Scheduling Algorithms Performance Evaluation in Grid Environments

Yang Zhang and Ken Kennedy and Charles Koelbel
Computer Science Department
Rice University
Houston, TX 77005
Email: {yzhang8,ken,chk}@rice.edu

***Abstract*— Effective scheduling is critical for the performance of an application launched onto the Grid environment [13], [14]. Deriving efficient algorithms for this scheduling has always been a challenging research area. Many scheduling algorithms have been proposed, studied and compared but there are few studies comparing their performance in Grid environments. The Grid environment has the unique property of drastic cost differences between inter-cluster and the intra-cluster data transfers. In this paper, we compare several scheduling algorithms that represent two major schemes. We further analyze the results to show how different resource environments and workflow application structures affect the algorithms' performances, and suggest directions for future research in Grid schedulers.**

I. Introduction

With the development of large-scale high-speed networks, the Grid, as a computational platform [13], [14] is attracting more and more high-performance parallel and distributed applications [32], [7], [33], [22]. Although grid technologies enable the sharing and utilization of widespread resources, the performance of parallel applications on the Grid is sensitive to the scheduling algorithms they use. Scheduling is the decision process by which application components are allocated to available resources to optimize performance metrics. In this paper, we examine the scheduling of workflow applications to minimize execution time. *Workflow applications* are an important class of grid applications which partition the overall task into multiple (usually coarse-grain) sub-tasks linked to each other by data dependences (usually implemented as file transfers). Each sub-task is considered atomic, consuming its inputs and producing outputs for dependent subtasks. We represent the overall application as a *directed acyclic graph* (DAG), where graph nodes represent sub-tasks and graph edges represent data transfers. A node must be assigned to a particular resource on the Grid, and the communication must occur over the network between the assigned processors. In this paper, we assume that the application is available in the form of a DAG and the scheduler will compute both the allocated resource for each node in the DAG and the order of execution if multiple nodes are assigned to the same resource.

Scheduling parallel and distributed applications in general is a known NP-complete problem [15], and many scheduling heuristics have been proposed in the literature. Among those heuristics, most of the work has attempted to schedule DAGs onto a homogeneous environment [8], [26], [24], [30], [17], [1], [27], [21]. These approaches fall into a few categories such as list-based, clustering and duplication-based. Among them, list-based is generally accepted as the best approach with low complexity and good results [20]. The list-based approach first orders the nodes in the DAG by a pre-calculated priority. Then the scheduler considers the nodes in order, assigning each to a resource that minimizes a suitable cost function. Other heuristics [31], [2], [16], [28], [34], [3], [25] attempt to schedule DAGs onto a heterogeneous environment. In a heterogeneous environment, the execution time of a node will depend on the resource to which it is assigned. Hence, the scheduling process becomes more involved in this environment than in a homogeneous environment. One approach to this complexity is to adjust the priority order to account for heterogeneity; we refer to this as the *list-based approach* for grid scheduling. Iverson [18], Illvarasan [12] and Atakan [10] argue that the pre-computed order for list-based approaches does not apply in heterogeneous environments and propose a new set of heuristics which we refer as the *level-based approach*. The level-based methods first organize the DAG into levels in which all the nodes are independent and then schedule the jobs within each level. A Grid environment usually consists of many clusters with special properties that we will describe more in Section II. This poses even more challenges for scheduling applications because not only are the processors heterogeneous but also the communication variance is larger. To the best of our knowledge, there has no published research that shows how well the list-based or the level-based approaches adapt to the Grid environment.

In this paper, we evaluate the schedules produced by several well-known list-based and level-based scheduling algorithms, using a variety of parameters on both the resource environments and the application DAGs. Via tens of thousands of experiment runs, we explore how the performance of these algorithms varies with differences in the resource environments and the application DAGs.

We also provide an explanation of why some scheduling algorithms perform better in certain settings, and pose future research questions based on these observations.

The rest of the paper is organized as follows. Section II discusses the basics of scheduling a DAG in a Grid environment. Section III briefly introduces all the scheduling algorithms that we evaluate in this paper. Section IV presents our application and Grid parameters and experimental environment. Section V gives our results. Section VI concludes the paper with a summary of contributions and perspectives on future work.

II. Problem Definition

This section presents the model of Grid computing environments and DAG applications used for scheduling. The concept of the Grid [13], [14] is to view the global resources as an active computational environment, connecting geographically distributed computers, databases, instruments and people into a seamless web of advanced capabilities. Miguel et al. [5] points out that a Grid environment usually has the characteristics of *heterogeneity, large scale* and *geographical distribution*. This paints a picture of a typical Grid environment consisting of many clusters, where the intra-cluster communication is fast (often as fast as 10 Gigabit/sec) but the inter-cluster communication can be 10 to 1000 times slower. Thus, the Grid is not just a heterogeneous resource pool, but also an unevenly distributed (but hierarchical) inter-connection network. Furthermore, while the processors in different clusters are often significantly heterogeneous, each cluster consists of many homogeneous processors. As Section V shows, these features have a big impact on how scheduling algorithms originally designed for homogeneous or heterogeneous platforms perform in Grid environments.

The directed acyclic graph (DAG) is an abstract description and is frequently used to represent an application. We define an *abstract* DAG, $G = (V, E)$, where V is the set of nodes each representing an application task and E is the set of edges each representing a data dependence between tasks. Our complexity measures will often use v as the size of set V and e as the size of set E. We will later refer to an abstract DAG as the *DAG model*. We assume that an abstract DAG has a single entry node and a unique exit node since we can always insert pseudo entry or exit nodes into the DAG.

