MATCHMAKING SCIENTIFIC WORKFLOWS IN GRID ENVIRONMENTS

Yili Gong, Marlon E. Pierce
Community Grids Lab
Indiana University
Bloomington, IN 47404
Email: gongy@indiana.edu, mpierce@cs.indiana.edu
Geoffrey C. Fox
Community Grids Lab, Department of Computer Science, School of Informatics
Indiana University
Bloomington, IN 47404
Email: gcf@indiana.edu

ABSTRACT
In this paper we analyze the scientific workflow matchmaking problem in Grid environments and combine workflow mapping and scheduling. Based on the characteristics of Grids, a new resource model is proposed. Motivated by the observations that not all jobs can run on all resources and that resource-critical jobs should be considered with their ancestor and descendant jobs when mapping, a novel resource-critical algorithm is designed based on a new Grid resource model. By means of experiments, it is shown to have good performance.

KEY WORDS
Grid Computing, Workflow, Resource Model, Matchmaking Algorithm

1 Introduction

Grids are attracting more and more scientific applications, such as those in earthquake science, computational chemical informatics, physics, astronomy, etc. These applications often include parallel computing and the processing of large scale data in steps. Because of the various requirements of jobs, the large number of jobs and the heterogeneity of resources, mapping large-scale collaborative workflows onto the heterogeneous resources in Grids is a non-trivial problem.

In the QuakeSim project [1], we are working on mapping scientific workflows onto TeraGrid resources for execution. There are some jobs which cannot run on all resources. For example, a the case of a program named GeoFEST, which can only run on IA-32 architecture and requires Pyramid library; only 3 out of 15 TeraGrid resources can satisfy its requirements. There are already some works in the workflow scheduling field but this kind of problem is not directly addressed.

In this paper, according to our target earthquake workflows and execution environment, TeraGrid, we make a few distinct assumptions. First, many works assume that a machine can only execute one job at the same time, i.e. it is exclusive. This holds true on a single processor but not for batch queuing systems in a cluster. Once two jobs have no data or logic dependency, they can probably run simultaneously on a computing resource (if not exceeding any limit). Based on this observation, we propose a new resource model. Second, previous works only assume that jobs can run on every resource and that the heterogeneity of resources only makes jobs' running time different. While in realistic Grids, due to access control policies, different software installation or version incompatibility, and particular hardware required (e.g. special visualization cards), etc, it's commonly seen that some jobs can never run on certain machines. This phenomenon makes the workflow scheduling in Grids quite unique and challenging. This is also why we call our workflow mapping "matchmaking", since it is not only scheduling the jobs onto resources, but also prior to that it matches the jobs with the resources which satisfy their requirements. The term "matchmaking" is borrowed from Condor [2], but in Condor matchmaking is only finding resources for single jobs, while our matchmaking system is aimed at finding resources for workflows. Under this assumption, the jobs which can run on every resource are more flexible than the resource-critical jobs which can only run on just a few resources. For a resource-critical job, considering the more resource-flexible jobs before and after it as a group for mapping should be better than mapping them individually. This is the key idea for our resource-critical workflow matchmaking algorithm.

Our main contributions are as follows. First, we propose a new resource model for cluster node in Grids; second, we propose a resource-critical algorithm for workflow matchmaking, which is proven to have good performance by experiments.

The remainder of the paper is organized as follows. Section 2 describes related work. Section 3 shows the system architecture of the workflow matchmaking system in Grids. The workflow matchmaking problem in Grids is formulized, a resource model is set up and a resource-critical algorithm is proposed in Section 4. In Section 5, the algorithm is evaluated under varying experimental settings. Finally, Section 6 concludes the paper.

2 Related Work

There is a large amount of work concerning DAG workflow scheduling. In recent years, much work focuses on scientific workflow scheduling and heterogeneous environments, especially Grids. Most of them only assume that jobs can run on every resource and that the heterogeneity of resources mainly makes jobs' running time different. While in realistic Grids, due to access control policies, different software installation or version incompatibility, storage limitation, etc, it is commonly seen that some jobs can never run on certain machines. On the other side, these works do not consider the global effect of the current scheduling decision. In [3], the authors claim that resources should meet certain hard requirements of jobs and use min-min heuristic for mapping, which is used as a comparative algorithm in our experiments. Some of them, such as HEFT in [4] and the hybrid heuristic in [5], are based on the assumption that no two jobs can be executed at the same time on a resource. But in Grid environments, the common resources are clusters, in which multiple jobs can run concurrently. In [6], the authors do take the storage constraint on resources into consideration. This can be easily incorporated into our model by stipulating that if a resource cannot accommodate the data files needed for a job, the job cannot run on the resource.

