Data Sharing Pattern Aware Scheduling on Grids

Young Choon Lee and Albert Y. Zomaya
Advanced Networks Research Group, School of Information Technologies,
The University of Sydney,
NSW 2006, Australia
{yclee,zomaya}@it.usyd.edu.au

Abstract

These days an increasing number of applications, especially in science and engineering, are dealing with a massive amount of data; hence they are data-intensive. Bioinformatics, data-mining and image processing are some typical areas of data-intensive applications. Such applications tend to be deployed on grids that provide powerful processing capabilities at reasonable cost. One fundamental scheduling issue, that arises when exploiting grids with these types of applications, is the minimization of data transfer. Therefore, the use of an efficient scheduling scheme that takes into account data transfers is rather essential in order to achieve both a shorter application completion time and efficient system utilization. In this paper, a novel scheduling algorithm, called the Shared Input data based Listing (SIL) algorithm for data-intensive bag-of-tasks (DBoT) applications in grid environments is proposed. The algorithm uses a set of task lists that are constructed taking the data sharing pattern into account and that are reorganized dynamically, based on performance of resources, during the execution of the application. The primary goal of this dynamic listing is to minimize data transfer, thus leading to shortening the overall completion time of DBoT applications. SIL further attempts to reduce serious schedule increases by adopting task duplication. In our evaluation study extensive simulation tests with three different types of the DBoT application model have been conducted. Based on the experimental results, SIL noticeably outperforms two previously proposed algorithms in schedule length.

1. Introduction

As the grid has emerged as a promising platform to tackle large-scale problems, an increasing number of applications in various areas, including bioinformatics, high energy physics, image processing and data mining, have been developed and ported for grid environments. In general, these applications are designed with parallel and/or distributed processing in mind. Two typical application models found among them are bag-of-tasks and workflow. A bag-of-tasks application consists of independent tasks and thus no specific order of task execution, whereas an application in the workflow model is composed of interdependent tasks. Bag-of-tasks applications can be further classified into compute-intensive and data-intensive. In the case of running applications in the former category, the performance of computing resources is the most influential factor. However, the management of data transfers plays a crucial role with applications in the latter category.

In this paper, data-intensive bag-of-tasks applications are of particular interest. The DBoT application model can be found in many scientific, engineering and enterprise applications, such as BLAST [1], MCell [2], INS2D [3] and data mining applications. Since tasks in a DBoT application are able to run independently and simultaneously, distributed computing systems, such as grids are indeed suitable for DBoT applications [4]. Although the tasks do not have any dependencies, they may share input data. This particular characteristic of DBoT applications (i.e. data sharing) raises one fundamental scheduling issue, namely the minimization of data transfer. Therefore, it is essential to use an efficient scheduling scheme that takes into account data transfers in order to achieve both shorter application completion time and efficient system utilization. However, designing such grid scheduling schemes involves a number of challenging issues mainly due to the dynamic nature of the grid. These issues include searching for resources in collections of geographically distributed heterogeneous computing systems and making scheduling decisions taking into consideration quality of service.

In recent years a number of grid scheduling algorithms for various application models including the DBoT application model have been proposed [5] [6] [7] [8] [9]. Despite efforts that these existing scheduling algorithms have been designed to provide good performance, they have difficulty guaranteeing the quality of schedules they produce. It can be said that performance prediction information on resources obtained using NWS [10] can be incorporated with scheduling algorithms as in XSufferage [5] to ensure the quality of scheduling. However, it is impractical to assume that perfect performance information on underlying resources in a grid is readily able to be obtained.

In this paper, a novel scheduling algorithm, called the Shared Input data based Listing (SIL) algorithm for DBoT applications on grids is proposed. The algorithm uses a set of task lists that are constructed taking the data sharing pattern into account and that are reorganized dynamically, based on performance of resources, during the execution of the application. The primary goal of this dynamic listing is to minimize data transfer thus leading to shortening the overall completion time of DBoT applications. SIL further attempts to reduce serious schedule increases, that occur because of inefficient task/host assignments, by adopting task duplication.

