Incorporating Data Movement into Grid Task Scheduling

Xiaoshan He[1], Xian-He Sun[1]

[1]Department of Computer Science, Illinois Institute of Technology
Chicago, Illinois, 60616, USA
{hexiaos, sun}@iit.edu

Abstract. Task Scheduling is a critical design issue of distributed computing. The emerging Grid computing infrastructure consists of heterogeneous resources in widely distributed autonomous domains and makes task scheduling even more challenging. Grid considers both static, unmovable hardware and moveable, replicable data as computing resources. While intensive research has been done on task scheduling on hardware computing resources and on data replication protocols, how to incorporate data movement into task scheduling seamlessly is unrevealed. We consider data movement as a dimension of task scheduling. A dynamic data structure, Data Distance Table (DDT), is proposed to provide real-time data distribution and communication information. Based on DDT, a data-conscious task scheduling heuristics is introduced to minimize the data access delay. A simulated Grid environment is set up to test the efficiency of the newly proposed algorithm. Experimental results show that for data intensive tasks, the dynamic data-conscious scheduling outperforms the conventional Min-Min significantly.

1 Introduction

Grid computing provides a seamless access to immerse network resources, such as high-performance computers and networks, or otherwise unavailable data files. The widely available network resources, however, are geographically distributed, heterogeneous, and under autonomous administration domains. Task scheduling is a vital issue of Grid computing, and, on the other hand, many technical challenges need to be addressed before an efficient task scheduling strategy can be developed. In this study, incorporating data movement and replication into task scheduling is proposed. Consequently, a light-weighted dynamic adjustable strategy for integrating data movement delay with task execution scheduling is introduced. A scheduling heuristic, which treats data as one dimension of the quality of service, is derived to address the issue of data-conscious task scheduling of Grid computing.

Intensive research has been conducted in parallel and distributed task scheduling [1, 2, 3]. Task scheduling can be classified as parallel scheduling, where the tasks may be from the same application and have inherent dependence relations, and metatask scheduling, where the tasks are independent from each other [4, 5, 6]. Current Grid scheduling research has been focusing on metatask scheduling. We will focus on metatask scheduling in this study as well. There are two categories of modes of metatask: Online mode and batch mode. Online mode schedules a task upon its arrival whereas batch mode schedules the tasks periodically after a fixed time period. Batch mode is a better choice for busy systems. Among various batch mode scheduling heuristics, the Min-Min is the most frequently used heuristic. In this work, we consider data movement as a factor of QoS requests, and we extend Min-Min to integrate the dynamic adjustment of data access cost, thus achieving a better scheduling of Grid tasks.

Data has been recognized as an important resource of Grid computing. Some recent works have addressed data movement in task scheduling. Current research is developed along two directions: allocate the task to where the data is, and move the data to where the task is. Researches on integrating data movement with task scheduling seamlessly, however, is rare. Xsufferage [7] considers the data location in task scheduling. It finds the "site-level sufferage value" to avoid unnecessary data replication. For each task and each site, the function *f* computes the minimum completion times of the task over the hosts in the site. This minimum is called site-level completion time. For each task, f returns the difference between the second minimum and the minimum site-level completion time. The difference is called site-level sufferage value. Xsufferage allocates task close to data in order to reduce data movement, but does not incorporate the data movement into the scheduling.

Other scheduling heuristics for data-intensive tasks include [8]. [8] tries to statistically predict the data request and get data ready before it is called. It schedules data movement based on its prediction, and a decoupling strategy is proposed to separate data movement scheduling from task scheduling. Within the decoupled framework, data movements are operated in a decoupled, asynchronous process on the basis of observed data access patterns and load. [8] contributes to data usage prediction and moves the data to where the task is, but tries to schedule data and task separately. In this study, we intend to incorporate data movement into task scheduling. We use a newly introduced data structure to measure the dynamic data movement cost, and integrate this cost into an extended Min-Min scheduling heuristics. We dynamically adjust data replica based algorithm to get data ready on under-utilized sites, before possible load imbalance occurs.

