Automatic Resource Specification Generation for Resource Selection

Richard Huang
Computer Science & Engineering
University of California, San Diego
1-858-822-2449
ryhuang@cs.ucsd.edu

Henri Casanova
Information and Computer Sciences
University of Hawai'i at Manoa
1-808-956-2649
henric@hawaii.edu

Andrew A. Chien
Computer Science & Engineering
University of California, San Diego
1-858-822-2458
achien@cs.ucsd.edu

ABSTRACT
With an increasing number of available resources in large-scale distributed environments, a key challenge is resource selection. Fortunately, several middleware systems provide resource selection services. However, a user is still faced with a difficult question: "What should I ask for?" Since most users end up using naïve and suboptimal resource specifications, we propose an automated way to answer this question. We present an empirical model that given a workflow application (DAG-structured) generates an appropriate resource specification, including number of resources, the range of clock rates among the resources, and network connectivity. The model employs application structure information as well as an optional utility function that trades off cost and performance. With extensive simulation experiments for different types of applications, resource conditions, and scheduling heuristics, we show that our model leads consistently to close to optimal application performance and often reduces resource usage.

Categories and Subject Descriptors
C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *distributed applications.*

General Terms
Performance.

Keywords
Resource requirement specification, scientific workflows.

1. INTRODUCTION
The current trend of steady growth in the number of deployed clusters is due to decreasing hardware prices, the existence of several cluster vendors, and the availability of open-source cluster management tools [1, 2]. Furthermore, improvements in network technology, with projects such as OptIPuter [3] that use optical fibers to create 10-40Gbps networks, make it possible to aggregate enormous amounts of these compute resources across large geographical regions into production large-scale distributed environments (LSDE). These LSDEs can be used for running large-scale, loosely synchronous applications such as scientific workflows. For instance, GriPhyN (Grid Physics Network) [4] is a physics research community that routinely requires LSDEs for data intensive science, with expected 100 Petabyte data requirements in the next decade.

The above is only possible with a *middleware* infrastructure for connecting and executing application components. Because of staggering and steadily increasing numbers of resources in LSDEs, when hundreds or thousands of users in tens or hundreds of institutions join in the collective sharing of resources, knowing where resources are and which resources are available is a challenge. As a result, middleware solutions exist for *resource discovery*, such as the Globus Monitoring and Discovery Services (MDS) [5] or the Internet Scout Project [6].

Another challenge is to *select* resources that are appropriate for executing an application. Much research [7-17] has explored resource selection. Resource selection systems encompass bilateral matchmaking [7] or multilateral gangmatching [8] in Condor [12], or a systems such as in SWORD [13] and RedLine [10, 11] where resources are selected via a constraint-solving process. Others employ relational databases (such as the vgES system [14, 15] in the VGrADS project [18]) to organize resources and apply nondeterministic queries [16] or other optimizations such as scoping or approximate queries [17] for fast searches. All these systems provide a resource requirement description language to express requirement specifications. Some languages allow for ranking functions which can be used to specify priority on resource characteristics.

The key problem (illustrated by **Error! Reference source not found.**) we address in this work, in the context of scientific workflow applications, is that most users do not know what resource requirement specification to provide to the resource selection service in order to obtain a resource set that optimizes application performance, while perhaps staying within some resource or resource cost budget. To the best of our knowledge this problem has not been studied in the literature. Users typically resort to naïve approaches, such as asking for a number of resources that matches the maximum parallelism of their workflow.

SC07, November 10–16, 2007, Reno, Nevada, USA.

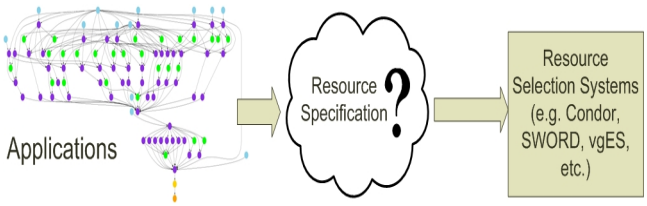

Figure 1: Key question of how to provide resource specifications to resource selection systems.

The key technical contributions in this paper are as follows:

1. A prediction model that outputs the best number of resources that should be requested given an application workflow and a scheduling heuristic. This model can be directly applied by users to generate resource requests.
2. Validation of our prediction model's accuracy and demonstration that its use leads to appropriate resource specification for near optimal application performance across wide range of randomly generated workflows and real application workflows.
3. Performance comparisons against the current practice of asking for a number of resources matching the maximum parallelism of the workflow which demonstrate that in all cases, our model leads to comparable or better application performance at reduced resource costs.
4. A sensitivity analysis of our model across varied levels of resource heterogeneity and several different scheduling heuristics. This analysis shows that our model enables good application performance across different resource conditions and different scheduling heuristics for a wide range of DAG applications.

The rest of this paper is organized as follows. In Section 2 we present our assumptions and methodology. In Section 3, we describe the resource specification generation problem. In Section 4, we derive our model for predicting the number of resources that should be requested and we validate this model in Section 5. Section 6 discusses the impacts of CPU heterogeneity and of different scheduling heuristics. Section 7 concludes and highlights future directions.

2. MODELS AND METHODOLOGY

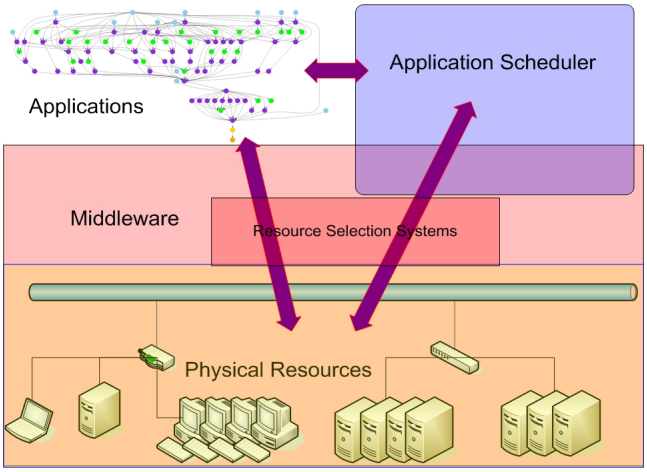

Figure 2: Overview of running applications on LSDEs.