The evaluation study in this paper is conducted with three different types of DBoT applications in various grid environments simulated using a grid simulator built with SimGrid [11], [12]. The characteristics of resources in the simulated grid environments are random and uniformly distributed among a predefined set of resource properties, such as processing speed, latency and bandwidth. In addition, the workload on each grid resource is simulated by workload traces obtained from actual systems deployed as the GrADS testbed at University of California, Santa Barbara [13].

The remainder of this paper is organized as follows. Section 2 introduces the scheduling model used for the algorithm. The proposed algorithm along with other algorithms used for comparative purposes is described in detail in Section 3. In Section 4, the evaluation results are presented and explained with conclusions following in Section 5.

2. Models

2.1. Grid Model

The grid G in our study consists of a number of sites in each of which a set of m computational hosts is participating in a grid. More formally,

$$G = \{S_1, S_2, ..., S_r\}, \text{ and } S_i,\ 1 \le i \le r, = \{H_{i,1}, H_{i,2}, ..., H_{i,m}\} \cup D_i$$

where S_i is the ith site participating in G, and H_i and D_i are a set of host machines and data repository/storage at S_i, respectively. Let $H = \{H_1, H_2, ..., H_r\}$ denote a set of all hosts in G.

Each site is an autonomous administrative domain that has its own local users who use the resources in it. These sites are connected with each other through WAN. Hosts are composed of both space-shared and time-shared machines with various processing speeds, i.e., CPU speed. These resources are not entirely dedicated to the grid. In other words, they are used for both local and grid jobs. Each of these hosts has one or more processors, memory, disk, etc. We assume that hosts in the same site are able to access each other's data repository as if they are accessing their own, i.e., a set of data repositories in a site can be represented as a single data repository. This assumption is made because a site connects its hosts through a high bandwidth LAN, in general.

The availability and capability of resources, e.g., hosts and network links, fluctuates over time. Therefore, the accurate completion time of a task on a particular host is difficult, if not impossible, to determine a priori. Moreover, the task may fail to complete due to a failure of the resource on which it is running. However, resource failures are not considered in the study.

2.2. Application Model

Bag-of-tasks applications are typically embarrassingly parallel type of applications that exist in many scientific and engineering fields. An application J of this model consists of a number of n heterogeneous independent tasks $\{T_1, T_2, ..., T_n\}$ without inter-task communications or dependencies and thus it is suitable for grids. A task T_i in J is associated with a set I_i of input data objects $\{I_{i,1}, I_{i,2}, ..., I_{i,d}\}$.

In our model, tasks are data-intensive; that is, the input data transfer for each task is a more influential factor than its computation for task execution. A bunch of tasks in an application may share one or more input data objects. This data sharing pattern varies between applications. Three typical data sharing patterns found in DBoT applications are shown in Figure 1.

It is assumed that all input data are initially stored on the host on which scheduling takes place. Therefore, input data required by a task need to be transferred from the scheduling host to the site on which the task is scheduled if not existing on the

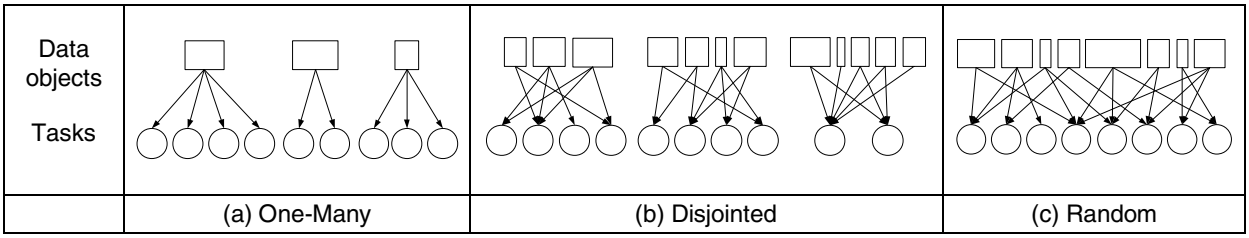


Figure 1. Data Sharing Patterns of DBoT applications

scheduled site. We also assume that there are no inter-site data exchanges.