The inputs to a scheduling algorithm are an abstract DAG, a set of resources P and two performance prediction matrices $M_p = V \times P$ and $M_n = P \times P$. Here, $M_p[i][j]$ represents the estimated computation cost of node n_i on processor p_j and $M_n[i][j]$ represents the estimated communication cost of transferring data from processor p_i to processor p_j. The estimated communication cost is defined as

$$T_{communication} = T_{latency} + Datasize/Bandwidth$$

where $T_{latency}$ is the latency(in sec) and the unit of Datasize is Byte and the unit of Bandwidth is Byte/sec. Our complexity measures will often use the term p for the size of P. We will later refer to P as the *resource model*, M_p as the *cost model* and M_n as the *network model*.

The output of a scheduling algorithm is a *concrete* DAG, $G = (V, E, M)$, where V and E are the same as in an abstract DAG and M is a map from V to P where $M[v_i]$ is the resource on which the node will be executed and the time it will start.

Before presenting the objective function, we need to define the EST (Earliest Start Time) and EFT (Earliest Finish Time) attributes as follows:

$$AFT(n_{entry}) = 0$$

$$EST(n_i, p_j) = max\{Evail[j], \max_{n_m \in pred(n_j)}\{AFT(n_m) + M_n[m][j]\}\}$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + M_p[i][j]$$

$$EST(n_i) = min_{p_j \in P}\{EST(n_i, p_j)\}$$

$$EFT(n_i) = min_{p_j \in P}\{EFT(n_i, p_j)\}$$

Here $AFT(n_i)$ is the actual finish time of node n_i as it is scheduled and $Evail[j]$ is the estimated earliest available time of resource P_j.Now we can define the schedule length which is also called the *makespan*. Since we only have one exit node, we define the makespan as:

$$makespan = AFT(n_{exit})$$

In this paper, the objective of the scheduling algorithms is to provide a map for an abstract DAG such that the actual scheduled length (makespan) is minimized.

III. Related Work

There has been considerable work in scheduling DAGs to minimize the schedule length (makespan) on either homogeneous [20] or heterogeneous processors [6]. As we mentioned in section I, our goal is to verify whether the level-based or list-based algorithms work well in the Grid environment and to compare their performances. In our experiment, we chose some well-known and representative algorithms in both categories. This section gives brief overviews of those algorithms.

A. Heterogeneous Earliest Finish Time (HEFT) Scheduling Algorithm

HEFT [16] is a well-established list-based algorithm with one of the better performances on the heterogeneous platforms [3], [16]. Like all the list-based scheduling algorithms, HEFT has two phases, namely, the node prioritizing phase and the processor selection phase.

In the node prioritizing phase, HEFT uses an *upward rank* which is defined as

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}(\overline{c_{i,j}} + rank_u(n_j)),$$

where $succ(n_i)$ is the set of immediate successors of node n_j, $\overline{w_i}$ is the average (estimated) computation cost of node n_i and $\overline{c_{i,j}}$ is the average (estimated) communication cost of edge $E(i,j)$. Averages are computed over the set of all Grid resources P or processor-processor connections $P \times P$, respectively. We assign $rank_u(n_{exit}) = 0$ and traverse the DAG edges backwards to compute the other upward ranks. HEFT then processes the nodes in descending order of their rank.

In the processor selection phase, HEFT assigns each node, in order, to the processor that gives the earliest finish time, i.e. the minimal EFT. HEFT uses an insertion-based policy to find the earliest available time of a processor p_j. Instead of using the time p_j finishes its last assigned node, HEFT tries to find an idle slot of p_j that is later than the available time of the node n_i (the earliest time that all n_i's predecessors finish) and long enough to finish the execution of n_i.

After taking a closer look, we found that the upward rank is equivalent to the definition of *b-level* [20] commonly used in list-based scheduling algorithms. Furthermore, we think it can be considered as an heterogeneously adept version of MCP(Modified Critical Path) [35], a list-based scheduling algorithm for homogeneous environments, in the sense that HEFT uses the same definitions as MCP, but adapted to use average costs in computing $rank_u$ rather than exact (fixed) costs. In addition, we didn't observe many opportunities to insert the execution of an node onto a processor other than at the end in our experiments. Since the idle slot search increases the complexity greatly, we implemented the second phase without it. The computation complexity of this version of HEFT is $O(v^2 + vp)$.

B. Levelized Heuristic Based Scheduling(LHBS) Algorithm

LHBS [23] is a level-based algorithm for Grid scheduling. As all level-based algorithms do, it proceeds by dividing the DAG into levels of independent nodes. Within each level, LHBS can use several heuristics to map the nodes to the processors:

- *Greedy Heuristic*: This heuristic works in a first-come first-serve way. For each node in the same level, assign it to the processor that gives the earliest finish time(minimal EFT).
- *Min-min Heuristic*: For each node in the same level, find the processor that has the minimum EFT and record a tuple (n_i, p_i, t_i), where n_i is the node, p_i is the processor and $t_{=}EFT(n_i, p_i)$. In the second step, Min-min selects the node n_i with the minimum t_i and assigns it to its corresponding p_i. Continue this process until every node is scheduled.
- *Min-max Heuristic*: The first step is exactly the same as the Min-min heuristic. In the node choice phase, however, Min-max heuristic chooses the node with the maximum t_i. The intuition behind it is to assign the longer job first.
- *Sufferage Heuristic*: This heuristic records both the minimum EFT (t_i^+) and the second lowest EFT (t_i^*) for each node. It then selects the node n_i with the biggest difference $t_i^+ - t_i^*$. The intuition behind Sufferage is to select the node that may suffer the most if it is not assigned optimally.

LHBS can choose to run all or some of the above heuristics to map the nodes in each level and select the best schedule length. The complexity of the LHBS using only the Greedy heuristic is $O(vp)$; we will refer this as *Simple LHBS*. The complexity of the LHBS using the other three heuristics is $O(v^2p)$; we will refer this variant as *Sophisticated LHBS*.

