A General Distributed Scalable Peer to Peer Scheduler for Mixed Tasks in Grids

Cong Liu, Sanjeev Baskiyar[*], and Shuang Li

Dept. of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849
{liucong, baskisa, lishuan}@auburn.edu

Abstract. We consider non-preemptively scheduling a bag of independent mixed tasks in computational grids. We construct a novel Generalized Distributed Scheduler (*GDS*) for tasks with different priorities and deadlines. Tasks are ranked based upon priority and deadline and scheduled. Tasks are shuffled to earlier points to pack the schedule and create fault tolerance. Dispatching is based upon task-resource matching and accounts for computation as well as communication capacities. Simulation results demonstrate that with respect to the number of high-priority tasks meeting deadlines, *GDS* outperforms prior approaches by over 40% without degrading schedulability of other tasks. Indeed, with respect to the total number of schedulable tasks meeting deadlines, *GDS* outperforms them by 4%. The complexity of *GDS* is $O(n^2m)$ where n is the number of tasks and m the number of machines. *GDS* successfully schedules tasks with hard deadlines in a mix of soft and firm tasks, without a knowledge of a complete state of the grid. This way it helps open the grid and makes it amenable for commercialization.

1 Introduction

A major motivation of grid computing [5] [6] is to aggregate the power of widely distributed resources to provide services. Application scheduling plays a vital role in pro- viding such services. A number of deadline-based scheduling algorithms already exist. However, in this paper we address the problem of scheduling a bag of independent mixed tasks in computational grids. We consider three types of tasks: hard, firm and soft [8]. It is reasonable for a grid scheduler to prioritize such mission critic- al tasks while maximizing the total number of tasks meeting deadlines. *Doing so may make the grid commercially viable as it opens it up for all classes of users.*

To the best of our knowledge, *GDS* is the first attempt at prioritizing tasks according to task types as well as considering deadlines and dispatch times. It also matches tasks to appropriate computational and link bandwidth resources. Additionally, *GDS* consists of a unique shuffle phase that reschedules mission critical tasks as early as possible to provide temporal fault tolerance. Dispatching tasks to peers is based upon both

[*] This research was supported in part by NSF-grant OCI 048136.

computational capacity and link bandwidth. Furthermore, *GDS* is highly scalable as it does not require a full knowledge of the state of all nodes of the grid as many other algorithms do. For *GDS*'s peer to peer dispatch, knowledge of peer site capacities is sufficient. One must consider that obtaining full knowledge of the state of the grid is difficult and/or temporally intensive.

The rest of this paper is organized as follows. A review of recent related works has been given in Section 2. In Section 3, we outline the task taxonomy used in this work. Section 4 describes the grid model. Section 5 presents the detailed design of *GDS*. Section 6 presents a comprehensive set of simulations that evaluate the performance of *GDS*. Conclusions and suggestions for future work appear in Section 7.

2 Related Work

Several effective scheduling algorithms such as *EDF* [9], *Sufferage* [11], and *Min-Min* [12] have been proposed in previous works. The rationale behind *Sufferage* is to allocate a site to a task that would "suffer" most in completion time if the task is not allocated to that site. For each task, *Min-Min* tags the site that offers the earliest completion time. Among all tasks, the one that has the minimal earliest completion time is chosen and allocated to the tagged site.

Few scheduling algorithms take into account both the task types and deadlines in grids. A deadline based scheduling algorithm appears in [16] for multi-client, multi-server environment existing within a single resource site. It aims at minimizing deadline misses by using load correction and fallback mechanisms. In [2], a deadline scheduling algorithm with priority appropriate for multi-client, multi-server environment within a *single* resource site has been proposed. Since preemption is allowed, it leaves open the possibility that tasks with lower priority but early deadlines may miss their deadlines. Also, it does not evaluate the fraction of tasks meeting deadlines.

Venugopal and Buyya [17] propose a scheduling algorithm that tries to minimize the *scheduling budget* for a bag of data-intensive applications on data grid. Casanova [3] describes an adaptive scheduling algorithm for a bag of tasks in Grid environment that takes *data storage* issues into consideration. However, they make scheduling decisions centrally, assuming *full knowledge* of current loads, network conditions and topology of all sites in the grid. Liu and Baskiyar [10] propose a distributed peer to peer grid scheduler that solves the scalability issue in grid systems. Ranganathan and Foster [15] consider dynamic task scheduling along with data staging requirements. Data replication is used to suppress communication and avoid data access hotspots. Park and Kim [14] describe a scheduling model that considers both the amount of computational resources and data availability in a data grid environment. a

The aforementioned algorithms do not consider all of the following criteria: task types, dispatch times, deadlines, scalability and distributed scheduling. Furthermore, they require a full knowledge of the state of the grid which is difficult and/or expensive to maintain.

