Relative Performance of Scheduling Algorithms in Grid Environments

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

***Abstract*— Effective scheduling is critical for the performance of an application launched onto the Grid environment [11], [12]. Finding effective scheduling algorithms for this problem is a challenging research area. Many scheduling algorithms have been proposed, studied and compared on heterogeneous parallel computers but there are few studies comparing the performance of scheduling algorithms in Grid environments. The Grid is unique because of the drastic cost differences between inter-cluster and the intra-cluster data transfers. In this paper, we compare several scheduling algorithms that represent two classes of schedulers used for Grid computing. We analyze the results to explain how different resource environments and workflow application structures affect the performance of these algorithms. Based on our experments, we introduce a new measurement called *effective ACP* that could drastically improve the performance of some schedulers.**

I. Introduction

With the development of large-scale high-speed networks, usually called "the Grid", becomes an attractive computational platform [11], [12] for high-performance parallel and distributed applications [14], [26], [20]. Although grid technologies enable the sharing and utilization of widespread resources, the performance of parallel applications on the Grid is sensitive to the effectiveness of the algorithms used to map them Grid resources. Scheduling is the decision process by which application components are assigned to available resources to optimize various performance metrics. In this paper, we focus on scheduling the important class of *workflow applications*, in which the overall task is partitioned into multiple (usually coarse-grain) sub-tasks linked to each other by data dependences, typically requiring file transfers. We represent the overall application as a *directed acyclic graph* (DAG), in which 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 represented by an edge must occur over the network between processors to which its endpoints are assigned. We assume that the application is available in the form of a DAG and the scheduler must 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.

In general, scheduling parallel and distributed applications is a known NP-complete problem [13]. Numerous heuristics have been proposed for scheduling DAGs onto a heterogeneous or homogenous computing environment [1], [15], [24], [16]. These strategies fall into several categories, including list-based, clustering, and duplication-based strategies. Among thee, list-based scheduling heuristics are generally accepted as the best overall approach, exhhibiting both low complexity and good results [19] .

The *list-based scheduling strategy* first orders the nodes in the DAG by a pre-calculated priority then considers the nodes in order, assigning each to a resource that minimizes a suitable cost function. However, Iverson [17], Illvarasan [10] and Atakan [8] argue that the pre-computed order for list-based strategy cannot be used in heterogeneous environments and propose a new heuristic class that we call the *level-based strategy*. Level-based methods first organize the DAG into levels, or echelons, such that within each all the nodes are independent and can be scheduled once all the node in the previous level have been completed; they then schedule all the nodes in each level, level-by-level from beginning to the end.

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. Looking over surveys of state-of-the-art Grid scheduling algorithms [29], [9], we can see that many Grid projects simply use dynamic dispatching mechanisms similar to condor [23]. Besides that, the list-based and the level-based algorithms are the only two strategies used in a Grid project. Blythe *et. al.* [3] reported that the level based strategy out performed the random matching strategy by more than 50%. However, to the best of our knowledge, there has been no published research that directly compares the performance of these two classes of algorithms in the Grid environment.

In this paper, we evaluate the schedules produced by several well-known list-based and level-based scheduling algorithms. Relying on tens of thousands of experimental runs, we show how the performance of these algorithms varies with differences in resource environments and ap-

plication DAGs. We analyze these results to explain why some scheduling algorithms perform better in certain settings and less well in others. Based on these observations, we introduce a promising new scheduling concept, called *effective aggregated computing power (ACP)* and demonstrate how it can be used in scheduling algorithms.

The rest of the paper is organized as follows. Section II covers the basics of scheduling DAGs in a Grid environment. Section III briefly introduces all the scheduling algorithms that we evaluate in this paper and their usage in the state-of-the-art Grid computing projects. Section IV presents our applications, the experimental environments we are using, and the Grid parameters we vary in the experiments. Section V presents our results; it also defines effective ACP and shows how it works in a scheduling algorithm. Section VI concludes the paper with a summary of contributions and perspectives on future work.