The interaction between an application and a LSDE is depicted in Figure 2. The physical resources are shown at the bottom of the figure. In this work, we focus on compute and network resources only. LSDE resources are heterogeneous in terms of software, hardware, and access policies. A middleware infrastructure (e.g., GTK [19-22]) is used to hide and manage this heterogeneity, at least partially. Resource selection systems are part of this middleware infrastructure. The application, depicted in the top left corner of Figure 2, is comprised of potentially many compute tasks with different resource requirements that may need to communicate. Finally, in the top right corner of the figure we depict the application scheduler, which is responsible for mapping application tasks and data to resources, and which might be implemented entirely or partially in the application. The application can inform the scheduler about its characteristics, and the scheduler can use the middleware infrastructure to discover necessary resource information. Either the application or the application scheduler needs to specify the desired set of resources to the resource selection system [7-17], which then selects the desired set of resources from the resource universe.

Previous work [23] has shown that the scheduler performance can benefit enormously from using a selected resource collection rather than working with the whole universe of resources. In this paper we investigate how to derive appropriate resource specification from application characteristics.

2.1 Application Model

A popular and general application model is the "task graph" model, by which an application is represented as a weighted Directed Acyclic Graph (DAG). The DAG application model is particularly relevant for *scientific workflows* [24]. The last few years have seen active development and deployment of many such workflows in various domains such as physics [25, 26], image processing [27], and astronomy [28] and these workflows require considerable amounts of computing power. Therefore, it is natural to explore the possibility of executing them on large-scale computing platforms such as LSDEs [29].

Formally, we define a DAG as (V,E) where V = $\{v_1, v_2, \ldots, v_n\}$ is a set of nodes and E = $\{e_1, e_2, \ldots, e_m\}$ is a set of edges. A node in the DAG represents a task in the "task graph". A task is a set of indivisible executables or instructions that must be executed on one processor. The computational costs for each task is denoted by $w_v(v_i)$, in units of seconds on a reference CPU. We assume "uniform" compute resources: each resource has a compute speed and the execution time of a task on a resource is proportional to its speed. Therefore, a resource is not intrinsically better for some tasks than for some others.

Each DAG node can have multiple inputs, or multiple edges pointing to it. An edge represents the need to communicate data between tasks. The communication cost for each edge e_k is denoted by $w_e(e_k)$, in seconds. We calculate $w_e(e_k)$ by dividing the data size by an arbitrary reference bandwidth of 10Gbps. Each directed edge e_k represents a dependency between two tasks and implies that if $v_i \rightarrow v_j$, then v_i is the parent (denoted by $p_e(e_k)$) and v_j is the child (denoted by $c_e(e_k)$) and v_j cannot start executing until v_i finishes and has sent its data to v_j. $P(v_i)$ denotes the set of parent nodes of node v_i. and $C(v_i)$ denotes the set of child nodes of node v_i. A node without any parents is

called an entry node and a node without any child nodes is called an exit node.

2.2 Resource Model

We generate synthetic resources and simulate the execution of applications on LSDEs. Indeed, we wish to experiment with very large-scale distributed environments that do not exist today but will in the short- to medium-term future. Furthermore, even if these environments existed today, running experiments on them would be very expensive, time consuming, and non-repeatable. While the majority of the large scale systems in use today (such as the TeraGrid [30]) uses batch queue systems that lead to non-deterministic queue waiting times, many batch queue systems (e.g., PBS Professional [31]) allow for advanced reservations. Thus, we assume that the resource environment is composed of dedicated or reservation-based resources and that the middleware layer can acquire dedicated resources on behalf of the user, with a possible initial known delay (i.e., until the beginning of a known reservation period) that most likely increases with the number of requested resources. In this paper we do not incorporate this delay explicitly but a deployed implementation of a system based on our work could easily account for such a delay. Non-deterministic queue waiting times pose more of a challenge. However, others are working on queue waiting time prediction techniques [32, 33], which ultimately could be integrated with our work.

We assume that compute resources have identical processor architectures and that their speed depends only on their clock rates. We simulate application execution based on performance models of the task runtimes. For some applications, such as the Montage [34] application, a performance model is readily available. For other, reasonable predictions are possible and other researchers have developed techniques to derive predictive performance models [35].

In terms of network connectivity, we ignore latencies as they are negligible in the overall execution times for loosely-coupled applications such as distributed workflows. We make the simplifying assumption that resources returned by a resource selection service are interconnected by a bandwidth of 10Gbps. This assumption is reasonable for two reasons. First, given current trends and models for cluster configurations [36] and given the abundance of resources in future LSDEs, we argue that finding sufficient resources that are collocated on a local-area network should be often possible for most workflow applications. Furthermore, with projects such as OptIPuter [3], high and homogeneous wide-area bandwidths between institutions (where scientific workflows are likely to be deployed) should also become increasingly common. Therefore we leave a study of heterogeneous bandwidths between selected compute resources for future work. The inclusion of heterogeneous bandwidths would not fundamentally change any of the results in this work. Second, we keep this bandwidth fixed at 10Gbps, but in our experiment we vary the application's communication-to-computation ratio, which is equivalent to varying the bandwidth for a given application. We do not address the problem of network contention as that is outside the scope of this work.

2.3 Application Scheduling Heuristics

Among all the DAG scheduling heuristics surveyed and evaluated in [37] we chose the MCP (Modified Critical Path) heuristic [38] and the DLS (Dynamic Level Scheduling) heuristic [39], as they are popular and competitive according the results in [37]. We also use two baseline heuristics: Fastest Compute-host Available (FCA) and First-Come-First-Serve (FCFS).

2.4 Performance and Cost Metrics

Our measure of application performance is turn-around time, which is the sum of the time to compute the schedule and of the application *makespan*. The makespan of the application is defined as the difference between the start time of the earliest entry node and the end time of the latest exit node. We measure the time to compute the schedule by running scheduling heuristics on a 2.80 GHz Intel Xeon CPU.

While turn-around time is a common metric, defining a metric for cost is more difficult. Rather than coming up with an arbitrary metric, we choose to use the same one as an existing production system that charges users consistently: Amazon's Elastic Cloud [40]. In this system, each "instance", that is a (virtual) 1.7GHz x86 processor machine, is $0.10 per hour. We simply scale this cost by our simulated resources clock rates and compute total cost for application executions.

3. GENERATING BEST RESOURCE SPECIFICATIONS

While the subject of resource selection has been extensively studied [7-17], to the best of our knowledge all resource selection systems require the user to input resource specifications. The problem we are addressing in this work is: How does one generate the best resource specification?

The resource selection system returns to the user a resource collection (RC), or a set of hosts on which the user can execute his/her application. The best resource specification is the one that results in the best RC. The characteristics of an RC (as well as the choice of a scheduling heuristic) affect application performance and cost. While the benefits of using appropriate resource collections are clear [23], it is not clear how to send resource requirement specifications to resource selection services so that the returned RC optimizes application performance and/or cost. We define the "best" RC as one that minimizes application turn-around time (for a given DAG and a given scheduling heuristic).