The organization of this paper is as follows. In section 2, the Grid scheduling model deployed in this study is introduced. Then, a dynamic adjusting strategy for moveable resources is proposed in section 3. Section 4 introduces a simulated Grid environment. In section 5, the experimental results are presented and discussed. We conclude this study in section 6.

2 The Scheduling Model in Grid

The Grid considered in this study is composed of a number of non-dedicated hosts and each host is composed of several computational resources, which may be homogeneous or heterogeneous. To simulate the computational hosts, we represent each host with three local parameters for the utilization, arrival rate, and standard deviation of service time, respectively. These parameters are the recorded average performance of the computing hosts. The host set is the set of available hosts in the Grid:

$$Hosts = \{H_1, H_2, \ldots, H_m\} \tag{1}$$

Each host in Hosts is composed of the processor sets P_m available in H_m, and the data Set D_m available in H_m.

$$H_m = P_m \cup D_m \tag{2}$$

where D_m is the set of all the data sets available in S_m

$$D_m = \bigcup_{d_n\_is\_on\_host\_m} d_n \tag{3}$$

The metatask to be scheduled is composed of a set of independent tasks that is to be executed remotely: $M = \{T_1, T_2, \ldots, T_l\}$. For each of those independent tasks T_i, it has a set of independent subtasks: $T_i = \{t_{i1}, t_{i2}, \ldots, t_{ij}\}$. t_{ij} denotes the subtask j of task T_i. I_{ij} denotes the input data requested by the subtask j of task T_i. Therefore, the input data requested by task i is I_i:

$$I_i = \bigcup_{j=1}^{|T_i|} I_{ij} \tag{4}$$

A scheduler allocates tasks by selecting the "best" match from a pool of available resources. Before moving to the scheduling heuristics, let us first review some terms and definitions [5, 6].

The expected execution time ET_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given that m_j has no load when t_i is assigned. The expected completion time CT_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let m be the total number of the machines in the H/C suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of task t_i be a_i, and the beginning time of t_i be b_i. From the above definitions, $CT_{ij} = b_i + ET_{ij}$. Let CT_i be CT_{ij}, where machine j is assigned to execute task i. The makespan for the complete schedule is then defined as $max\{t_i \in K\}$ (CT_i). Makespan is a measure of the throughput of the heterogeneous computing system. The objective of Grid scheduling is to minimize the makespan. It is well known that the problem of deciding on an optimal assignment of tasks to resources is NP-complete. Heuristics are developed to solve the NP-complete problem. In this paper, we mount dynamic data adjusting to task scheduling heuristics to achieve efficient Grid task scheduling.

A *scheduling hole* is defined as an idle period of a host between two busy periods. Scheduling holes may be a result of the mismatching of resources and tasks, e.g. the idle host is not qualified to execute waiting tasks. For tasks with data requests, a host without the requested data has to wait for the qualified tasks, so it may cause idling. Meanwhile, tasks that need to access critical data have to wait in the waiting queue to access the data, which even more likely lead to scheduling holes. Integrating data movement with task scheduling will reduce the scheduling holes and, therefore, maximize the host utilization and achieve a better makespan.

3 Dynamic Adjusting Strategy for Moveable Resources

In Grid task scheduling, we not only are concerned with the availability of those hardware static resources, but also with moveable software resources, such as data that reside in a host. For this type of the resource, the load imbalance resulted from resource constraints could be alleviated by adjusting the resource attributes of the hosts. In this section, we intend to investigate those moveable resources, such as data replica, as one dimension of QoS. A dynamic adjusting strategy for such problems is proposed to address the QoS scheduling in data-intensive applications.

In a Grid environment, if a data is only statically available in one host, the executions of tasks that request the data are restricted to that host. Thus, an undesirable congestion brought by data requests from tasks occurs on the host with the critical data. To avoid such congestion, we could leverage the data distribution of the hosts. We define the data whose replica is requested by tasks as a QoS in the system as critical data. The critical data could be replicated to other hosts, so that more hosts are able to provide a certain level of QoS. However, we have to be aware that the replication of data needs communication cost. A model is proposed to implement the scheduling strategy that decides if the replication is worthwhile by evaluating if the cost of the communication surpasses the computation delay brought by scheduling congestion. We have made the assumption that the replicated data we referred to here are the input data that are read-only. Such cases occur in the application, such as gene comparison, etc.