The amount of output data produced by DBoT applications considered in this study is assumed to be much smaller and negligible compared to input data.

Hereafter, application and job are used interchangeably.

2.3. Grid Scheduling Problem

The grid scheduling problem addressed in this study is task scheduling of a set J of n independent tasks, comprising a bag-of-tasks application, onto $|H|$ heterogeneous hosts dispersed across multiple sites in a grid. The primary goal of this scheduling is to make as many appropriate task-host matches as possible, so that the *makespan*, also called schedule length, of a bag-of-tasks application can be minimized. The makespan in this study is defined as the amount of time taken from the time the first input data transfer starts to the time the last task completes its execution.

3. Scheduling Algorithms

In this section, some existing scheduling algorithms are described first and then the proposed algorithm is presented.

3.1. Storage Affinity (SA)

The Storage Affinity algorithm primarily aims at the minimization of data transfer by making scheduling decisions incorporating the location of data previously transferred [6]. In addition, it considers task replication as soon as a host becomes available between the time the last unscheduled task gets assigned and the time the last running task completes its execution.

SA determines task/host assignments based on 'the storage affinity metric.' Storage affinity of a task to a host is the amount of the task's input data already stored in the site to which the host belongs. Although the scheduling decision SA makes is between task and host, storage affinity is calculated between task and site. This is because in the grid model used for SA each site in the grid uses a single data repository that can be fairly accessible by the hosts in the site.

For each scheduling decision SA calculates storage affinity values of all unscheduled tasks and dispatches the task with the largest storage affinity value. If none of the tasks has a positive storage affinity value one of them is scheduled at random. By the time this initial scheduling gets completed there would be as many as $|H|$ running tasks leaving all $|H|$ hosts busy. On the completion of any of these running tasks SA starts task replication. Now each of the remaining running tasks is considered for replication and the best one is selected. The selection decision is based on the storage affinity value and the number of replicas.

3.2. List scheduling with Round-robin order Replication (RR)

RR [14] is a grid scheduling algorithm for independent coarse-grained tasks. As the name implies its distinctiveness comes from the round-robin order replication scheme that makes replicas of running tasks in a round-robin fashion after conducting list scheduling for all the unscheduled tasks. RR first randomly assigns a task to each host in the grid and then waits until one or more of those assigned hosts complete their tasks. On the completion of a task the next unscheduled task is dispatched to the host that on which the completed task has run. This tends to result in fast resources get more tasks. Once all the tasks are dispatched RR starts replicating running tasks hoping that some or all of these replicas finish earlier than their originals. Note that, RR performs scheduling without any dynamic information on resources and tasks. Nevertheless, the algorithm is compelling and comparable to other scheduling heuristics that require such performance information.

3.3. The SIL Algorithm

Two very influential factors that should be taken into account when scheduling DBoT applications on grids are data transfer and the dynamicity of grid resources. In this section we present SIL that incorporates these issues into its scheduling.

The heuristic judiciously groups tasks into a number of dynamic lists based on their data sharing patterns. Each of these lists is intended to be scheduled into the same site in the grid in order to minimize data transfer which is critical to shorten the overall completion time of DBoT applications in particular. Since the performance of grid resources fluctuates over time the lists are reorganized dynamically during application runtime.

Note that SIL does not use any prediction information on the performance of resources and applications, except the information on input data, i.e., size and location, which is easily obtainable.

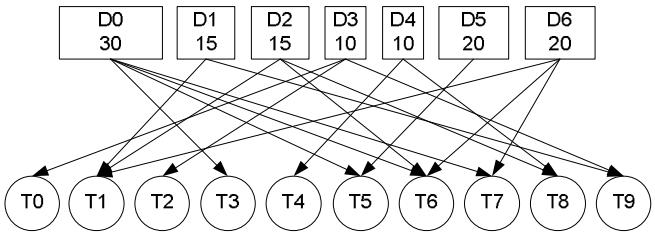
SIL consists of two major phases:

Function GroupTasks
/** **Input**: A set J of tasks, a set H of hosts, a set G of sites
Output: A set L of task lists **/
1. Let $L = \emptyset$
2. **while** J is not empty **do**
3. Remove the first task and tasks in J, such that all pairs of the tasks have data sharing
4. Create a task list, L_l and insert it to L
5. Associated L_l with S_i, if $i \leq |G|$, otherwise L_l is called unassociated
6. Insert the removed tasks to L_l
7. Create a task list, *tempL*
8. Remove the first task in L_l and insert it to *tempL*
9. **while** L_l is not empty **do**
10. Remove task, T_j in L_l that:
    - shows data sharing with some tasks in *tempL*
    - the amount of its data transfer is the smallest among tasks in L_i if it is scheduled after tasks in *tempL* and they are all scheduled into the same site
11. Insert T_j to *tempL*
12. **end while**
13. Let L_l = *tempL*
14. **end while**
15. **for** each site, S_i in G
16. Associate L_l which is associated with S_i with a host, $H_{i,1}$ in S_i
17. **for** each host, $H_{i,k}$ in S_i, except $H_{i,1}$
18. Find the longest task list, *longestL*, $|$ *longestL* $| > 1$ in L in order of:
    lists associated with S_i, unassociated lists and lists associated with other sites.
19. **if** $|$ *longestL* $|$ is greater than 1 **then**
20. Break *longestL* into two lists as 1st PT in *longestL* a delimiter /* don't break if *longestL* is unassociated */
21. Associate the latter list with $H_{i,k}$
22. **else**
23. Break
24. **end if**
25. **end for**
26. **end for**
27. **return** L

Figure 2. The task grouping algorithm

As an attempt to efficiently deal with the dynamicity of grid resources SIL adopts task duplication that is particularly helpful to avoid serious schedule increases. For example, a couple of tasks may be running unexpectedly long increasing the overall schedule significantly due to the overload or abnormal behaviors of the resources on which they are running or being transferred.

- Task Grouping Phase (Figure 2) – groups tasks into a set of lists based on their data sharing pattern, associates these task lists with sites, and further breaks and/or associates them with hosts.



(a) A sample task list

Initial List				Rearranged List				
Task	Data object	TA	ATA	Task	Data object	TA	ATA	
T0	D3	10	10	T0	D3	10	10	
T1	D1, D2, D6	50	50	T2	D3	10	0	
T2	D3	10	0	T9	D1, D3	25	15	
T3	D0	30	30	T1	D1, D2, D6	50	35	
T4	D4	10	10	T8	D2, D4	35	10	
T5	D0, D5	50	20	T4	D4	10	0	
T6	D0, D2, D6	65	0	T6	D0, D2, D6	65	30	
T7	D0, D6	50	0	T3	D0	30	0	
T8	D2, D5	35	0	T7	D0, D6	50	0	
T9	D1, D3	25	0	T5	D0, D5	50	20	

(b) Task lists before and after task rearrangement

Figure 3. An example of task rearrangement

- Scheduling Phase (Figure 4) – assigns tasks to hosts dynamically reorganizing task lists and duplicates tasks when all tasks are scheduled and some tasks are still running.

3.3.1. Task Grouping Phase

The major performance gain of SIL is achieved through its distinctive task grouping scheme that tends to make a substantial reduction of data transfer. A DBoT application may show a certain data sharing pattern as shown Figure 1. It is obvious that assigning tasks sharing input data to the same site reduces data transfer. This is a primary motivation for task grouping. In the task grouping phase the tasks are first grouped into a number of task lists and then tasks in each of these lists are rearranged as shown in steps from 9 to 12. Each of the first $\lfloor G \rfloor$ task lists created at this stage is associated with a site. This task rearrangement further clusters tasks in each list according to actual transfer amount (ATA) of task. The actual transfer amount of a task is defined to be

$$ATA(T_j, S_i) = TA(T_j) - \sum_{d \in (I_j \cap DAT(D_i))} |d|$$

where $TA(T_j)$ is the original transfer amount of T_j, $DAT(D_i)$ is the data already transferred to D_i and $|d|$ is the amount of data object d.