We consider two parameters that characterize a RC: (i) number of hosts, or size; (ii) coefficient of variance of host clock rates within the RC. Existing resource selection systems [8-10, 12, 13, 15-17] allow users to specify resource requirements for different instantiations of these characteristics (sometimes quantitatively, sometimes qualitatively, sometimes only partially). Different values of these characteristics correspond to various tradeoffs. For example, increasing the RC size could shorten the makespan but could lengthen the scheduling time. Such trade-offs are the reasons why composing an optimal RC is challenging.

4. PREDICTION MODEL FOR RESOURCE SPECIFICATION

Our solution to the problem of resource specification generation is to use an empirical prediction model. Our vision, depicted in Figure 3, is that of a prediction model that takes as input the target DAG and an optional utility function that the user can specify to trade off high performance for low cost. The output is a resource specification that the user can use as the input to different resource selection systems to acquire the best set of resources for the application. The resource specification prediction model is composed of two components: the *heuristic prediction model* determines which heuristic should achieve the best performance for the target DAG; and the *size prediction model* predicts the best RC size given the target DAG and the scheduling heuristic.

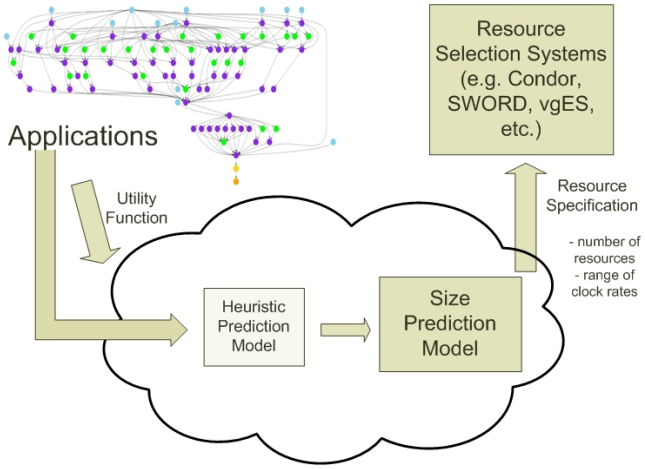

Figure 3: Resource Specification Prediction Model.

Due to space limitations, this paper focuses solely on the size prediction model. To predict RC size in this section we assume homogeneous compute resources and we use a reference scheduling heuristic, MCP. In Section 6, we evaluate the sensitivity of our prediction to resource heterogeneity and to different scheduling heuristics

Our empirical strategy for formulating an empirical prediction model for RC size is as follows:

1. Determine DAG characteristics that are likely to impact the choice of resources
2. Define what the best RC size should be
3. Execute the (reference) scheduling heuristic on an observation set of DAG configurations while varying the relevant DAG characteristics
4. Derive a model from the observation set results that predicts the best RC size

4.1 Relevant DAG Characteristics

Before we can formulate our prediction model, we need to identify the relevant DAG characteristics that affect the choice of resources for best performance. We identify and define the following four characteristics:

1. Dag size (n)
2. Communication-to-computation ration (CCR)
3. Parallelism (α)
4. Regularity (β)

The DAG *size* refers to the number of tasks in the DAG, n. A *level* of a node, denoted by level(v_i), is defined as the length (in number of nodes) of the longest path from an entry node to node v_i. Nodes in the same level could be processed in parallel. We define the height of the DAG (h), as the number of levels in the DAG. We denote the set of all levels $L = \{l_1, l_2, \ldots, l_h\}$ where l_1 contains the first entry node and l_h contains the last exit node. We define size(l_k) to denote the number of tasks in level k. We define the average number of tasks per level to be τ, where $\tau = n/h$.

We define the *Communication-to-Computation Ratio* (CCR) as the average of the cost of each edge e_k divided by the weight (task execution time) of the parent node of each (e_k) for all $e_k \in E$ (both are in units of seconds):

$$CCR = \frac{1}{m}\sum_{k=1}^{m}\frac{w_e(e_k)}{w_v(p_e(e_k))}$$

Note that we account for the data transfers between tasks by varying the CCR throughout our experiments.

The *parallelism* (α) parameter is derived from the number of tasks per level in the graph. Intuitively, we want to equate low α with low parallelism and high α with high parallelism so that when the number of tasks per level is 1 (in the case of a chain), α is 0 and when the number of tasks per level is equal to the DAG size (and there is only 1 level in the DAG), α is 1. We define parallelism as:

$$\alpha = \frac{ln(\tau)}{ln(n)}$$

The DAG *regularity* (β) characterizes the regularity of the distribution of the number of tasks at each level in the DAG. A regularity value of 1 would mean that all levels have the same number of tasks. The lower the regularity value the higher the variance in the number of tasks per level. We define regularity as:

$$\beta = 1 - \frac{R}{\tau}$$

where τ is the average number of tasks per level and R is defined as $|size(l_i) - \tau|_{max}$ where $\{i=1, \ldots, k\}$.

We base our RC size prediction model on DAG size, CCR, parallelism and regularity. We have determined experimentally that the DAG height and the average number of tasks in the DAG are in fact subsumed by the above four characteristics.

4.2 Best RC Size

Before constructing our model, we need to define what constitutes the best RC size for our model. For illustrative purposes, Figure 4 shows turn-around time as a function of the RC size for a DAG of size 5000, CCR of 0.01 and parallelism (α) of 0.7. The different curves are for different regularity values. As we can see from the figure, increasing the RC size improves application turn-around time up to a point. Beyond a certain threshold, the application turn-around time actually increases with more hosts in the RC due to increased scheduling time in spite of MCP having "low" polynomial complexity. The

time increase would be faster for more costly scheduling heuristics leading to narrower "good" regions. We define the best RC size, or "knee" as the smallest RC size such that a bigger RC size would improve turnaround time by less than (an arbitrary threshold of) 0.1%. It is necessary to pick a threshold value slightly greater than 0% so that small fluctuations in scheduling heuristic execution times cannot skew the results.

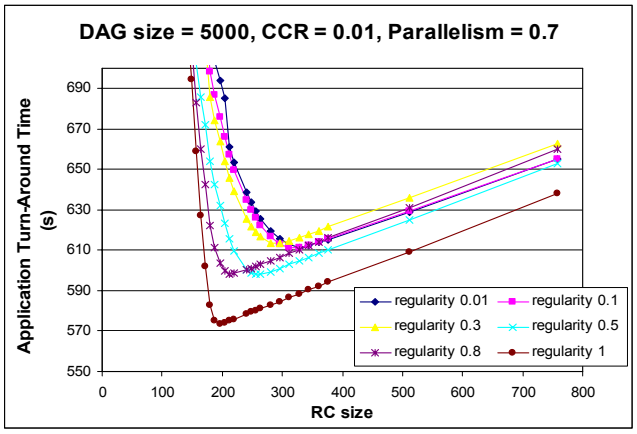

Figure 4: Resource Specification Prediction Model.