3.1 Data Distance Table

To make the leverage between communication cost of the data replication and the queuing time for the host with data, we have to formulize the information of data replica placement of the hosts and the communication cost of each data replication. A dynamic *data distance table* (DDT), is generated to record the information.

The *data distance* represents the distance between data copy d_n and host H_m. The *data distance* decides the communication cost of replicating data d_n to host H_m. *Data distance* is defined as:

$$dist_{nm} = \begin{cases} 0 & \text{if } d_n \in D_m \\ \dfrac{\min(rep\_time\_mk)}{rep\_time\_std} & \text{if } d_n \notin D_m \end{cases} \quad (5)$$

So, if data n is in host m, then $dist_{ij}$ will be 0. If data n is not in host m, then $dist_{ij}$ will depend on the communication time from host m to the nearest host k that has data n. The communication cost of data replication is normalized by a standard communication time, which is chose as the shortest path between any two hosts. To reflect the real time Grid data distribution, the table is updated once a replication of data happens. Therefore, the data replica placement and the communication cost of replication are represented by a dynamic data distance table, which may look like:

Table 1. A sample data distance table

	h_1	h_2	h_3	h_4
d_1	0	1.2	2	0
d_2	1.5	0	1.5	3
d_3	0	3	0	1.5
d_4	4	2	2	0

A row in the table represents a dataset's distances to each host. A column in the table represents the distance of different datasets to a host. The value of each entry $dist_{ij}$ depends on both the replica of data n and host m, and the communication cost of data *n* to host *m*.

The DDT not only provides the information of data distribution, but also the real time network performance. It enables the calculation of communication cost to be independent from the scheduling heuristic, and thus makes the scheduling heuristic light-weighted. In the heuristic presented in the next subsection, the replication decision is made based on the DDT. DDT is updated periodically based on the data distribution, and the network performance. By looking up the DDT, the nearest data are located, so that the heuristic is well informed when making data replication decision. Hence, a light-weighted data-conscious scheduling heuristic is achieved.

3.2 A Data-conscious Scheduling Heuristic

We have to leverage the communication cost of a data replication with the queuing time of the task in the host with critical data. Intuitively, a data replication can be considered when 1) the communication time can overlap with computing time; and 2) the communication time is less than the data access queuing time on the host with data. However, we have to consider other factors, such as processing overhead and the data update cost.

If we do not replicate the data, undesirable scheduling holes may exist on the receiver hosts due to the data access delay. If we replicate the data at the execution time of tasks, the scheduling holes may also appear, resulting from the data replication delay. Since the scheduling holes could be filled by executing tasks that do not need the critical data, the detriment brought by scheduling holes varies in different situations. So it is necessary to make a tradeoff between the detriment brought by scheduling holes and computation time gain.

However, if the extra copy of the data is created, extra communication cost will be paid for the data replication. The extra communication cost includes the notification of the modification and another round of sending data. A straightforward way to compromise is to compare the two makespan brought by scheduling with and without data replication. If the scheduling with data replication results in a smaller makespan, we replicate the critical data. Otherwise, we do not replicate.

In our heuristics, the replication of datasets should take place when replication of some datasets could bring a shorter makespan of the metatask. The source host of the dataset to be replicated should be an over-loaded host, a host that is queued while scheduling holes exist at other hosts due to the queuing delay. We select a threshold of the queue to decide the time to replicate. Here in our algorithm, we select the threshold as 0. This means we will test to see if we need replication once there are tasks in the queue. As the complexity of the algorithm is low, we trade the computation cost of scheduling heuristic for less cost on waiting for the data.