An example of a task list after task rearrangement is show in Figure 3. As one can easily see the ATAs of the rearranged tasks exhibit a certain pattern of curve. Here, *T1*, *T6* and *T5* are denoted as peak tasks (PT). Note that, the first task in the rearranged list is not considered for peak task in order to avoid a task list continuously being associated with shifting from one particular site to another particular site (steps from 17 to 25).

The peak tasks come into play when the task lists, constructed and associated with sites in the first *while* loop, are further divided (steps from 15 to 26) in order for each host to get the share of tasks that are targeted to be assigned to it. That is, the peak task is used as a delimiter when breaking a site-wide task list into a number of host-wide lists. A peak task in a task list normally implies data sharing is small between those tasks before the peak task and those tasks after it and itself; hence it is suitable for being used as a delimiter.

3.3.2. Scheduling Phase

The work in the scheduling phase of SIL (Figure 4) is significantly lightened by its intuitive task grouping. SIL simply dispatches a task in each task list to the host the list is associated with, waits until any of the assigned hosts become available and assigns more tasks to available hosts. This process is repeated until one or more of the task lists become empty. Once this happens and there are still some tasks unscheduled, SIL searches among all unscheduled tasks for an unscheduled task whose ATA to the available host is the smallest, but not 0. An unscheduled task with an ATA of 0 might be selected if all unscheduled tasks have their ATAs being 0. Note that SIL does not just take that task, but it also takes all the tasks after that task and appends them to the end of the emptied task list. This is because the amount of data transfer of the following tasks is likely to increase if the selected task's data transfer to the site to which their associated host belongs has not taken place.

Algorithm SIL
/** **Input**: A set *J* of tasks, a set *H* of hosts, a set *G* of sites
Output: A schedule of *J* onto *H* **/
1. Call GroupTasks
2. Assign the first task in each task list in *L* to its associated host
3. Wait until any host, *H** becomes available
4. **while** any unscheduled task or running task exists **do**
5. Get next task, *T** in the task list, *L** associated with *H**
6. **if** *T** is empty **then**
7. Find task list, *L*** in *L* that contains unscheduled task, *T*** that:
*ATA(T**, S*)*, *H**∈ *S** is the smallest, but not 0 unless *ATA* of all unscheduled tasks is 0
8. **if** *L*** is not empty **then**
9. Break *L*** into two lists as *T*** a delimiter
10. Append *L*** to the end of *L**
11. Let *T** = *T***
12. **end if**
13. **end if**
14. **if** *T** is empty **then**
15. Find the running task, *T*** with min. *ATA(T**, S*)*, *H**∈ *S** and the smallest # replicas
16. Let *T** = replica of *T***
17. **end if**
18. Assign *T** to *H**
19. Wait until any host, *H** becomes available /* either task completed or canceled */
20. Kill all the replicas of the task just completed
21. **end while**

Figure 4. The SIL algorithm

There are two main reasons for the adoption of task duplication in the proposed algorithm. First, due to the absence of the use of performance information in SIL some tasks might be assigned to resources that show a drastic performance drop during the time they are in process, i.e., they are running on hosts or their input data are being transferred. In addition, hosts that become available, when there are no more tasks to schedule, could complete some of running tasks if they are duplicated and assigned to these hosts.

As implied above task duplication only starts when all tasks are scheduled, some tasks are still running and one or more hosts are available. The selection for a task to be duplicated is determined based on ATA and the number of replicas.

4. Experiments

The comparative evaluation of the SIL algorithm is presented in this section. Comparisons have been conducted between two previously proposed scheduling algorithms (i.e., SA and RR) introduced in Section 3 and SIL. This selection is made based on their proven performance and the performance information independence; that is, their scheduling decisions, like SIL's, are made without using any performance information on resources and applications. Note that the target application model of RR is a computation intensive bag-of-tasks which is different from that of SIL and SA. Despite this factor it

is selected for the present comparison study. This is mainly to study the impact of neglecting data transfer on scheduling quality.