4.3 Sample Set of DAGs for Observation

Our strategy to obtain a predictive model of the best RC size is to use a sample set of DAGs for observation. We construct this observation set by choosing sample values for the four DAG characteristics identified in Section 4.1. We choose DAG sizes ranging from 100 tasks to 10,000, which is representative of current workflow applications. We pick CCR values between 0.01 and 1.0. For extreme communication intensive applications (CCR much greater than 1), a RC with a single host would be the best choice. When the computational cost is more than one hundred times the communication cost (as is the case when CCR is below 0.01), the communication costs becomes negligible and we can use the value of 0.01 for CCR to predict the RC for smaller CCR values.

When parallelism is low, the average number of tasks per level becomes small and a small RC would be sufficient to achieve good performance. Similarly, when parallelism is high (i.e., close to 1), all tasks can be executed in parallel and the best RC size would be equal to the DAG size. Thus, we choose values between 0.3 and 0.9. For regularity, we choose values ranging from 1.0 to 0.01. A regularity value of 1.0 means that all levels have the same number of tasks. A regularity value of 0.01 implies that a level can have at most twice as many tasks as the average. Our observation set is composed of 1260 DAG configurations. The values for each DAG characteristic are summarized in Table 1.

Table 1: Observation set DAG parameter values

DAG Characteristic	Values
DAG size	100, 500, 1000, 5000, 10,000
CCR	0.01, 0.1, 0.3, 0.5, 0.8, 1.0
Parallelism (α)	0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Regularity (β)	0.01, 0.1, 0.3, 0.5, 0.8, 1.0

For each DAG configuration, we instantiate 10 random DAGs. We vary the number of tasks per level randomly while maintaining the same DAG size, parallelism, and regularity value for each instantiation. For each instance, we simulate application execution using the MCP scheduling heuristic on RCs of various sizes and compute the turnaround time. From this data we empirically determine threshold (knee) values.

4.4 Prediction Model Formulation

After determining knee values for the observation set we need to formulate a model based on our four DAG characteristics. For better tractability we first consider only two: parallelism (α) and regularity (β). For example, Table 2 shows the knee values for DAGs with fixed DAG size 5000 and CCR of 0.01. These values are the average knee values for the 10 different DAG instantiations. Based on the data in Table 2, and others like it, we hypothesize that the knee value could be modeled as:

Knee = $2^{(a\alpha + b\beta + c)}$.

Table 2: Sample observation set knee values

α\β	0.01	0.1	0.3	0.5	0.8	1.0
0.3	34	32	22	18	14	14
0.4	52	36	28	24	22	20
0.5	80	62	58	50	56	42
0.6	136	140	128	112	94	128
0.7	328	312	280	248	212	196
0.8	464	456	448	448	448	432
0.9	496	496	440	440	432	392

Thus for each (size, CCR) pair, we need to solve for a, b, and c. Figure 5 plots the logarithm of knee values of various parallelism and regularity values for DAGs of size 5000 and CCR of 0.01. We obtained very similar planar shapes for all sample (size, CCR) value pairs, thereby suggesting that we can indeed use a planar fit and easily solve for a, b, and c for each (size, CCR) pair. It turns out that a planar fit to the data can be done in all cases with mean relative error of at most 16% for a DAG size of 5000.

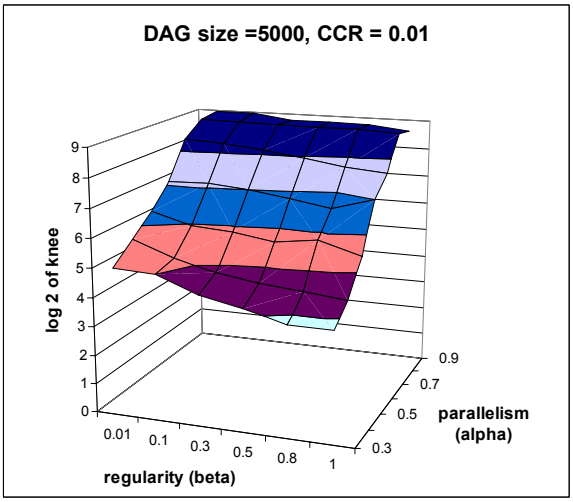

Figure 5: Logarithm of knee values.

The last remaining step is to account for the two remaining application characteristics: DAG size and CCR. We show a representative plot of varying knee values as a function of DAG sizes in Figure 6 for DAGs with fixed CCR of 0.01 and fixed parallelism at 0.7 for different regularity values. We also show a representative plot of varying knee values (for different parallelism values) as a function of CCR in Figure 7 with fixed DAG size of 5000 and fixed regularity at 0.01. Unfortunately, although these curves at first glance look logarithmic and exponential, it is difficult to find a simple model with a good fit. Therefore we resort to an empirical approximation based on linear interpolation between experimental data points on these curves. We interpolate in both dimensions when both DAG size and CCR value fall between values used in the observation set.

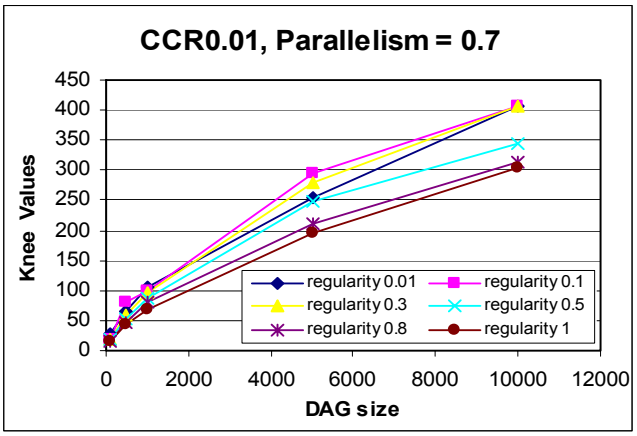

Figure 6: Knee values versus DAG sizes.

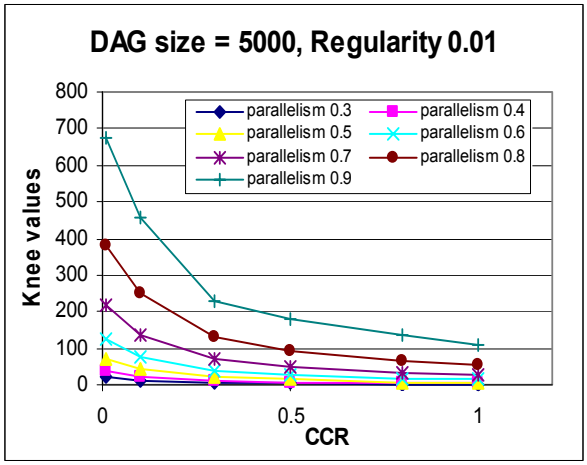

Figure 7: Knee values versus CCR values.

