State Independent Resource Management for Distributed Grids

Aysan Rasooli
Department of Computing and Software
McMaster University
Hamilton, Canada
rasooa@mcmaster.ca

Douglas G. Down
Department of Computing and Software
McMaster University
Hamilton, Canada
downd@mcmaster.ca

***Abstract*—Advances in network technologies and computing resources have led to the possibility of deploying large scale computational systems, such as those following a Grid architecture (or related architectures). The scheduling problem is a significant issue in the Grid environment. In practice, a scheduling algorithm should consider multiple objectives. Typically, there are two kinds of objectives. The first is the performance of the system in terms of quantities related to the completion time of tasks, the second is the amount of state information required, which is often measured in terms of quantities such as communications costs. These two objectives are often in tension with one another. For example, gathering large amounts of state information can lead to low completion times. In this work, we introduce a scheduling algorithm which simultaneously addresses the objectives listed above namely, minimizing completion times, while requiring *zero* dynamic state information. Using simulation, we show the promising performance of our algorithm, and its robustness with respect to errors in parameter estimates.**

***Keywords*-Grid Scheduling; Scheduling Algorithms; Shadow Routing Approach**

I. Introduction

Innovations in computational and network technologies have led to the emergence of computational Grid systems [1]. Task scheduling is an integral part of a distributed computing system. These scheduling algorithms involve matching of application needs with resource availability. Several optimization criteria are considered for scheduling in Grids, making the problem a multi-objective one in its general formulation. With these criteria and other complexities of scheduling in the Grid environment, the scheduling problem is known to be NP-complete [2].

In recent years, several analogies from natural and social systems have been leveraged to form powerful heuristics for Grid scheduling, which have proven to be highly successful in attacking several NP hard global optimization problems [2]. These scheduling policies may use different system information to make reasonable scheduling decisions; some parameters are typically periodically estimated such as resource execution rate, and some are highly dynamic information, which should be gathered in real-time such as current resource loads.

As mentioned, Grid scheduling is a problem with multiple objectives, and it is extremely difficult to satisfy all of the objectives with one scheduling algorithm. To the best of our knowledge there is no single Grid scheduling algorithm which is the optimum over all Grid systems with their different applications and features. In this work, we address the scheduling problem of Grid systems whose resources are widely distributed, and there is a considerable communication cost between the resources. The most well-known application of these Grid systems is in the Enabling Grids for E-sciencE (EGEE) [3] project. The main contributions of our work are:

- We bring a theoretical idea from the queuing literature (the so-called Shadow Routing algorithm), to a practical scheduling algorithm implemented in Grid systems.
- We modify this basic theoretical approach to be efficient for Grid systems, and study the advantages of the proposed algorithm in widely distributed Grid systems.

In general, the Grid scheduling algorithm should improve the performance of the Grid system, which can be evaluated by various criteria such as Flowtime or Makespan. Furthermore, if an algorithm reduces the amount of state information required at the time of scheduling, this leads to reductions in the communication cost and synchronization overhead. In fact, a large system that requires full state information for scheduling may suffer from increasing limitations due to significant communication and synchronization overheads.

The Shadow Routing method is a robust, generic scheme, introduced in [4] for routing of arriving tasks in systems of parallel queues with flexible, many-server pools. This algorithm has proven to achieve good performance levels in queuing systems. However, its structure is designed in a way which is not directly applicable in Grid systems. In this paper we modify the structure of the Shadow Routing approach in two key ways, and introduce a scheduling algorithm for Grid systems, called the Grid Shadow Routing algorithm. First, we change the structure of the basic Shadow Routing algorithm to be applicable for a typical Grid workload model [7]. Second, we add to the basic Shadow Routing algorithm, which causes significant improvement in its performance in Grid systems.

The Grid Shadow Routing algorithm uses virtual queues to keep track of the loads on resources in the system. These are used as estimates of the actual queue lengths in the system, thus removing the need to gather real-time load information. The only information that the Grid Shadow Routing algorithm requires are estimates of the lengths of tasks and the execution rates of resources in the system. By using this information, the algorithm properly balances the loads on the resources. The two estimated parameters are used in most Grid scheduling algorithms. In order to provide this information various prediction methods are applied to forecast them, and in turn to guide task scheduling and load balancing strategies to achieve high performance and more efficient resource usage [5, 6]. An important advantage of our algorithm is that it does not require highly accurate estimates- even highly inaccurate estimates can be satisfactory for our algorithm.