The performance metric used for the comparison is makespan. Typically, the makespan of a job generated by a scheduling algorithm is used as the main performance measure of the algorithm.

4.1. Simulation Configuration

The grid simulator used for this study is implemented with SimGrid since its rich set of simulation facilities empowers us to easily develop and evaluate scheduling algorithms for heterogeneous distributed computing environments, e.g., computational grids. Another tool used for simulating grids is Tiers [15], a random network topology generator that produces random network models analogous with the structure of the Internet.

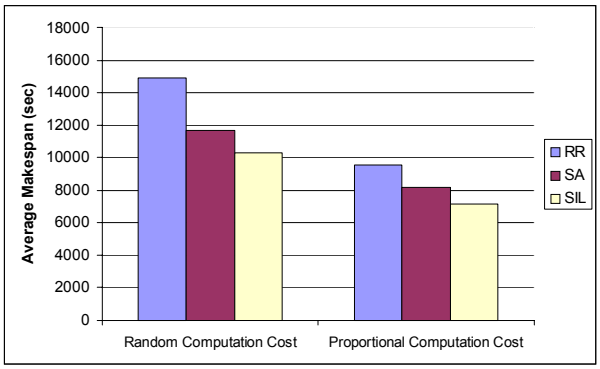
Properties of resources and jobs in the simulations conducted in this study are random and uniformly distributed among a predefined set of resources and job parameters shown in Table 1. Each host and network link is simulated by workload traces obtained from actual systems deployed as the GrADS testbed at University of California, Santa Barbara.

Table 1. A predefined set of resource and job parameters

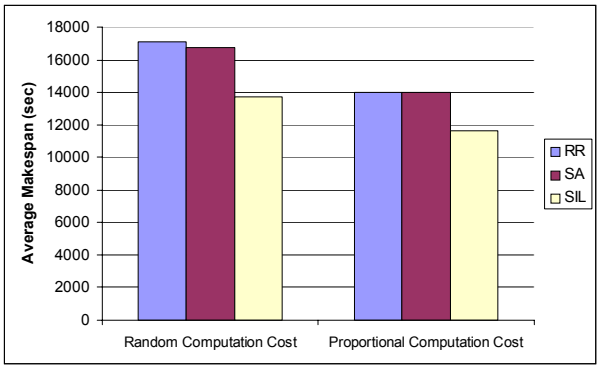
	Property	Value
Computing	The number of sites	3 – 25
	The number of hosts per site	2 – 8
	Relative processing speed	1 – 7.5
Communication	Bandwidth (Mbps)	0.1 – 100
	Latency (ms)	1 – 300
Job	The number of tasks	100 – 1000
	Random computation time of task (sec)	100 – 500
	Proportional computation time of task (sec/MB)	1 – 5
	The size of data per task(MB)	4 – 100

4.2. Experimental Results

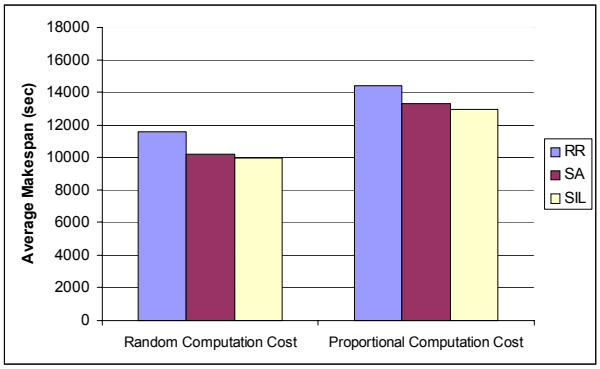
The proposed algorithm and the two previously proposed algorithms, SA and RR are extensively experimented with using a total of 18,000 simulations, i.e., 6,000 simulations for each. More specifically, three groups (i.e. the three data sharing patterns shown in Figure 1) of 20 jobs are first generated. With respect



(a) One-many



(b) Partitioned



(c) Random

Figure 5. Simulation Results for different data sharing patterns of DBoT applications

to each group of 20 jobs, the computation time of each task in one 10 job lot is random and uniformly distributed between 100 and 500 seconds and that in the other 10 job lot is proportional to the size of the input data. Each 10 job lot is run in 10 different simulated grids (i.e. 100 grid-job pairs per 10 jobs),

and these 600 grid-job pairs are run 10 times with different host and network link workload traces.