3 System Architecture

This work is intended to map scientific workflows onto resources in Grids. Figure 1 shows our implementation of a workflow matchmaking system, which reuses services from several existing projects. Since the target execution environment is TeraGrid, which use the Globus Toolkit to provide remote job submission and management, we select Condor DAGMan to describe and submit workflow jobs with its support by Condor-G. That is, we use Condor-G as a client to Globus services. Two services are used to get resource information required by workflow matchmaking. Static information, e.g. CPU number, OS, etc, is retrieved through GPIRQuery, the web service provided by GPIR [7]. Dynamic information, such as network bandwidth, host load, etc, is acquired by NWS [8]. QBETS [9] can predict the execution time

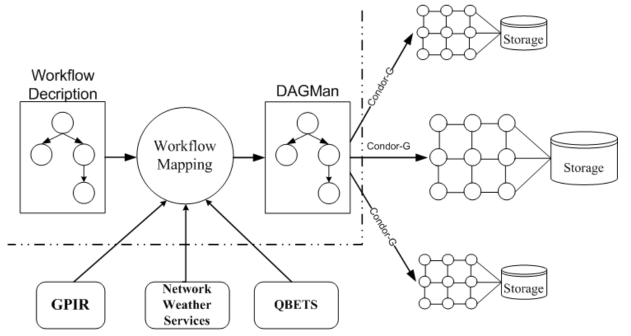

Figure 1. System architecture.

of jobs on resources. All these services are installed and running on TeraGrid to serve queries. The input of the workflow mapping system is a description of the workflow. It follows the grammar of DAGMan and Condor-G on submitting, except that it does not need to specify the actual location information about which resources the jobs are about to run on. The mapping system fills out the blanks and submits the completed submit files to DAGMan for execution.

4 A Resource-Critical Algorithm for Workflow Matchmaking

4.1 Problem Statement

This sub-section gives the formal description of the optimization problem.

Given a DAG (Directed Acyclic Graph) of the workflow representation of the application, $G=(V, E)$, $V=\{v_1, \ldots, v_N\}$ is the set of jobs in the workflow and N is the total job number. vol_{ij} denotes the volume of data generated by i and is required by j, $i, j \in V$ and $ij \in E$.

Let the set of Grid resources be $R=\{r_1, \ldots, r_m\}$ and M is the number of resources (machines) in the Grid. c_{ij} is the computation cost of job i on resource j. Let $C(G, R)=\{c_{ij} | i \in V, j \in R\}$. If job i cannot run on resource j, c_{ij} is infinity.

In batch systems, after they are submitted, jobs typically have to wait some time before actually running; w_{ij} is the waiting time for job i on resource j. Let $W(G, R)=\{w_{ij} | i \in V, j \in R\}$. Using QBETS we can get predicted waiting times for jobs on TeraGrid resources.

tr_{kl} is the transfer rate from resource k to l, $k, l \in R$. t_{ij}^{kl} is the communication cost between i and j, when i is executed on k and j on l, and is vol_{ij}/tr_{kl}, $i, j \in V$, $k, j \in R$. When i and j are executed on the same resource, the communication cost is zero. Let $T(G, R)=\{t_{ij}^{kl} | i, j \in V, k, l \in R\}$.

Let $parents(v)$ be the parent(s) of the job v and $children(v)$ be the child(children) of the job v, $v \in V$. Here, we assume that the DAG has a single entry node

v_0 which has no parent and a single exit node v_{N-1} which has no child; any of the other nodes has at least one parent and one child.

Assume the function *map(v): V→R* is the mapping from the jobs to the resources.