Generally, based on the information that can be used, and the timing of the scheduling decision, scheduling algorithms are classified as either static or dynamic algorithms. Static scheduling algorithms do not use any dynamic state information, but there can be a huge performance degradation in comparison to dynamic algorithms. On the other hand, dynamic scheduling algorithms can make better scheduling decisions, while increasing the communication cost. As we mentioned before, there is no single scheduling algorithm which performs well in all Grid systems. If a Grid system has low communication overhead, a dynamic scheduling algorithm with full state information can make a significant improvement in the performance of the system. On the other hand, in a widely distributed system in which the time to gather full state information is significant (as in the systems in which we are interested), a dynamic scheduling algorithm which requires full state information can potentially create severe additional overheads. The large amount of overhead can lead to performance degradation. The main advantage of our proposed Grid Shadow Routing algorithm is that it requires *zero* state information from resources at the time of scheduling, and it can achieve much better performance than dynamic algorithms that require full state information. This is particularly advantageous for large, highly loaded systems with widely distributed resources, where communications costs are significant.

We use simulation to study the performance of our proposed algorithm in Grid systems. We evaluate the Grid Shadow Routing algorithm by comparing to two other algorithms. Minimum Completion Time (MCT) is a dynamic scheduling algorithm, commonly used in Grid systems. This algorithm greedily attempts to reduce the mean completion time, and does not consider the communication overhead in scheduling. The second algorithm that we employ is Join the Shortest Queue, which uses partial state information and does not require task lengths in making scheduling decisions. In order to evaluate the performance of our algorithm in a real system in which the parameters may be estimated inaccurately, we implement our algorithm in a system which has various levels of error in the estimates. Moreover, we evaluate our algorithm by using inaccurate parameters, to show that the significant performance gain of our algorithm can be achieved even in an environment in which there are errors in estimating required parameters.

The remainder of this paper is organized as follows. The problem of task scheduling and our workload model are described and formally introduced in Section II. An overview of several Grid scheduling algorithms is given in Section III. Then, in Section IV, the Shadow Routing algorithm is introduced and Section V provides the details of our proposed scheduling algorithm. In Section VI, details of the environment in which we study our algorithm are provided. In Section VII, we study the performance of our algorithm in various Grid systems with different error models in estimating the parameters of the system. Finally, we provide some concluding remarks and discuss possible future work in the last section.

II. Workload and System Model

In order to describe and evaluate our scheduling algorithm, we define a workload model based on a typical Grid workload [7]. As mentioned before, most Grid scheduling algorithms assume that estimates of task lengths and resource execution rates are available at the time of scheduling. In our workload model, we let the number of resources in the system be M. The actual resource execution rate for resource r is given by μ_r, and the task length for task k is given by L_k. We assume the use of one of the available estimation methods to provide estimates of resource execution rates for all resources and the length of each incoming task. We define the estimated length of task k as $\hat{L}_k$, and the estimated execution rate of resource r as $\hat{\mu}_r$.

In order to model a widely distributed Grid system, we assume that the Grid network has associated delays. The delay in the network is calculated based on the bandwidth and the load on the Grid network. When a task arrives to the system, the Grid scheduling algorithm is used to route the arriving task to one of the available resources in the system. Here we assume that all local schedulers are using the classical FIFO algorithm, however in general each resource can use its own local scheduling algorithm.

III. Current Algorithms and Related Work

A large number of algorithms have been designed to schedule independent tasks on computational Grid resources. In this section, rather than presenting a complete survey of current Grid scheduling algorithms, we list two of the commonly used scheduling algorithms for Grid environments, those for which we compare performance with our proposed algorithm.