3 Task Taxonomy

We consider three types of tasks: hard, firm and soft. *GDS* uses such a task taxonomy that considers the consequence of missing deadlines, and the importance of task. Hard tasks are mission critical since the consequences of failure are catastrophic, e.g. computing the orbit of a moving satellite to make real-time defending decisions [13]. For firm tasks a few missed deadlines will not lead to total failure, but missing more may. For soft tasks, failures only result in degraded performance.

An example of mission-critical application is the Distributed Aircraft Maintenance Environment [4], a pilot project which uses a grid to the problems of aircraft engine diagnosis and maintenance. Modern aero-engines must operate in highly demanding environments with extreme reliability. As one would expect, such systems are equipped with extensive sensing and monitoring capabilities for real-time performance analysis. Catastrophic consequences may occur if any operation fails to meet its deadline.

An example of a firm task with deadline is of financial analysis and services [7]. The emergence of a competitive market force involving customer satisfaction, and reduction of risk in financial services requires accuracy and fast execution. Many corresponding solutions in the financial industry are dependent upon providing increased access to massive amounts of data, real-time modeling, and faster execution by using grid job scheduling and data access. Such applications do have deadlines; however, the consequences of missing them are not that catastrophic.

Applications which fall in the category of soft tasks include coarse-grained task-parallel computations arising from parameter sweeps, Monte Carlo simulations, and data parallelism. Such applications generally involve large-scale computation to search, optimize, and statistically characterize products, solutions, and design space but normally do not have hard real-time deadlines.

4 Grid Model

In our grid model, as shown in Fig. 1, geographically distributed sites interconnect through WAN. We define a site as a location that contains many computing resources of different processing capabilities. Heterogeneity and dynamicity cause resources in grids to be distributed hierarchically or in clusters rather than uniformly. At each site, there is a main server and several supplemental servers, which are in charge of collecting information from all machines within that site. If the main server fails, a supplemental server will take over. Intra-site communication cost is usually negligible as compared to inter-site communication.

5 Scheduling Algorithm

The following are the design goals of *GDS*:

- Maximize number of mission-critical tasks meeting their deadlines
- Maximize total number of tasks meeting their deadlines
- Provide temporal fault tolerance to the execution of mission-critical tasks
- Provide Scalability

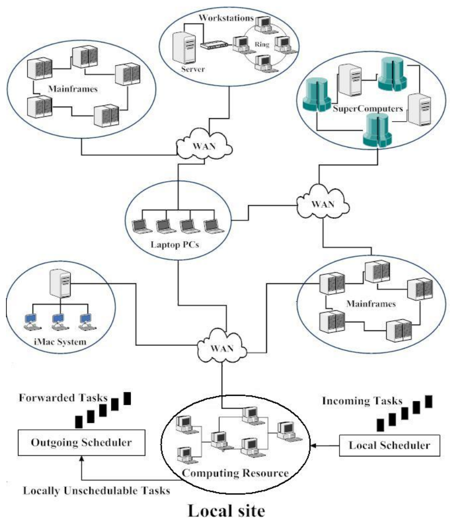


Fig. 1. Grid Model

Since neither EDF nor using priorities alone can achieve the above goals, we proposed *GDS*. *GDS* consists of three phases. First incoming tasks at each site are ranked. Second, a shuffling based scheduling algorithm is used to assign each task to a specific resource on a site, and finally those tasks that are unable to be scheduled are dispatched to remote sites where the same shuffling based algorithm is used to make scheduling decisions. The pseudo code of *GDS*'s main function is shown in Fig. 2.

5.1 Notations

The following notations have been used in this paper.

