New Optimal Load Allocation for Scheduling Divisible Data Grid Applications

M. Othman*, M. Abdullah, H. Ibrahim, and S. Subramaniam

Department of Communication Technology and Network,
University Putra Malaysia, 43400 UPM Serdang, Selangor D.E., Malaysia
mothman@fsktm.upm.edu.my, monabdullah@hotmail.com

Abstract. In many data grid applications, data can be decomposed into multiple independent sub-datasets and distributed for parallel execution and analysis. This property has been successfully employed by using Divisible Load Theory (DLT), which has been proved as a powerful tool for modeling divisible load problems in data-intensive grid. There are some scheduling models have been studied but no optimal solution has been reached due to the heterogeneity of the grids. This paper proposes a new model called Iterative DLT (IDLT) for scheduling divisible data grid applications. Recursive numerical closed form solutions are derived to find the optimal workload assigned to the processing nodes. Experimental results show that the proposed IDLT model obtains better solution than other models (almost optimal) in terms of *makespan*.

Keywords: Divisible Load Theory, Data Grid, Load Balancing.

1 Introduction

In the last decade, data grids have increasingly become popular for a wide range of scientific and commercial applications [1]. Load balancing and scheduling play a critical role in achieving high utilization of resources in such environments [2]. Scheduling an application is significantly complicated and challenging because of the heterogeneous nature of a grid system. Grid scheduling is defined as the process of making scheduling decisions involving allocating jobs to resources over multiple administrative domains [8,9]. Most of the scheduling strategies try to reduce the *makespan* or the maximum completion time of the task which is defined as the difference between the time when the job was submitted to a computational resource and the time it completed. *makespan* also includes the time taken to transfer the data to the point of computation if that is allowed by the scheduling strategy [5].

In other hand, in many data intensive grid applications, data can be decomposed into multiple independent sub datasets and distributed for parallel

* The author is also an associate researcher at the Lab of Computational Science and Informatics, Institute of Mathematical Research (INSPEM), University Putra Malaysia.

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 165–174, 2009.

execution and analysis. High Energy Physics (HEP) experiments fall into this category [7]. HEP data are characterized by independent events, and therefore this characteristic can be exploited when parallelizing the analysis of data across multiple sites. In [11], the DLT paradigm has emerged as a powerful tool for modelling data-intensive computational problems incorporating communication and computations issues [5]. An example of this direction is the work by [3] where the DLT is applied to model the grid scheduling problem involving multiple sources to multiple sinks. In that model, they did not consider the communication time. Whereas, the scheduling in grid applications must consider communication and computation simultaneously to achieve high performance. Some related materials to the problem addressed in this paper can be found in [4,7,8,9,11].

2 Scheduling Model

We consider the problem of scheduling large-volume loads (divisible loads) within in multiple sites. Communication is assumed to be predominant between such cluster nodes and is assumed to be negligible within a cluster node [3,4]. This section describes the scheduling model, the notations, the cost model, and the optimality criterion that are used in our research.

We use the scheduling model that was used by [3,4,8]. It can be described as follows. The target data intensive applications model can be decomposed into multiple independent sub tasks and executed in parallel across multiple sites without any interaction among sub tasks [5]. Lets consider job decomposition by decomposing input data objects into multiple smaller data objects of arbitrary size and processing them on multiple virtual sites. For example in theory, the HEP jobs are arbitrarily divisible at event granularity and intermediate data product processing granularity. Assume that a job requires a very large logical input data set D consists of N physical datasets and each physical dataset (of size L_k) resides at a data source (DS_k, for all $k = 1, 2, \ldots, N$) of a particular site. Figure 1 shows how D is decomposed onto networks and their computing resources.

The scheduling problem is to decompose D into datasets (D_i for all $i = 1, 2, \ldots, M$) across M virtual sites in a Virtual Organization given its initial physical decomposition. We assume that the divisible data can be analyzed at any site.

2.1 Notations and Definitions

All notations and their definitions used throughout this paper are shown in Table 1.