- *Minimum Completion Time (MCT)*: assigns each task to the resource which has the minimum expected com-

pletion time for that task [8]. The expected completion time for a newly arriving task will be computed at each resource; the scheduler collects this state information from all resources and chooses the resource with the minimum expected completion time for execution of the new task. This algorithm can cause a significant improvement in maximum completion time of all tasks. However, it has the cost of requiring full state information, and consequently may have a large communication cost. This algorithm requires the following parameters: 1) estimated length of incoming task, 2) current estimated resource execution rate (considers the fluctuations in each resource execution rate), 3) estimated current available bandwidth in resources, and between resources and scheduler, 4) real-time load on each resource. The final three parameters should be collected from all resources at the time of scheduling.
- *Join the Shortest Queue* (JSQ*)*: This algorithm assigns each task to the resource which has the smallest number of waiting tasks in its queue. The advantage of this approach is that it does not require the length of tasks to make a scheduling decision. It simply uses the number of tasks in all of the queues as the state information. This algorithm just requires one parameter: the real-time number of tasks in each resource queue. However, this parameter should be collected from all resources at the time of scheduling.

IV. Shadow Routing Algorithm

The Shadow Routing algorithm was first introduced in [4] as an algorithm for routing in systems of parallel queues. This algorithm proposes a generic routing algorithm, which properly balances resources' loads and automatically identifies the best set of matchings, without the knowledge of the flow input rates and without explicitly solving any optimization problem. The algorithm is based on the Shadow Routing algorithm in a virtual queueing system.

In order to describe the basic Shadow Routing algorithm, first we define the queuing system model in which the algorithm is introduced in [4]. The model is defined as follows: It is assumed to have I classes (types) of tasks, and J pools of resources in the system in which the resources in each pool are homogeneous. The mean execution time of a task of class i on a resource of pool j is given by $\mu_{i,j} \geq 0$ (if $\mu_{i,j} = 0$, this means that tasks of class i can not be executed on resources of pool j). It is assumed that the input rate of tasks, and number of resources in each pool are increased simultaneously with scaling parameter $r \to \infty$. This means that the growth rate of incoming tasks to the system is $O(r)$, and the number of resources in pool j is given by $\beta_j r$, with parameter $\beta_j > 0$. Our brief description of the model is simply to introduce the basic Shadow Routing algorithm; more details about the model are provided in [4].

According to [4], in a heavily loaded queuing system, if the routing algorithm chooses only certain matchings of tasks to pools of resources, it can keep task queues stable and provide asymptotically optimal performance. By asymptotic optimality, we mean that if the number of resources in all pools and the input rates of tasks scale up simultaneously by a factor r (which grows to infinity), the Shadow Routing algorithm keeps the queuing system load within $O(\sqrt{r})$ of its optimal capacity.

- The initial values of Q_j are set to $\eta Q_j = 1/J$ for all $j \in (1...J)$.
- Upon each new task arrival, the algorithm does the following:
  1) Find the class which the incoming task belongs to, say class i.
  2) A virtual queue
  $$m \in \{\arg\min_j Q_j/(\beta_j \mu_{i,j})\},$$
  is identified, and the arriving task is sent to pool m.
  3) The virtual queue of the chosen pool is updated to :
  $$Q_m = Q_m + 1/(\beta_m \mu_{i,m})$$
  4) If the condition
  $$\eta \sum_j Q_j \geq 1$$
  holds, the following update is performed:
  $$Q_j = [Q_j - c]^+, \text{ for each } j,$$
  where $c = (\max\{1/(\beta_j \mu_{i,j}) \;|\mu_{i,j} > 0\})$, and $[x]^+ = \max\{x, 0\}$.

Figure 1. The Basic Shadow Routing Algorithm

Figure 1 presents the basic Shadow Routing algorithm. This algorithm maintains a virtual (shadow) queue Q_j for each pool of resources j - these are used to keep the loads in the system balanced. The algorithm makes each routing decision based on the values and simple updates of virtual queues; virtual here means that the queues are simply variables maintained by the algorithm. The parameter $\eta > 0$ is a small number (we elaborate later on how it should be chosen), which controls the tradeoff between responsiveness of the algorithm and its accuracy.

When a new task arrives to the system, the algorithm first defines the class of the incoming task, and compares the ratio of the virtual queue length to execution rate of that class on each pool of resources. Among all the pools in the system, the algorithm chooses the pool with the smallest ratio. As mentioned before, the algorithm keeps track of the load of each pool by using virtual queues. In fact, the virtual queue length provides an estimate of the (relative) time that the pool will be busy with executing previously assigned tasks. To summarize step 1, the algorithm trades off a small (virtual) queue length versus a fast execution rate. When the algorithm selects a pool for the new task, it adds the mean execution time of the task on the selected pool to the virtual queue of that pool. If the loads on faster pools of resources increase such that the proper balancing of loads is going to be violated, the total virtual queue length of all pools will reach a predefined limit. In this case, the virtual queue lengths of all pools are reduced by a prespecified amount. By doing this, the algorithm is making the virtual queue lengths of slower pools of resources smaller, and is increasing the chance of slower pools being chosen for executing future tasks.