- t_i: task i
- e_{ijk}: estimated execution time of t_i on $machine_k$ at $site_j$
- c_{ij}: estimated transmission time of t_i from current site to $site_j$
- l_{ijk}: latest start time of tasks t_i on $machine_k$ at $site_j$
- e_i: instruction size of t_i
- d_i: deadline of t_i
- CCR_{ij}: communication to computation ratio of $task_i$ residing at $site_j$
- n_j: number of machines within $site_j$
- cc_{jk}: computing capacity of $machine_k$ at $site_j$
- S_{pkj}: start time of the p^{th} slack on m_k at s_j
- E_{pkj}: end time of the p^{th} slack on m_k at s_j
- CC_j: average computing capacity of $site_j$
- Ave_CC_i: average computing capacity of all the neighboring sites of $site_i$
- Ave_C_{ij}: estimated average transmission time of t_i from $site_j$ to all the neighbors

A task is composed of execution code, input and output data, priority, deadline, and CCR. Tasks are assigned one of the priorities: high, normal, or low, which correspond to mission-critical, firm, and soft tasks. A task's CCR-type is decided by its Communication to Computation Ratio (CCR), which represents the relationship between the transmission time and execution time of a task. It can be defined as:

$$CCR_{ij}=Ave\_C_{ij}\,/\,(e_i\,/\,Ave\_CC_i) \tag{1}$$

If CCR_{ij} >>1, we assign a CCR-type of communication-intensive to task t_i. If CCR_{ij} <<1, we assign a CCR-type of computation-intensive to t_i. If CCR_{ij} is comparable to 1, we assign a CCR-type of neutral to t_i. In estimating CCR, we assume that users can estimate the size of output data. This assumption can be valid under many situations particularly when the size of input output data are related.

Each site contains a number of machines. The average computing capacity of $site_j$ is defined as:

$$CC_j = \sum_{k=1}^{n_j} cc_{jk} \Big/ n_j \tag{2}$$

```
GDS
// $Q$ is a task queue in site $S$
    Sort $Q$ by decreasing priority then by decreasing CCR-type
    then by increasing deadline
        Schedule
    If unscheduled tasks remain in $Q$
        Send message to each $m \in S$ to execute Shuffle
        Schedule
    endif
    If unscheduled tasks remain in $Q$
        Dispatch
    endif
end GDS
```

Fig. 2. GDS

5.2 Multi-attribute Ranking

At each site, various users may submit a number of tasks with different priorities and deadlines. Our ranking strategy takes task priority, deadline and CCR-type into consideration. The scheduler at each site puts all incoming tasks into a task queue. First, tasks are sorted by decreasing priority, then by decreasing CCR-type and then by increasing deadline. Sorting by decreasing priority allows executing mission-critical tasks as soon as possible. Sorting by decreasing CCR-type allows executing most communication-intensive tasks locally. If we were to dispatch such tasks to a remote site, the transfer time may be negative to performance. Experimental results show that sorting by CCR-type gives us good performance.

5.3 Scheduling Tasks Within Slacks

To schedule task t_i on a site s_j, each machine m_k at s_j will check if t_i can be assigned to meet its deadline. If tasks have already been assigned to m_k, slacks of varying length will be available on m_k. If no task has been assigned, slacks do not exist, thus:

$$S_{pkj}=0 \ \&\& \ E_{pkj}=\infty \tag{3}$$

The scheduler checks whether t_i may be inserted into any slack while meeting its deadline. The slack search starts from the last to first. The criteria to find a feasible slack for t_i are:

$$e_{ijk} + max(S_{pkj}, c_{ij}) <= E_{pkj} \ \&\& \ e_{ijk} + max(S_{pkj}, c_{ij}) <= d_i \tag{4}$$

If the above conditions are satisfied, we schedule t_i to the p^{th} slack on m_k at s_j, and set its start time to:

$$l_{ijk} = min(d_i, E_{pkj}) - e_{ijk} \tag{5}$$

Setting tasks start time to their latest start times creates large slacks, enabling other tasks to be scheduled within such slacks. Also, if s_j is the local site for t_i, the transmission time is ignorable; in other words, $c_{ijk} = 0$. The pseudo code of *Schedule* is shown in Fig. 3.

```
Schedule
  for each unscheduled task $t \in Q$
    dofor each machine $m \in S$ //visit in random order to balance load
      Visit slacks from latest to earliest
      If $t$ fits within slack   // while meeting deadline
          Schedule $t$ on $m$ at the latest possible time within the slack
          Mark $t$ scheduled
          Update count of unscheduled tasks in $Q$
      endif
    until $t$ is scheduled
  endfor
end Schedule
```