2.2 Cost Model

The execution time cost (T_i) of a subtask allocated to the site i and the turn around time ($T_{Turn\_Around\_Time}$) of a job J can be expressed as follows

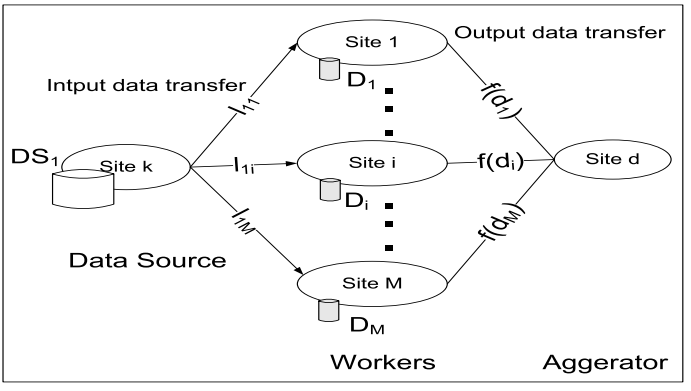

Fig. 1. Data decomposition and their processing

Table 1. Notation and Definition

Notation	Definition
M	The total number of nodes in the system
L	The loads in data file
α_i	The fraction of load that node i will receive from the data file
L_i	The amount of load that node i will receive from the data file
w_j	The inverse of the computing speed of node i
Z_in_i	The link between node i and the data source
Z_out_i	The link between node i and the aggrator
T_i	The processing time in node i

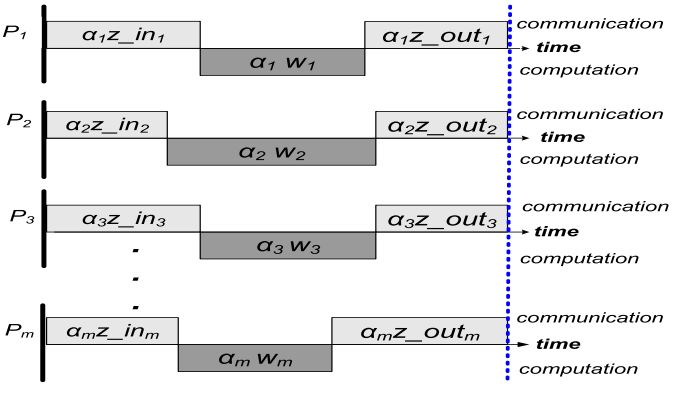

Fig. 2. The communication and computation of sources within the system (optimal case)

$$T_i = T_{input\_cm}(i) + T_{cp}(i) + T_{output\_cm}(i, d)$$

and

$$T_{Turn\_Around\_Time} = \max_{i=1}^{N}\{T_i\},$$

respectively. The input data transfer $T_{input\_cm}(i)$, computation $T_{cp}(i)$, and output data transfer to the client at destination site d, $T_{output\_cm}(i, d)$ are presented as

$$T_{input\_cm}(i) = L_i \cdot \frac{1}{Z_i}, \quad T_{cp}(i) = L_i \cdot w_i \cdot ccRatio$$

and

$$T_{output\_cm}(i, d) = f(L_i) \cdot Z_{id}$$

respectively. Where $L_i = L \cdot \alpha_i$ and the function $f(d_i)$ is an output data size and *ccRatio* is the non-zero ratio of computation and communication. The turn around time of an application is the maximum among all the execution times of the sub tasks.

The problem of scheduling a divisible job onto M sites can be stated as deciding the portion of original workload (D) to be allocated to each site, that is, finding a distribution of α_i which minimizes the turn around time of a job. The proposed model uses this cost model when evaluating solutions at each generation.

2.3 Optimality Criterion

In all literatures related to divisible load scheduling [7,9], an optimality criterion is used to derive an optimal solution which stated as follows. It states that in order to obtain an optimal processing time, it is necessary and sufficient that all the sites that participate in the computation must stop at the same time. Otherwise, load could be redistributed to improve the processing time. The timing diagram for this distributed system in optimal case is depicted in Fig. 2.

3 Proposed IDLT Model

The load scheduling problem is to decompose D into datasets (D_i for all $i = 1, 2, \ldots, M$) across M virtual sites in a Virtual Organization given its initial physical decomposition. This model includes two steps

3.1 Initial Solution

The proposed model will start from a good initial solution. ADLT and A^{2}DLT models will be used for this purpose. The best solution (minimum *makespan*) will be considered as an initial solution of the iterative model. As it is explained in [8,9,11], ADLT and A^{2}DLT models produced good results for computation and communication intensive applications, respectively.