II. Problem Definition

A fundamental concept underlying the Grid is the view of the global network of resources as an active computational environment, connecting geographically distributed computers, databases, instruments and people into a seamless web of advanced capabilities. Miguel *et al.* [4] 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) interconnection network. Furthermore, while many homogeneous processors reside in any one cluster, the processors in different clusters are often significantly different. 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* as a pair $G = (V, E)$, where V is a set of nodes, each representing an application task, and E is a 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 dummy entry and 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. 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 such that $M[v_i]$ is a pair (r_i, t_i), where r_i is the resource on which the node will be executed and t_i the time it will start. In this paper, the objective of the scheduling algorithms is to output a concrete DAG corresponding to an abstract DAG such that the actual scheduled length, usually called the *makespan*, is minimized.

III. Related Work

There has been considerable work in scheduling DAGs to minimize the schedule length (makespan) on either homogeneous [19] or heterogeneous processors [5]. As we mentioned in section I, the level-based and list-based algorithms are the most used ones in Grid environments and we want to compare their performance. For our experiments, we have chosen some representative and effective algorithms in both categories. This section gives a brief overview of each of those algorithms.

A. List Scheduling Algorithm: HEFT

Heterogeneous Earliest Finish Time(HEFT) [15] is a well-established list-based algorithm known to perform well on heterogeneous platforms [2], [15]. Like all the list-based scheduling algorithms, HEFT has two phases, namely, the node prioritizing phase and the processor selection phase. For more detail, refer to the paper by Topcuoglu, Hariri, and Wu [15].

Upon examination, we discovered that the rank used in HEFT is a heterogeneous adeptation of the definition of *b-level* commonly used in list-based scheduling algorithms. Thus, it can be considered as a heterogeneous version of MCP(Modified Critical Path) [27] algorithm which performs very well in homogeneous environment [19]. Both Ma and Buyya [21] and Cao *et. al.* [6] use HEFT to help schedule application DAGs onto Grid resources. The computation complexity of this version of HEFT is $O(v^2+vp)$.

B. Levelized Scheduling Algorithm: LHBS

Levelized Heuristic Based Scheduling(LHBS) [22] is a level-based algorithm for Grid scheduling. As all level-based algorithms do, it proceeds by partitioning the DAG into levels of independent nodes. Within each level, LHBS can use Greedy, Min-min,Min-max or Sufferage heuristics [5] to map the nodes to the processors. Both the GrADS [3] and Pegasus [14] schedulers use a version of LHBS. The complexity of the LHBS using only the greedy heuristic is $O(vp)$; we will refer this as *Greedy LHBS*. The complexity of the LHBS using the other three heuristics is $O(v^2p)$; we will refer this variant as *Heuristic LHBS*.

C. Hybrid Scheduling Algorithm: HHS

Hybrid Heuristic Scheduling (HHS) [25] is refers to a class of algorithms that use a hybrid of the list-based and level-based strategy. The version we study in this paper first computes levels as in LHBS, then processes nodes in each level following the prioritized order used by HEFT. This version has the same complexity as HEFT: $O(v^2 + vp)$. Sakellariou [25] reports that it can achieve better performance than HEFT.

IV. Experimental Methodology

In order to study how well these scheduling strategies perform in the Grid environment, we have implemented the algorithms described in Section III and compared the schedules produced on 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 combinations of DAGs and grids.

A. DAG Model

We use DAGs taken from two real Grid applications, along with three classes of artificially-generated, but realistic, DAGs. EMAN [20] and Montage [26] are two real workflow applications that we have previously studied [3], [22]. The generated DAGs abstract certain characteristics of these applications. For details, please refer to Zhang *et. al.* [31].

1) 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 [2], [15], 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 level immediately above. In the choke DAG, there always exists one level (the choke point) that 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: α represents the ratio of the DAG height (i.e. number of levels) to the width (i.e. maximum number of nodes in a level). The height and the width of the DAG are generated using the method described by Topcuoglu, Hariri, and Wu [15], which takes α and λ as parameters.
- 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 (CCR)*: Following Blythe *et al.* [3], 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, as described by Kee. Casanova, and Chien [18]. 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 sub-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 latency matrices according to the actual latency data collected over the Internet [30]. Following the experiment results of Yang *et al.* [28] and Denis *et al.* [7] we assigned the bandwidth based on the latency. Low-latency links had high bandwidth, consistent with the data in Bo *et al.* [30]. Given the latency and bandwidth of each network link, it was a simple matter to compute the communication cost matrix M_n.

