Double Auction-Inspired Meta-Scheduling of Parallel Applications on Global Grids

Saurabh Kumar Garg[1], *Srikumar Venugopal*[2], *James Broberg*[1] AND *Rajkumar Buyya*[1]

[1]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory,*
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia,
[2]*School of Computer Science and Engineering*
The University of New South Wales, Sydney, Australia
Email: {sgarg, brobergj, raj}@csse.unimelb.edu.au

Meta-schedulers map jobs to computational resources that are part of a grid, such as clusters, that in turn have their own local job schedulers. Existing Grid meta-schedulers either target system-centric metrics, such as utilization and throughput, or prioritize applications based on utility metrics provided by the users. The system-centric approach gives less importance to users' individual utility, while the user-centric approach may have adverse effects such as poor system performance and unfair treatment of users. Therefore, this paper proposes a novel meta-scheduler, based on the well-known double auction mechanism that aims to satisfy users' service requirements as well as ensure balanced utilisation of resources across a grid. We have designed valuation metrics that commodify both the complex resource requirements of user applications and the capabilities of available computational resources. Through simulation using real traces, we compare our scheduling mechanism with other common mechanisms widely used by both existing market-based and traditional meta-schedulers. The results show that our meta-scheduling mechanism not only satisfies up to 15% more user requirements than others, but also improves system utilization through load balancing.

Keywords: Grid Computing, Resource Allocation, Meta-Scheduling, Auction

1. INTRODUCTION

Grids are composed of distributed high-performance commodity clusters and supercomputers managed by batch job schedulers such as Portable Batch Scheduler (PBS) [1]. These distributed resources in the production grids are mostly managed by meta-schedulers that interact with the local job schedulers at each resource site in a grid to determine the most appropriate resource for executing an application submitted by a user. Meta-scheduling is different from cluster-level scheduling as it involves matching of the multiple concurrent applications to different distributed resources rather than dispatching applications to individual cluster nodes within a single domain. Examples of such meta-schedulers include the Maui/Moab scheduling suite [2], gLite Workload Management System [3], and GridWay [4].

Whilst Grids have matured with respect to the integration of different components, users have also developed sophisticated Quality of Service (QoS) requirements for application execution, and are ready to compensate resource providers for delivering an agreed level of QoS. Two examples of such requirements are completing an application by a certain deadline and ensuring a minimum number of CPUs for executing an application. These QoS requirements increase the challenge of application scheduling due to a number of reasons. First, the requirements of different applications can conflict with one another, thereby rendering the system unable to satisfy all users. Second, deadline conditions and fixed requirements of CPUs can induce fragmentation in application queues, which reduces system utilization and leads to poor user satisfaction. Finally, a meta-scheduler not only has to take into account these problems, but also has to contend with the changing conditions of grid resources that are spread across different administrative domains.

Historically, meta-schedulers have given priority to improving system-centric performance metrics such as utilization, average load and turnaround time for user applications [5]. They were not designed to cater for the sophisticated QoS needs of an application, particularly when the demand for resources exceed the supply. In recent years, a number of researchers have explored applying well-known economic mechanisms such as markets to address user requirements in meta-scheduling [6][7][8]. In a grid with variable resource availability, it is difficult to determine accurate resource and application valuations to take advantage of market-based mechanisms. The parallel applications, that have rigid processor requirements, are not comparable to each other and cannot be commodified easily so as to be used in auction mechanisms. From the users' perspective, it is difficult to come up with a valuation that ensures that their application is provided with the required amount of resources and executed by the deadline. Therefore, we need scheduling mechanisms that not only ensure effective utilization of Grid resources, but also take into account users' interest, demand on resources, and allocate resources in a fair manner such that no application is starved.

In this paper, we present a novel grid meta-scheduling mechanism that takes inspiration from auction principles in allocating resources to parallel applications with competing QoS demands. The objective of our meta-scheduling mechanism is to satisfy the QoS requirements of the users as well as to ensure maximal utilisation of resources simultaneously, while avoiding starvation, and minimising the effect on other measures such as waiting time and slowdown. We have considered parallel applications having multiple communicating processes that are to be allocated to a rigid number of processors available from a single computational resource. We have designed valuation metrics that enable mapping of slots in different application queues at grid resources to multiple applications with fixed processor requirements. In this manner, resource shares are commodified so that the principles of auctions can be used in the design of the meta-scheduling mechanism to benefit both users and resource providers. Then, by using simulation on real workload traces of parallel applications from supercomputing centers, we show that our scheduler performs much better than classical scheduling mechanisms in similar working conditions.

The rest of the paper is structured as follows. Section 2 discusses related scheduling and economy-based resource management projects. Section 3 presents the system model, and the details of our scheduling mechanism are presented in Section 4. Sections 5 presents the experimental setup used for performance evaluation, and discusses the results. Finally, we conclude the paper and present future steps in this direction in Section 6.

2. RELATED WORK

The general problem of creating a schedule for a set of jobs to run on distributed resources is called *list scheduling* and is considered to be NP-complete [9]. The simplest strategy, First Come First Served (FCFS), handles jobs in the order of arrival at the scheduler and submits them to the first available resource. It is considered adequate for the most purposes, and in conjunction with gang scheduling and backfilling, able to effectively utilise resources [10][11]. Currently, meta-schedulers such as GridWay [4] and gLite Workload Management System [3] use FCFS in operation. Moab also has a FCFS batch scheduler with easy-backfilling policy [2]. Condor-G [12] uses either FCFS or matchmaking with priority sort as scheduling policies [13]. However, FCFS is a simple heuristics that can only be used for traditional system metrics such as system utilisation and application waiting time. Additionally, even though these schedulers give the administrator the capability to integrate any mechanism, they do not have mechanisms to handle conflicts between concurrent users with overlapping QoS requirements. Hence, we propose a new meta-scheduler that uses mechanisms from established areas such as economics to handle such requirements.

In recent years, many economy-based scheduling mechanisms have been proposed to handle concurrent user requirements. A framework for auction protocols is suggested by Chard et al. [Chard presented a novel architecture for a Virtual Organization (VO) based distributed economic metascheduler in which members of the VO collaboratively allocated Grid resources in

REXEC [14] and Tycoon [7] are proportional share systems in which a task is allocated a share of the resource depending on the proportion of its bid (price) to the total sum of the bids of all tasks executing on that server. Wieczorek et al. [15] proposed a Grid resource allocation model based on Continuous Double Auctions (CDAs) to schedule workflow applications on Grid resources. Pourebrahimi et al. [16] presented some research on applying CDAs in resource allocation on the Grid. They proposed an agent-based Grid model encompassing three types of agents: buyer, seller, and auctioneer agent. They introduced four tunable parameters which can be used to modify different behavior of Grid participants. Kant et al. [17] proposed and compared various types of double auctions to investigate its efficiency for resource allocation in Grid. In addition, Tan et al. [18] proposed the stable CDA to overcome the problem of high volatility. Vanmechelen, et al. [19] developed centralised and decentralised algorithms for economic resource management using futures market to maximize realized consumer value. DRIVE [20] is a meta-scheduling framework to enable secure auction in Grid environment. LibraSLA [21] prioritises users on the basis of application deadlines and the user-specified penalties for not meeting them. Bellagio [6] is an auction-based system that seeks to allocate resources for distributed computing infrastructure in an economically efficient fashion to maximise aggregate end-user utility.

Aforementioned market-based systems and mechanisms primarily aim either to improve the profitability and utilisation of the resource providers or utility satisfaction of the users, but not both at the same time. These systems use only the application valuations provided by the users. However, users cannot be expected to provide accurate valuations as they lack complete information about resource availability in a dynamic environment such as Grids [8]. As mentioned before, the problem is also to design valuation metrics that commoditise resource requirements and availability so as to take advantage of the efficiency of market-based mechanisms [22]. Moreover, the application of the two sided market based mechanism, such as double auction, to computational resource allocation has focused mainly on those cases where services can be easily traded as commodities, which is clearly not the case in Grids where applications can require simultaneously multiple resources which are difficult to commoditise. Therefore, a variant of the double auction is designed in our context.

Xiao et al. [23] present an incentive-based scheduling scheme, which employs a peer-to-peer decentralised scheduling framework, to maximise the success rate of application which require one processor for execution, and minimise fairness deviation among resource providers. However, the incentive-mechanisms in this case is primarily profit-based which tries to ensure that each provider has an equal chance of attracting jobs. However, there is a need to ensure that system-centric objectives such as minimising waiting time for applications are met as well.

Scheduling of non-malleable parallel jobs [24] on grid resource sites is still in its infancy. Most of research in this area has been concentrated on single supercomputing systems [25]. A survey of scheduling parallel jobs on the computational Grid by Feitelson *et al.* [26] classify the scheduling on the Grid into single-site (non-co-allocation) and multi-site (co-allocation). Our work focuses on scheduling rigid single-site jobs on Grids. Many Genetic Algorithm (GA)-based solutions have been proposed to improve the performance of parallel job scheduling mechanisms [27, 28, 29, 30]. As GAs require a long time to execute, they are not suitable for a dynamic environment such as Grids where schedules have to be recomputed regularly since resource availability changes rapidly. Abawajy *et. al* [31] presented an online dynamic scheduling policy that manages multiple job streams across both single and multiple cluster computing systems with the objectives of improving the mean response time and system utilization. This work assumes that parallel jobs are moldable and thus the number of processors can be changed. Sabin *et.al* [32] presented an algorithm to optimize turnaround time and utilization by submitting jobs simultaneously on multiple heterogeneous resources. Similar model is considered by Tchernykh *et. al* [33] who theoretically analyzed the meta-scheduling problem for multi-site environments. The designed scheduling policies in these works are for multi-cluster environments which are generally under single administrative domain but our work is for the Grids that extend across multiple administrative domains. Moreover, none of these works considered user QoS requirements such as deadline while scheduling parallel jobs.

In our previous publication [34], we have presented a double auction-based meta-scheduling mechanism to increase fairness and user satisfaction for the Bag-of-Task applications. This paper extends the application model to parallel applications with rigid processor requirements. In this paper, therefore, we focus on designing a meta-scheduling mechanism using auction principles for parallel applications having rigid processor requirements to benefit both the user, by aiming to meet the QoS requirements of applications, and the resources, by balancing load across them. The scheduling of such applications on Grid resources is a complex $0-1$ Knapsack Problem that is more challenging than traditional scheduling on parallel systems due to: fixed number of processors required by the application; dynamic availability of resources with different capabilities in different administrative domains; and continuous arrival of applications at the meta-scheduler [35].