5. MODEL VALIDATION

We use two workloads to validate our model: (i) randomly generated DAGs; and (ii) DAGs from a real-world application. We validate our prediction model by comparing the performance achieved by using the predicted RC size and the optimal performance. Further, we compare the performance of our prediction model to the current practice of using the maximum parallelism as the RC size.

5.1 Validation with Random DAGs

We test our model with DAGs generated with the same DAG characteristics as our observation set (best case), and DAGs generated with characteristics halfway between consecutive DAG characteristic values of our observation set, which we call "midpoint" values (most likely worst case due to linear interpolation). We summarize parameter values in Table 3.

We evaluate the goodness of our model in terms of turn-around time using "deviation from best". We determine the best knee value via exhaustive search (which uses an amount of simulation that would be prohibitive in practice at runtime). We can then compare how far our model is from "best" in terms of knee value and in terms of turnaround time. Lastly, we compare the cost of using the RC size predicted by our model relative to the cost when using the RC size necessary to achieve the best turn-around time.

Table 3: DAG characteristic values for validation suite

DAG characteristics	Observation Set DAG characteristic values	Midpoint between two observation set DAG values
DAG Size	100, 500, 1000, 5000, 10,000	300, 750, 3000, 7500
CCR	0.01, 0.1, 0.3, 0.5, 0.8, 1.0	0.05, 0.2, 0.4, 0.65, 0.9
Parallelism	0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
Regularity	0.01, 0.1, 0.3, 0.5, 0.8, 1.0	0.01, 0.1, 0.3, 0.5, 0.8, 1.0

5.1.1 Results for Observation Set CCR Values

Table 4: Results for Observation Set CCR Values

Observation Set DAG Sizes	Average Prediction Size Difference	Average Performance Degradation	Relative Cost
100	9.59%	0.18%	-6.75%
500	11.49%	0.22%	-5.29%
1000	9.62%	0.32%	-4.32%
5000	13.27%	0.77%	-4.72%
10,000	14.53%	1.82%	-2.94%
Midpoint DAG Sizes			
300	13.41%	0.34%	-11.31%
750	11.85%	0.29%	-5.59%
3000	14.97%	1.08%	-9.98%
7500	16.71%	1.93%	-5.28%

Table 4 shows results for DAGs with CCR values identical to CCR values in the observation set. The top part of the table shows results for DAGs with DAG size values identical to DAG sizes in the observation set. We see that the average performance degrades slightly from best as the DAG size increases, ranging from degradation of 0.18% (for the smallest DAGs) to 1.82% (for the biggest DAGs) a cost saving between

6.75% and 2.94%. The bottom part of the table shows results for DAGs with DAG sizes corresponding to midpoint values. We can see that in spite of the linear interpolation approximation, the performance is only slightly worse (even in the worst case, the average performance degradation is only 1.93%. The negative numbers for relative costs suggests that our model is under-estimating, leading to a smaller than optimal but cheaper RC size.

A closer examination of our results revealed that the predictive model had the greatest errors for DAG configurations with lower CCR and higher parallelism (α) values. The peak performance degradation was at 16.71% for DAGs of size 7500. Less than 0.37% of the DAG configurations had performance degradation greater than 10%

5.1.2 Results for Midpoint CCR Values

Table 5 shows results similar to those in Table 4, but for DAGs that have midpoint CCR values. Our expectation was to see performance degradation similar to the degradation experienced by the model when interpolating between DAG sizes; however, we observed the opposite. With all other DAG parameters being equal except CCR values, we observed better average performance from the interpolated CCR values than from the observation set CCR values!

Ironically this behavior is due to underestimation from our model. Figure 7 shows that linear interpolation between two observation set CCR values actually predicts a larger RC size than if we had been able to fit a smooth curve between all the observation set points. In the end the error we had expected from interpolation overcompensated for the RC size underestimation of our model for DAGs requiring bigger RC sizes, leading to better turn-around times.

Table 5: Results for Midpoint CCR Values

Observation Set DAG Sizes	Average Prediction Size Difference	Average Performance Degradation	Relative Cost
100	11.04%	0.16%	-3.66%
500	11.11%	0.13%	-0.81%
1000	10.10%	0.13%	2.34%
5000	10.65%	0.34%	-0.40%
10,000	12.56%	0.86%	-6.97%
Midpoint DAG Sizes			
300	10.89%	0.19%	-6.76%
750	9.42%	0.14%	-0.42%
3000	11.95%	0.50%	-4.48%
7500	12.69%	0.98%	-1.12%

5.2 Performance-Cost Tradeoff

So far we have only presented results about optimizing turnaround time. However, different users may have different notions of utility and different economic constraints. We enhance our model by allowing the user to specify simple notions of for trading off performance for lowering cost. We accomplish this by generating predicted sizes based on various thresholds for defining the knee values. Previously, we only used a threshold of 0.1%. Now we expand our model to also use thresholds of 0.5%, 1.0%, 2.0%, 5.0%, and 10.0%.

As an example, a user may wish to trade off 1% performance degradation for every 10% in cost. Figure 8 shows the utility for different thresholds. With the 1% / 10% utility above, our prediction model would use a threshold value of 2.0% as the curve minimizes the combination of degradation and cost. Alternatively, users can input the budget for running the application and our model can choose the threshold corresponding to the best performance degradation while staying within budget.

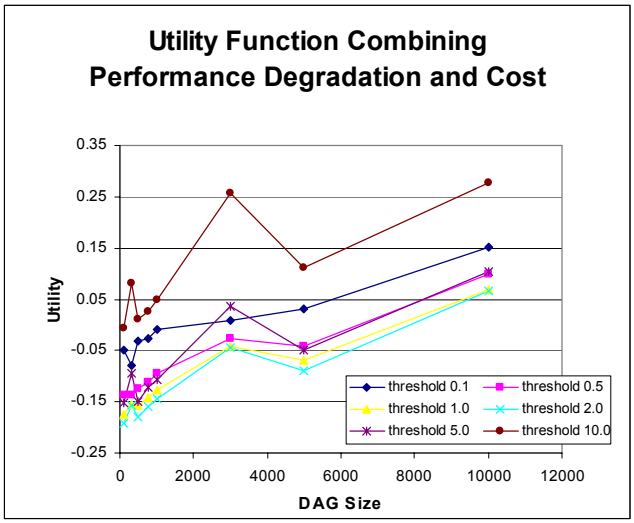

Figure 8: Utility vs. DAG size for various threshold values.