C. Hybrid Heuristic(HHS) Scheduling Algorithm

HHS [29] takes a hybrid approach that has three phases. In the first phase, it computes the upward rank for each node and sorts them the same as in HEFT. In the second phase, it groups the nodes into independent sets. This can be done in several ways. One version of HHS creates "levels" based on the rank computed in HEFT. It puts two equally-ranked nodes in the same level if neither is a parent of the other. Any heuristics used in LHBS can be used within the same group.This version is closer to LHBS. We implement it using the Min-min heuristic and It has a complexity of $O(v^2p)$.

Another version is to first computes levels as in LHBS, then processes nodes in each level following the upward rank order. This version is closer to HEFT, and has the same complexity of $O(v^2 + vp)$. We refer to this version as the List Hybrid Heuristic Scheduling (LHHS) Algorithm. In this paper, we will mainly show the results of LHHS because we have only managed to get partial results from the first version. (We expect to have fuller results for the first version in time for the final paper submission).

IV. Experimental Methodology

In order to study how well these scheduling algorithms work in the Grid environment, we implemented the algorithms we described in Section III and compared the schedules produced for a variety of DAGs and Grids. To achieve a thorough comparison, we developed a platform to create test cases. The platform consists of three key components: the DAG generator described in Subsection IV-A, the cost generator described in Subsection IV-B, and a Grid generator described in Subsection IV-C. As Subsection IV-D discusses, our experiments combined these to schedule and evaluate over 10,000 DAG/environment combinations.

A. DAG Model

We use DAGs taken from two real Grid applications and three classes of artificially-generated (but realistic) DAGs. The real applications are EMAN [22] and Montage [33], which are both workflow applications that we have used previously. The generated DAGs abstract certain characteristics of these applications. All are described below.

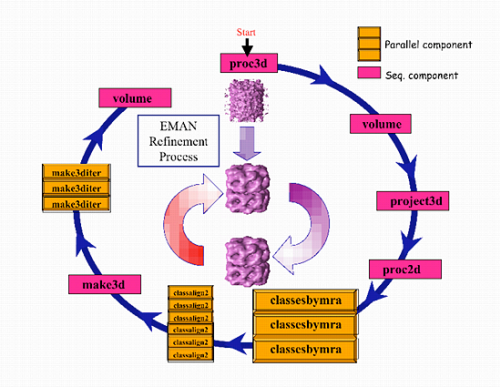


Fig. 1. EMAN Refinement Workflow

1) EMAN: EMAN (Electron Micrograph Analysis) is a bio-imaging application developed at the Baylor College of Medicine. It deals with 3D reconstruction of single particles from electron micrographs. Human expertise is needed to construct a preliminary 3D model from the 'noisy' electron micrographs. The computationally-intensive refinement from a preliminary model to the final 3D model is fully automated that can benefit from Grid computing. Figure 1 shows the structure of the DAG of this refinement process. It is essentially a linear workflow with both sequential and parallel stages. The important and time-consuming steps are the large parameter sweep steps like "classesbymra". Different problem sizes generate different numbers of tasks in the parallel stages. We use this to generate a variety of EMAN cases, with DAG sizes from under 100 to over 3000 nodes.

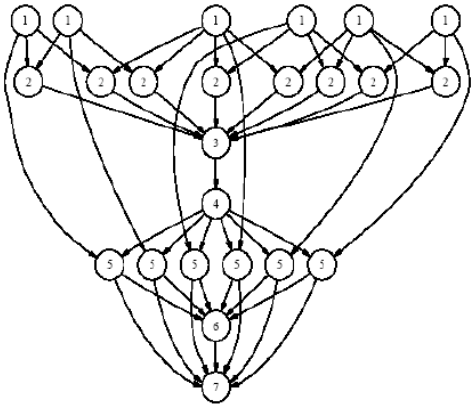


Fig. 2. A Small Montage Workflow

2) Montage: Montage is a data-intensive astronomy application to create custom image mosaics of the sky on demand. It consists of four steps: (i) re-projection of input images to a common spatial scale; (ii) modeling of background radiation in images to achieve common flux scales and background levels; (iii) rectification of images to a common flux scale and background level; and (iv) co-addition of re-projected, background-corrected images into a final mosaic. Figure 2 shows the structure of a small Montage workflow. The workflow consists of some highly parallel sections that can benefit from execution over multiple grid sites. This application is data-intensive, transferring potentially large files on the edges of the workflow. The size of a Montage DAG depends on the size and resolution of the mosaic being created. The Montage graphs in our tests ranged from under 50 nodes to over 4000.

3) DAG generator: Besides the DAGs from real applications, we also implemented a DAG generator that can generate various formats of weighted pseudo-application DAGs. The following input parameters were used to create a DAG.

- Type of DAG: Unlike other DAG generators [3], [16], our DAG generator can generate different formats of DAGs. Currently, we support *fully random, level,* and *choke* formats. In a random DAG, each node can be connected to any node on a higher level (to ensure that the graph is acyclic). In the level DAG, a node can only connect to nodes on the adjacent level. In the choke DAG, one level (the choke point) has only one node; it connects to all the nodes on the levels above and below it. Nodes in other levels are connected as in the random graph.
- Total number of nodes in the DAG, λ.
- Shape parameter, α: The height of the DAG is randomly generated from a uniform distribution with mean value equal to $sqrt(\lambda)/\alpha$. The width of the DAG at each level is randomly selected from a uniform distribution with mean equal to $\alpha \times sqrt(\lambda)$. In order to prevent the creation of extremely narrow or flat DAGs, we confined the random number generator to only generate numbers between 0.35 and 0.65.
- Out degree of a node, η : Each node's out degree is randomly generated from a uniform distribution with mean value equal to η.

B. Cost Model

Given a DAG, whether from a real application or automatically generated, we generate base costs for the nodes and edges using three parameters.