Based on these considerations, we propose our data-conscious scheduling heuristic in Figure 1, where MCT denotes the Minimum Completion Time of the current metatask according to the scheduling.

```
While there are tasks to be scheduled
    For each task $i$ in $M$ but not run yet
        For all $H_m$ in $Hosts$
            Compute $MCT_{im}$.
        Endfor
        $MCT_{i0} = \min(MCT_{im})$
        If subtask $t_{ij}$ needs any data $d_n$ in set $I_i$
            For all $H_m$ in $Hosts$
                Compute $MCT'_{im}$ based on copying nearest
              copy of $data_j$ to $m$ by looking up
                the $DDT$
            Endfor
            $MCT'_{i0} = \min(MCT'_{im})$
            If $MCT'_{i0} < MCT_{i0}$
              Copy $data_j$ from host 1 to host $r$
                $MCT_{min} = MCT'_{i0}$
            Else
              $MCT_{min} = MCT_{i0}$
        Else
            $MCT_{min} = MCT_{i0}$
    Endfor
    Schedule $MCT_{min}$
Endwhile
```

Fig. 1. Data-conscious Scheduling Heuristic

In the data-conscious scheduling heuristic, we consider the matching of the data request and data replica between the tasks and hosts based on the conventional Min-Min. Similar to Min-Min, the data-conscious scheduling heuristic computes MCT_{im}, the minimum completion times of each task on all the hosts at the start. Then, we get the MCT_{i0} according to the Min-Min heuristic. Instead of selecting the task/host pair, we take a test on all the tasks that request data in the metatask: For each task i with data request $data_j$, firstly, compute the MCT'_{im} based on the replication of $data_j$ for all the m hosts available using the information provided by dynamic *data distance table (DDT)*. Secondly, among the m data replications, algorithm finds the one with the minimum completion time MCT'_{i0}. If MCT'_{i0} is less than the minimum completion time before the replication MCT_{i0}, we make the data replication that result in MCT'_{i0}, and set it as MCT_{min}. Otherwise, we do not make the replication. By repeating the same process as we described on all the tasks, we get a group of data replications and the minimum completion time MCT_{min} for each task. We schedule the data replication, if necessary, and the task/host pair that results in minimum MCT_{min}. The scheduling process will be repeated till all tasks are scheduled. The data-conscious scheduling heuristic will be executed at every scheduling event.

3.3 A Sample Scenario

Tables 2 and 3 give a scenario in which the data-conscious scheduling heuristic outperforms the Min-Min. It shows the expected execution time of four tasks on two hosts. The hosts are assumed to be idle in the beginning. In this particular case, the Min-Min heuristic gives a makespan of 11, whereas the data-conscious scheduling heuristic gives a makespan of 9. Figures 2 and 3 give a pictorial representation of the assignments made for the case in Table 2 and 3.

Table 2 shows the four tasks with their data requests. The entry of required data indicates the data d that is requested by task *i*. Task 4 does not have data request, so it is x. The execution time of the four tasks is assumed to be the same on the two hosts. Table 3 shows the data distribution on two hosts. In this case, data 1, 2, 3 initially reside in host 1, and data 4 is in host 2. The communication cost of data replication is assumed to be 1.