5.3 Comparison with Current Practice

Table 6: Using DAG width as the RC size

DAG Size	Observation Set CCR		Midpoint CCR	
Observation Set DAG Sizes	**Average Perf. Degrad.**	**Relative Cost**	**Average Perf. Degrad.**	**Relative Cost**
100	0.50%	144.8%	0.33%	209.2%
500	0.20%	425.7%	0.10%	437.3%
1000	0.45%	562.9%	0.36%	586.8%
5000	22.66%	998.1%	22.45%	1007.9%
10,000	130.93%	3360.9%	133.22%	3417.5%
Midpoint DAG Sizes				
300	0.30%	219.2%	0.15%	307.6%
750	0.26%	503.0%	0.18%	528.0%
3000	6.80%	759.8%	6.60%	782.2%
7500	81.22%	1429.8%	73.31%	1212.2%

While comparing performance to the best that could be achieved is interesting, it is also interesting to compare our approach to what a reasonable user might do. Without the aid of a predictive model, the natural and current practice is to use the DAG width

as the RC size, that is, the maximum number of tasks that can be run in parallel. In fact, the DAG width is an upper bound on the optimal RC size.

Table 6 presents results similar to those in Table 4 and Table 5, showing average performance and cost degradation from best, but obtained when using DAG width as the RC size. The first prominent numbers that provided stark contrast to the numbers for our prediction model are the relative costs. Using the DAG width as the RC size incurs enormous costs. Although using the DAG width may achieve similar performance as using our prediction model (for smaller DAGs), the costs may be prohibitive. When the DAG sizes are larger, the scheduling times become larger and using excessive resources leads to worse performance than when using our model. Therefore, our predictive model can achieve equal or better (or much better for larger DAGs) application performance at only a fraction of the cost.

5.4 Validation with Real Applications

In addition to validating our predictive model with randomly generated DAGs, we also validate the usefulness of our predictive model by applying it to DAGs from real applications. For applications that are computationally intensive, such as EMAN [41], choosing the DAG width as the RC size would yield the best application turn-around time. For other applications such as DAGs from the Southern California Earthquake Center (SCEC) [42], our predictive model is also unnecessary due to the specific structure of the applications. For example, the SCEC DAGs are composed of parallel chains. For such DAGs, the optimal size would equal the number of chains in the DAG.

We choose to validate our prediction model with Montage [43], an astronomy application that creates a mosaic image of a portion of the sky on demand. The size of the Montage DAG corresponds to the size of the mosaic. We test two Montage sizes – 1629 tasks and 4469 tasks. The 1629-tasks DAG corresponds to a three square degree mosaic and the 4469-tasks DAG corresponds to a five square degree mosaic. All tasks on level k have a parent task on level k-1. The top-level tasks (level 1) are not dependent on any other tasks. Table 7 summarizes the number of tasks in each level for the two Montage DAGs. We use the average runtimes of Montage tasks on a 1.5Ghz host as reported in [34].

Table 7: Tasks per level for the two Montage DAGs

Level	Task Name	Number of Tasks (1629)	Number of Tasks (4469)
1	mProject	334	892
2	mDiffFit	935	2633
3	mConcatFit	1	1
4	mBgModel	1	1
5	mBackground	334	892
6	mImgtbl	12	25
7	mAdd	12	25

Montage DAGs are extreme when compared to our observation set in two ways:

1. Low regularity: Recall that DAG configurations in our observation set have regularity values between 0 and 1. Both Montage DAGs have negative regularity numbers. (Our predictive model accepts negative numbers for regularity values.)
2. Low CCR values: Montage DAGs have small intermediate files ranging from 200 bytes to 8 Mbytes. Consequently we use a low CCR value of 0.01 for the two DAGs.

Table 8 summarizes the results of applying our predictive model to the Montage DAGs. We varied the application performance degradation thresholds from 0.1% to 10%. We can see that even relatively large threshold values do not lead to very large performance degradation. For budget-constrained users, a 10% threshold would be reasonable.

Table 8: Results for Montage DAGs

	1629-task DAG		4469-task DAG	
Threshold	**Perf. Degr.**	**Relative Cost**	**Perf. Degr.**	**Relative Cost**
0.1%	0.08%	11.20%	0.00%	0.00%
0.5%	0.04%	7.52%	0.00%	-2.40%
1.0%	0.01%	0.62%	0.00%	-4.03%
2.0%	0.89%	-13.48%	1.35%	-21.21%
5.0%	0.75%	-30.80%	1.81%	-30.41%
10.0%	4.18%	-48.22%	4.67%	-50.98%

For these Montage DAGs, using the current practice of DAG width as the RC size can have similar application turn-around time for the 1629-task DAG, but worse performance degradation (6.53%) for the 4469-task DAG. However, the relative cost when using the DAG width as the RC size are 89.08% and 195.9%, for the 1629-task DAG and the 4469-DAG respectively.

6. SENSITIVITY STUDIES

Our previous results were based on homogeneous resources with a reference scheduling heuristic. We also wish to determine how different levels of clock rate heterogeneity and different scheduling heuristics impact application performance and cost when our model is used.

6.1 Impact of CPU Heterogeneity

We construct our prediction model for clock rate heterogeneity by applying the same methods as in Section 4.4. We define clock rate heterogeneity as the coefficient of variance of the host clock rates in the RC. In this work we use values of 0%, 1%, 5%, 10%, 20%, and 30%. We maintain a constant mean clock rate in our experiments.

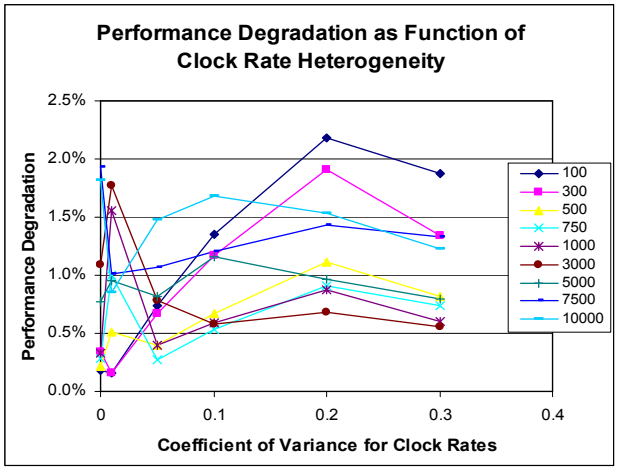

Figure 9: Performance degradation as function of clock rate heterogeneity for various DAG sizes.

