or resources. User enters the value of deadline, load, bandwidth and processing capability of processor. This section consists of two parts i) computing platform ii) comparison with previous algorithm.

A. Computing Platform

Here a single cluster is considered as a computing platform. In UMR: A multi-Round Algorithm [10] for scheduling divisible workloads a master/worker model with m worker processes running on n processors. The master sends out chunk to worker/processors over a network. And author assumed that the master uses its network connection in sequential fashion. In our proposed approach makespan is computed for both way serial and concurrent communication. If the graph plots in between the chunk and makespan then it is found that makespan value for the concurrent communication is lesser in comparison of sequence communication.

Method for computing comm. time and comp. time

1. For UMR communication

$Comm.Time = No\ of\ processors \times W_m \times Chunk_i$

$Comp.Time = W_p \times Chunk_i$

$Makespan = No\ of\ processors \times W_m \times Chunk_i + W_p \times Chunk_i$

Cost (RS.) = Makespan × Cost per Sec

2. For DBSA communication

$X = (D-S) / (W_M + W_P)$

$Chunk = MIN(X, Load)$

$Comm.Time = W_m\ X\ Chunk_i$

D=D-Comm.Time

$Comp.Time = W_p \times Chunk_i$

Cost (RS.) = Makespan × Cost per Sec

Here we take example for DBS and UMR Algorithm in which $LW_m = 0.2$, $W_p = 0.6$, We assume that cost for 1 sec = 10INR.

B. Comparison with UMR Algorithm

In this section DBS algorithm is compared to UMR algorithm [10]. The performance of these algorithms is compared with respect to two parameters such as makespan and cost as shown in table 1. Graphs are plotted in between load vs. makespan and load vs. cost as shown in fig. 2 and 3. Here load is same for both the algorithm, with the same load both the algorithm has concluded for different makespan and cost.

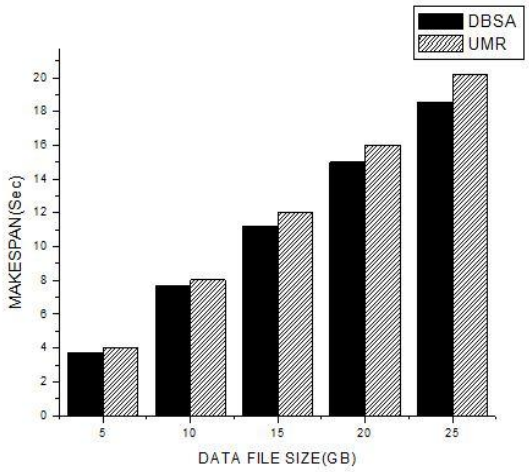

Fig. 2 Comparison with Makespan and Chunk data size.

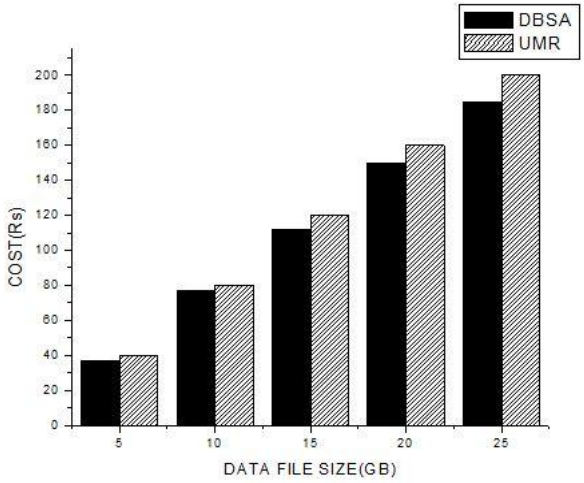

Fig. 3 Comparison with Cost and Chunk data size.

TABLE I

DBSA			UMR	
CHUNK	**MAKESPAN**	**Cost(Rs)**	**MAKESPAN**	**Cost (Rs)**
5	3.7	37	4.008	40.08
10	7.7	77	8.034	80.34
15	11.2	112	12	120
20	15	150	16.008	160.08
25	18.5	185	20.016	200.16

Comparison between DBSA and UMR based on Makespan and Cost

In the above graphs it is concluded that when the Load is same, our proposed algorithm DBS shows minimum makespan and cost in comparison to UMR algorithm. So DBS algorithm gives better result than UMR.

VI. CONCLUSION

The proposed model maximize Divisible-load distribution among the processors of a selected resource and computes the application within the given deadline and cost in a homogeneous environment. The performance of the above algorithms is compared with respect to makespan and cost. Graphs are plotted; load vs. makespan and load vs. cost. When the Load is same, DBS shows minimum makespan and cost and balance the load more efficiently in comparison to UMR. So DBS algorithm gives better result than UMR. In future, the model can be extended to support a heterogeneous grid environment.

REFERENCES

[1] Jennifer M. Schopf, "Ten actions when grid scheduling - The User as a Grid Scheduler", http://www.mcs.anl.gov/uploads/cels/papers/P1076.pdf.

[2] D. Thilagavathi and Dr. Antony Selvadoss Thanamani, "HEURISTICS IN GRID SCHEDULING" *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),Volume 2 Issue 8, August 2013.*

[3] Thomas G. Robertazzi and Dantong Yu, Multi-Source Grid Scheduling for Divisible Loads, *40th annual conference on information sciences and systems*, princeton university, march 22–24, 2006.

[4] D. Yu and T. G. Robertazzi, "Divisible Load Scheduling for Grid Computing", *the 15th IASTED International Conference on Parallel and Distributed Computing And Systems*, November, 2003, Marian Del Rey, CA, USA.

[5] K. Ranganathan, I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," *11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[6] M. Abdullah, M. Othman, H. Ibrahim and S. Subramaniam, "A New Load Balancing Scheduling Model in Data Grid Application", *International Symposium*

on Information Technology Volume:1, 2008, pages 1-5.

[7] Othman, M., M. Abdullah, H. Ibrahim and S. Subramaniam, 2007. A^2DLT: Divisible load balancing model for scheduling communication intensive grid applications: Computational science. *Lecture Notes Comput. Sci.,* 5101: 246-253. DOI: 10.1007/978-3-540-69384-0_30.

[8] X. Lin, Y. Lu, J. Deogun, and S. Goddard. Real-time divisible load scheduling for cluster computing. Technical Report TR-UNL-CSE-2005-0014, CSE Department, University of Nebraska-Lincoln, December 2005.

[9] X. Lin, Y. Lu, J. Deogun, S. Goddard Multi-round real-time divisible load scheduling for clusters *15th International Conference on High Performance Computing, Springer (2008), pp. 196–207.*

[10] Yang, Y.; Casanova, H., "UMR: a multi-round algorithm for scheduling divisible workloads," *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* , pp.9, 22-26 April 2003