Fig. 3. Schedule

5.4 Shuffle

If after executing *Schedule*, unscheduled tasks remain, a shuffling procedure is executed on each machine of the site. *Shuffle* tries to move all mission-critical tasks as early as possible. Next, it moves other tasks as close as possible to their earliest start times. In doing so, *Shuffle* creates larger slacks for possible use by unscheduled tasks. The pseudo code of *Shuffle* is shown in Fig. 4. An example of *GDS*'s ranking, scheduling and shuffling phases are given in Fig. 5. The advantages of shuffling are two fold:

- Longer slacks may be obtained by packing tasks.
- Executing mission-critical tasks early provides temporal fault-tolerance.

```
Shuffle
    for each task t // select tasks from highest priority to lowest priority
        Re-Schedule t to the earliest available slack
    endfor
end Shuffle
```

Fig. 4. Shuffle

5.5 Peer to Peer Dispatching

Each task is assigned a ticket, which is a very small file that contains certain attributes of a task. A ticket [1] has several fields: ID, priority, deadline, CCR-type, instruction size, input data size, output data size, schedulable flag and route information. Since tickets are small they are dispatched in scheduling decisions, rather than the tasks themselves. If a task can not be scheduled locally, its ticket is dispatched to a remote site to find a suitable resource.

In dispatching, previous works have selected a remote site randomly or used a single characteristic, such as computing capacity, bandwidth, or load. *GDS* uses both the computing capacity and bandwidth in dispatching. Furthermore, *GDS* helps decrease communication overhead since each site only needs to maintain its immediate neighbors' basic information such as bandwidth and average computing capacity.

Every site always maintains three dispatching lists which are used for the three CCR-type tasks. In each list, immediate neighbors are sorted according to different attributes. The order of neighbors represents the preference of choosing a target neighboring site for dispatch. For computation-intensive tasks, the corresponding list has neighboring sites sorted by decreasing average computing capacity. For communication-intensive tasks, neighboring sites are sorted by decreasing bandwidth. For neutral-CCR tasks, neighboring sites are sorted by decreasing rank. The rank of $site_j$, a neighbor of $site_i$, is defined as:

$$Rank_{ji} = CC_j \Big/ \sum_{k=1}^{r} CC_k + BW_{ij} \Big/ \sum_{k=1}^{r} BW_{ik} \qquad (6)$$

where r is the number of neighbors of $site_i$. The three lists are available at each site and are periodically updated. A site will check whether any of its neighbors can consume a task within deadline or not. Neighbors are checked breadth-first. If none can, the most

Task	Priority	Exec. Time	Deadline
1	Mission-critical	1	3
2	Mission-critical	1.5	7
3	Mission-critical	1	11
4	Firm	2	14
5	Firm	0.5	1
6	Soft	1	4.5
7	Soft	1.5	9

Ranked Tasks at a Resource Site

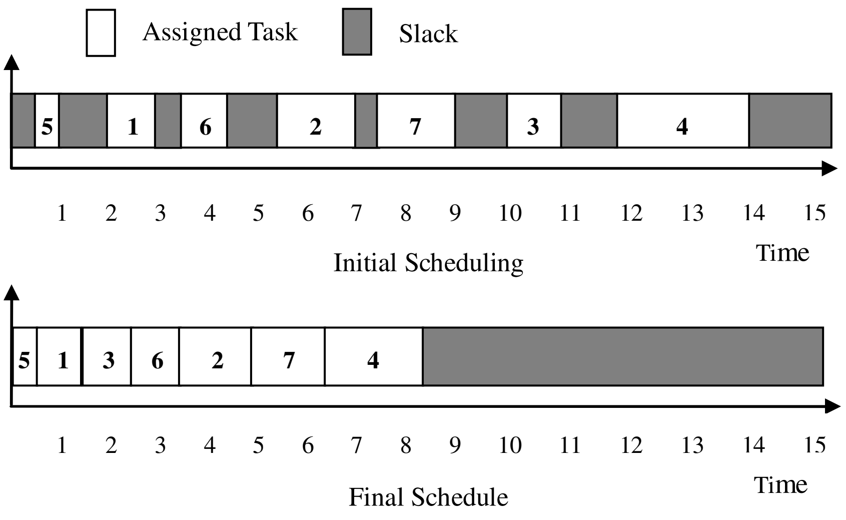


Fig. 5. An example of GDS schedule

favorite neighbor will search its neighbors. This process continues until suitable remote resource has been found, or all sites have been visited. The pseudo code of *Dispatch* is shown in Fig. 6.