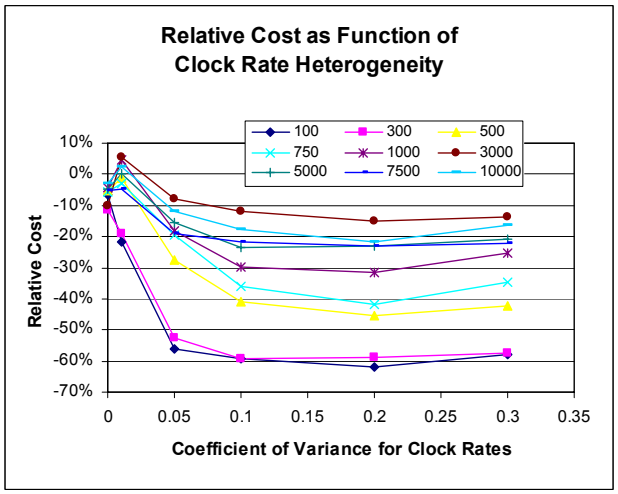

Figure 10: Relative cost as function of clock rate heterogeneity for various DAG sizes.

Figure 9 shows the degradation from best turnaround time versus clock rate heterogeneity (lines represent different DAG sizes) for DAG configurations with observation set CCR values only (recall that these performed worse). Figure 10 shows the relative cost of using our model (line representing different DAG sizes) as a function for clock rate heterogeneity. The main conclusion is that our model still leads to performance close to optimal at reduced costs even as heterogeneity increases.

With clock rate heterogeneity in resource collections, it is now possible to improve application performance, or improve savings, or both. Given that our reference scheduling heuristic (MCP) can take advantage of faster resources; we expect that the application turn-around time to be faster. Figure 11 shows that as the coefficient of variance of clock rate increases, the application turn-around time decreases (faster time) for all DAG sizes (represented by different lines). We also see that the turn-around times for bigger DAGs are less impacted by the same clock rate heterogeneity. This is most likely because more fast hosts are required to improve the overall performance of bigger DAGs.

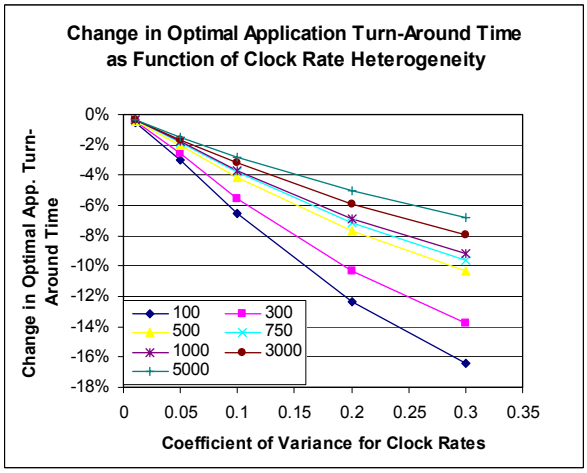

Figure 11: Change in optimal turn-around time as function of clock rate heterogeneity for various DAG sizes.

From Figure 9, Figure 10, and Figure 11, we draw the conclusion that using our prediction model, for applications capable of tolerating clock rate heterogeneity (most workflow applications are), users can achieve better application performance at cheaper costs than using homogeneous resources.

6.2 Impact of Scheduling Heuristics

We are also interested in seeing whether our model remains valid with other scheduling heuristics. We perform experiments with DAG configurations with CCR values of 0.01 and 1.0, parallelism values of 0.4 and 0.8 and keep regularity constant at 0.5 (because regularity impact the model the least).

We present Figure 12 and Figure 13 as the worst-case scenarios for our predictive model (DAG size 5000, parallelism value of 0.8) for different scheduling heuristics under different resource conditions. For different resource conditions, the average performance degradation for all heuristics stayed within 7.3% from optimal with savings at 65% for FCA and around 40% for DLS and MCP. As expected, the FCFS heuristic is more random and performance degradation increases with higher heterogeneity among the resources and costs are more unpredictable.

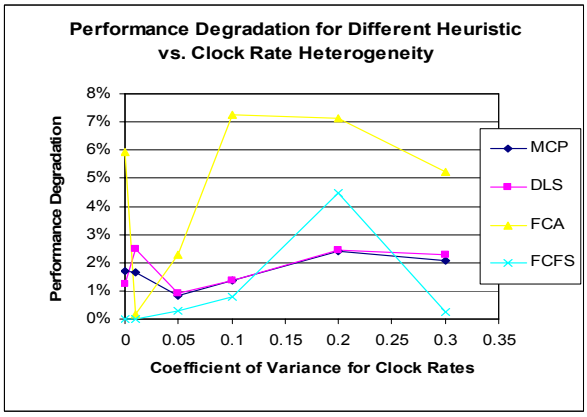

Figure 12: Performance degradation for different heuristics under different resource conditions.

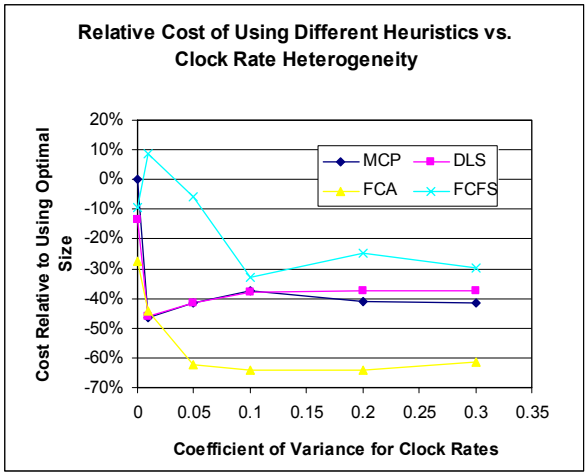

Figure 13: Relative cost of using different heuristics over different resource conditions.

7. CONCLUSION

In this paper, we have provided part of the missing link between scientific applications and resource selection systems. We have constructed a prediction model based on relevant workflow characteristics to output the best resource specification to optimize application performance (or some other utility that the user may specify). In this paper, we focused solely on the component of the prediction model that predicts the best number of resources to use in resource requirement specifications sent to a resource selection service, while considering performance and cost tradeoffs. In extensive simulation over a wide range of workflow configurations, we showed that our prediction model consistently allowed workflows to achieve performance within a few percent of optimal. When applied to a real application, we showed that our prediction model lead to almost optimal performance. Furthermore, when comparing the usage of our prediction model with current practice, we have found that using our model is far more cost effective while achieving better performance. Lastly, we showed that our model consistently achieved good performance for different resource conditions and different scheduling heuristics.