3.2 The Iterative Model

The optimality criterion that discussed in section 2.3 will be used in the design of the load distribution strategy. The IDLT model involves the following steps

1. First, we divide the load using one of the adaptive DLT models.
2. Calculate the *makespan* using the cost model.
3. If all nodes finish at the same time, go to step 8 else go to step 4.
4. Then, we calculate the summation (Sum) of the processing time of the nodes.
5. Next, we calculate the average time by $avg = Sum/M$.
6. After that, we redistribute the load depending on the average (avg) based on the iterative numerical equation that will be discussed later in this section.
7. Go to step 2
8. The current time is the final *makespan*.
9. End

Here, we will discuss step by step the derivation of a closed form equation by which one can calculate the optimal fraction of the load that has to be assigned to each processing node in order to achieve the minimum *makespan* and the optimal data allocation for each processor. The processing nodes (w_i), communication links (Z_i)and applications types ($ccRatio$) were assumed to have different values.

After we calculate the *makespan* as an initial solution to the IDLT model, we will compute the $\sum_{i=1}^{M} T_i$ where M is the number of the processing nodes. Based on cost model, we have

$$T_i = L_i \cdot z\_in_i + w_i \cdot L_i \cdot ccratio + L_i \cdot z\_out_i \tag{1}$$

where z_in_i and z_out_i is the input communication time and output communication time of link i, respectively. Then, compute the average of completion time as

$$avg = \frac{\sum_{i=1}^{M} T_i}{M}. \tag{2}$$

In any iteration, the *makespan* of any nodes must equal to (avg) in order to get the optimal solution. It means that, the load is distributed among the processing node M equally. While the average time is the processing time of load α that is calculated by (1), we can recalculate the amount of α if we have the (avg). In general, if avg is

$$avg = \alpha \cdot z + \alpha \cdot w \tag{3}$$

The load fraction α can be calculated by (4),

$$\alpha = \frac{avg}{z + w} \tag{4}$$

In this case, for any iteration, after we calculate the time average of processing any load, we can calculate the amount of this load. Furthermore, if the processing time of the first node to process α_1 is

$$avg = \alpha_1 \cdot z\_in_1 + \alpha_1 \cdot w_1 \cdot ccRatio + \alpha_1 \cdot z\_out_1 \tag{5}$$

The α_1 will be calculated by:

$$\alpha_1 = \frac{avg}{z\_in_1 + (w_1 \cdot ccRatio) + z\_out_1} \tag{6}$$

Consequently, the α_2 of the second node will be calculated as

$$\alpha_2 = \frac{avg}{z\_in_2 + (w_2 \cdot ccRatio) + z\_out_2} \tag{7}$$

Finally, the α_M of the M^{th} node will be

$$\alpha_M = \frac{avg}{z\_in_M + (w_M \cdot ccRatio) + z\_out_M} \tag{8}$$

Equation (8) is correct only in the last iteration (when we get the optimal solution). But in the first iteration, the last node will take the rest of the load only as

$$\alpha_M = (1 - (\alpha_1 + \alpha_2 + \cdots + \alpha_{M-1})) \cdot L \tag{9}$$

There are M-1 equations. An additional equation is called a normalization equation, which states that the summation of the all the allocation fractions should be 1.

$$\alpha_1 + \alpha_2 + \cdots + \alpha_M = 1 \tag{10}$$

The load of the last node does not equal the load that produces the (avg) time. Thats means, the last node still does not take the optimal load. Here, we will compute the new *makespan* (the *makespan* will be calculated by cost model that discussed in Section 2.2 every iteration).

Consequently, we will carry out these steps in the second iteration. In this iteration, the *makespan* will be reduced but still not optimal. Therefore, we will carry out these steps until certain termination condition happens. The termination condition here is the optimality criterion. All nodes must finish the load processing at the same time.

The IDLT model for single source produces almost optimal solution after some iterations. There is a very small different among the nodes processing time (small fractions). In this model, all nodes will take the new load based on the new average. It means that all nodes will finish at the same time. For the last node will take the rest of load without considering the average. For the next iteration, the new average will be reduced. The last node will take more load than previous iteration. After some iterations the last node will finish as same time as the others.