The costs we generated are static, although actual Grids can have dynamic costs due to load. However, we claim that the static data help us focus on performance of the algorithms and factor out the uncertainties of resource and network behavior. We are embarking on separate research to explore the effects of dynamic costs on the algorithms

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 all of these 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 [19], [15], [2], 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 use the estimated *critical path length*. Because the costs of nodes depend on where they are mapped, in this calculation 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

$$SLR = makespan/CPIC$$

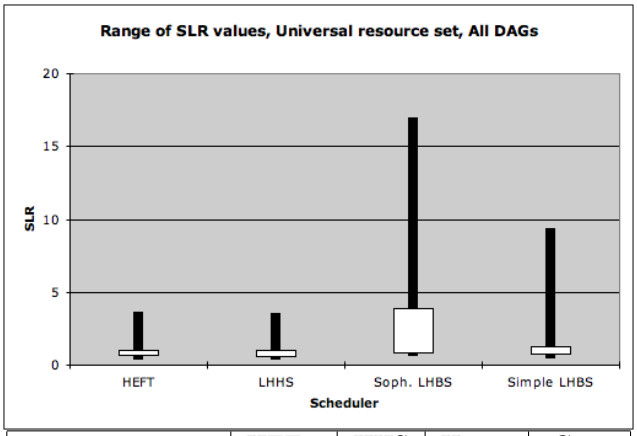
Intuitively, a small SLR is indicative of a better schedule than a large SLR. An SLR of 1 occurs when all nodes and edges are mapped to average processors and network links, and no bottlenecks occur due to lack of resources. An SLR can be below 1 when some nodes are mapped to faster-than-average resourcess, and above 1 due to resource limits or use of slower-than-average resources. 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

Over the entire set of DAGs and Grids, SLRs ranges from 0.06 to 88. (The range of makespans is even greater.) Moreover, the algorithm that produces the best schedule for each DAG varies with no obvious pattern. Once the results are aggregated, however, a somewhat clearer picture emerges.

A. Results Analyses

Figure 1 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



	HEFT	HHS	Heuri. LHBS	Gree 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. 1. Aggregate behavior of scheduling methods

outliers, but that the bulk of the results from the HEFT, HHS, and Greedy 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 HHS 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 HHS often computed equivalent schedules, particularly for choke DAGs. This would lead us to believe that HEFT or HHS produce better schedules than level-based methods on average. However, we did not observe the clear advantages of HHS over HEFT reported by Sakellariou and Zhao [25].

The difference in behavior was not, however, consistent across types of DAGs, as shown by Figure 2. In particular, all of the methods produced good schedules for EMAN. Most of the differences are statistically significant (the exceptions are HEFT and HHS 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

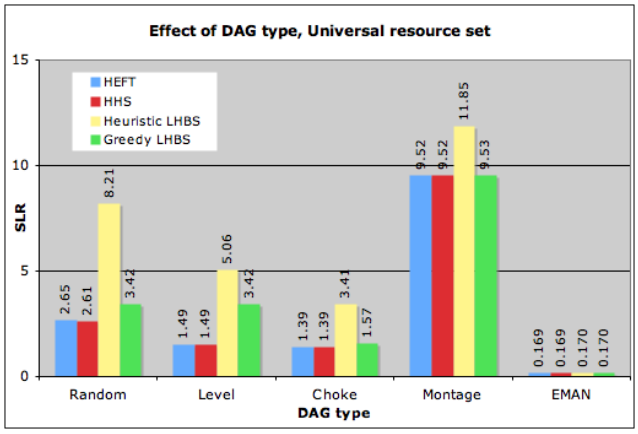


Fig. 2. Results for different DAG types

resource sets, as Figure 3 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 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