5.6 Complexity

Let n be the number of incoming tasks, m the number of machines within each site, and s the number of sites. Then, the complexity of *Shuffle* is $O(n)$, of *Schedule* is $O(n^2m)$ and of *Dispatch* is $O(ns)$. The complexity of *GDS*'s ranking phase is $O(nlogn)$. Therefore, the complexity of *GDS* is $O(n^2m)$, assuming $s < nm$. If in *Schedule*, the slacks within each machine were to be evaluated in parallel by each machine in a non-blocking fashion, the complexity of *GDS* would be $O(n^2)$. We note that the complexity of *Sufferage* and *MinMin is* $O(n^2m)$.

```
Dispatch
   for each unscheduled task $t \in Q$
      for each neighbor $N$ of $S$
      // visit neighbors in order depending upon CCR-type of $t$
         Send $t$'s ticket to $N$
         if $N$ can successfully schedule $t$
            Send $t$ to $N$
            Mark $t$ scheduled
      endfor
   endfor
end Dispatch
```

Fig. 6. Dispatch

6 Simulations

We conducted extensive simulations to evaluate *GDS*. The goal of simulations was two fold: (i) to compare *GDS* against other heuristics, and (ii) to evaluate the merits of each component of *GDS*.

We generated 17 sites with each site having a random number of computers between 20 and 50. The CCR value of each task was varied between 0.05-20. We varied other parameters to understand their impact on different algorithms. The deadlines and number of tasks were chosen such that the grid system is *close to its breaking point where tasks start to miss deadlines*. We varied the instruction size, size of input and output data, bandwidth between sites, and each machine's processing capability. Each data point is an average of 20 runs. The *Critical* Successful Schedulable Ratio (*Critical SSR*) and the *Overall SSR* have been used as the main metrics of evaluation. They are defined as:

$$Critical\ SSR = \frac{number\ of\ mission\ critical\ tasks\ meeting\ deadlines}{total\ number\ of\ mission\ critical\ tasks}$$

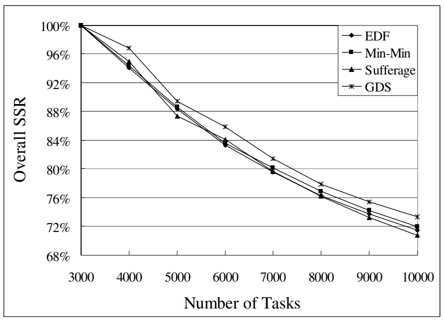
$$OverallSSR = \frac{number\ of\ tasks\ meeting\ deadlines}{total\ number\ of\ tasks}$$

6.1 Performance

The first experiment set was to evaluate the performance against other algorithms. We compared *GDS* against three other heuristics: *EDF*, *Min-Min*, *and Sufferage*.

For *Critical SSR*, from Fig. 7, we observe that *GDS* yield 41% better performance on average than others especially when the number of tasks is high. The other three heuristics do not consider task priority, which results in a number of un-schedulable mission-critical tasks. Also, ranking tasks by CCR-type brings benefits to *GDS* through executing communication-intensive tasks locally and dispatching computation-intensive tasks to other sites.

With respect to *Overall SSR*, as shown in Fig. 8, the performance difference among the five heuristics diminishes. Although *EDF*, *Min-Min* and *Sufferage* do not consider priorities of tasks, overall they are very effective. But, the fact that *GDS* still outperforms them by 4% on average is important. Thus, *GDS* not only maximizes the number of mission-critical tasks meeting deadlines, but it does so without degrading the *Overall SSR*.



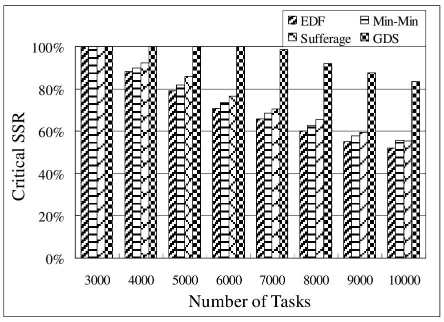
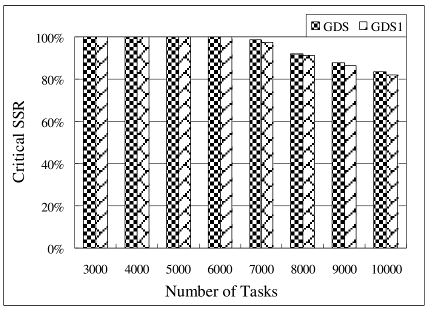