Table 2. The data request of 4 tasks

Task	Execution Time	Required Data
t_1	4	1
t_2	5	2
t_3	2	3
t_4	3	x

Table 3. The data distribution on 2 hosts

Machine	Data Hosted
1	1, 2, 3
2	4

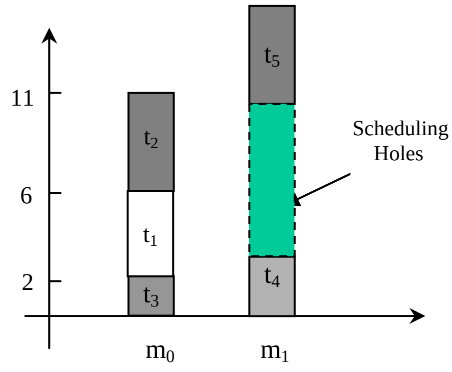

Fig. 2. Min-min gives a makespan of 11

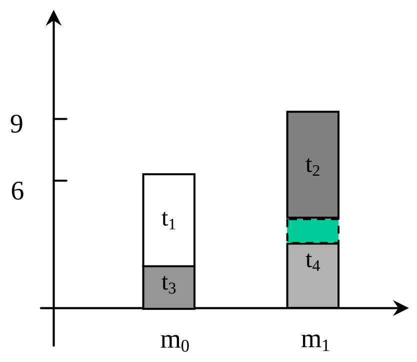

Fig. 3. Data-conscious scheduling heuristic gives a makespan of 9

4 Implementation Details

The goal of implementing the simulation model is to demonstrate the efficiency of our dynamic adjusting strategy. The model consists of a simulated Grid environment and a scheduler that implements different scheduling strategies. The system is illustrated in Figure 4.

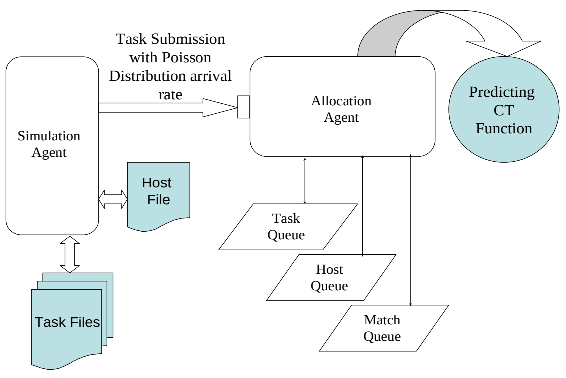

Fig. 4. The scheduling model in a simulated Grid environment

Task Attributes. For each task, the attributes of the task include interarrival time, number of subtasks, data request, and the workloads of the subtasks. The task arrival rate follows the Poisson distribution. The number of the subtasks is generated to randomly fall from 1 to 20. The data request is generated according to different request ratios. The workloads of the subtasks are generated randomly between 1000 and 2000.

Machine Attributes. For each set of machines, the attributes of the machine include number of workstations (or computing nodes), replica of the data, and the information of each workstation, which includes the utilization, arrival rate, and standard deviation of service time. Each data initially has only one copy in the system. Data sizes vary from 0 to 5, and 20 to 40 unit sizes in two experiments. The network speeds between hosts are set to vary from 0 to 1.

5 Experimental Results

5.1 Experiments Overview

We have developed a simulator to compare the performance of the data-conscious scheduling heuristic and the general Min-Min [9] under the same load conditions representing a wide variety of system states. In this simulator, we fix the parameters of the hosts during all these simulations with four data request and distribution scenarios. In addition, we compare the scheduling performance under different communication cost of the data replication. The metric of the comparison is the makespan of the tasks.

Based on the analysis above, the experimental evaluation of the scheduling algorithm is performed in two parts. In the first part, the data-conscious scheduling heuristics and the general Min-Min heuristics will be compared in four data request and distribution scenarios, respectively, with smaller data set size, and thus smaller communication cost. In the second part, the effect of the scheduling frequency will be discussed using a larger data size, thus, larger communication cost, which is averagely about four times as much as in the first experiment.

5.2 Host and Task Setup

As we mentioned before, all configurations about the hosts remain the same during all the simulations. The four non-dedicated networks are settled as follows. Each network has 20 workstations and the local parameters, such as *util*, *arri* and *serstd*, are generated randomly in given scope, such as *util* is from 0.0 to 1.0, arriving rate is from 0.01 to 0.15 and *serstd* is about 25. This setup makes this Grid non-dedicated and heterogeneous. For the data status, we assume there are 60 different data on four hosts. Initially, there is only on copy for each data. The distribution of the data on four hosts varies in the four scenarios. In scenarios (a) and (b), the distribution of the data is uneven. Host 1 has 30 data whereas the host 2, 3, and 4 each has 10 data. In scenarios (c) and (d), the data is evenly distributed. All four hosts have 15 data.