- The lower and upper bound of the *data size*, ϵ, ϕ: The data size attached to each edge in a generated DAG is randomly generated from a uniform distribution between the lower and upper bound. In level graphs, all edges between two adjacent levels have identical data size; in random and choke graphs, we generate costs for every edge independently. For EMAN and Montage DAGs, we use actual data sizes from the application.
- *Communication-Computation Ratio*: Following Blythe et al. [4], we define the CCR of a DAG as

$$CCR = \frac{total\ communication\ cost}{number\ of\ nodes \times AvgCompCost}$$

We can set this ratio as a parameter and combine it with the total data size and average bandwidth in the resource pool to compute the average computation cost for a node:

$$AvgCompCost = \frac{total\ file\ size/avg\ bandwidth}{number\ of\ nodes \times CCR}$$

- *Range*: The node computation costs for generated DAGs are independently randomly generated from a uniform distribution from $AvgCompCost \times (1 - range)$ to $AvgCompCost \times (1 + range)$. For EMAN and Montage DAGs, we use uniform costs for each level, reflecting the behavior of the actual applications.

This gives us a base cost for every node, which will be modified by the Grid model.

C. Grid Model

Our resource model is based on a tool that generates populations of representative compute clusters [19]. This tool uses empirical statistical models of cluster characteristics (e.g., number of processors, processor clock rate) obtained from a survey of 114 real-world clusters. Using this tool we generated a resource pool that contains over 18,000 processors grouped in 500 clusters, which we refer as *the universal environment*. We also semi-manually generated two smaller resource pools. They both have roughly 300 processors, but one groups them into 20 clusters while the other only has only 4 clusters. We will later refer the resource pool with 20 clusters as *the many-cluster environment* and the other as *the big-cluster environment*.

Given the resource model, we computed the computational cost matrix $M_p[i][j]$ by scaling the base cost for DAG node i by the clock rate of processor j.

Our network model is based on a tool that generates end-to-end latencies matrixes according to the real latency data collected over the Internet [11]. Following the experiment result of Yang et al. [36] and Denis et al. [9] we assigned the bandwidth based on the latency. Low-latency links had high bandwidth, consistent with the data in Bo et al. [11]

Given the latency and bandwidth of each network link, it was a simple matter to compute the communication cost matrix M_n.

D. Experimental Setup

We used our DAG generator to produce DAGs with the following parameters:

- Type = {random, level, choke}
- λ = {300, 1000, 3000}
- α = {0.5, 1.0, 5.0}
- η ={1.0, 2.0, 5.0}

We generated 5 random DAGs for each possible parameter combination. In addition, we used 30 EMAN DAGs and 30 Montage DAGs. For each of those DAGs, we applied our cost model with the following parameters:

- $\{\epsilon, \phi\}$ = { {20,1000},{100,1000},{500,1000} }
- CCR = {0.1, 1.0, 10}
- Range ={0.15, 0.4, 0.85}

With three Grids and four scheduling algorithms, we collected about 120,000 schedules and their associated makespans.

The makespans usually vary widely among DAGs, making it difficult to take meaningful averages or make cross-DAG comparisons. Following the methodology of other scheduling work [20], [16], [3], we use *Schedule Length Ratio* (SLR) as the main metric for the comparisons so that the results will not be sensitive to the size of the DAG. Conceptually, the SLR is a normalization of the makespan to an estimate of the best possible schedule length of a given DAG in a given environment. In a perfect world, we would use an optimal schedule for this estimate; however, since finding the optimal makespan is NP-complete, we instead estimate critical path length. The *critical path* through a DAG is the most costly path from the entry node to the exit node, while the *critical path length* is the total cost of nodes and edges along this path. Because the costs of nodes depend on where they are mapped, we approximate the computation cost of a DAG node by its average cost over all possible processors. Similarly, we approximate the communication cost of a DAG edge by its average over all possible processor pairs. We compute the *Critical Path Including Communication (CPIC)* as the cost of the critical path using these estimates, and define

$$SLR = makespan/CPIC$$

Intuitively, if a schedule has an SLR of 1 then it matches the "expected" best time.

We note a few characteristics of the above definition of SLR:

- Small SLRs are better than large SLRs.
- An SLR below 1 can occur when some DAG nodes are mapped to faster-than-average processors. Similarly, mapping a DAG edge to a low-latency or high-bandwidth network link can reduce the SLR below 1.
- An SLR above 1 can occur when some DAG node is mapped to a slower-than-average processor, or when several independent DAG nodes are serialized on a single processor. Similarly, mapping a DAG edge to a low-bandwidth network link can increase the SLR above one.
- Our definition of SLR differs slightly from the usual definition of SLR that uses CPES (critical path excluding communication). We prefer our definition because it includes an approximation of communication cost, thus providing a more realistic standard of comparison.

V. Results

Figure 3 gives a sense of our experience. It is a scatterplot of 3825 DAGs, each scheduled with 4 methods, on the

universal resource set, indexed by the actual DAG width. (That is, the figure shows about 1/5 of our experiments). We index the DAGs by the width rather than size because it tends to spread the cases apart more. Figure 4 is simply a magnification of the lower left corner. Obviously, there is a wide variation in the quality of generated schedules. Moreover, the algorithm that generates the best schedule for any particular DAG varies without a clear pattern.