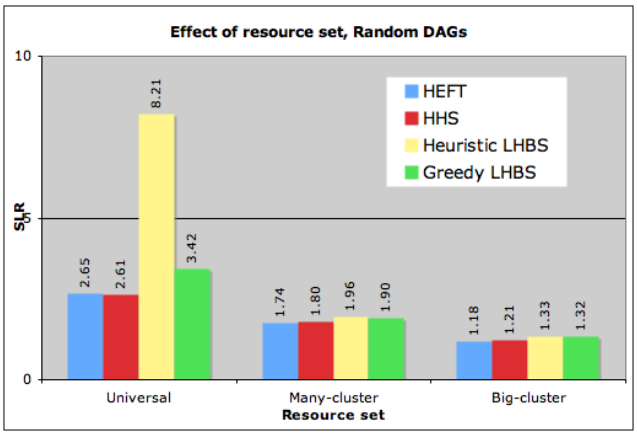


Fig. 3. Algorithms Performance on Different Resource Models

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, HHS. 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 4 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 there are more opportunities for inappropriate parallel assignment. Figure 5 shows this for wide ($\alpha = 5$), square ($\alpha = 1$), and narrow ($\alpha = 0.5$) DAGs. Figures 4 and 5 consider only the random, level and choke graph types, since we have not yet generated complete data for Montage nor EMAN DAGs. (Note: 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 environment much more than the others. The connection is in the characteristics of the resource pools. Our previous work [31] shows that our algorithms typically select clusters with the fastest nodes. 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

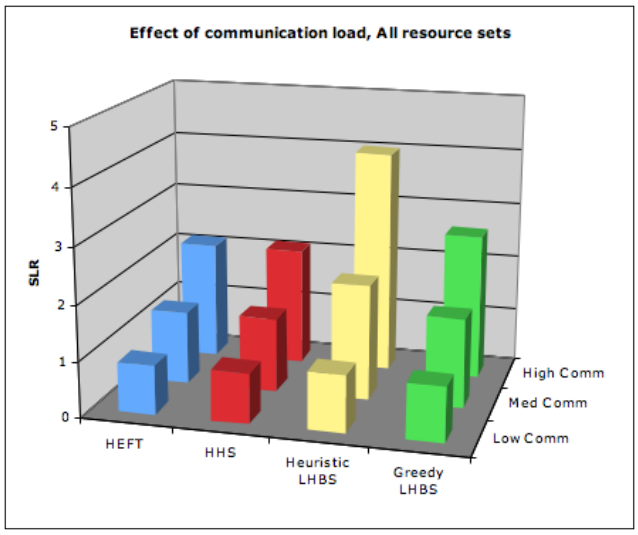


Fig. 4. Results for varying communication-computation ratios (CCR)

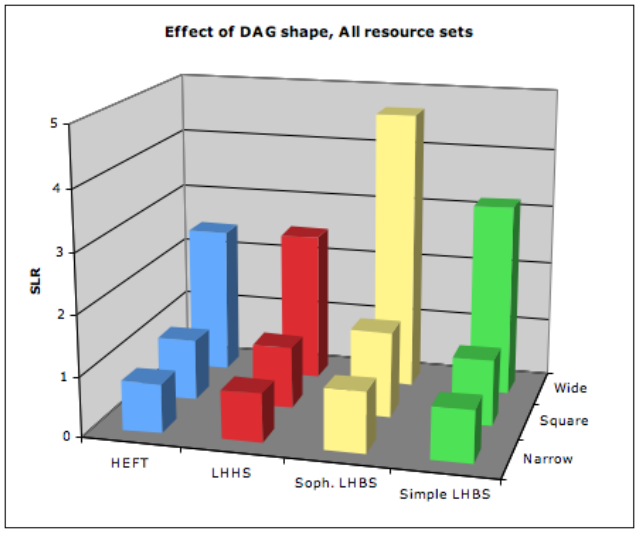


Fig. 5. Results for varying shapes (α)

	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

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 6 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. The above observations suggested that we could improve

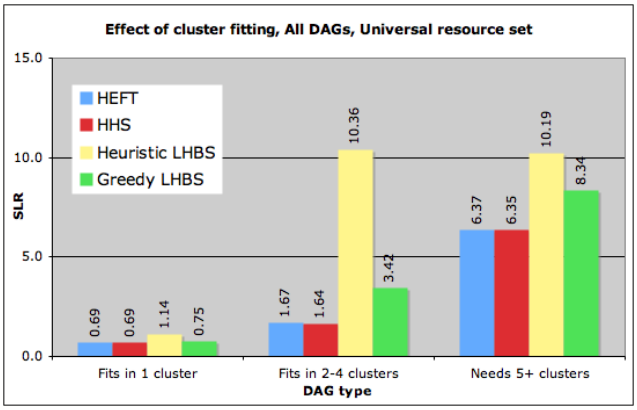


Fig. 6. Random DAG Performance in Universal Resource Environment with Different Widths

the quality of schedules for Grid environments by choosing the clusters on which to run more intelligently.