Hence, the **contributions of this paper** are: 1) a meta-scheduling mechanism for parallel applications that takes advantage of both auctions and system-based schedulers to meet the needs of users as well as balance load across resources to ensure effective utilisation, 2) A new valuation metrics for modeling both user and resource requirements. The valuation metrics commodify both the available resource share and the users' application requirements such that they can be compared and matched using principles from double auctions and 3) Detailed performance analysis of meta-scheduling algorithms using different performance metrics.

3. SYSTEM COMPONENTS

The meta-scheduler, considered in this work, follows the model commonly found in large computing installations across educational and research institutions [36] as shown in Figure 1. In this model, resources are managed at different sites by administrators who have to cater for their local users' needs. Batch scheduling systems that manage these resources are generally organised as a collection of user-accessible job queues where a queue may only allow the submission of specific applications that meet certain criteria (e.g. within a maximum application size) [37]. The resource

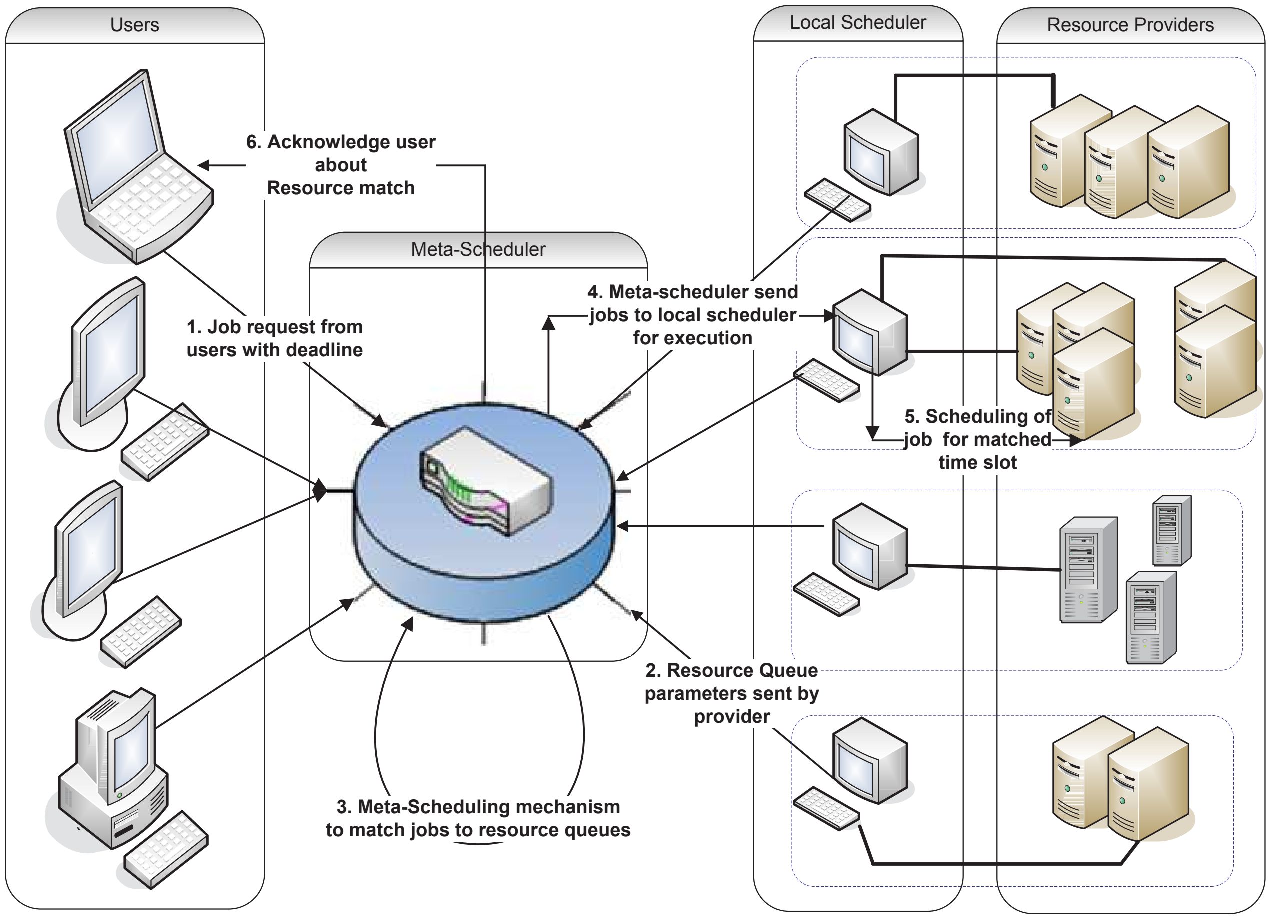

Users
Local Scheduler
Resource Providers
Meta-Scheduler
6. Acknowledge user about Resource match
1. Job request from users with deadline
4. Meta-scheduler send jobs to local scheduler for execution
5. Scheduling of job for matched time slot
2. Resource Queue parameters sent by provider
3. Meta-Scheduling mechanism to match jobs to resource queues

FIGURE 1. Interactions between a meta-scheduler, users and local schedulers at the resource sites

management system (local scheduler) at a site may employ policies such as easy or conservative backfilling in order to improve the utilisation and responsiveness for small applications [38]. The pre-emption of executing applications may not be allowed. The meta-scheduler uses the information supplied by providers and users to match applications to the appropriate queues on the resources. The meta-scheduler runs the scheduling algorithm at periodic intervals to satisfy both the users' and the resource providers' objectives. It may have control over allocation to some or all of the processors in a resource or may only be allowed to access certain queues within a resource. After matching applications to resource queues, the meta-scheduler transfers the user applications to local schedulers of the resource site for execution. Therefore, other than the meta-scheduler, there are two principal participants in this system, namely, the resource sites and the users.

- **Resource Sites:** We consider a grid with m resource sites, $R_1, R_2...R_m$ with k job queues. Resource sites supply information about available slots, load and waiting times of each queue to the meta-scheduler at regular intervals. A *slot* is a unit of resource allocation which is described by a start time, a finish time, and the number of processors available for that duration. A resource site also supplies an initial valuation for running an application in its queue slots. This initial valuation may be based on the processors provided to the queue and the load expected to be generated on the resource by the jobs in that queue. The objective of the meta-scheduler is to distribute jobs across all the resource sites to ensure effective utilisation through load balancing, and minimal slowdown and waiting time for applications.
- **User:** In this work, we consider the user application to follow a compute-intensive parallel application model composed of multiple communicating processes. An application has a rigid number of processor requirements that need to be satisfied at a single resource site. Most parallel applications are of this nature, as they are sensitive to latency of message passing, unless they have been explicitly designed to be executed across multiple resources. The objective of the users is to have their applications completed by a deadline. It is assumed that deadlines are hard, i.e. a user will benefit only if his/her application is executed by its deadline.

Users will also provide an initial valuation of the application to the meta-scheduler. This valuation can be based on the importance of the application to the user. The estimated execution time of an application by the user is considered to be accurate [39] in order to facilitate comparison between the algorithms proposed in this paper and those cited in Section 2 .

4. DOUBLE AUCTION-INSPIRED META-SCHEDULER (DAM)

The **D**ouble **A**uction-inspired **M**eta-scheduling process proposed in this paper, hereafter referred to as DAM, is a sequence of three broad stages as shown in Figure 2. The first stage (*collection*) consists of gathering information about resources and applications such as queue slot availability and waiting time for the former, and processor and QoS requirements for the latter. In the next stage (*valuation*), the valuations are computed for all applications and resources by the meta-scheduler. It is important to note that the valuation is private to the meta-scheduler and hidden from both users and resource providers. Finally, the last stage of scheduling is to perform *matching* of user applications to the available resources based on these valuations. Those applications that are not matched are then retained at the meta-scheduler. In the next scheduling cycle, resources and application valuations are recomputed using new information and the matching is carried out again.

As described in Section 2, auction-based mechanisms have been the subject of many previous studies. Grosu *et al.* [40] have compared resource allocation protocols using First-Price, Second-Price Vickrey and Double Auction (DA), and have concluded that DA favours both users and resources, while the first-price auction is biased towards resources and the Vickrey auction towards users. Therefore, we have opted to use principles of DA, also known as Call auction, within the meta-scheduler.

4.1. Call Auction

In a typical Call auction, sellers and buyers submit offers (*asks*:$a_j, 1 \leq j \leq m$) and requests (*bids*:$b_i, 1 \leq i \leq n$) respectively to an auctioneer who continually ranks them from highest to lowest in order to generate demand and supply profiles. The ordering of asks and bids after sorting is the following:

$$a_1 < a_2 < \ldots a_j \ldots < a_m$$
$$b_1 > b_2 > \ldots b_i \ldots > b_n$$

A seller is allowed to participate in the match process if $a_m < b_n$. From the profiles, the maximum quantity exchanged can be determined by matching asks, starting with the lowest price and moving up, with the bids, starting with the highest price and moving down. This format allows buyers to make offers, and sellers to accept those offers at any particular moment. After matching process, the auctioneer decides the amount of payment received by the provider from the user based on the ask and bid values.

Call auctions present an efficient framework for general resource allocation, especially if carried out over a long period of time. However, the meta-scheduler in this paper carries out the matching internally without any explicit involvement of buyers and sellers. Also, the applications considered for scheduling are different enough, that they cannot be commoditised and compared using single values. The same holds for resources as well. Therefore, call auctions cannot be directly applied to this scenario. However, we have designed a valuation mechanism that allows us to take advantage of the efficiency of call auctions for our matching process.

4.2. Valuation Mechanism

In call auctions, the maximum bid is matched to the minimum ask. In the meta-scheduler, the application that has the earliest deadline (and thus, the most urgent) must be matched to the fastest queue that has enough processors available. Therefore, we have constructed our valuation mechanism such that the resource valuations are the asks and the application or job valuations are the bids. The valuations of the resources and the applications depend on many factors such as supply and demand, resource loads and user deadlines. We have to reconcile these multiple measures to a single value that can be used for comparison of valuations. To this end, we apply Multi-Attribute Utility Theory (MAUT) [41][42][43], which provides a logical, consistent and tractable approach for quantifying an individual's preferences by consolidating them into a single objective. This theory first addresses the identification of attributes and the desirability function for each attribute and, then the aggregation of these desirability functions into a single scalar utility value.

Resource Valuation (Ask) Batch job schedulers that manage resources are generally constructed as sets of queues with different attributes. The load of a queue is defined as the ratio of the number of processors occupied by jobs to the total number of processors available to the queue. In order to balance load across independent grid sites, the meta-scheduler tries to give preference to the least loaded queues while submitting applications. Also, the most urgent application must be matched to the fastest queue. Hence, the valuation metric of resource queues should be such that the queue with the least waiting time should have the least ask value.