Let $EST(v,r)$ and $EFT(v,r)$ be the earliest start time and earliest finish time of job v on resource r respectively. For the entry node, $EST(v_0,r) = 0$, $r \in R$. For the other jobs, $EST(v,r)$ means the earliest time at which all of v's parent jobs have finished, the data it requires have been transferred to resource r and it is ready to run. Here, we assume that the data transferring and the job waiting can be concurrent. Thus it can be derived that $EST(v,r) = \max_{\forall u \in parents(v)} (EFT(u, map(u)) + \max(t_{uv}^{map(u),r}, w_{vr}))$. Here u is a parent of v, $EFT(u, map(u))$ is the earliest finish time of node u on resource $map(u)$ and $\max(t_{uv}^{map(u),r}, w_{vr})$ is the bigger of the transmission time from u to v and the waiting time of v on resource r. $EFT(v,r) = EST(v,r) + c_{vr}$. The makespan, i.e. the overall execution time of the workflow, is the earliest finish time of the exit job, v_{N-1}, i.e. $EFT(v_{N-1})$.

The workflow matchmaking problem is summarized as follows: Given {*G, R, C(G, R), W(G, R), T(G, R)*}. Select the mapping $map(v)$ to minimize the makespan of the workflow, $EFT(v_{N-1})$.

4.2 Resource Model

In early work, resource in a heterogeneous computing environment is modeled as a single processor on which no two jobs can run concurrently. While in Grids, the computing resources are mainly clusters, many but not an infinite number of jobs can run at one time. We propose a new resource model to describe resources in workflow matchmaking in Grids.

Each node in a Grid has a capability number, such as the CPU number of the cluster, and each job has a required capability number, such as the number of CPUs it needs. At any time, the sum of the required capability numbers of the jobs running on a resource cannot exceed the resource's capability. It is true that more jobs can run on a cluster concurrently, but the computing time of each job will suffer due to frequent context switching.

4.3 Mechanism and Algorithm

Since it has been proven that the workflow matchmaking problem is NP-complete, we try to find a good heuristic to solve it.

The key idea of the algorithm is to consider the more resource-flexible jobs before and after a resource-critical job as a group for better than mapping them individually.

The proposed algorithm is given in Algorithm 1. The input of the algorithm is a DAG G and two matrices: W gives the execution cost of each node on each machine and C gives the communication cost between two nodes/jobs connected by an edge on all combinations of different resources where two nodes can run; the cost is zero if the two jobs are executed by the same machine.

Since a job may not run on all resources, we define $MR(v)$ as the match ratio of the number of resources on which the job v can run and the number of all resources, $v \in V$. By checking the computation cost array, it is easy to get $MR(v)$ by calculating the number of c_{vr} which is not equal to infinity, $r \in R$.

Algorithm 1 The resource-critical workflow matchmaking algorithm.

```
 1: Set the computation costs of jobs and communication costs of edges with mean values.
 2: Compute the rank for all jobs by traversing DAG upward, starting from the exit node.
 3: Sort the jobs in a non-ascending order of the rank values. { 4 - 16: Group nodes.}
 4: $G_0 = \{\}$; $i = 0$.
 5: repeat
 6:    Get a node $v$ in the order of nodes' rank values.
 7:    if $v$ has not been grouped then
 8:       Add $v$ to $G_i$.
 9:       for all $u$ such that $u$ is $v$'s descendants do
10:          if all ancestors of $u$ have been grouped, all nodes on the path from $v$ to $u$ is in $G_i$ and $MR(u) \le \alpha$ then
11:             add $u$ to $G_i$.
12:          end if
13:       end for
14:       $i$++; $G_i = \{\}$.
15:    end if
16: until there are no more nodes.
17: for all group $G_i$, in ascending order of $i$. do
18:    Schedule the jobs in $G_i$.
19:    Choose the schedule with the smallest EFTs for the end nodes.
20: end for
```

The algorithm consists of three phases: ranking, group creation, and scheduling a group.

In the first step, a weight is assigned to each node and edge of the DAG, which is the mean value of all possible values. The weight of a node is the mean of its computation cost on all matched resources. The weight of an edge should be the mean of the maximum of the communication cost and the waiting time of all possible combinations of resources.

Using this weight, upward ranking is computed and a rank value is given to each node. The rank value, $rank_i$, of a node i is recursively defined as follows: $rank_i = nw_i + \max_{\forall j \in children(i)} (ew_{ij} + rank_j)$, where

nw_i is the weight of node i, and ew_{ij} is the weight of the edge connecting node i and j.