We will see that a significant advantage of the Shadow Routing algorithm is in properly balancing the load of the system without requiring any state information. However, the basic Shadow Routing algorithm is defined on a workload model which is not applicable in Grid systems. In this work, we modify the structure of the Shadow Routing algorithm, and introduce a new scheduling algorithm which is applicable for typical Grid workloads. Also, we change the basic Shadow Routing algorithm in a way that leads to significant performance improvement.

V. Grid Shadow Routing Algorithm

In this Section, we introduce a new Grid scheduling algorithm, called the *Grid Shadow Routing algorithm (Grid Shadow)*, which is presented in Figure 2. In order to apply the idea of the Shadow Routing algorithm in Grid systems, first we should eliminate the class-based setting of the basic Shadow Routing algorithm. As we know, a Grid is a dynamic system in which the resources may join and leave at any time, and various types of tasks may be assigned to the system. So, it is unrealistic to assume that we have predefined types (class) of tasks, and that the execution rate of each class on each resource is known.

Instead of using the workload model of the basic Shadow Routing algorithm, we consider each task separately. We introduce our algorithm based on the typical Grid workload model defined in Section II. As mentioned before, various estimation methods have been introduced in the literature to provide estimates of task lengths and resource execution rates (see [5, 6] for example). Rather than going into detail on any particular estimation method, we simply assume that such estimates have been provided, with associated errors. By applying these two parameters, we estimate the expected execution time of task k on resource r by $\frac{\hat{L}_k}{\hat{\mu}_r}$, where $\hat{L}_k$ and $\hat{\mu}_r$ are estimates of the length of task k and the execution rate of resource r, respectively. Also, we assume that each resource has a virtual queue (Q_r), which is used to estimate loads on resources, and the parameter $\eta > 0$ is used to control the tradeoff between responsiveness of the algorithm and its accuracy. The required parameters of the Grid Shadow Routing algorithm are: 1) estimated incoming task length, and 2) estimated mean resource execution rate. To achieve these parameters, it is not required to contact any of the resources at the time of scheduling.

```
• For all resources in the system, set the initial values of $Q_r$ to $\eta Q_r = 1/M$ for all $r$.

• Upon the arrival of task $k$, the algorithm performs the following steps :

    1) Among all the resources available in the system (i.e. those with $\hat{\mu}_r > 0$), choose the resource $m$, such that :
           $m \in \arg\min_r \{(Q_r + \frac{\hat{L}_k}{\hat{\mu}_r}) \times \frac{\hat{L}_k}{\hat{\mu}_r}\}$
       and submit the arriving task to resource $m$.

    2) The virtual queue of resource $m$ is updated to :
           $Q_m = Q_m + \frac{\hat{L}_k}{\hat{\mu}_m}$

    3) If condition
           $\eta \sum_r Q_r \geq 1$
       holds, the following update is performed:
        for each resource $r$:
            If $(Q_r - c_k > 0)$ then
                $Q_r = Q_r - c_k$
            else
                $Q_r = 0$
       where $c_k = 2 \times (\max\{\frac{\hat{L}_k}{\hat{\mu}_r} \mid \mu_r > 0\})$.
```

Figure 2. The Grid Shadow Routing Algorithm

In Grid scheduling, taking into account the load that an incoming task adds to each resource, is important when the resources are heterogeneous, and the load of the system is moderate or light. The basic Shadow Routing algorithm just considers the current load of each resource in making the scheduling decision for each incoming task. So, we changed the first step of the basic Shadow Routing algorithm to make our scheduling decision also consider the expected load of the incoming task on each resource. In our scheduling decision, instead of comparing the current loads, we consider the current size of the virtual queue plus the expected load of the incoming task on the corresponding resource. Another way to look at this is that from the analytic perspective, if the load on the system approaches 1 (as in [4]), then the effect of the incoming task is negligible. This may not be true in practice, and should be accounted for.