The valuation metric should also take into account the initial valuation given by the resource provider, and also the demand and supply levels of resources in the

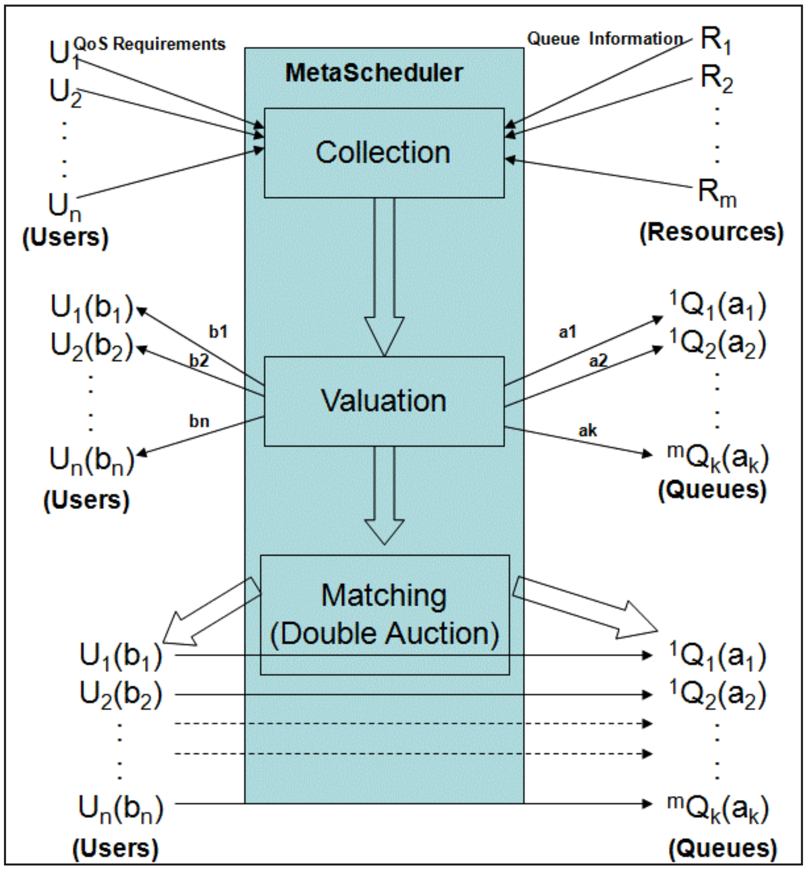

QoS Requirements
Queue Information
MetaScheduler
Collection
Valuation
Matching (Double Auction)
U_1 U_2 U_n (Users)
R_1 R_2 R_m (Resources)
$U_1(b_1)$ $U_2(b_2)$ $U_n(b_n)$ (Users)
b1 b2 bn
a1 a2 ak
${}^1Q_1(a_1)$ ${}^1Q_2(a_2)$ ${}^mQ_k(a_k)$ (Queues)

FIGURE 2. Auction-based Meta-Scheduling

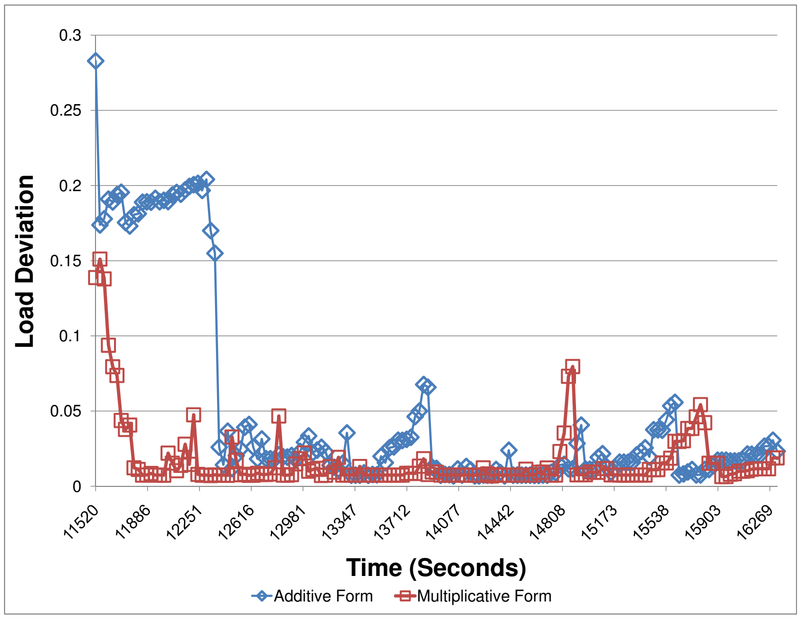


FIGURE 3. Comparison of Multiplicative and Additive forms of Valuation Metrics

system (denoted as *Demand* and *Supply*). *Demand* is the sum of the processors required by applications to be allocated and *Supply* is the total number of processors available at all resources. Let $l_{k,t}$ be the load on the resource queue k at time t. Let $c_{k,t}$ be the initial resource valuation given by the provider. Let $w_{k,t}$ be the application waiting time for queue k at time t. Thus, the desirability functions are proportional to $w_{k,t}$, $c_{k,t}$, *Demand*, *Supply*, and $l_{k,t}$. Since each of these attributes are preferentially independent [43], thus the valuation metric can be formed by aggregating these attributes either in multiplicative or additive form. When we compare the effects of each metric on distribution of load on the resource sites, we observed that in the case of additive form, the deviation in load[1] across the resource sites is much more than that for the multiplicative form as shown in Figure 3. In addition, when more than two attributes are involved, an accurate additive aggregation requires the exact trade-offs between attributes, which may not be apparent. Thus, we have used a multiplicative aggregation to form a valuation metric from normalised attributes, where $c_{max,t}$ is the maximum initial valuation given by a resource provider and $l_{max,t}$ is the maximum load on the resources. This valuation metric $a_k(t)$ of queue k at time t is given by following equation:

$$a_k(t) = O_k \times w_{k,t} \times \frac{c_{k,t}}{c_{max,t}} \times \frac{l_{k,t}}{l_{max,t}} \times \frac{Demand}{Supply}, \quad (1)$$

where O_k is a proportionality constant which is taken to be 1 for simulation purposes.

Job Valuation (Bid) Similar to resources, a user application has also many attributes such as the number of CPUs required, deadline and run time. The valuation metric of application i at time t is designed in such a way that the maximum value is assigned to applications with urgent deadline. Also, if an application has not been mapped in the previous scheduling cycle, its corresponding bid (valuation) should be increased. This is to reduce the possibility of this application getting starved of resources by others with higher valuations (bids) that have arrived at the meta-scheduler in the meanwhile. The $(d_i - t)$ indicates how urgently user want application i to be executed, where d_i is the user-supplied deadline for application i. Let st_i be the submission time of the application. Similar to that for resources, the application metric should also take into account the initial valuation of the application given by the user, and also demand and supply levels of resources in the system (denoted as *Demand* and *Supply*). Let v_i be the initial application valuation given by the user. Thus, the desirability functions are proportional to v_i, st_i, *Demand*, *Supply*, and $(d_i - t)$. Let v_{max}, d_{max} and st_{min} be the highest initial valuation, largest deadline and earliest start time, respectively. The resultant valuation metric for application i at time t, formed by multiplying all the normalized attributes, is the following:

$$b_i(t) = H_i \times \frac{v_i}{v_{max}} \times \frac{d_{max} - t}{d_i - t} \times \frac{Demand}{Supply} \times \frac{(t + 1 - st_i)}{t + 1 - st_{min}}, \quad (2)$$

where H_i is a proportionality constant which is taken to be 1 for simulation purposes, and $d_i \neq t$.

4.3. The Meta-Scheduling Algorithm

As noted in the previous sections, the commodity unit, matched on behalf of the resource site, is a slot where a set of processors are bounded by start and finish times. Figure 4 demonstrates the method using which the slots can be generated within a Grid site. At time t, the local job scheduler can provide the time slots that are currently free, so applications can be scheduled by the meta-scheduler to fill up the queue to a specific time horizon T_H. The slots start from the first available time and contain as many processors that are not occupied for a specific duration. In Figure 4, the slots S_1 to S_5 are examples of such free slots. The list of available slots can be generated by the local job scheduler using the estimated execution time of the applications. As shown in Figure 4, after current time t, two processors are free in queue Q_3 upto time T_H since jobs $J6$ and $J7$ are estimated to finish before time t. S_5 is therefore, an available time slot with available $time = T_H - t$. In the case of Queue Q_2, jobs $J4$ and $J3$ are finishing after time t and also estimated to complete at different times. Thus, the simultaneously free processors after time t result in two time slots S_2 and S_4, each with different number of processors and available time unit. Similarly, another time slot S_1, consisting of one processor on the queue Q_1, is available as well. This approach, where slot can be of different sizes, was also used by Singh et. al [30]. In this case, the local scheduler at the resource site can use backfilling to minimise the fragmentation in the schedule such that execution of applications in the queue does not get delayed.

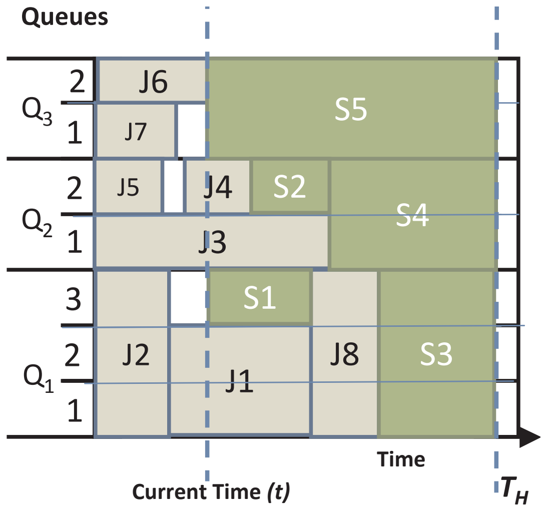


FIGURE 4. Available queue slots.