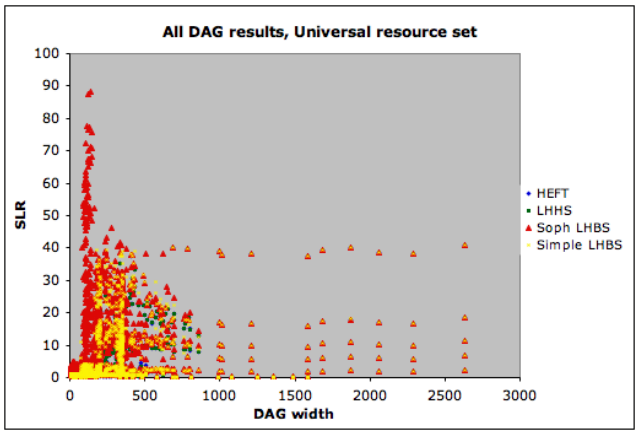


Fig. 3. Scatter-plot of all DAGs scheduled onto universal resource set

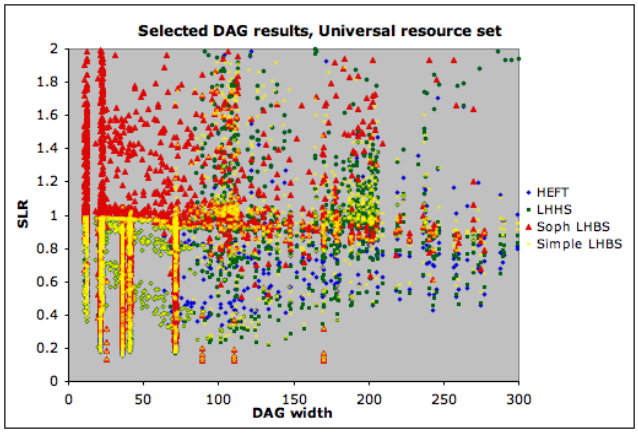
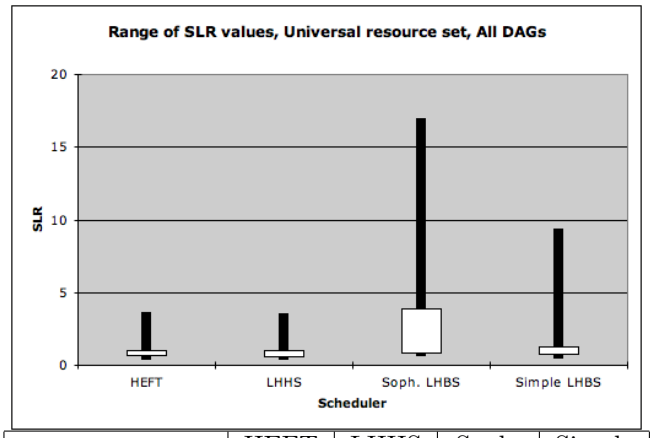


Fig. 4. Scatter-plot of selected DAGs scheduled onto universal resource set

In aggregate, however, a somewhat clearer picture emerges. Figure 5 shows the range of quality for each scheduling method on all DAGs for the universal resource set. The top and bottom of the white boxes are the 75th and 25th percentile SLRs for each scheduler, while the top and bottom of the black lines are the 90th and 10th percentile. It is clear that all the methods have many high-SLR outliers, but that the bulk of the results from the HEFT, LHHS, and simple LHBS methods are comparable. The included table shows the average results for each method. Despite the high variance of data, the differences between the means are statistically significant at levels far less than $p = 0.001$ (according to paired t-tests). Even the 1% difference between HEFT and LHHS has a statistical significance of $p = 6 \times 10^{-6}$, although that difference may not be noticeable in practice. The last two lines of the table show how often each method returned the best and worst result for the same DAG among the four algorithms we tested. The percentages do not add up to 100% due to ties; HEFT and LHHS often computed equivalent schedules, particularly for choke DAGs. This would lead us to believe that HEFT or LHHS produce better schedules than level-based methods on average.



	HEFT	LHHS	Soph. LHBS	Simple LHBS
Mean SLR	1.99	1.97	5.58	2.90
Std. Dev. SLR	4.12	4.12	10.35	6.05
Median SLR	0.90	0.89	1.21	0.92
% of best cases	56%	60%	1%	29%
% of worst cases	0.5%	0.3%	80%	19%

Fig. 5. Aggregate behavior of scheduling methods

The difference in behavior was not, however, consistent across types of DAGs, as shown by Figure 6. In particular, all of the methods produced good schedules for EMAN. Most of the differences are statistically significant (the exceptions are HEFT and LHHS results for level and EMAN DAGs), but many are too small to be important in practice. Nor was the difference between methods true of all resource sets, as Figure 7 shows for random DAGs. We can clearly see that the LHBS algorithms perform much worse in the larger resource pool. As for the universal resource set, the differences are statistically significant (except for the two LHBS algorithms in the big-cluster resource set), but many are likely smaller than the uncertainties in our simulation.

After examining some of the schedules, we hypothesized that most of the differences were due to LHBS methods emphasizing parallelism over communication costs. One scenario is that LHBS might assign some DAG nodes to clusters that have a earlier start time in order to to achieve a shorter makespan in one level. If these nodes required