As mentioned, this modification is a significant improvement to the basic Shadow Routing algorithm, in particular when loads are moderate or light, and the system is heterogeneous. For each incoming task, we aim to increase the total amount in the virtual queues by the minimum possible amount, then the normalization step will be triggered less frequently. This results in less overhead due to scheduling decisions, which improves the performance of the basic Shadow Routing algorithm significantly. Science Grid systems have a large number of resources, and step 3 of the basic Shadow Routing algorithm requires searching over all resources and updating all of their virtual queues. If we modify the algorithm in a way that reduces the number of times step 3 is triggered, we will reduce the overhead of the algorithm and improve the performance of the Grid system.

When a new task arrives to the system, our new scheduling algorithm considers three factors for choosing a resource: the current load on each resource, estimation of the execution time of the arriving task on each resource, and how much load the incoming task is going to add to each resource. So, we define the first step of our algorithm as follows: the algorithm compares the quantity $((Q_r + \frac{\hat{L}_k}{\hat{\mu}_r}) \times \frac{\hat{L}_k}{\hat{\mu}_r})$ for all resources, and chooses the resource with the smallest value. So, the algorithm has a trade-off between a resource which finishes the currently assigned tasks earlier, and a resource which executes the incoming task faster. Also, it aims to minimize the load which is going to be added to each resource. When the algorithm selects a resource for the new task, it adds the estimated execution time of the task on the selected resource to the virtual queue of that resource, which is given by $\frac{\hat{L}_k}{\hat{\mu}_m}$ for task k on resource m.

If the loads on faster resources increase such that the proper balancing of loads is going to be violated, the total virtual queue length of all resources will reach a predefined limit. In this case, the virtual queue lengths of all resources are reduced by a specific amount. This is a normalization step of the algorithm, in which the virtual queues are reduced by the maximum load that the incoming task can add to the resources. By doing this, the algorithm is making the virtual queue length of slower resources smaller, and is increasing the chance of slower resources being chosen for executing future tasks.

The parameter η should be chosen based on the features of the system. If η is a large number, the algorithm tries to equally divide the loads, and reduces the impact of differing execution rates of resources. So, η should be a small number, and the smaller the parameter is chosen, the more accurate matching of resources to tasks would be. As a result, the loads would be balanced properly based on both the load and speed of each resource. However, the rate at which the algorithm adapts to changes in the system parameters is proportional to η (the smaller η, the slower the rate) and thus η cannot be chosen too small. For the workloads considered in this work, we conclude that a good value of η is $1/300$.

Since the Grid Shadow Routing algorithm makes each decision based on the values of virtual queues, if task input rates, or resource execution rates change in the system, no explicit detection of such an event (or any other input rate measurement/estimation) is necessary. The virtual queues automatically readjust and the algorithm starts routing along the new best matchings of resources to tasks.

VI. Experimental Set-up

We use simulation to evaluate the scheduling algorithms. This section gives details of the simulation toolkit used, the performance metrics applied, and the experimental set-up.

A. Simulation toolkit & Performance metrics

Simulation models were implemented with the Java package GridSim [9]. GridSim is a toolkit for modeling and simulation of Grid resources and application scheduling. It provides a comprehensive facility for the simulation of different types of resources, users, tasks, and schedulers.

Depending on the Grid scenario and applications run in the system, there exist different performance metrics for evaluating Grid scheduling algorithms. We use two of the most important performance metrics to evaluate the algorithms from different aspects. The metrics that we consider are: Makespan (the maximum completion time of all tasks), and Flowtime (the average completion time of all tasks).

B. Experimental Environment

We consider a Grid system consisting of 50 dedicated resources with different CPU speeds, working in parallel with an overall high load. To simulate a widely distributed Grid system, and because the bandwidth between elements of the system which are far from each other is low, we set the bandwidth inside the elements of the system to be 1 Gbps, and the bandwidth between the scheduler and each of the 50 resources to be 10 Mbps. The GridSim simulator calculates the network delay between the elements of the system, depending on the bandwidth of each network and the load on each network at any given time.

As mentioned before, our proposed algorithm is mostly advantageous for EGEE Grids, so we evaluate our algorithm in a real workload from the CERN Grid project. We use a workload from the Grid Workload Archive, in a typical Grid Workloads Format (GWF). This workload is collected from the LCG project. The LCG testbed represents the Large Hadron Collider (LHC) Computing Grid. We use the LCG trace, version 0.1 which is provided by the Grid Workloads Archive [7]. We use the first $20,000$ tasks in this trace for our experiment. We run our simulation until $20,000$ tasks arrive to the system and then turn off the arrival streams and wait until the system becomes empty.