4 Experimental Results

To measure the performance of the proposed IDLT model against CDLT, ADLT and A^2DLT models, randomly generated experimental configurations were used [5,8,9]. The estimated expected execution time for processing a unit dataset on

Table 2. Results IDLT Model: Step-by-Step

Steps	Node 1	Node 2	Node 3	Node 4	Node 5
0	5643.84	1440.41	2993.02	843.48	1108.35
5	1822.61	1822.61	1822.61	1822.61	1757.71
10	1799.71	1799.71	1799.71	1799.71	1798.61
11	1799.49	1799.49	1799.49	1799.49	1799.00

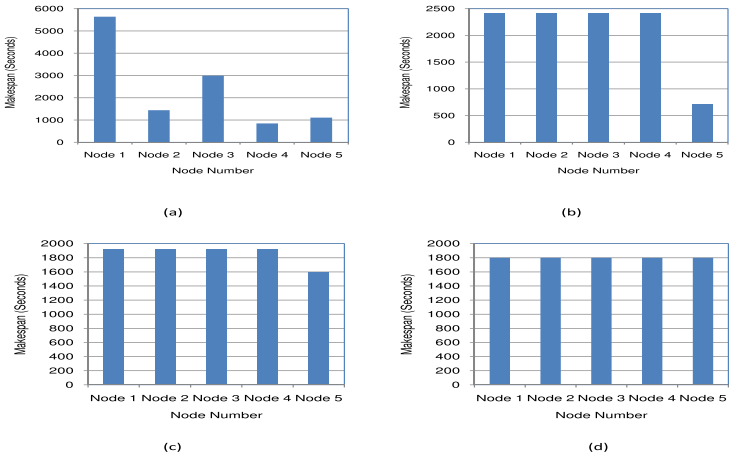

Fig. 3. Comparison of various iterations for IDLT model, (a) initial step, (b) and (c) intermediate steps, and (d) final step

each site, the network bandwidth between sites, input data size, and the ratio of output data size to input data size were randomly generated with uniform probability over some predefined values. The network bandwidth between sites is uniformly distributed between 1 Mbps and 10 Mbps.

We examined the overall performance of each model by running them under 100 randomly generated Grid configurations. We varied the parameters, *ccRatio* (0.001 to 1000), r_{cb} (10 to 500), and M (1 to 100). Thus, from series of experiments, it can be concluded that the IDLT gives an optimal solution and all nodes stop at the same time.

Let considers the system and loads with processing nodes M=5 and *ccRatio* =0.001, respectively. Using the IDLT model, we have the experimental results as shown in Table 2 and Fig. 3. In an initial solution, the maximum completion time or *makespan* is 5643.84 seconds. It is reduced until *makespan* 1799 seconds. In the last step, all nodes finished the processing at the same time (at 1799 Seconds). Furthermore, when we increase the number of iteration, all nodes will finish at almost the same time (1799.32 seconds) and it is improved approximately 68%.

Figure 3 showed that, the five processing nodes finish at almost same time. It means that, in the last iteration, the load is distributed to the processing node almost equally.

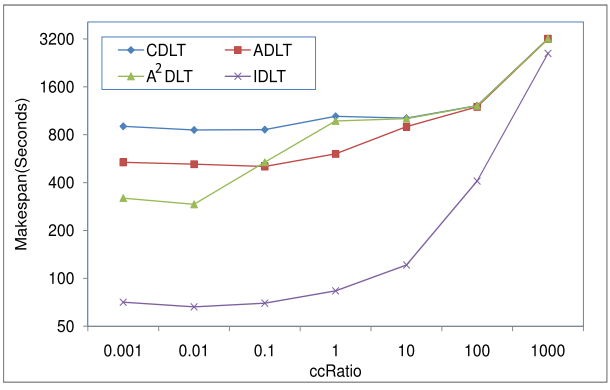

Fig. 4. *Makespan* vs. *ccRatio* for CDLT, ADLT, A^{2}DLT and IDLT(M=100)

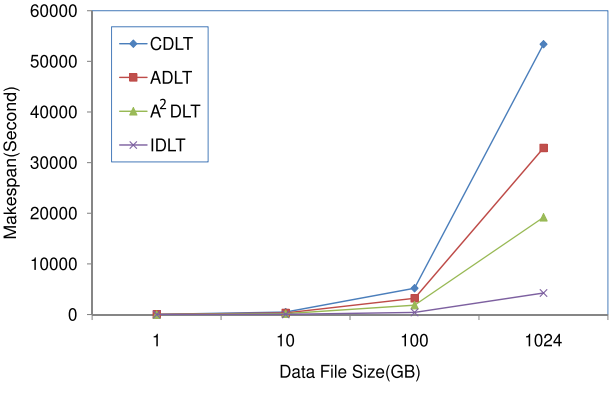

Fig. 5. *Makespan* vs. data file size for CDLT, ADLT, A^{2}DLT and IDLT(M=100 and *ccRatio*=0.001)

To show how these models perform on different type of application (different *ccRatio*=100), we executed the model and plotted all the results as shown in Fig. 4. From the plotted graph, the IDLT model is the best for any type of application. As expected, the IDLT model produce the almost optimal solution from single source.