[1]The experiment is conducted with the same configurations as given in the Section 5. The"load deviation" is used as a metric for comparison which indicates the standard deviation of the resource load across the grid from average.

```
1  while $current\_time < next\_schedule\_time$ do
2      RecvResourcePublish(P);
       // P contains information about providers
3      RecvJobQoS(Q) ;
       // Q contains QoS information about users
4  Add information of pending applications from previous
   scheduling cycle to RecvJobQoS;
5  Calculate the Demand and Supply for resources;
6  Update the value of bids and asks using eqn. 2 and 1;
7  Sort asks in ascending order ;
8  Sort bids in descending order;
9  while no all applications are assigned to resource queues do
10     $i = 1$, $j = 1$;
11     if bid $b_i$ is greater than ask $a_j$ then
12         if $QueueWaitingTime(j) + ExecTime(i) < Deadline(i)$ then
13             if check processor availability on resource j then
14                 Schedule the application $i$ to the resource j;
15                 add application with matched resource site
                   in Schedule List (Schd_List) ;
16                 update the value of available time slots
                   from resource $j$;
17                 $i++$;
18
19         else
20             add user application to pending application list;
21     $j++$;
22 foreach $element \in Schd\_List$ do
23     notifyuser();
```

Algorithm 1: Double Auction-inspired Meta-scheduling in each scheduling cycle

The meta-scheduler schedules the applications in regular time intervals, therefore in the beginning of a scheduling cycle, it gets the updated available queue slots using the above approach from local job schedulers. Our proposed scheduling algorithm is shown in Algorithm 1. In each scheduling cycle, the meta-scheduler schedules the parallel applications after collecting all users' requests and resource performance information such as queue waiting times and free time slots (Line 1-3). At the end of each scheduling cycle, the meta-scheduler computes the demand for resources and their supply (Line 4). Then, based on the information collected from users and resources, the meta-scheduler assigns valuation to user applications (bids) and resource's queue (asks) using the valuation mechanisms presented in the previous section (Line 6).

From the sorted bid list, the bid (b_i) with the highest value will be matched to the resource queue with the minimum ask (a_j) that can satisfy the user's requirement. Whether user application i will be scheduled to resource queue j (corresponding to ask a_j) depends on the applications' processor and deadline requirements. Thus, the first deadline of application i is checked using waiting time of resource queue j (Line 12), and then processor availability is checked on resource queue j (Line 13). If there is an ask that satisfies the application's QoS requirements, then the bid is matched to ask; and the matched user and the resource provider are informed of the decision. The application i is then scheduled on to resource queue j (Line 14) and then added to the schedule list (Line 15). The available time slots (commodity units) on resource queue j are updated correspondingly (Line 16). If the deadline requirement of application i can be satisfied by resource queue j, then no other ask can be matched to the application's bid (Line 20). Therefore, the application is removed from the bids list in that scheduling cycle. If the required number of processors is not available on resource queue j, bid b_i is matched with the next ask in the sequence (Line 21). Matching for other bids and asks which are in pending list will be done in the next scheduling cycle with new requests. All the users whose applications have been scheduled are notified by the meta-scheduler about the matching (Line 23). In each scheduling cycle, the process is repeated with recalculation of ask and bid values until either all bids or asks are matched. It should be also noted that the valuation process is such that unmatched bids will get higher value in the next scheduling cycle to avoid starvation of jobs.

5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our proposed algorithm in various experimental scenarios. Therefore, we present the details of experiments conducted to compare the performance of Double Auction-Inspired Meta-Scheduler (DAM) with current state-of-art meta-scheduling algorithms and approximate theoretical model of the meta-scheduler.

5.1. Comparison of DAM with State-of-Art Meta-scheduling Algorithms

5.1.1. Experimental Configuration

For our experiments, Feitelson's Parallel Workload Archive (PWA) [35] is used to model the parallel application workload. Since this paper focuses on studying the HPC parallel applications of users, the PWA meets our objective by providing the necessary characteristics of real parallel applications collected from supercomputing centers. To avoid the effect of initial setup phase of the HPC center, our experiments utilize the traces from the second week of the Lawrence Livermore National Laboratory (LLNL) Thunder supercomputer (January 2007 to June 2007). The LLNL Thunder trace is chosen due to its highest resource utilization of 87.6% among available traces to ideally model a heavy workload scenario. From this trace, we obtain the submit time, requested number of processors, and actual run time of applications. The characteristics of the traces are given in Table 1. The submission time of a parallel application is divided by 1000 to increase the number of applications submitted per schedule interval as per the methodology presented by Feitelson and Wiseman [44]. Since the workload trace

TABLE 1. Workload Characteristics

Mean Inter-Arrival Time	16.176 sec.
Average Job Runtime	2328.911 sec.
Standard Deviation for Job Runtime	7790.11 sec.
Average CPU requirement	44 CPUs

does not contain any information about the user's deadline and initial valuation, these are generated synthetically. For a user application with a runtime r, the deadline is generated from a uniform random distribution between r and $3r$. The trace data of utility grid applications are currently not released and shared by any commercial utility grid providers, thus this information also has to be generated using a random distribution. The average initial valuation of applications is chosen randomly between 90000 and 160000 currency units using uniform distribution, so that it is always greater than application execution cost. The application execution cost can be computed by using resource_initial_value*execution_time*number_of_CPUs. The user valuations are assigned so that at least half of users can afford to execute their application on the resources with the highest valuation. The grid modelled in our simulation contains 10 resource sites spread across five countries derived from European Data Grid (EDG) testbed [45]. The configurations assigned to the resources in the testbed for the simulation are listed in Table 2. The configuration of each resource is decided such that the modelled testbed reflects the heterogeneity of platforms and capabilities that is normally the characteristic of such installations. Each of the resources are simulated as a cluster that employs a multi-partition Easy backfilling policy for local resource allocation [46].

The processors associated with each cluster in Table 2 are exclusively managed by the meta-scheduler (i.e. all users are going through meta-scheduler). We have sub-divided the allocated processors of each cluster into 3 queues in ratio of 1:2:3 of the total number of processors in the cluster. The processing capabilities of the processors are rated in terms of Million Instructions per second (MIPS) so that the application requirements can be modelled in Million Instructions (MI). An initial valuation, randomly computed between 4.5 and 9.5 currency units per processor per second, is assigned to each resource.

We have compared our double auction-inspired meta-scheduling algorithm against five other well-known traditional and market based algorithms listed below:

- **Shortest Job First (SJF):** In this algorithm, applications are prioritized on the basis of estimated runtime. This is a very common algorithm used in cluster management.
- **First Come First Serve (FCFS):** An application is assigned to the first available queue. This is a common algorithm employed by many meta-schedulers such as GridWay [4].
- **Earliest Deadline First-Fastest Queue (EDF-FQ):** In this algorithm, the applications with the earliest deadline are scheduled on to the resource queue slot with the least waiting time (Fastest Queue (FQ)).
- **Highest Valuation to Fastest Queue (HVFQ):** In this algorithm, the application with the highest user valuation is assigned to the queue slot with the least waiting time. This algorithm is generally used in auction mechanism such as Vickrey auction. Vickrey auction is used in resource management systems such as Spawn [47] and Bellagio [6].
- **FairShare or Proportional Share:** In this algorithm, each application is assigned queue slots proportional to the ratio of its valuation to the combined valuation of all the applications. This algorithm is employed in REXEC [14].

TABLE 2. Simulated EDG Testbed Resources

Site name (location)	**Number of processors**	**Single processor rating (MIPS)**
RAL (UK)	2050	1140
Imperial College (UK)	2600	1330
NorduGrid (Norway)	650	1176
NIKHEF (Netherlands)	540	1166
Lyon (France)	600	1320
Milano (Italy)	350	1000
Torina(Italy)	200	1330
Catania (Italy)	250	1200
Padova (Italy)	650	1000
Bologna (Italy)	1000	1140

The following criteria are used to compare fairness and user satisfaction provided by these algorithms:

- **Urgency vs. Success Ratio:** The user's urgency to get their application completed, is defined as:

$$u = \frac{deadline - start\_time}{execution\_time} - 1 \qquad (3)$$

  where $start\_time$ and $execution\_time$ are attributes of the application. The deadline is considered very urgent when $u < 0.25$, urgent when $0.25 < u < 0.5$, intermediate when $0.5 < u < 0.75$, relaxed when $0.75 < u < 1$ and very relaxed when $u > 1$. "Success Ratio" is ratio of number of applications finished successfully before deadline to the total number of applications in a particular urgency group. This criterion relates to how the scheduler deals with users with different demands on time.
- **Valuation vs. Success Ratio**: The valuation provided by the user for an application is divided by the required number of processors in order to normalize it. We examine how the schedulers

allocate resources fairly among different users with different application valuations. If ($u < 0$) for an application then the application will not be scheduled by the meta-scheduler.

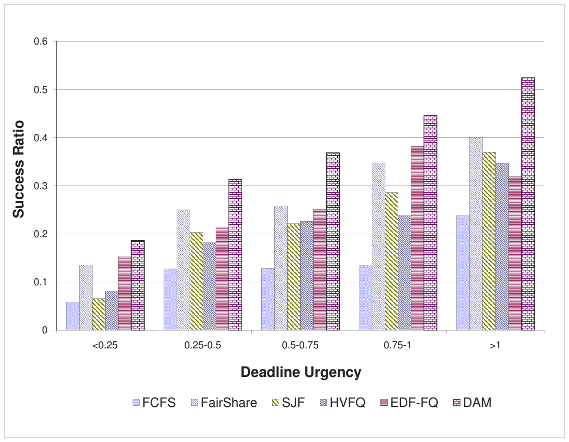
- **Number of deadlines missed** with increase in number of user applications. We use this criterion to examine how the scheduling algorithms are able to cope with user requests when demand for resources exceeds supply.
- **Load Deviation:** The load of a resource is the ratio of the number of processors occupied to total number of processors available at the resource site. We average the load over the grid resources and measure the standard deviation. This informs about how well the scheduling mechanism was able to balance load across the grid.

5.1.2. Analysis of Results

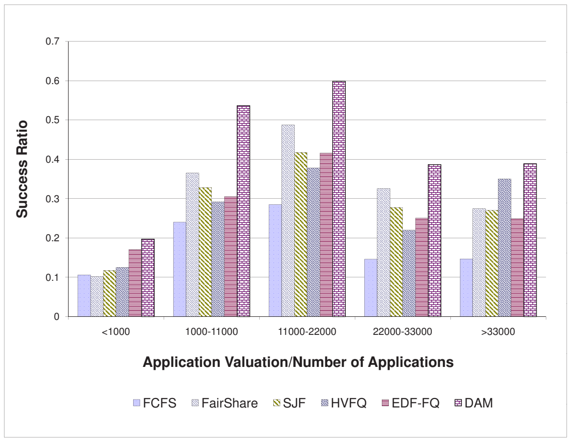
In this section, we discuss the results of our evaluation.

Benefit for Users: This section shows how our meta-scheduler benefits users by not only completing the most number of applications with different QoS needs but also benefiting every user in different urgency and budget groups.