Our algorithm uses estimates of the task lengths and resource execution rates. However, various estimation methods may have different levels of accuracy. So, to show that our algorithm can achieve significant performance gain even with considerable errors in estimates of parameters, we evaluate our algorithm in a system that has various levels of error in the estimated task lengths and resource execution rates. In order to completely study the robustness of our algorithm, we examine cases that have 0% to 40% error in our estimations; however, typically these errors are on the order of 10% [10]. We evaluate our proposed algorithm by considering the error model discussed in [11] for estimating task lengths and resource execution rates. Generally the two models of error in these estimates are:

- *Over and Under Estimation Error.* Consider actual task lengths and resource execution rates to be L_k for task k and μ_r for resource r, respectively. Let $\hat{L}_k$ denote the (corresponding) estimated task length, and $\hat{\mu}_r$ denote the estimated resource execution rate. In our simulations, $\hat{L}_k$ and $\hat{\mu}_r$ are obtained using the following relations: $\hat{L}_k = L_k \times (1+E_k)$ and $\hat{\mu}_r = \mu_r \times (1+E_r)$. Here, E_k and E_r are the errors for task lengths and resource execution rates, respectively, which are sampled from the uniform distribution $[-I, +I]$, and I is the maximum error.
- *Over Estimation Error.* The main error models are obtained using the relations $\hat{L}_k = L_k \times (1 + E'_k)$ and $\hat{\mu}_r = \mu_r \times (1 + E'_r)$. The parameters E'_k and E'_r are the errors for task lengths and resource execution rates, respectively, and are sampled from the uniform distribution $[0, +I]$ for over estimation, in which I is the maximum error. This model is used for systems which always over estimate the parameters (powers of resources are estimated to be the maximum amount without considering fluctuation of their power caused by increasing the load).

VII. Experimental Results

In this Section we consider the various error models. First, we consider an environment with over and under estimation errors in task lengths without any error in estimating resource execution rates. Then we evaluate our algorithm in a system with over and under estimation errors in both task lengths and resource execution rates. We repeat the simulation for these two cases with the over estimation error model.

A. Over and Under Estimation

In this part, we assume the over and under estimation error model applies. First, we evaluate our algorithm in an environment with accurate resource execution rates, and errors in estimating task lengths, whose results are provided in Figures 3 and 4, from the Flowtime and Makespan perspectives, respectively. Then, we consider an environment with errors in estimating both task lengths and resource execution rates, whose results are provided in Figures 5 and 6. Different error levels are considered in these Figures.

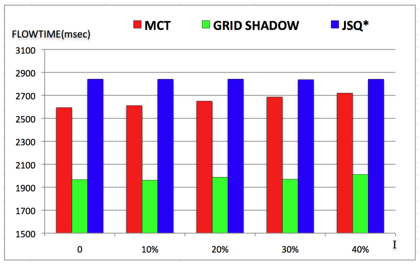

Figure 3. Flowtime-over & under estimating task length

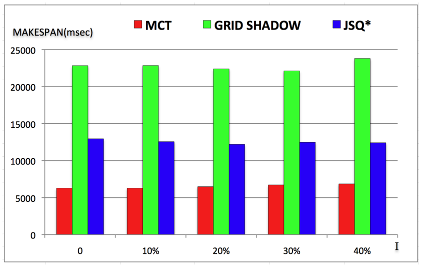

Figure 4. Makespan-over & under estimating task length

Among the algorithms presented in this work, the MCT algorithm is the only one that uses full state information in order to make scheduling decisions. So, we expect that in the absence of overhead, this algorithm should achieve the smallest Makespan and Flowtime, and should lead to a good balance between the loads of resources. Since our simulations consider a highly loaded system, in which gathering full state information causes large overhead, the MCT algorithm can not achieve good Flowtime compared to our proposed algorithm. However, the MCT algorithm does achieve the best Makespan by minimizing the completion time for each individual incoming task. Generally, minimizing Flowtime can be at the expense of the largest task taking a long time, whereas minimizing Makespan asks that no task takes too long, at the expense of most tasks generally taking a longer time. In summary, minimization of Makespan can result in maximization of Flowtime. By considering this fact, and since the MCT algorithm uses the greedy approach of minimizing the completion time for each individual task, it does not perform well for the average completion time of all tasks. So, the poor Flowtime performance of the MCT algorithm results from the combination of its high overhead and its greedy approach in minimizing the completion time for each individual task.