In the second step, nodes are sorted in non-ascending order of their rank values. Tie-breaking is done randomly. Based on this order, nodes are divided into groups. The first node (i.e. the node with the highest rank value) is added to a group numbered 0. Check each of its children if its ancestors are grouped and its match ratio is below a certain valve α. If so, add the child node into the group, mark it as grouped and check its children further on. If no additional such node is found, make the next ungrouped node as a new group, and so on. The outcome of this process is a set of ordered group, each of which consists of a node and its descendants on the path to which the match ratios of the nodes are all lower than the valve α.

In the third step, a group of nodes are mapped, where any algorithm for scheduling a DAG could be used. Since when scheduling a group, the mapping is probably incomplete, the makespan of the whole workflow is not a proper metric to value different assignments. Given a mapping, an end node is defined as a node which either has no children or whose children have not all yet been mapped. Comparing the two mappings, the one with the smaller largest EFT of all end nodes is preferred; if they have the same largest EFT, the one with the second smaller EFT is better; and so on. If all EFTs of the end nodes are the same, choose either of them randomly. So far, we adopt an enumerative algorithm to try all combinations of resources for a group and choose the one with the best EFTs of all end nodes.

On one side, with the valve α properly set, the size of a group is not large; on the other side, since the match ratios of nodes in a group, except the ancestor, are lower than a constant α, the number of combinations of resource assigning is not big. In practice, the running time is insignificant, since there are only low-cost operations involved in the algorithm.

4.4 Comparative Algorithm

Since we adopt a different resource model and different assumptions, most of the existing workflow scheduling approaches are not applicable. The minimum EFT algorithm, which is used in [3], is simple and easy to be extended to our resource model as well as satisfy our assumptions.

In the minimum EFT algorithm, nodes are sorted in non-ascending order of their rank values in the same way as the first two steps of the resource-critical algorithm, shown in Algorithm 1. Then nodes are scanned in the order of their rank values. For each node, the resource which makes it finish earliest is chosen.

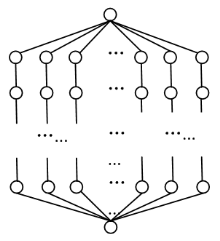

Figure 2. The structure of a parameter sweep DAG.

5 Experimental Evaluation

This section evaluates our resource-critical workflow matchmaking algorithm against the minimum EFT algorithm. First, the settings of the experiments are described. Then we define the metrics for evaluation. Finally the simulation results are shown and discussed.

5.1 Settings

Many factors influence the performance of a workflow matchmaking system in Grids.
DAG Generator: We generate parameter sweep DAGs, whose structure is shown in Figure 2. Parameter sweep applications are the typical usage scenarios in the QuakeSim and CICC project. Every DAG has one start node and one end node. Jobs on the same level in different branches have same resource requirements and similar execution time. We vary the branch number and the depth respectively from 4 to 12 and from 8 to 24, and correspondingly the number of node are from 34 to 290.
Heterogeneity Model: In our model, there are 15 resources and the factors indicating their computational power are the same as they are in TeraGrid. The base execution time of a job is chosen using a random uniform distribution over the interval [10, 100].
Match ratio: This is a new factor introduced by considering the fact that some jobs can never run on certain resources. The match ratio for a job is the ratio of the matched and total resource numbers. The ratios are randomly generated among the values from 0 to 1 while a job at least has one matched resource on which it can run.
Communication Bandwidth: The communication bandwidth between any two resources is a random number between 5M/s and 300M/s, which is the bandwidth range we measured among resources in TeraGrid.
Communication-to-Computation-Ratio (CCR): CCR of a parallel program is defined as its average communication cost divided by its average computation cost on a given system.
Match Ratio Threshold (MRT): This value is used by the resource-critical algorithm to decide which nodes should be grouped together for mapping. If the

MRT is so small that no job's match ratio below it, every node is a group and the resource-critical algorithm acts the same as the minimum EFT algorithm. If the MRT is too large, the groups will expand, for example, if MRT = 1, all nodes will form one group; to find the best solution for a big group is time consuming. In the experiments, we set MRT from 0.1 to 0.5.