In a view to fully automating the interaction between users and resource selection services, a natural next step for this work is to suggest alternative specifications with graceful degradation in performance when the resource selection system cannot return the desired resources. Another useful enhancement to the model is to suggest to the application user the best scheduling heuristic in conjunction with the best resource specification to provide the optimal application turn-around time at the optimal cost. Our ultimate goal is to build this model inside a usable system that automatically generates resource requirement specifications for a number of deployed resource selection services [12, 13, 15].

8. REFERENCES

1. http://rocks.npaci.edu/Rocks/.
2. Papadopoulos, P., M. Katz, and G. Bruno. *NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters*. in *In the Proceedings of the IEEE International Conference on Cluster Computing*. 2001.
3. http://www.optiputer.net/.
4. http://www.griphyn.org.
5. *Globus Monitoring and Discovery System (MDS).* http://www-unix.globus.org/toolkit/mds/.
6. *Internet Scout Project,* http://scout.cs.wisc.edu/scout.
7. Raman, R., M. Livny, and M. Solomon. *Matchmaking: Distributed Resource Management for High Throughput Computing*. in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*. 1998. Chicago, IL.
8. Raman, R., M. Livny, and M. Solomon. *Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching*. in *Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing*. 2003. Seattle, WA.
9. Coleman, N., et al., *Distributed Policy Management and Comprehension with Classified Advertisements*. 2003, University of Wisconsin-Madison Computer Sciences Technical Report #1481.
10. Liu, C., et al. *Design and Evaluation of a Resource Selection Framework for Grid Applications*. in *Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing*. 2002. Edinburgh, Scotland.
11. Liu, C. and I. Foster, *A Constraint Language Approach to Grid Resource Selection*. 2003, University of Chicago Department of Computer Science Technical Report (TR-2003-07).
12. Litzkow, M., M. Livny, and M. Mutka. *Condor -- A Hunter of Idle Workstations*. in *Proceedings of the 8th International Conference of Distributed Computing Systems*. 1988.
13. Oppenheimer, D., et al., *Scalable Wide-Area Resource Discovery*. 2004, UC Berkeley Technical Report UCB//CSD-04-1334.
14. Chien, A., et al., *The Virtual Grid Description Language: vgDL*. 2004, UCSD Technical Report CS2005-0817.
15. Kee, Y.-S., et al. *Efficient Resource Description and High Quality Selection for Virtual Grids*. in *Proceedings of the IEEE Conference on Cluster Computing and the Grid*. 2005.
16. Dinda, P. and D. Lu. *Nondeterministic Queries in a Relational Grid Information Service*. in *Proceedings of the Supercomputing Conference*. 2003.
17. Lu, D., P. Dinda, and J. Skicewicz. *Scoped and Approximate Queries in a Relational Grid Information Service*. in *Proceedings of the 4th International Workshop on Grid Computing*. 2003.
18. *The Virtual Grid Application Development Software Project,* http://vgrads.rice.edu.
19. http://www.globus.org.
20. http://www.globus.org/toolkit/mds/.
21. Foster, I. and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit.* International Journal of Supercomputer Applications, 1997. **11**(2): p. 115-128.

22. Foster, I. *Globus Toolkit Version 4: Software for Service-Oriented Systems*. in *IFIP International Conference on Network and Parallel Computing*. 2005: Springer-Verlag LNCS 3779.
23. Huang, R., H. Casanova, and A.A. Chien. *Using Virtual Grids to Simplify Application Scheduling*. in *IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*. 2006.
24. http://www.extreme.indiana.edu/swf-survey/.
25. Barish, B. and R. Weiss, *Ligo and detection of gravitational waves.* Physics Today, 1999. **52**(10).
26. Deelman, E., et al. *GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists*. in *Proceedings of the IEEE High Performance Distributed Computing*. 2002.
27. Hastings, S., et al. *Image Processing on the Grid: a Toolkit for Building Grid-enabled Image Processing Applications*. in *Proceedings of the International Symposium on Cluster Computing and the Grid*. 2003.
28. Berriman, G.B., et al. *Montage: a Grid Enabled Engine for Delivering Custom Science-Grade Image Mosaics on Demand*. in *Proceedings of the SPIE Conference on Astronomical Telescopes and Instrumentation*. 2004.
29. Foster, I. and C. Kesselman, eds. *Computational Grids: Blueprint for a New Computing Infrastructure*. 2nd ed. 2003, M Kaufman Publishers, Inc.
30. http://www.teragrid.org/.
31. *PBS Professional:* http://www.altair.com/software/pbspro.htm.
32. Brevik, J., D. Nurmi, and R. Wolski, *Predicting Bounds on Queueing Delay in Space-Shared Computing Environments*. 2005, University of California, Santa Barbara: Santa Barbara, CA.
33. Brevik, J., D. Nurmi, and R. Wolski. *Predicting Bounds on Queueing Delay for Batch-scheduled Parallel Machines*. in *In Principles and Practice of Parallel Programming*. 2006.
34. Singh, G., C. Kesselman, and E. Deelman, *Optimizing Grid-Based Workflow Execution*. 2005, University of Southern California 05-851 PDF.
35. Marin, G. and J. Mellor-Crummey. *Cross Architecture Performance Predictions for Scientific Applications Using Parameterized Models*. in *Proceedings of the joint ACM SIGMETRICS-Performance 2004 Conference on Measurement and Modeling of Computer Systems*. 2004.
36. Kee, Y.-S., H. Casanova, and A. Chien. *Realistic Modeling and Synthesis of Resources for Computational Grids*. in *Proceedings of the ACM Conference on High Performance Networking and Computing*. 2004.
37. Kwok, Y.-K. and I. Ahmad, *Benchmarking and Comparison of the Task Graph Scheduling Algorithms.* Journal of Parallel and Distributed Computing, 1999. **59**(3): p. 381-422.
38. Wu, M.-Y. and D.D. Gajski, *Hypertool: A Programming Aid for Message-Passing Systems.* IEEE Transactions on Parallel and Distributed Systems, 1990. **1**(3): p. 330-343.
39. Shih, G.C. and E.A. Lee, *A Compile-Time Scheduling Heuristic for Interconection-Constrained Heterogeneous Processor Architectures.* IEEE Transactions on Parallel and Distributed Systems, 1993. **4**(2): p. 75-87.
40. *Amazon Elastic Cloud:* www.amazon.com/ec2.
41. *EMAN,* http://ncmi.bcm.tmc.edu/~stevel/EMAN/doc.
42. *Southern California Earthquake Center.* http://www.scec.org/.
43. Jacob, J.C., et al. *The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets*. in *Proceedings of the Earth Science Technology Conference (ESTC)*. 2004.