It can be easily seen from Figure 5 that SIL outperforms SA and RR by a significant margin, except for DBoT applications that show a random data sharing pattern. According to the experimental results SIL can be a natural choice for DBoT applications that show one-many and partitioned data sharing patterns in particular. As mentioned earlier this superior performance of SIL to the other two is achieved through its distinctive task grouping scheme and dynamic task reorganization taking data transfer into account.

The results for DBoT applications with random data sharing, presented in Figure 5 (c) indicate that the average makespan of SIL is only slightly over 2% higher than that of SA. This can be explained as follows. When tasks in a DBoT application are sharing input data randomly it is often the case that all pairs of tasks share some input data; hence a single task list is constructed when SIL is used. In this case some data transfers to different sites containing the same data tend to be frequent.

The average makespan of SIL computed based on the entire simulations is 13% and 26% on average smaller than those of SA and RR, respectively. It is observed that with some data sharing patterns SIL outperforms SA and RR by 22% and 40%. The poor performance of RR is a strong indication that scheduling for DBoT applications in grids should take data transfer into account.

5. Conclusion

In this paper, we have presented a novel scheduling algorithm, called the SIL algorithm, for data intensive bag-of-tasks applications in grid environments. SIL delves into the data sharing pattern of DBoT applications in order to efficiently group and reorganize tasks. In addition, its adoption of task duplication helps lead to better schedules. Based on a number of intensive experiments with various test configurations SIL mostly outperformed the two algorithms by a noticeable margin, especially when scheduling DBoT applications with one-many and partitioned data sharing patterns. The simulation results presented in this paper clearly show this promising performance of SIL.

6. References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Molecular Biology*, 1(215):403–410, 1990.

[2] J. Stiles, T. Bartol, E. Salpeter, and M. Salpeter, "Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes," *Computational Neuroscience*, pp. 279-284, 1998.

[3] S. Rogers and D. Ywak, "Steady and Unsteady Solutions of the Incompressible Navier-Stokes Equations," *AIAA Journal*, 29(4):603-610, Apr. 1991.

[4] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, San Francisco, CA, USA, 1999.

[5] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," In *Heterogeneous Computing Workshop*, Cancun, Mexico, pages 349–363, May 2000.

[6] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting Replication and Data Reuse to Efficiently Schedule Data-Intensive Applications on Grids," In *Workshop on Job Scheduling Strategies for Parallel Processing*, New York, NY, USA, pages 210-232, Jun. 2004.

[7] K. Ranganathan, and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," In *Int'l Symp. High Performance Distributed Computing*, Edinburgh, Scotland, pages 352–358, Jul. 2002.

[8] H. Mohamed and D. Epema. "An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters," In *Int'l Conf. Cluster Computing*, pages 287-298, Sep. 2004.

[9] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms," *IEEE Trans. Parallel and Distributed Systems*, 15(4):319-330, Apr. 2004.

[10] R. Wolski, "Dynamically Forecasting Network Performance Using the Network Weather Service," *Cluster Computing*, 1(1):119-132, Mar. 1998.

[11] H. Casanova, "Simgrid: A Toolkit for the Simulation of Application Scheduling," In *Int'l Symp. Cluster Computing and the Grid*, pages 430-437, May 2001.

[12] A. Legrand, L. Marchal and H. Casanova, "Scheduling Distributed Applications: the SimGrid Simulation Framework," In *Int'l Symp. Cluster Computing and the Grid*, pages 138-145, May, 2003.

[13] http://pompone.cs.ucsb.edu/~rich/data/.

[14] N. Fujimoto and K. Hagihara, "Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid," In *Int'l Conf. Parallel Processing*, pages 391–398, Oct. 2003.

[15] M. Doar, "A Better Model for Generating Test Networks," In *Global Telecommunications Conference*, pages 86-93, Nov. 1996.