For any given branch number and depth, we generate 200 DAGs with their own job computing times, job-resource match ratios and resource communication bandwidths, and each is called a case. With each combination of the branch number, depth, CCR and MRT, these two algorithms will be run on the 200 cases of every combination.

5.2 Metrics

Two metrics are used to evaluate the algorithms.
Difference Ratio: First, we define a metric named Normalized Schedule Length (NSL), which is the ratio of the makespan divided by a fixed cost of the critical path.

$$NSL = \frac{L}{\sum_{n_j \in CP} w(n_i)} \tag{1}$$

The denominator is the sum of computation costs on the critical path and is a lower bound on the schedule length. Considering communication cost, such a lower bound is probably not possible to achieve and the calculated schedule length should be larger than this bound.

Our resource-critical algorithm does not always outperform the minimum EFT algorithm. The difference ratio is the ratio of the difference of NSLs for two algorithms and the bigger NSL of two. If a difference ratio is below zero, the NSL of the resource-critical algorithm is larger than that of the minimum EFT algorithm; if above zero, the resource-critical algorithm has a better mapping result than the minimum EFT algorithm; if equal to zero, the two algorithms are deemed to perform the same.
Average Improvement Ratio: As its name shows, the average improvement ratio is the average of difference ratios of all cases in a certain setting.

5.3 Results

In the first set of experiments, the branch numbers and the depth numbers of the DAGs are all set as 4 and 8 respectively. The other settings have similar results.

The influence of Match Ratio Threshold (MRT) on difference algorithms is shown in Figure 3. Here CCR (Communication-to-Computation-Ratio) = 1.0. It can be noticed that in some cases, the resource-critical algorithm performs worse than the minimum EFT algorithm. The reason is that when mapping a group, the resource-critical algorithm always try to minimize the finish times of current nodes while sometimes compromising the current finish times can achieve better results in the long run. But in most cases, it outperforms the minimum EFT algorithm.

In Figure 4, the average improvement ratios of 200 cases are shown. As the MRT grows, the average improvement ratio of the resource-critical algorithm over the minimum algorithm grows from 6.31% to 23.13%. Combining these two figures together, it can be concluded that the bigger the MRT is, the better the difference ratios are. MRT is used to control which nodes should be grouped and mapped together: the bigger it is, the more nodes could be grouped together, thus there is a better chance to find a better mapping.

Figure 5 gives the difference ratios of 200 cases under different CCR values with MRT = 0.5. For all CCRs, in 7.5% - 8.5% cases among all 200 cases the resource-critical algorithm performs worse than the minimum EFT algorithm; in 13% - 19% cases they perform the same; in 72% - 78% cases, the former outperforms the latter. In Figure 6, the average improvement ratio increases from 11.65% to 23.69% as the CCR increases. This shows that the resource-critical algorithm works better where communication cost plays a heavier role. In the extreme circumstance in which there is no communication cost, our algorithm will degrade to the minimum EFT algorithm.

In the second set of experiments, we discuss the influence of the branch number, the depth and the node number on the performance of these algorithms. Here CCR (Communication-to-Computation-Ratio) = 1.0 and Match Ratio Threshold (MRT) = 0.5.

First, the depth of the workflows is set as 24 and the branch number varies from 4 to 12. As Figure 7 shows, the average improvement ratio does not deviate much from 45%. This is decided by the characteristic of the parameter sweep applications: the branches are independent from each other except that they share the same start and end nodes; thus the branch number does not much influence grouping and correspondingly does not much influence mapping.

Second, we set the branch number of the workflows as 4 and vary the depth from 8 to 24. As seen from Figure 8, the average improvement ratio of the resource-critical algorithm over the minimum EFT algorithm increases from 23.13% to 43.45% as the depth increases. This is because the greater the depth, the more chances there are that more nodes can be grouped together to find the best mapping by trying all possible combinations.

6 Conclusion and Future Work

In this paper, we investigate the problem of matchmaking scientific workflow onto resources in Grid en-

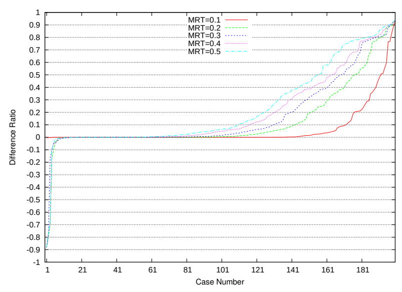

Figure 3. Difference ratio under various MRTs with CCR=1.