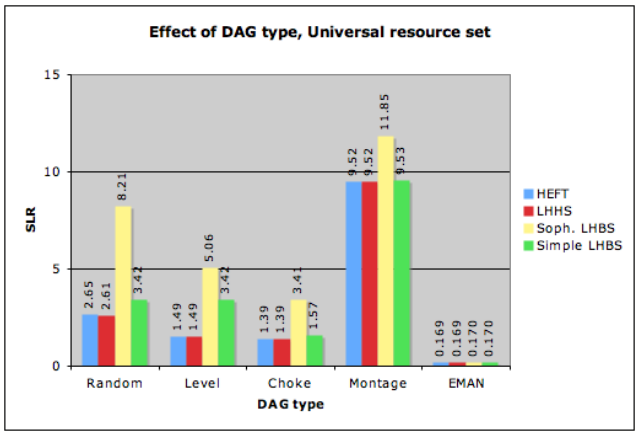


Fig. 6. Results for different DAG types

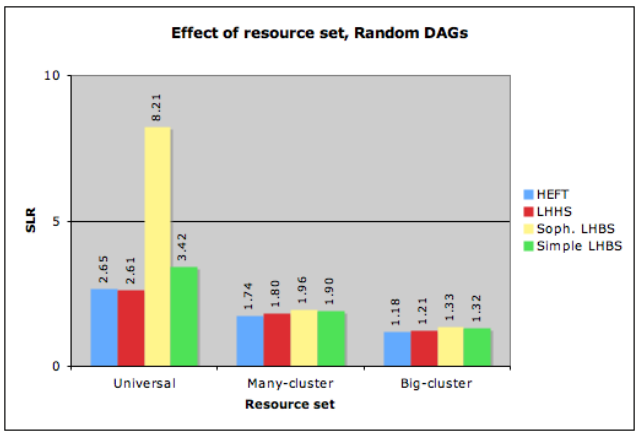


Fig. 7. Algorithms Performance on Different Resource Models

input from two or more clusters, the estimated communication costs might be equivalent for that level. At the next level, however, having the nodes on different clusters might require additional inter-cluster communications. This scenario would obviously have more impact when a DAG required more point-to-point communication. (All-to-all communication, as in EMAN, does not necessarily suffer, because the inter-cluster communication is almost always required.) This may have a smaller impact on HEFT and, to a lesser extent, LHHS. It is because nodes with high future communications requirements are scheduled earlier, when the resources nearby (i.e. processors within the same cluster) may have not yet been allocated.

To test this, we examined the sensitivity of the algorithms to various DAG attributes. Figure 8 shows the average SLR for high-communication (CCR=10), medium-communication (CCR=1), and low-communication (CCR=0.1) DAGs. We can see that the performance difference among algorithms is very sensitive to CCR. We think it is because high communication costs affect the performance of LHBS the most. Wide DAGs should also show the effect, since

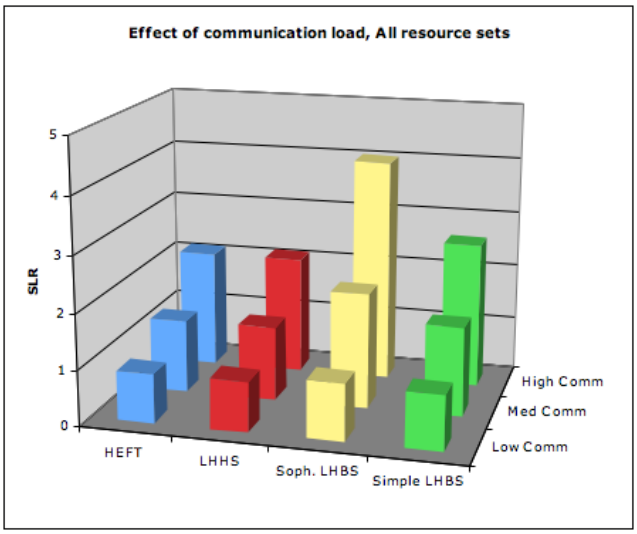


Fig. 8. Results for varying communication-computation ratios (CCR)

there are more opportunities for inappropriate parallel assignment. Figure 9 shows this for wide ($\alpha = 5$), square ($\alpha = 1$), and narrow ($\alpha = 0.5$) DAGs. Denser DAGs

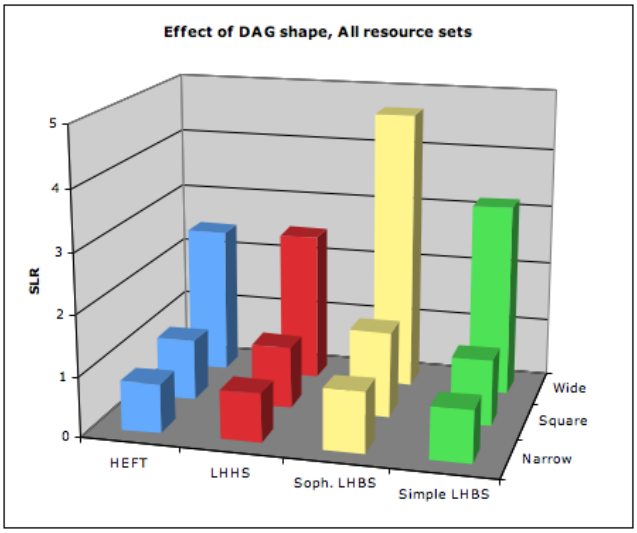


Fig. 9. Results for varying shapes (α)

(i.e. with higher out-degree) should show the effect more. Figure 10 shows little difference in this regard, however. This is likely because all of the tested graphs are relatively sparse, with average out-degrees of 5 (sparse), 2 (sparser), and 1 (sparsest). Figures 8 and9 consider only the random, level and choke graph types, since we have not yet generated complete data for Montage nor EMAN DAGs. If this paper is accepted, we plan to finish those experiments and update the graphs accordingly.

It may be less apparent why our hypothesized parallelism/communication trade-off affects the large universal

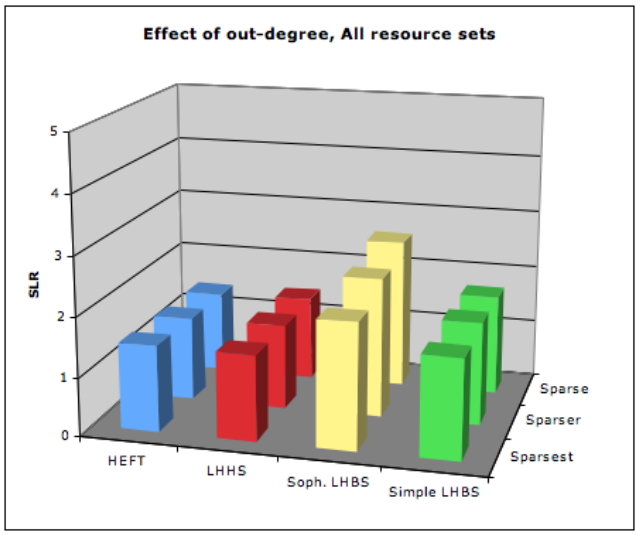


Fig. 10. Results for varying out-degree (ν)