As mentioned before, our proposed Grid Shadow Routing algorithm does not use any state information; one might

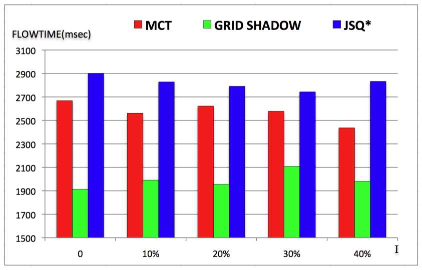

Figure 5. Flowtime-over & under estimating task length and resource rate

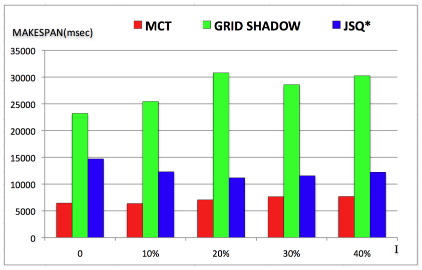

Figure 6. Makespan-over & under estimating task length & resource rate

expect a large difference between the performance of our algorithm and the MCT algorithm. However, the results illustrate that our algorithm has much better Flowtime than the MCT algorithm. This is due to the fact that the Grid Shadow Routing algorithm does not use the greedy view point of the MCT algorithm in optimizing the completion time for any single task. Instead, our proposed algorithm considers a long term approach for minimizing the completion times, and balancing the loads in the system, so it can achieve good performance for aggregated metrics like Flowtime. Another reason for better Flowtime of our algorithm compared to the MCT algorithm is the large overhead of the MCT algorithm in gathering full state information from the system, while our proposed algorithm has no such overhead. As discussed before, minimizing Flowtime can result in maximizing the Makespan. As the Grid Shadow Routing algorithm does not have the goal of minimizing the completion time for each individual task, and it considers overall balancing of loads, this can increase the completion time for a small number of tasks, which results in larger Makespans for the Grid Shadow Routing algorithm. Still, its Makespan is competitive with the JSQ* algorithm. As discussed before, our algorithm is most useful for EGEE like Grid systems in which gathering full state information for scheduling each incoming task causes significant overhead for the system. We believe that in these Grid systems the average completion time (Flowtime) (which is interpreted as QoS [12]) of the system is more important than maximum completion time of tasks (Makespan) (which is interpreted as throughput of the system). According to the results, even in systems which have large estimation errors, our proposed Grid Shadow Routing algorithm still has much better Flowtime than the MCT algorithm. This result is the consequence of the Grid Shadow Routing approach in properly balancing the loads in all resources based on their estimated loads and execution rates.

B. *Over Estimation*

In this part, we assume the over estimation error model applies. First, we evaluate our algorithms in an environment with accurate resource execution rates, but with various error levels in estimating task length, whose results are provided in Figures 7 and 9, from the Flowtime and Makespan perspectives, respectively. Then, we consider an environment with error in both task length and resource execution rate estimation. Figures 8 and 10 compare the Flowtime and Makespan of the scheduling algorithms in an environment with errors in all estimates. Different error levels are considered in these Figures.

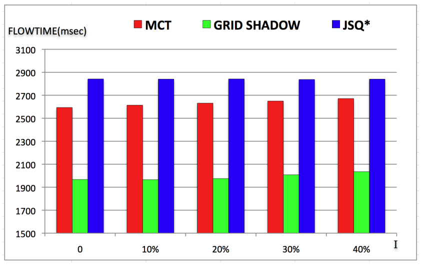

Figure 7. Flowtime-over estimating task length

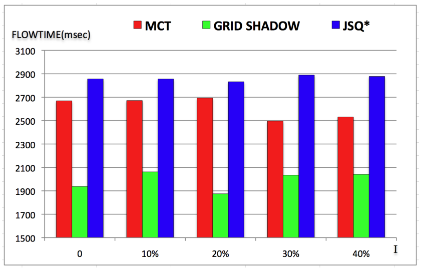

Figure 8. Flowtime-over estimating task length and resource rate

Based on the results, up to 40 percent over estimation does not significantly affect performance of our algorithm. The Grid Shadow Routing algorithm has the best Flowtime and the MCT algorithm has the best Makespan.