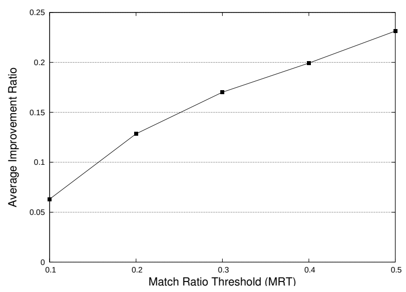

Figure 4. Average improvement ratio under various MRTs with CCR=1.

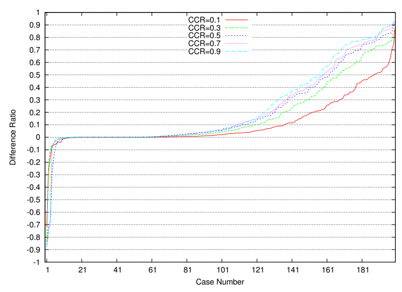

Figure 5. Difference ratio under various CCRs with MRT=0.5.

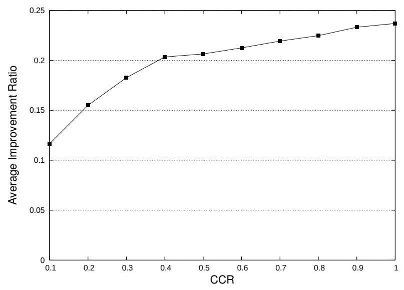

Figure 6. Average improvement ratio under various CCRs with MRT=0.5.

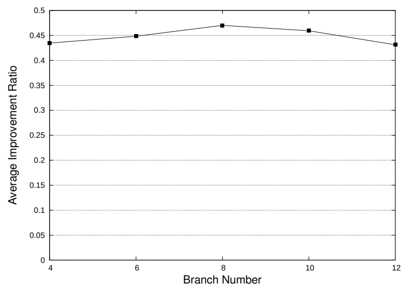

Figure 7. Average improvement ratio under various branch numbers with CCR=1.0, MRT=0.5, depth=24.

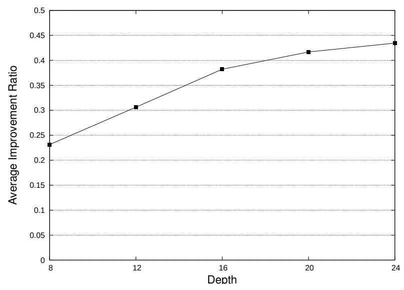

Figure 8. Average improvement ratio under various depths with CCR=1.0, MRT=0.5, branch number=4.

vironments. We combine the workflow mapping and scheduling together and present a new resource model to formalize the problem. By exploiting the fact that some jobs can only run on some resources, we propose a novel algorithm of resource-critical workflow mapping, which take advantage of the fact that some jobs can only run on a small number of resources and that mapping them together with their ancestor jobs can achieve better mapping results. By means of modeling experiments, we discuss the factors affecting the performance of the algorithm and it has been demonstrated that the resource-critical algorithm outperform the minimum EFT algorithm in various conditions.

This work is funded by the National Aeronautics and Space Administration's Advanced Information Systems Technology program.

References

[1] The QuakeSim project. `Http://quakesim.jpl.nasa.gov/`.

[2] D. Thain, T. Tannenbaum, and M. Livny, Distributed computing in practice: The Condor experience, Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[3] 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, Proc. of HPDC'05.

[4] H. Topcuoglu, S. Hariri and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, vol. 13, No. 3, pp. 260-274, March 2002.

[5] R. Sakellariou and H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems. Proc. of IPDPS'04.

[6] A. Ramakrishnan, G Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers and M. Samidi, Scheduling data-intensive workflows onto storage-constrained distributed resources, Proc. of CCGrid'07.

[7] `http://gridport.net/services/gpir/`.

[8] R. Wolski, N. Spring, and J. Hayes, The network weather service: A distributed resource performance forecasting service for metacomputing, Journal of Future Generation Computing Systems, Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

[9] D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue bounds estimation from time series, Proc. of 3th Workshop on Job Scheduling Strategies for Parallel Processing, June, 2007.