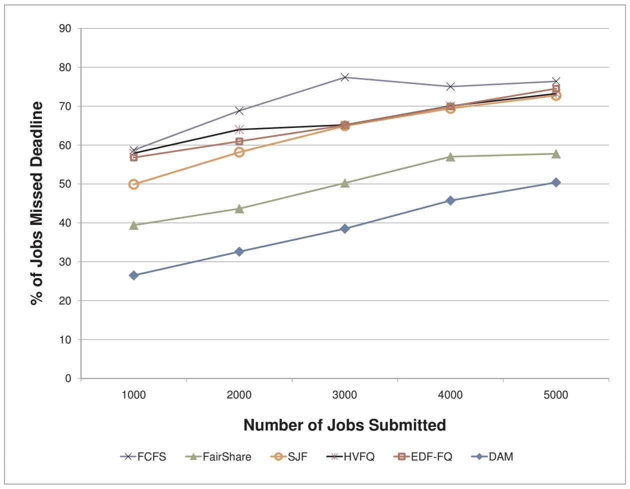
- **Effect of User Urgency:** Figure 5(a) shows the percentage of total applications completed successfully against the users' urgency values. Figure 5(a) shows that DAM has scheduled a larger number of applications than other algorithms in every urgency group. For example, in the intermediate group ($0.5 - 0.75$), DAM schedules 12% more applications than its closest competitor (FairShare). This is in contrast to the performance of FCFS and SJF which is the worst in almost every case. This is due to the fact that DAM is designed to increase an application's value with urgency, while in others this is not considered. The performance of FairShare is very close to DAM and it has scheduled an number of applications almost equal to that of DAM when deadline urgency is less than 0.25. This is because DAM tries to reduce the waiting time of applications with relaxed deadline by increasing their valuation. Thus, when the deadline urgency is greater than 1, then DAM schedules about 12% of more applications than FairShare. Jobs with relaxed deadlines progressively gain in valuation (or, float to the top of the bid list) when they are held at the scheduler over time in DAM, and are therefore not starved. This can be seen by comparing the performance of DAM with EDF-FQ, which prioritizes urgent applications but performs poorly with relaxed deadlines. Since the users' objective is to complete their applications by the deadline, delaying an application at the scheduler is appropriate as long as the deadline is met.
- **Effect of User Valuation:** From Figure 5(b), we can see that DAM completes more number of applications across all valuations than the other algorithms. Even though Fareshare again performed very close to DAM for medium valuation groups, DAM outperforms FairShare for all other groups by scheduling atleast 10% more applications. For applications with very low valuation (< 1000), the DAM has managed to schedule about 20% of the applications when compared to 11% for FairShare which performs as well as DAM, when the application valuations are medium. This is because the latter assigns the lowest proportion of resources to the users with the lowest valuation. Therefore, in this case, most of the parallel applications fail to execute due to lack of sufficient processors. It is also interesting to note that HVFQ, which is supposed to favour users with high budget, has scheduled almost 4% less number of applications than DAM for $Budget > 33000$. This is because HVFQ does not consider other requirements of applications such as deadline.
- **Number of deadlines missed:** From Figure 5(c), we can clearly see that as the demand for resources (number of applications) increases, the number of applications that missed their deadline also increases due to the scarcity of resources. But DAM is able to complete around 8% to 15% more applications than other algorithms. As SJF resulted in better packing of jobs at resource sites, SJF performs relatively better than the other algorithms such as EDF-FQ, HVFQ and FCFS. FCFS performs the worst as it does not consider the effect of deadlines and queue waiting times.
- **Variation in user's urgency and arrival rate:** We conducted this experiment to understand the effects of various valuation parameters on our algorithm's performance. Figure 6(a) and 6(b) shows how user valuation parameters, such as urgency and arrival rate, effect successful completion of applications. The deadlines of all users is varied from high to low which is calculated in terms of 'Urgency' level (Figure 6(a)). Thus, for a particular experimental scenario, all users have same urgency level. Similarly, to vary the submission time of jobs arriving for scheduling, job arrival rate is changed from 'very low' to 'very high'(Figure 6(b)).
  In Figure 6(a), the decrease in success ratio in case of all the algorithms is due to limited number of resources which cannot execute all the urgent jobs simultaneously. Similarly, with increase in job arrival rate, except DAM and FairShare, all the algorithms resulted in lower success ratio. It is clear from the experiments that schedules obtained from DAM always results in the highest successful completion of applications (almost 15% more) in all cases. It shows that by using our proposed meta-



(a) Effect of user urgency

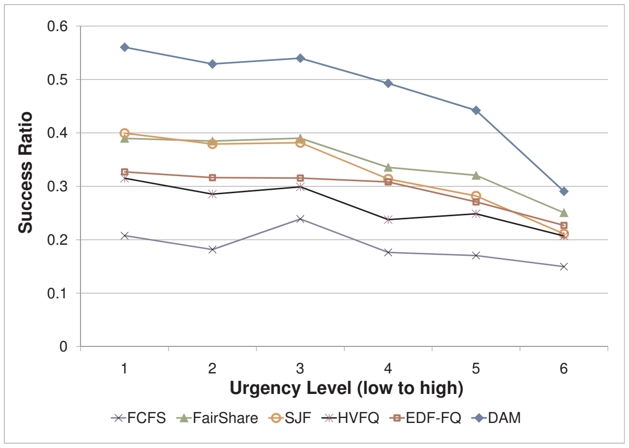


(b) Effect of user valuation

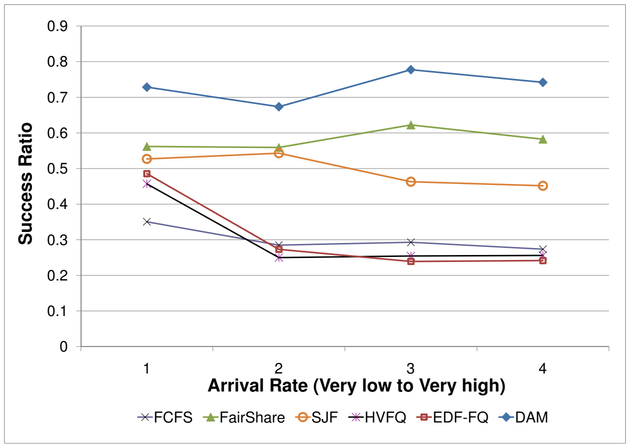


(c) Percentage of Deadlines Missed

FIGURE 5. Benefit for Users



(a) Effect of variation in urgency



(b) Effect of variation in arrival rate

FIGURE 6. Benefit for Users by Varing Valuation Parameters

scheduling algorithm, users have higher chance of job completion.

Benefit for Resources: Simulation results in Figure 7 show how DAM affects the load on different resources. The figure shows the standard deviation in resource loads against the time period of the execution. It can

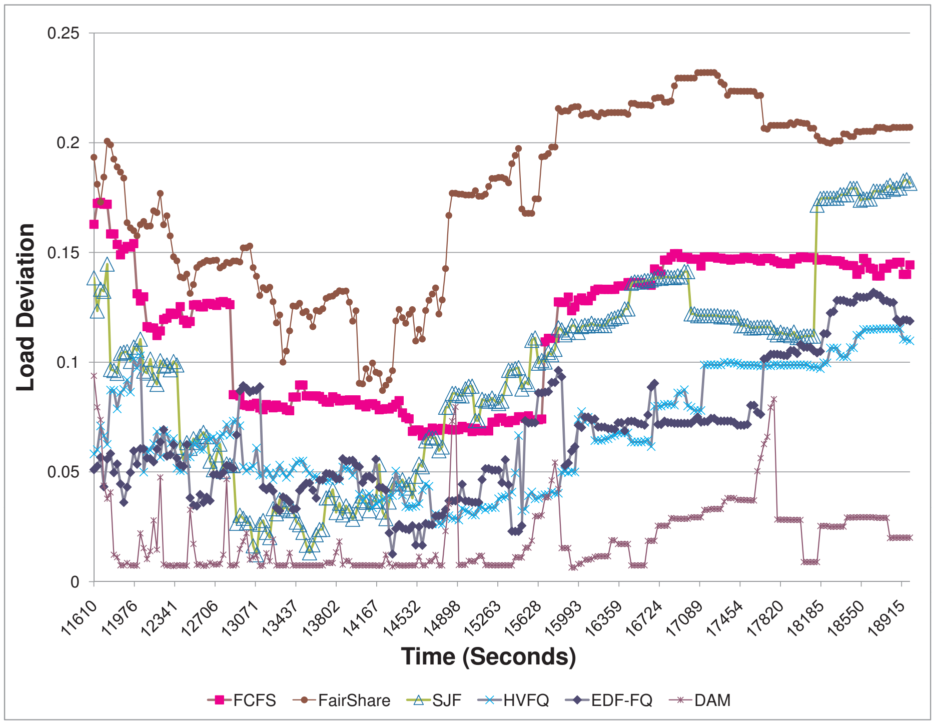


FIGURE 7. Variation in Load across Resources.

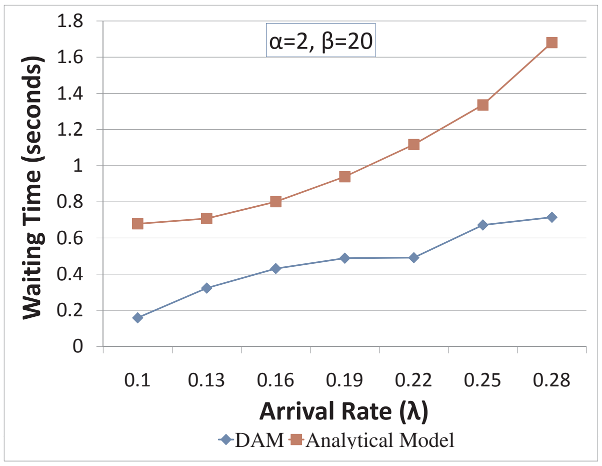
be noted that while the deviation across resources for other algorithms is steadily increasing, on average DAM has kept the load deviation, consistently, almost close to 0. This implies that the DAM was able to successfully balance the load across all the resource sites. This is due to the fact that the resource queue's valuation is increased when its load is increased and therefore, heavily loaded queues are sorted to the bottom of the ask list. The performance of the EDF-FQ algorithm, which is closest to that of DAM, also results in on average five times more Load Deviation than DAM. Moreover, from Figures 5(a) and 5(b), it can be seen that EDF-FQ does not schedule as many applications as DAM, even though EDF-FQ also tries to balance load across the resources by submitting according to queue waiting time. However, FairShare algorithm which benefits users in similar way as DAM, has resulted in the maximum load imbalance which is even worst than HVFQ. Thus, DAM is not only providing benefit to users but also providing benefit to resource providers by equally dividing load between them.