Different size of data files are used considering large scale data grid. It is varied from 1 GB to 1 TB. Figure 5 clearly showed that the IDLT model produce better results for all sizes.

The convergence metric records how the initial *makespan* value minimized during the iteration between the initial and optimal solutions. Figure 6 depicts the convergence of the models for the average out of ten executions for when the *ccRatio* is equal to 1000. All models have same results, whereas the result of IDLT model is significantly reduced as the number of iteration increased.

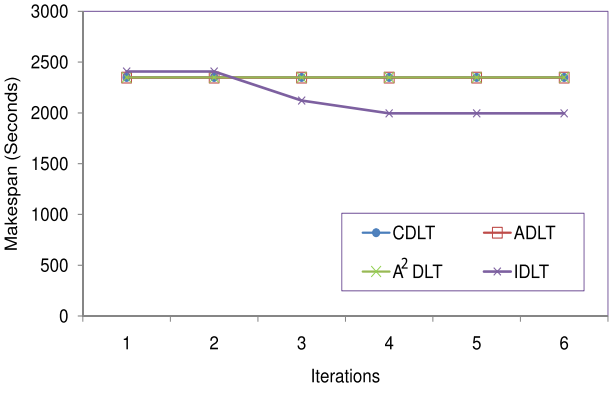

Fig. 6. Convergence for IDLT model for single source ($ccRatio$=1000))

5 Conclusion

In this paper, we have developed an effective iterative model for optimal divisible load allocation. The IDLT model is proposed for load allocation to processors and links for scheduling divisible data grid applications. The experimental results showed that the proposed IDLT model is capable of producing almost optimal solution for single source scheduling. Hence, the proposed model can balance the processing loads efficiently and can be embedded into the existing data grid schedulers.

References

1. Tierney, B., Johnston, W., Lee, J., Thompson, M.: A Data Intensive Distributed Computing Architecture for Grid Applications. Future Generation Computer Systems 16(5), 473–481 (2000)
2. Xiao, Q.: Design and Analysis of a Load Balancing Strategy in Data Grids. Future Generation Computer Systems 16(23), 132–137 (2007)
3. Tang, M., Lee, B.-S., Tang, X., Yeo, C.-K.: The Impact of Data Replication on Job Scheduling Performance in the Data Grid. Future Generation Computer Systems 22(3), 254–268 (2006)
4. Wong, H.M., Veeravalli, B., Dantong, Y., Robertazzi, T.G.: Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints. In: Proceeding of the IASTED Conference on Parallel and Distributed Computing and Systems, Marina del Rey USA, 7-11 (2003)
5. Kim, S., Weissman, J.B.: A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications. In: IEEE Proceeding of the International Conference on Parallel Processing, Washington DC, USA, vol. 1, pp. 406–413 (2004)
6. Venugopal, S., Buyya, R., Ramamohanarao, K.: A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing. ACM Computing Surveys 38(1), 1–53 (2006)
7. Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics for Scheduling Jobs on Computational Grids. In: Proceedings of 8th IEEE International Conference on Advanced Computing and Communications, pp. 45–52 (2000)

8. Othman, M., Abdullah, M., Ibrahim, H., Subramaniam, S.: Adaptive Divisible Load Model for Scheduling Data-Intensive Grid Applications. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 446–453. Springer, Heidelberg (2007)
9. Othman, M., Abdullah, M., Ibrahim, H., Subramaniam, S.: A^2DLT: Divisible Load Balancing Model for Scheduling Communication-Intensive Grid Applications. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part I. LNCS, vol. 5101, pp. 246–253. Springer, Heidelberg (2008)
10. Viswanathan, S., Veeravalli, B., Robertazzi, T.G.: Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid Systems. IEEE Transaction of Parallel and Distributed Systems 18(10), 1450–1461 (2007)
11. Bharadwaj, V., Ghose, D., Robertazzi, T.G.: Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. Cluster Computing 6(1), 7–17 (2003)