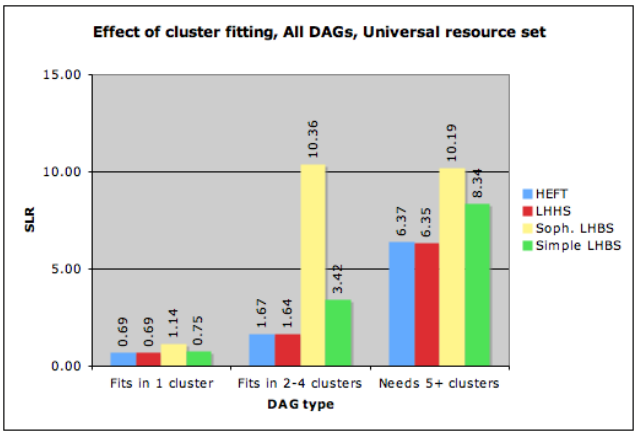


Fig. 11. Random DAG Performance in Universal Resource Environment with Different Widths

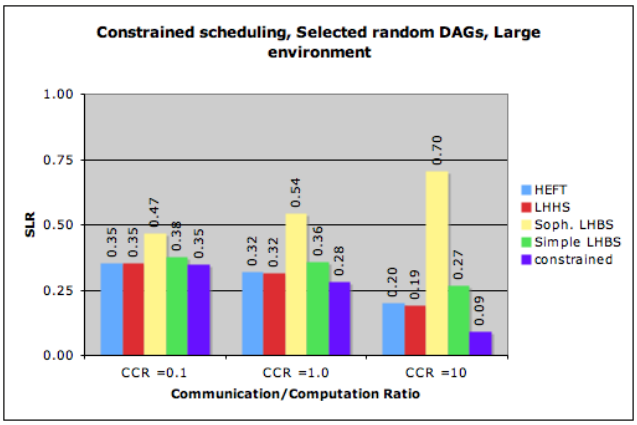


Fig. 12. Constraining the set of resources

environment more than the others. The connection is in the characteristics of the resource pools. Our previous work [37] shows that all of our algorithms usually use clusters with the fastest nodes most of the time. Table I lists the number of nodes and their speed in the four highest-GHz clusters in each of the three Grid environments. Clearly, the per-node speeds of these clusters in the universal resource environment are closer than in the other environments. At the same time, the top cluster in the universal environment is larger than in the others. Therefore, a relatively narrow DAG (e.g. width=40) can be run entirely on a single, fast cluster in the universal environment. Running the same DAG run on the many-cluster or big-cluster environment must either use slower cluster (e.g. the second cluster in the big-cluster environment) or multiple clusters (e.g. all four displayed clusters in the many-cluster environment). Figure 11 illustrates this effect. When the DAG's width is less than the number of nodes of the fastest cluster or is larger than all the nodes in the fastest four clusters, the difference between algorithms are much smaller than when the DAG's width is in between. In other words, when the choices between clusters are obvious, all the algorithms perform relatively the same, while when the choices are tough, different algorithms can perform very differently.

	Universal		Big-Cluster		Many-cluster	
	nodes	speed	nodes	speed	nodes	speed
First	78	4.2 Ghz	38	4.2 Ghz	13	4.2 Ghz
Second	6	4.2 Ghz	52	3.0 Ghz	18	3.8 Ghz
Third	103	4.1 Ghz	88	2.8 Ghz	17	3.7 Ghz
Fourth	118	4.1 Ghz	34	2.0 Ghz	6	3.6 Ghz

TABLE I
THE CONFIGURATION OF THE FASTEST FOUR CLUSTERS IN THE RESOURCE POOL

The above observations suggested that we could improve the quality of schedules for Grid environments by choosing the clusters on which to run more intelligently. In particular, we tried a quick experiment by running the HEFT algorithm on the 300-node random DAGs using the third fastest cluster in the universal Grid. We also ran a series of tests of the HEFT algorithm on 1000-node random DAGs on the third and fourth fastest clusters in the same Grid. Figure 12 and 13 show these experiments compared to our standard scheduling methods. The leftmost set of bars of Figure 12 represents DAGs that has low communication cost(CCR =0.1). In this case, the constrained scheduler does not have a clear advantage over the original HEFT and the other scheduling algorithms. The middle set represents DAGs that has medium communication cost(CCR =1.0); The rightmost set represents the most communication intensive DAGs(CCR =10), this is where we thought that the standard methods would likely to make bad trade-offs between parallelism and communication, and it confirms that we were right. Similarly, Figure 13 shows that the constrained scheduler has a better performance

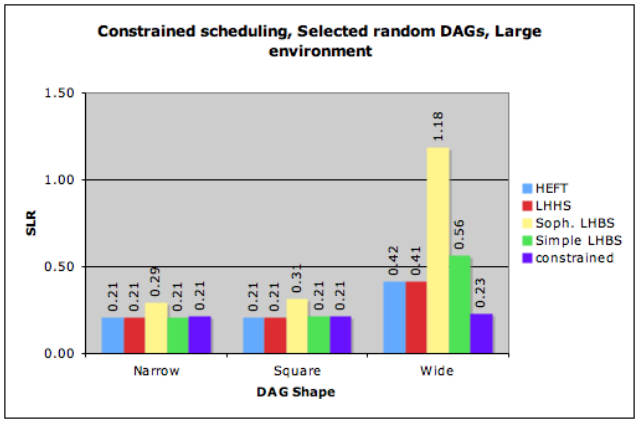


Fig. 13. Constraining the set of resources

over our standard scheduling methods when the DAG is wide($\alpha = 5.0$). Figure 12 and 13 show that reducing inter-cluster communication for the communication intensive or wide DAGs is every effective and the fastest clusters are not always obvious choices.

In summary, our experiments show that list-based(or hybrid) scheduling algorithms can be applied successfully in a Grid environment. In our tests, they often outperformed level-based scheduling methods, although this depended on the details of the environment and DAG being scheduled. Further improvements in scheduling appear possible by incorporating knowledge of the Grid environment. Specifically, ensuring that the scheduler does not attempt to spread the application over inappropriately many clusters is a key to performance.