To summarize the observations in this section, in a real Grid workload, the Grid Shadow Routing algorithm has much better Flowtime than the MCT algorithm, and the Grid Shadow Routing algorithm achieves this performance without collecting any state information. The MCT algorithm achieves the best Makespan at the cost of collecting full state information. So, our Grid Shadow Routing algorithm is a promising candidate for widely distributed, and highly loaded Grid systems.

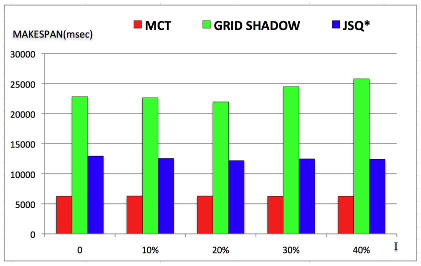

Figure 9. Makespan-over estimating task length

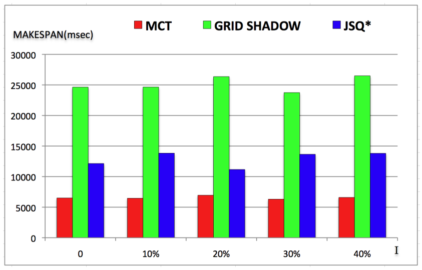

Figure 10. Makespan-over estimating task length and resource rate

VIII. Conclusion and Future Work

We introduced a new scheduling algorithm for Grid systems, called the Grid Shadow Routing algorithm. We evaluated the performance of this algorithm by simulation, and the results show the promising performance of our algorithm for aggregate measures such as Flowtime. Our introduced algorithm and the MCT algorithm use the same system parameters, but they differ in the amount of dynamic state information used for scheduling, in which our algorithm requires *zero* state information. We conclude that in cases where the system elements are not tightly coupled, and the communication cost is considerable, applying the Grid Shadow Routing algorithm is highly recommended over the MCT algorithm.

As potential future work, we propose making modifications in the Grid Shadow Routing algorithm to also consider data intensive applications. This would require refining the relations in steps 1 and 2 of the algorithm, to minimize the data transfer and storage costs. So, the algorithm finds the best matchings of resources and tasks such that it reduces both the costs of data transfer and storage, in addition to what it currently considers.

IX. Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada. The LCG Grid traces are provided by the HEP e-Science group at Imperial College London.

References

[1] I. T. Foster and C. Kesselman, "Computational grids," in *Proceedings of the 4th International Conference on Vector and Parallel Processing*. Springer, 2000, pp. 3–37.

[2] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids," in *Proceedings of the 8th IEEE International Conference on Advanced Computing and Communications*, Tata McGraw -Hill, India, 2000, pp. 45–52.

[3] L. E and B. Jones. (2009) Enabling grids for e-science: The egee project, egee-pub- 2009-001. [Online]. Available: http://www.eu-egee.org/

[4] A. L. Stolyar and T. Tezcan, "Control of systems with flexible multi-server pools: a shadow routing approach," 2009, Bell Labs Technical Memo, revised.

[5] null Yuanyuan Zhang, null Wei Sun, and null Yasushi Inoguchi, "Predicting running time of grid tasks based on cpu load predictions," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (Grid06)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 286–292.

[6] D. Lu, H. Sheng, and P. Dinda, "Size-based scheduling policies with inaccurate scheduling information," in *Proceedings of the 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 31–38.

[7] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, and et al. (2006, Nov) The Grid Workloads Archive. http://gwa.st.ewi.tudelft.nl/.

[8] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queens University, Kingston, Ontario, Canada, Tech. Rep. 504, 2006.

[9] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[10] S. Akioka and Y. Muraoka, "Extended forecast of cpu and network load on computational grid," in *Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid(CCGrid'04)*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 765–772.

[11] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-oftasks in large-scale distributed systems," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, 2008, pp. 97–108.

[12] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the 8th Heterogeneous Computing Workshop*. Los Alamitos, CA, USA: IEEE Computer Society, 1999, p. 30.