Fig. 7. Critical SSR **Fig. 8.** Overall SSR

6.2 Impact of Shuffling

In this experiment, we investigate the use of the shuffling component of *GDS*. To do so, we use *GDS*$_1$, which is the scheduler obtained upon removing the shuffling portion from *GDS*. From Fig.9, we see that *GDS*'s *Critical SSR* is almost identical to *GDS*$_1$. However, From Fig. 10 we observe that *GDS*'s *Overall SSR* is higher than *GDS*$_1$ by 5%. In other words, *Shuffle* schedules more tasks with firm and soft deadlines while maximizing the number of mission-critical tasks that meet deadlines. It also provides temporal fault tolerance to mission-critical tasks by re-scheduling them earlier.



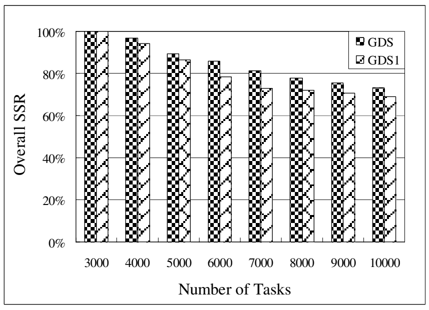


Fig. 9. Critical SSR **Fig. 10.** Overall SSR

7 Conclusion

In this paper, we proposed a novel algorithm to schedule independent tasks with different priorities and deadlines in grid systems. Detailed simulations demonstrate that *GDS* significantly increases both the *Critical SSR* and the *Overall SSR* of all incoming tasks. In the future, we will investigate the schedulability analysis of *GDS* in order to provide deadline guarantees as well as address temporal fault tolerance.

References

[1] Baskiyar, S., Meghanathan, N.: Scheduling and load balancing in mobile computing using tickets. In: Proceedings of the 39th ACM Southeast Conference (2001)

[2] Caron, E., Chouhan, P.K., Desprez, F.: Deadline Scheduling with Priority for Client-Server Systems on the Grid. In: Proceedings of the 5th International Workshop on Grid Computing (2004)

[3] Casanova, H., Obertelli, G., Berman, F., Wolski, R.: The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In: Proceedings of the 13th International Conference for High Performance Computing, Networking, Storage and Analysis (2000)

[4] Distributed Aircraft Maintenance Environment [Online]. Available: [Accessed May 8, 2007], http://www.cs.york.ac.uk/dame

[5] Foster, I., Kesselman, C.: The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers, San Francisco (1998)

[6] Foster, I., Kesselman, C.: The Grid2. Morgan Kauffmann Publishers, San Francisco (2003)

[7] Joseph, J., Fellenstein, C.: Grid Computing. Prentice Hall, Englewood Cliffs (2004)

[8] Laplante, P.A.: Real-Time Systems Design and Analysis, Wiley-IEEE Press (2004)

[9] Liu, C., Layland, J.: Scheduling Algorithms for Multiprogramming in a hard Real-Time Environment. Journal of the ACM (1973)

[10] Liu, C., Baskiyar, S., Wang, C.: A distributed peer to peer grid scheduler. In: Proceedings of the 18th International Conference on Parallel and Distributed Computing and Systems (2006)

[11] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings of the 8th Heterogeneous Computing Workshop (1999)

[12] Menasce, D., Saha, D., Porto, S.: Static and Dynamic Processor Scheduling disciplines in Heterogeneous Parallel Architectures. Journal of Parallel and Distributed Computing (1995)

[13] National Aeronautics and Space Admin. [Online]. Available : [Accessed May 8, 2007], http://liftoff.msfc.nasa.gov/academy/rocket_sci/satellites

[14] Park, S., Kim, J.: Chameleon: A Resource Scheduler in a Data Grid Environment. In: Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid (2003)

[15] Ranganathan, K., Foster, I.: Identifying Dynamic Replication Strategies for a High Performance Data Grid. In: International Workshop on Grid Computing (2001)

[16] Takefusa, A., Casanova, H., Matsuoka, S., Berman, F.: A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In: Proceedings of the 10th IEEE Symposium on High Performance and Distributed Computing (2001)

[17] Venugopal, S., Buyya, R.: A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids. In: Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (2005)