B. Effective ACP

To investigate further, we introduce the notion of *effective aggregated computing power (ACP)* and apply it within the *two-level scheduling* approach from our previous work [31]. Briefly, our two-level scheduler performs a very fast selection phase to select a suitable subset resource from the large resource base represented by the real Grid. It then performs a more complex scheduling step, such as LBHS, to map the application to nodes within the virtual grid.

We define ACP for a cluster A as

$$ACP(clusterA) = \sum_{B \in A} computing\ power\ of\ \ node\ B$$

We use the node's clock rate as an approximation of the computing power, although we could use more sophisticated performance models as well. ACP represents the peak computing power of a cluster, but this may not all be usable on a particular DAG. For example, consider running 20 independent tasks on two clusters. Cluster A consists of 100 processors running at 1GHz, while cluster B consists of 30 processors running at 2 GHz. Our unit of comparison is one processor running at one GhZ. Although A apparently has a higher ACP (100 units vs. 60 units), the DAG can utilize at most 20 processors in either cluster.

Therefore, we introduce the notion of *effectiveness* which only aggregates the computing power up to the width of the DAG. In our example, cluster B has 40 effective ACP units while cluster A has 20.

Within the two-level scheduling algorithm described above, the selection phase chooses nodes from clusters with the highest *effective ACP* for the given DAG. After this selection, we apply the HEFT, LHBS, or HHS algorithms to the smaller universe of resources. Below we will refer to this as the *effective ACP version* or simply the *EACP version* of each standard algorithm. Figure 7 and 8 show

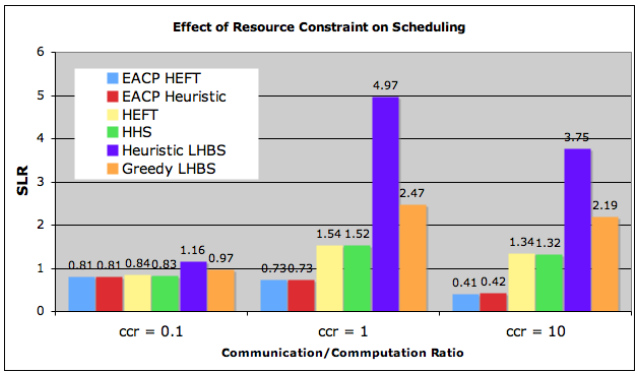


Fig. 7. comparing EACP version algorithms with the standard version

how the EACP versions of HEFT and Heuristic LHBS compared to the corresponding standard algorithms under the universal resource environment with all three kinds of DAGs. The EACP versions of the other algorithms exhibited very similar results. The leftmost set of bars of Figure 7 represents DAGs that have low communication cost(CCR =0.1). In this case, the EACP version algorithms do not have significant advantage over the standard HEFT or the other scheduling algorithms. The middle set represents DAGs that have medium communication cost(CCR =1.0) and the rightmost set represents the most communication intensive DAGs(CCR =10). We thought that the standard methods would be more likely to make bad trade-offs between parallelism and communication in these cases. The results confirm our beliefs. The EACP versions of HEFT and Heuristic LHBS outperformed their standard versions by factors of two to nine in aggregate. Both EACP algorithms performed better than any standard algorithm. Similarly, Figure 8 shows that the EACP version algorithms have better performance than the standard algorithms when the DAG is wide($\alpha = 5.0$). Taken together, figure 7 and 8 show that 2-level selection based on effective ACP can vastly reduce the inter-cluster communication cost when communication is significant. In addition, the EACP version algorithms are more scalable in very large Grid environments since they are only applied to a subset of the universal resources.