A hundred independent tasks are created based on Poisson distribution. Each task has randomly 1-20 subtasks and each subtask has randomly 1000-2000 work demand. The running time interval is one time unit. For data request, we assume each task requests exactly one data or requests none. The data requests of 100 tasks vary in the four scenarios. In scenarios (a) and (c), the data request percentage from tasks is 50%, whereas in scenarios (b) and (d), the data request percentage from tasks is 35%. Based on the actual world, the four different scenarios simulated are shown in Table 4:

Table 4. The four scenarios of data request and distribution

Scenarios	**Date Request Task %**	**Data Distribution on the 4 Hosts**
a	50%	30, 10, 10, 10
b	35%	30, 10, 10, 10
c	35%	15, 15, 15, 15
d	50%	15, 15, 15, 15

5.3 The comparison based on smaller data size

For each scenario presented in the previous section, we compare the performance of the general Min-Min with the data-conscious scheduling heuristics. For each scenario, we create the 100 tasks 100 times independently and get the average makespan by two algorithms 100 times. The data size is randomly between 0 to 5 unit sizes. Figure 5 and Table 5 show the comparison. The data is in unit size.

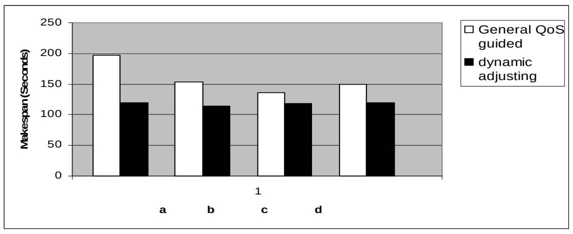

Fig. 5. The comparison based on data size of 0 to 5 unit size

As shown in Figure 5, for all four scenarios the dynamic adjusting scheduling heuristics for data-intensive task outperforms the general Min-Min heuristics. The makespan using data-conscious scheduling heuristics can be as much as 46.02% shorter than that using the general Min-Min. For scenario (a), where the tasks that request the data contribute 50% of the metatask and the data are unevenly distributed, the performance gain reaches as high as 38.57%. For scenario (b), where the tasks that request data are in lower density (35%), a satisfactory performance gain of 25.46% is acquired. For scenario (c), where only 35% of the tasks request data and the data distribution becomes even, the performance gain of the data-conscious scheduling heuristics is relatively small, i.e., 13.21% better than the general Min-Min. Finally, in scenario (d), the data request rate of tasks becomes 50% with all data evenly distributed on the hosts, and the performance gain is 19.70%.

5.4 The comparison based on larger data size

The size of the data plays an important role in the efficiency of the data-conscious scheduling heuristics, since the larger the data size is, the larger the communication cost is. If the communication cost is too high, we would be less likely to overlap the communication of the data by the computation of execution of other tasks.

In this subsection, we compare the performance of the general Min-Min heuristics with the data-conscious scheduling heuristic under different data sizes. The data sizes are randomly generated between 20 to 40 unit sizes. For each scenario, we create the 100 tasks 100 times independently and get the average makespan by two algorithms 100 times. Table 5 and Figure 6 show the comparison. The data is in unit size.

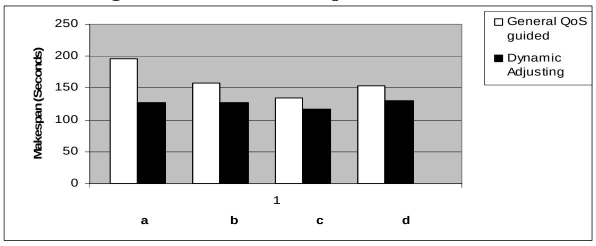

Fig. 6. The comparison based on data size of 20 to 40 unit size

As shown in Figure 6, for all four scenarios the general Min-Min heuristic is outperformed by data-conscious scheduling heuristics. The best performance gain of the data-conscious scheduling heuristics over general Min-Min can be as much as 33.20%. For scenario (a), where the tasks that request the data contribute 50% of the metatask and the data are unevenly distributed, the performance gain reaches as high as 33.20%. For scenario (b), where the tasks that request data are in lower density (35%), a satisfactory performance gain of 18.24% is acquired. For scenario (c), where only 35% of the tasks request data and the data distribution becomes even, the performance gain of the data-conscious scheduling heuristics is relatively small, i.e., 12.96% better than the general Min-Min. Finally, in scenario (d), the data request rate of tasks becomes 50% with all data evenly distributed on the hosts, and the performance gain is 13.95%.