5.2. Comparison of DAM with Theoretical Results

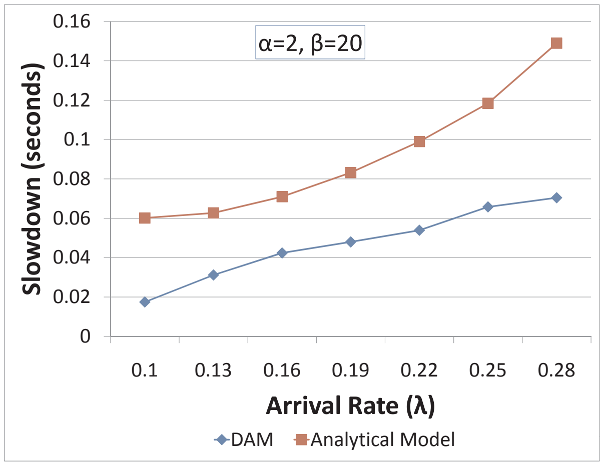
In order to understand the generalized behavior of our proposed algorithm, we compared the performance of DAM with theoretical results from a queueing model of meta-scheduler in terms of other system-based metrics such as slowdown and waiting time. An application's slowdown is its waiting time divided by its execution time. Mean slowdown is considered because users generally desire that their application delay should be proportional to its execution time [48][49]. For instance, users with lighter processor requirements will generally prefer to wait for lesser time than those with heavier requirements. The details of queueing model of meta-scheduler are given in Appendix A.

The optimal mean waiting time and mean slowdown is calculated using the approximate queuing model for meta-scheduling, by solving Equation A.24 and A.25 using the NMinimize function in Mathematica. This is achieved by finding the r_i values in each instance that produce the local minima for expected waiting time (E(W)) and slowdown (E(S)). Since the analytical model proposed in Appendix A is an approximation to the meta-scheduling problem, thus the solution obtained is actually near-optimal for the meta-scheduling problem at steady state.

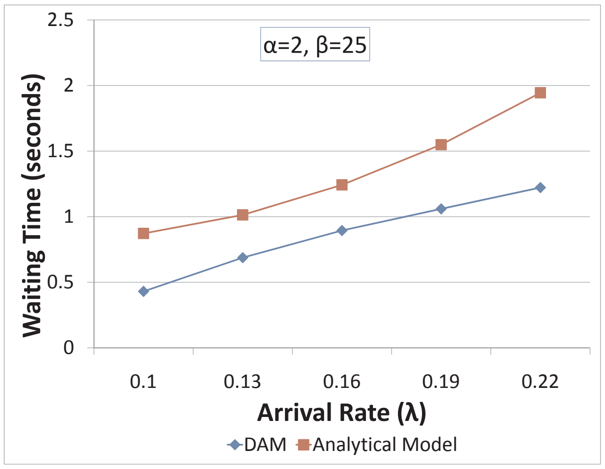
Experimental Methodology: Li et al. [50] analyzed various Grid workloads and found that the Weibull distribution is best fitted to model runtime of applications. Hence, the application runtime is generated using a Weibull distribution (α, β) [50]. The arrival rate application is assumed to be Poisson distribution with parameter λ. Since our aim is to compare both analytical and simulation results, the probability distributions for arrival rate and runtime of applications are the same in both analytical and simulation experiments. However, since the analytical model is an approximation of the real meta-scheduling systems, thus the r_i values can differ between the analytical model (where they are numerically solved)



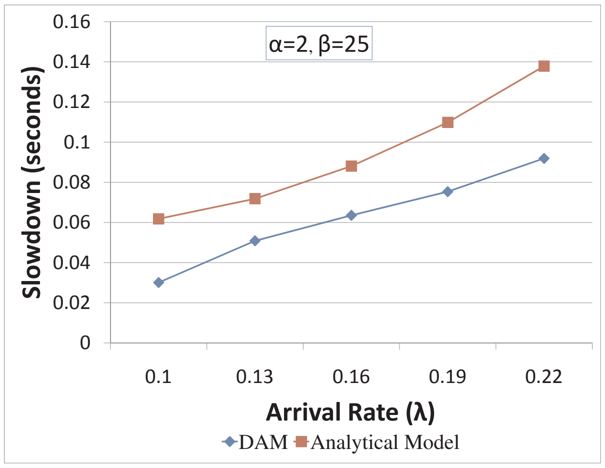
(a)



(b)



(c)



(d)

FIGURE 8. Comparison of DAM with analytical results

and the simulated real systems (where the scheduling is performed using heuristics). The arrival rate of i^{th} priority class applications (λ_i) depends on probability p_i. The p_i is obtained during the valuation process of DAM through simulation. Each resource considered is assumed to have same number of processors to make the simulation scenario as close as possible to the analytical model. To make the solution of the analytical model tractable, we have considered five Grid resources with 128 processors each and four priority class applications.

A large range of λ values are considered demonstrating a wide spectrum of load and arrival rate. The performance metrics are computed for these arrival rates each with different mean application runtime (represented by the combination of values of α and β). Two set of results with different mean application runtime are presented in Figure 8. The different mean in two scenario is obtained by scaling up the value of β which results in jobs with longer average runtime and with large variance.

In order to obtain steady state results, we have ignored scheduling of the first 5000 applications and measure mean waiting time and slowdown for the next 5000 applications. For each value of λ, experiment is repeated 30 times, and average of their results is used for comparison.

Discussion of Results: Figure 8 shows that the simulation results of DAM follow similar increasing trend as the analytical model. This not only validates the correctness of the DAM heuristic but also indicates that the performance of DAM is near optimal in terms of metrics such as mean waiting time and mean slowdown. DAM gives consistently lower values for waiting time and slowdown than the optimal values obtained from analytical model. There is a significant gap between DAM and analytical model results, for example, when $\beta = 25$, the gap between DAM and analytical model is about 25% to 30%. The reason for the gap is that the analytical model is an approximation of real meta-scheduling systems and thus it does not model the backfilling policies used by the local scheduler of resources which decreases the slowdown and waiting time of applications more than the optimal solution obtained from the analytical model.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, we have presented a meta-scheduler for allocating parallel applications with rigid processor requirements on distributed resources within a grid. The resource sites are organised as a collection of queues with different capabilities and waiting times. The goals of the scheduler are to benefit the users by taking into account their deadlines and the value attached to their applications, and to benefit the resources by allocating the applications such that the load is balanced across the Grid. The scheduler also has to benefit all users by preventing starvation of applications that have relaxed deadlines or low valuation, while keeping the effect on other crucial system metrics such as mean waiting time and slowdown to a minimum.

Given many objectives of the scheduler, we adopted the principles of an efficient double auction protocol to design the core mechanism in the scheduler. We have met the challenge of designing valuation metrics that commodify user applications with different requirements and resource queues with different waiting times to bids and asks respectively, so that they can be matched and cleared in the Double Auction-Inspired Meta-Scheduler (DAM).

Experimental evaluation of the proposed mechanism against common algorithms such as SJF, HVFQ, EDF-FQ, FCFS, and FairShare used in other meta-schedulers has showed that DAM is able to benefit both users and resource providers across all the target metrics. The double auction mechanism is not only able to schedule **about 8% to 15% more user** applications than others but also has the highest success ratio in almost all the groups for applications with different deadlines and different valuations. For users with the lowest budget (< 1000), the success ratio of their applications is increased **by almost 10%.** Similarly, DAM also benefitted resource providers by equally distributing the workload according to capacity of resource sites. Thus, DAM is able to improve the balancing of load across the constituent resources of the Grid with almost zero load deviation. While FairShare satisfies users at the same level as DAM, the difference in resource loads was the highest for the former. This means FairShare resulted in the schedule where resources were disadvantaged highest. The key reason for the better performance of DAM over others is that the valuation metrics capture the information which is important to both users and resource providers, and therefore resulted in effective scheduling. Thus, we demonstrate that by the inclusion of both system metrics and market-parameters, we can get more effective scheduling that will satisfy both users and resource providers. We have also demonstrated how classical economic mechanisms, adapted suitably, can deal with multiple QoS requirements of the users more effectively than the state-of-the art algorithms used in today's schedulers. This motivates further enquiry into exploration of adapting other economic mechanisms to solve particular problems in job scheduling.

In the future, we will integrate DAM in an actual meta-scheduler implementation such as GridWay and apply it to real-world scenarios. We also intend to experiment with different valuation methods, and examine the applicability of the mechanism to other distributed application models. In the future, we plan to extend this work in the context of Cloud computing for developing new dynamic pricing and market models. In addition, our work is particularly relevant to notion of brokering across multiple Clouds which is becoming popular in the form of InterClouds.

ACKNOWLEDGEMENTS

We would like to thank Adam Barker, Mukkadim Pathan, Kapil Gupta, Professor Rao Kotagiri and Marco Netto for their comments and suggestions on this paper. This research is funded by the Australian Department of Innovation, Industry, Science, Research (DIISR) and Australian Research Council (ARC).We would also like to thank reviewers of the journal for their comments and suggestions that helped in improving the paper substantially.

REFERENCES

[1] Henderson, R. (1995) Job Scheduling Under the Portable Batch System. *Proceedings of the 1995 International Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, USA.*

[2] Bode, B. et al. (2000) The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, USA.*

[3] Laure, E. et al. (2006) Programming the Grid with gLite. *Computational Methods in Science and Technology*, **12**, 33–45.

[4] Huedo, E., Montero, R., and Llorente, I. (2004) A framework for adaptive execution in grids. *Software Practice and Experience*, **34**, 631–651.

[5] Sabin, G., Sahasrabudhe, V., and Sadayappan, P. (2005) Assessment and enhancement of meta-schedulers for multi-site job sharing. *Proceedings of 14th IEEE International Symposium on High Performance Distributed Computing, Research Triangle Park, NC, USA.*

[6] AuYoung, A., Chun, B., Snoeren, A., and Vahdat, A. (2004) Resource allocation in federated distributed computing infrastructures. *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure, Boston, USA.*

[7] Lai, K., Rasmusson, L., Adar, E., Zhang, L., and Huberman, B. (2005) Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, **1**, 169–182.

[8] Broberg, J., Venugopal, S., and Buyya, R. (2008) Market-oriented Grids and Utility Computing: The state-of-the-art and future directions. *Journal of Grid Computing*, **6**, 255–276.

[9] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, **61**, 810–837.