However, the results may vary depend on the Grid used. For example, the results of similar experiments using the

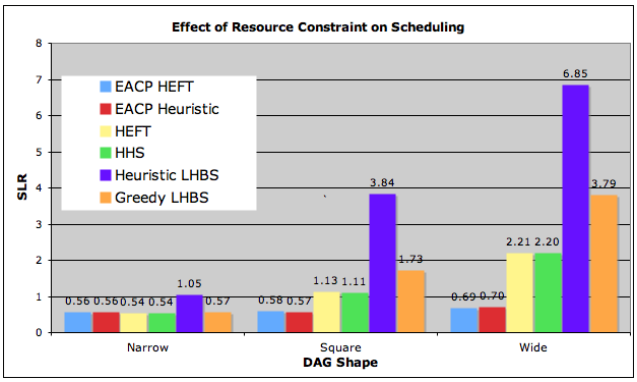


Fig. 8. comparing EACP versions with standard version of the same algorithms

big cluster environment shows EACP version of HEFT can perform 10% worse than the standard HEFT algorithm . We can explain this from the entries of Table I. In the big-cluster grid, the highest ACP cluster(the third) has relatively slow processors, so the fastest two clusters are likely to have the highest effective ACP for many DAGs. However, it happens that the network connection between these two clusters is slow in our experimental setting. Thus, selection based on effective ACP actually increases communication costs because it puts data movement on a slow link. More work is clearly needed to take effects like this into account.

VI. Conclusion and Future Work

In this work, we have compared the performance of several algorithms that represent alternative major approaches to scheduling on three different Grid environments. Our experiments show that the list-based, and hybrid, scheduling algorithms are effective in a Grid environment, outperforming level-based scheduling methods on many combinations of environments and DAGs. The experiments also show how different factors in a Grid computing environment affect the performance of the scheduling algorithms. The most critical question for scheduling in the Grid environment is whether to assign a node to a cluster different from its parents: performance of the algorithms are highly sensitive to this question. Finally, the experiments demonstrate that using effective aggregate computing power (EACP) in the selection phase of a two-level algorithm, then scheduling to the resulting virtual grid with a standard algorithm, can produce significantly improved schedules over the standard version of the same algorithm.

In the future, we will try to combine the concept of effective ACP and the concept of good network connection used in Zhang *et. al.* [31]. We will devlop methods for more intelligently dividing work among clusters and we will also conduct experiments to investigate the robustness of the resulting schedules, reflecting the fact that performance estimates are imperfect and the resources are dynamic. We

expect this research to lead to a new class 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 such an experiment as an ideal application for the Grid environment. 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.

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

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

[3] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken 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.

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

[5] 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.

[6] Junwei Cao, Stephen A. Jarvis, Subhash Saini, and Graham R. Nudd. Gridflow: Workflow management for grid computing. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 198, Washington, DC, USA, 2003. IEEE Computer Society.

[7] A. Denis and etc O. Aumage. 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.

[8] 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.

[9] F. Dong and Selim G. A.l. Scheduling algorithms for Grid computing: State of the art and open problems. Technical Report TR06-504, School of Computing, QueenâĂŹs University, 2006.

[10] P. Thambidurai E. Illvarasan. Levelized scheduling of directed acyclic 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.

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

[12] I. Foster and C. Kesselman. *The Grid 2*. Morgan Kauffmann Publishers, Inc., 2003.

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

[14] Ewa Deelman Gurmeet Singh, Carl Kesselman. Optimizing grid-based workflow execution. *Journal of Grid Computing*, 3(3):201–219, 2005.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] Tianchi Ma and Rajkumar Buyya. Critical-path and priority based algorithms for scheduling workflows with parameter-sweep tasks on global grids. In *SBAC-PAD '05: Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, pages 251–258, Washington, DC, USA, 2005. IEEE Computer Society.

[22] 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.

[23] Dagman MetaScheduler. http://www.cs.wisc.edu/condor/dagman.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for Grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[30] B. Zhang and T. S. etc Eugene. Measurement based analysis, modeling, and synthesis of the internet delay space. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 85–98, New York, NY, USA, 2006. ACM Press.

[31] 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.