Table 5. The experiment result of the comparison

Scenarios	**Gain (0-5)**	**Gain (20-40)**	**Date Request Task %**	**Data Distribution on the 4 Hosts**
a	38.57%	33.20%	50%	30, 10, 10, 10
b	25.46%	18.24%	35%	30, 10, 10, 10
c	13.21%	12.96%	35%	15, 15, 15, 15
d	19.70%	13.95%	50%	15, 15, 15, 15

Table 5 shows the performance gain of the comparison. The green parts in the table show the performance gains of the four scenarios, according to different data sizes. The first column is measured based on the relatively smaller data sizes that are randomly between 0 to 5 unit sizes. The second column is measured based on the relatively larger data sizes that are between 20 and 40 unit size.

From the table, we can conclude that our dynamic adjusting heuristics outperforms the general Min-Min. Especially, when the data is not evenly distributed, our dynamic adjusting strategy greatly reduces the makespan of a metatask. This is because our dynamic adjusting strategy can dynamically replicate the data among hosts. Hence, it is able to alleviate the load balance caused by the improper distribution data replica. Further, the metatask with a larger portion of tasks with data requests will benefit more from our dynamic adjusting strategy. The reason is that the denser the tasks with data request (constraints), the more likely hosts are to be imbalanced with load. Our strategy performs real-time adjusting on the data replica distribution. Therefore, it results in better makespan on metatask with larger portion of tasks with data request.

6 Conclusion

In this study, we have investigated the scheduling of data-intensive tasks in Grid environments. We have first formulized the Grid resources, including static resources, as well as the movable resources. Then, we have incorporated data movement into Grid task scheduling. The scheduling holes caused by the data access delay have been reduced through integrating data movement into the traditional Min-Min scheduling heuristics. By replicating the data, a tradeoff has been made between the communication cost and load balance of the hosts. The lack of administration-level information in a Grid computing environment has been redeemed by the newly introduced dynamic data distance table (DDT) data structure, which provides the sufficient real-time information for the data replication decision making. A dynamic adjusting scheduling heuristics has been proposed.

To verify our scheduling heuristics, a simulation architecture of Grid environment has been introduced. The simulation is composed of two parts: the simulation agent generates the task and host information, whereas the allocation agent executes the scheduling algorithms. We have carried out a series of experiments based on the simulation environment. Experimental results show that the new data-conscious dynamic adjusting scheduling heuristics outperforms the general Min-Min significantly for data intensive application, especially when the critical data are unevenly distributed.

Acknowledgments
This research was supported in part by national science foundation under NSF grant EIA-0130673, ANI-0123930, and by Army Research Office under ARO grant DAAD19-01-1-0432.

References

1. Buyya, R., Murshed, M., Abramson, D.: A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids. 2002 Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, Nevada, USA, (2002)
2. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for Scheduling Parameter Sweep applications in Grid environments. Proceedings of the 9th Heterogeneous Computing workshop (HCW'2000) 349-363.
3. Raman, R., Livny, M., Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing. Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL
4. T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao: A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. IEEE Workshop on Advances in Parallel and Distributed Systems (1998) 330-335.
5. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, ``Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," 8th IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico (1999) 30-44
6. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, Prentice Hall, Englewood Cliffs, NJ (1995)
7. Henri Casanova, Graziano Obertelli, Francine Berman and Rich Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", Proceedings of the Super Computing Conference (SC'2000), (2000)
8. Kavitha Ranganathan and Ian Foster. “Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications”. International Symposium for High Performance Distributed Computing (HPDC-11), Edinburgh, July 2002.
9. Xiaoshan He, Xian-He Sun, and Gregor von Laszewski, "QoS Guided Min-Min Heuristic for Grid Task Scheduling", Journal of Computer Science and Technology, Special Issue on Grid Computing, 18(4), 2003.