[10] Feitelson, D. and Jettee, M. (1997) Improved utilization and responsiveness with gang scheduling. *Proceedings of the 1997 International Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, Switerland.*

[11] Schwiegelshohn, U. and Yahyapour, R. (1998) Analysis of first-come-first-serve parallel job scheduling. *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete algorithms*, San Francisco, California, United States.

[12] Frey, J., Tannenbaum, T., Livny, M., Foster, I., and Tuecke, S. (2002) Condor-G: A computation management agent for multi-institutional Grids. *Cluster Computing*, **5**, 237–246.

[13] Raman, R., Livny, M., and Solomon, M. (2000) Resource management through multilateral matchmaking. *Proceedings of the Ninth IEEE Symposium on HPDC, Pittsburgh, Pennsylvania.*

[14] Chun, B. and Culler, D. (2002) User-centric Performance Analysis of Market-based Cluster Batch Schedulers. *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany.*

[15] Wieczorek, M., Podlipnig, S., Prodan, R., and Fahringer, T. (2008) Applying double auctions for scheduling of workflows on the Grid. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Austin, TX.*

[16] Pourebrahimi, B., Bertels, K., Kandru, G., and Vassiliadis, S. (2006) Market-based resource allocation in grids. *Proceedings of Second IEEE International Conference on e-Science and Grid Computing, Amsterdam, The Netherlands.*

[17] Kant, U. and Grosu, D. (2005) Double auction protocols for resource allocation in grids. *Proceedings of the International Conference on Information Technology: Coding and Computing, Washington, USA.*

[18] Tan, Z. and Gurd, J. (2007) Market-based grid resource allocation using a stable continuous double auction. *Proceedings of 8th IEEE/ACM International Conference on Grid Computing, Austin, Texas, USA.*

[19] Vanmechelen, K., Depoorter, W., and Broeckhove, J. (2008) Economic Grid resource management for CPU bound applications with hard deadlines. *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, France.*

[20] Chard, K. and Bubendorfer, K. (2008) A distributed economic meta-scheduler for the grid. *Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, France.*

[21] Yeo, C. and Buyya, R. (2005) SLA based allocation of cluster resources: Handling penalty to enhance utility. *Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, USA.*

[22] Shneidman, J., Ng, C., Parkes, D., AuYoung, A., Snoeren, A., Vahdat, A., and Chun, B. (2005) Why markets could (but don't currently) solve resource allocation problems in systems. *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating Systems, Santa Fe, NM, USA.*

[23] Xiao, L., Zhu, Y., Ni, L., and Xu, Z. (2008) Incentive-based scheduling for market-like computational grids. *IEEE Transactions on Parallel and Distributed Systems*, **19**, 903–913.

[24] Shmueli, E. and Feitelson, D. (2005) Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel and Distributed Computing*, **65**, 1090–1107.

[25] Bucur, A. and Epema, D. (2000) The Influence of the Structure and Sizes of Jobs on the Performance of Co-allocation. *Proceedings of 2000 Workshop on Job Scheduling Strategies for Parallel Processing, Cancun, Mexico*

[26] Feitelson, D., Rudolph, L., and Schwiegelshohn, U. (2004) Parallel job schedulinga status report. *Proceedings of 2004 Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pp. 1–16. Springer.

[27] England, D. and Weissman, J. (2004) Costs and benefits of load sharing in the computational grid. *Proceedings of 10th International Worskhop on Job Scheduling Strategies for Parallel Processing, New York, NY, USA.*

[28] Di Martino, V. and Mililotti, M. (2004) Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, **30**, 553–565.

[29] He, L., Jarvis, S., Spooner, D., Chen, X., and Nudd, G. (2004) Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. *Proceedings of the 5th International Workshop on Grid Computing, Pittsburgh, PA, USA.*

[30] Singh, G., Kesselman, C., and Deelman, E. (2007) A provisioning model and its comparison with best-effort for performance-cost optimization in grids. *Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, CA, USA.*

[31] Abawajy, J. H. and Dandamudi, S. P. (2003) Parallel job scheduling on multicluster computing system. *Proceedings of 2003 IEEE International Conference on Cluster Computing (Cluster 2003), Boston, Massachusetts, USA.*

[32] Sabin, G., Kettimuthu, R., Rajan, A., and Sadayappan, P. (2003) Scheduling of parallel jobs in a heterogeneous multi-site environment. *Job Scheduling Strategies for Parallel Processing*, pp. 87–104. Springer.

[33] Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., and Zhuk, S. (2006) Two level job-scheduling strategies for a computational grid. *Parallel Processing and Applied Mathematics* , **?**, 774–781.

[34] Garg, S. K., Venugopal, S., and Buyya, R. (2008) A meta-scheduler with auction based resource allocation for global grids. *Proceedings of the 14th International Conference on Parallel and Distributed Systems (ICPADS), Melbourne, Australia.*

[35] Zhang, W., Cheng, A., and Hu, M. (2006) Multisite co-allocation algorithms for computational grid. *Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece.*

[36] Huedo, E., Montero, R., Llorente, I., Thain, D., Livny, M., van Nieuwpoort, R., Maassen, J., Kielmann, T., Bal, H., Kola, G., et al. (2005) The GridWay framework for adaptive scheduling and execution on grids. *Software-Practice and Experience*, **6**.

[37] Pacifici, G., Spreitzer, M., Tantawi, A., and Youssef, A. (2003) Performance management for web services. *Research Report RC22676, IBM TJ Watson Research Center, Yorktown Heights, NY,USA*, **10598**.

[38] Lifka, D. (1995) The ANL/IBM SP Scheduling System. *Proceedings of the 1995 International Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, USA*.

[39] Feitelson, D., Rudolph, L., Schwiegelshohn, U., Sevcik, K., and Wong, P. (1997) Theory and practice in parallel job scheduling. *Proceedings of 1997 International Workshop on Job Scheduling Strategies for Parallel Processing, Geneva, Switzerland*.

[40] Grosu, D. and Das, A. (2004) Auction-based resource allocation protocols in grids. *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems, Cambridge, USA*.

[41] Butler, J., Morrice, D., and Mullarkey, P. (2001) A multiple attribute utility theory approach to ranking and selection. *Management Science*, **47**, 800–816.

[42] Wallenius, J., Dyer, J., Fishburn, P., Steuer, R., Zionts, S., and Deb, K. (2008) Multiple Criteria Decision Making, Multiattribute Utility Theory: Recent Accomplishments and What Lies Ahead. *Management Science*, **54**, 1336.

[43] Keeney, R. and Raiffa, H. (1993) *Decisions with multiple objectives: Preferences and value tradeoffs*. Cambridge University Press.

[44] Wiseman, Y. and Feitelson, D. (2003) Paired Gang Scheduling. *IEEE Transcatiosn on Parallel and Distributed Systems*, **14**, 581–592.

[45] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., and Stockinger, K. (2000) Data management in an International Data Grid Project. *Proceedings of 1st International Workshop on Grid Computing*. Bangalore, India.

[46] de Assunção, M. and Buyya, R. (2009) Performance analysis of allocation policies for interGrid resource provisioning. *Information and Software Technology*, **51**, 42–55.

[47] Waldspurger, C. A., Hogg, T., Huberman, B. A., Kep hart, J. O., and Stornetta, W. S. (1992) Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, **18**, 103–117.

[48] Rudolph, L. and Smith, P. (2000) Valuation of Ultra-scale Computing Systems. *Proceedings of 2000 Job Scheduling Strategies for Parallel Processing, Cancun, Mexico*.

[49] Harchol-Balter, M. and Downey, A. (1997) Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, **15**, 253–285.

[50] Li, H., Groep, D., and Wolters, L. (2004) Workload characteristics of a multi-cluster supercomputer. *Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing, New York, NY, USA*.

[51] Kleinrock, L. and Gail, R. (1976) *Queueing systems*. Wiley New York.

[52] Mu'alem, A. and Feitelson, D. (2001) Utilization, predictability, workloads, and user runtime estimatesin scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, **12**, 529–543.

[53] Cobham, A. (1954) Priority assignment in waiting line problems. *Journal of the Operations Research Society of America*, **2**, 70–76.

[54] Harchol-Balter, M. (2002) Task assignment with unknown duration. *Journal of the ACM (JACM)*, **49**, 260–288.

[55] Broberg, J., Tari, Z., and Zeephongsekul, P. (2006) Task assignment with work-conserving migration. *Parallel Computing*, **32**, 808–830.

[56] Nelson, R. and Philips, T. (1989) An approximation to the response time for shortest queue routing. *ACM SIGMETRICS Performance Evaluation Review*, **17**, 181–189.

[57] Nelson, R. and Philips, T. (1993) An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation*, **17**, 123–139.

[58] Nozaki, S. and Ross, S. (1978) Approximations in finite-capacity multi-server queues with Poisson arrivals. *Journal of Applied Probability*, **15**, 826–834.

[59] Harchol-Balter, M., Crovella, M., and C.D., M. (1999) On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, **59**, 204–228.

APPENDIX A. FORMULATION OF QUEUING NETWORK MODEL

In this section we present the detailed derivation of the meta-scheduler's queueing network model. The analytical model of the meta-scheduler is based on queuing theory which has been used extensively for modelling scheduling problems in distributed systems. Queuing network model provides a powerful stochastic and probabilistic approach to analytically model the mechanisms of scheduling policies [51]. Using this model, we obtain measures for metrics, such as mean waiting time and mean slowdown of applications, which is used to compare the performance of our proposed meta-scheduling algorithm with the (near-)optimal.