VI. Conclusion and Future Work

In this work, we have compared the performance of several algorithms that represent two major approaches to scheduling on three different Grid environments. Our results showed that the list-based(or hybrid) algorithms can be applied successfully to the Grid environment and showed how different factors in a Grid computing environment affect the performance of the scheduling algorithms. We also showed that the most critical question in scheduling in the Grid environment is whether to assign a node to a cluster different from its parents. The performance of the algorithms can be drastically different based on the approach to this question.

In the future, we will devlop methods for more intelligently dividing work among clusters. We will also contine experiments to investigate the robustness of the resulting schedules, reflecting the fact that performance estimates are imperfect. We expect this research to lead to a new set of scheduling algorithms specifically targeting the Grid resource environment.

Since our universal resource environment is large, we encountered some difficulties in gathering enough computational power to finish the experiments. We used three heterogeneous clusters to run about 4000 jobs, each of which can take from 30 minutes to 24 hours. We see our own experiment as an ideal application for the Grid environment since the whole experiment is embarrassingly parallel, until the final data collection. In the future we will try to use Grid software to conduct such experiments, controlled by the scheduling algorithms we develop. This will provide both simulation and real-world data to validate our methods and our simulations.

Acknowledgments

This material is based on work supported by the National Science Foundation under Cooperative Agreement No. CCR-0331645 (the VGrADS Project). This work was supported in part by the Rice Terascale Cluster funded by NSF under Grant EIA-0216467, Intel and HP.

References

[1] Ishfaq Ahmad and Yu-Kwong Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 9(9):872–892, 1998.

[2] Rashmi Bajaj and Dharma P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):107–118, 2004.

[3] Sanjeev Baskiyar and Christopher Dickinson. Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. *J. Parallel Distrib. Comput.*, 65(8):911–921, 2005.

[4] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[5] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gomez-Sanchez. Grid characteristics and uses: a grid definition.

[6] T. Braun, H. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[7] Charlie Catlett and et al. http://www.griphyn.org, 2002.

[8] Sekhar Darbha and Dharma P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Trans. Parallel Distrib. Syst.*, 9(1):87–95, 1998.

[9] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal. Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

[10] A. Dogan and R Ozguner. LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing (ICPP'02)*, page 352, Washington, DC, USA, 2002. IEEE Computer Society.

[11] P. Druschel, A. Nandi, T.S. Ng, R. Riedi, G. Wang, and B. Zhang. Measurement-based analysis, modeling, and synthesis of the internet delay space for large scale simulation. Technical Report TR06-872, Rice University, 2006.

[12] P. Thambidurai E. Illvarasan. Levelized scheduling of directed a-cyclic precedence constrained task graphs onto heterogeneous computing system. In *First International Conference on Distributed Frameworks for Multimedia Applications (DFMA'05)*, pages 262–269. IEEE Computer Society, 2005.

[13] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann Publishers, Inc., 1999.

[14] I. Foster and C. Kesselman. *The Grid2*. Morgan Kauffmann Publishers, Inc., 2003.

[15] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

[16] H.Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 2(13):260–274, 2002.

[17] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee. Scheduling precedence graphs in systems with inter-processor communication costs. *SIAM Journal of Computing*, 2(18):244–257, 1989.

[18] M. Iverson, F. Ozguner, and G. Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *4th Heterogeneous Computing Workshop (HCW '95)*, pages 93–100, Apr 1995.

[19] Y.-S. Kee, H. Casanova, and A. A. Chien. Realistic Modeling and Svnthesis of Resources for Computational Grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2004. IEEE Computer Society.

[20] Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms . *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[21] Y. Kwok, I. Ahmad, and J. Gu. FAST: A Low-Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors . 2, 1996.

[22] S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated software for high resolution single-particle reconstructions. *J. Struct. Biol*, (128):82–97, 1999.

[23] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *14-th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, pages 125–134, 2005.

[24] A. K. Nanda and L. N. Bhuyan. Mapping applications onto a cache coherent multiprocessor. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 368–377, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[25] H. Oh and S. Ha. A Static Scheduling Heuristic for Heterogeneous Processors. In *Euro-Par '96: Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*, pages 573–577, London, UK, 1996. Springer-Verlag.

[26] Santosh Pande, Dharma P. Agrawal, and Jon Mauney. A scalable scheduling scheme for functional parallelism on distributed memory multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(4):388–399, 1995.

[27] C. I. Park and T. Y. Choe. An optimal scheduling algorithm based on task duplication. *IEEE Trans. Comput.*, 51(4):444–448, 2002.

[28] Samantha Ranaweera and Dharma P. Agrawal. A scalable task duplication based scheduling algorithm for heterogeneous systems. In *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, page 383, Washington, DC, USA, 2000. IEEE Computer Society.

[29] R. Sakellariou and H. Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 111. IEEE Computer Society, 2004.

[30] V. Sarkar. *Partitioning and scheduling parallel programs for execution on multiprocessors.* PhD thesis, Stanford, CA, USA, 1987.

[31] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993.

[32] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. A metadata catalog service for data intensive applications. In *In ACM Supercomputing Conference (Phoenix, AZ, Nov. 2003)*, 2003.

[33] G. Singh, E. Deelman, and G. Bruce Berriman et al. Montage: a Grid Enabled Image Mosaic Service for the National Virtual Observatory. *Astronomical Data Analysis Software and Systems*, (13), 2003.

[34] Jan Janecek Tarek Hagras. A Simple Scheduling Heuristic for Heterogeneous Computing Environments . page 104, 2003.

[35] M. Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. Parallel Distrib. Syst.*, 1(3):330–343, 1990.

[36] L. Yang, J. M. Schopf, and I.Foster. Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[37] Y. Zhang and Mandal A. etc. Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling. In *Proceedings of the 6th IEEE Symposium on Cluster Computing and the Grid (CCGrid'06)*, May 2006.