We can model the meta-scheduling system under consideration as a network of three queues (Figure A.1) to get the optimal bound for various system parameters. In the meta-scheduler, each application is assigned a priority or valuation and matched to resources according to its priority. This component of the meta-scheduler can be modelled by a priority queuing system. Since processing time of each application may not be exponential, we have chosen M/G/1 priority queue system to analyze this part of the meta-scheduler. An application is then assigned and dispatched to a resource to balance load across all queues in the system. This component of the meta-scheduler is analyzed using

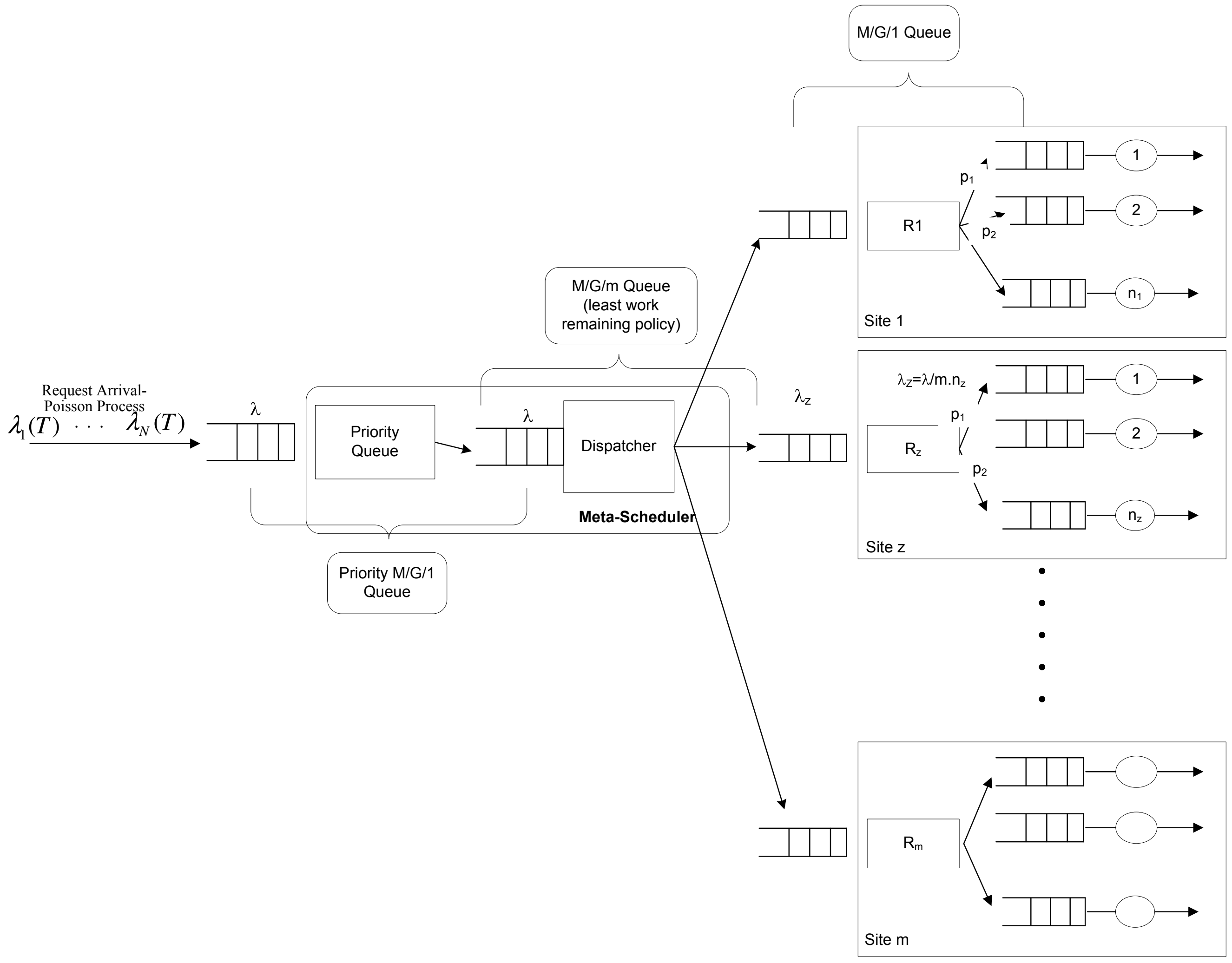

M/G/1 Queue
M/G/m Queue (least work remaining policy)
Request Arrival-Poisson Process
$\lambda_1(T) \cdots \lambda_N(T)$
λ
Priority Queue
λ
Dispatcher
Meta-Scheduler
Priority M/G/1 Queue
λ_z
R1
p1
p2
1
2
n1
Site 1
$\lambda_z=\lambda/m.n_z$
Rz
1
2
nz
Site z
Rm
Site m

FIGURE A.1. Queuing Network Model of Meta-scheduling

a comparable queueing system i.e. central queueing system with least work remaining policy (M/G/m), where m is the number of Grid sites.

At each resource, an application is executed on more than one machine at the same time. The local scheduler at the resource end also uses the backfilling policies and different types of queues to increase their utilization [52]. To analyze this part of meta-scheduling, we have taken an approximate model of real local scheduling systems. We have divided the processors in a resource into multiple partitions, where each partition acts as a M/G/1 queue. This resource with multiple partition is an approximation of real local scheduling systems. The CPU/machine processing requirements of each application also follows a general distribution (denoted as G). Let there be N_z servers/processors at resource site z ($1 \leq z \leq m$), which are divided between n_z queues (partitions). Each partition f on resource site z is initially assigned r_{zf} processors.

Thus, in meta-scheduling, each application goes through three queuing systems before it starts executing. Hence, by analyzing the combination of the following sequential network of three queuing systems, we get an approximate analytical model for meta-scheduling system:

- Valuation or prioritization (M/G/1 Queue with Priority)
- Matching or Dispatching (M/G/m Queue policy)
- Scheduling at a resource for execution (M/G/1 Queues)

The applications' arrival at the meta-scheduler follows a Poisson distribution with mean λ. Let the service time is random variable X by a general distribution. The mean service time of the applications is $E(X)$ and the second moment of service time is σ or $\mathrm{E}(X^2)$. The distribution of processor requirement by each application is given by C. Let the number of the resource sites be m.

Appendix A.1. Valuation or prioritization (M/G/1 Queue with Priority)

In the first part of DAM, applications are assigned valuations so that they can be re-sequenced for scheduling. Thus, this scenario can be modelled as a single server queuing system as shown in Figure A.1. There are K priority classes of applications. The mean service time of priority class j is $\frac{1}{\mu_j}$ and the second moment of service time is $\sigma_j^{'}$. The overall second moment of service rate of applications by the server is $\sigma^{'}$. Since the time taken to assign priorities is very small in DAM algorithm, the mean service rates of applications will also be very small in compare to their actual processing time. Thus, for experiments, we have modelled variance $\sigma_j^{'}$ very low to keep service time for all the jobs in the first queuing system to be almost equal and very low.

Let p_j be the probability with which j^{th} priority class applications arrived at the server. Then, the mean arrival rate (λ_j) of the j^{th} priority applications is given by:

$$\lambda_j = p_j \times \lambda \tag{A.1}$$

Since the service time of all priority classes is $\frac{1}{\mu_j}$, thus the system load due to j^{th} priority class applications is given by:

$$\rho_j = \frac{\lambda_j}{\mu_j} \tag{A.2}$$

Let $E(w_j)$ is expected waiting time for the applications with priority level j. Then using the classic result on non-preemptive priority queue by A. Cobham [53] we obtain the mean waiting time and slowdown $(E(s_j))$ of class-j applications:

$$E(w_j) = \frac{\frac{\lambda \times \sigma^{'}}{2}}{(1 - \sum_{i=1}^{j-1} \rho_i)(1 - \sum_{i=1}^{j} \rho_i)} \tag{A.3}$$

Thus, the total mean waiting time and slowdown of applications in the system is given by:

$$\bar{W_1} = \sum_{j=1}^{K} [(p_j \times (E(w_j) + \frac{1}{\mu_j})) \tag{A.4}$$

$$\bar{S_1} = \sum_{j=1}^{K} (p_j \times (E(w_j) + \frac{1}{\mu_j})) \times E(X_j^{-1})] \tag{A.5}$$

Appendix A.2. Matching or Dispatching (M/G/m Queue)

After applications are served by the above queuing system based on priority, applications in the outgoing queue will be served by the second queuing server on a FCFS basis. It can be considered as third component of the meta-scheduler i.e. matching. As an approximation, the applications arriving into the second queueing system are assumed to follow a Poisson process. This is true if the service distribution is exponential, but is an approximation otherwise. This approximation is commonly used in many advanced queuing models due to the difficulty of modeling a General (G) arrival/departure process, and solving any resultant models [54][55]. In reality, the arrival into the meta-scheduler would be less bursty (i.e. more uniformly random) than those of a Poisson process, but this has a minimal effect on the overall model. For the assignment of these applications to the resource sites, we have used the central queue with m servers policy. This policy has been proven to be equivalent to least-work-remaining allocation policy, which is claimed to be optimal by Nelson et. al [56][57]. This policy is not analytically tractable under M/G/m queueing system. Nonetheless, several good approximations exist in the literature, many of which are empirical. In this study, we use the approximation given by Nozaki et. al [58] which is also adopted in several other works [59] [49]. The approximation for mean queue length is stated as:

$$E(N_{M/G/m}) = E(N_{M/M/m}) \frac{E(X^2)}{E(X)^2}, \tag{A.6}$$

where X: Service Requirement and N: Queue Length.
The load of system is given by:

$$\rho = \lambda \times E(X) \tag{A.7}$$

Let $E(W_{M/M/m})$ be the average waiting time in a M/M/m queueing system and ρ be the system load. Then, using well known Pollaczek-Khinchin formula in queuing theory, the average queue length and waiting time for M/M/m queueing system is given by:

$$E(N_{M/M/m}) = \frac{P_N \rho}{1-\rho}, \tag{A.8}$$

$$\text{where } P_N = [\sum_{z}^{m-1} \frac{(m\rho)^z}{z!} + \frac{m^m \rho^m}{m!1-\rho}]^{-1}$$

$$E(W_{M/M/m}) = \frac{E(N_{M/M/m})}{\lambda} \tag{A.9}$$

Thus, using Equation A.6, A.8 and A.9, the mean waiting time and slowdown in the queue by using central queue policy is given by:

$$\bar{W_2} = E(W_{M/G/m}) = E(W_{M/M/m}) \frac{\sigma}{E(X)^2} \tag{A.10}$$

$$\bar{S_2} = \bar{W_2} E(X^{-1}) \tag{A.11}$$

Appendix A.3. Scheduling at a Resource for Execution (M/G/1 Queues)

After the application is assigned to the resource queue, the application is needed to be scheduled on multiple servers. Unlike the most of the commonly used queuing systems where an application requires only one server for execution, here each application needs more than one server (processors in our context) at the same time. Moreover, the local scheduler at a resource site uses different backfilling policies to decrease slowdown of applications [52]. Since, it is analytically intractable to solve this system, the designed analytical system for this component of meta-scheduling differ slightly from real systems. We divided the processors of the resource into multiple disjoint partitions, with one queue per partition. For example, let the total number of processors at a site are '6' and we divided them into 3 partition with each partition having 1, 2, and 3 processors respectively. Thus, application from each queue will be executed on one of these partition. If application requires one processor, it will be executed on partition 1. If application requires processors between range (1,2) or (2,3), then application will be executed on partition 2 or partition 3.

Thus, formally, each partition f on resource z is initially assigned $1 \leq r_{zf} \leq N_z$ processors. Each of these queues processes the applications, which require processors within range of $(r_{z(f-1)}, r_{zf})$, on FCFS basis. Let there be N_z servers/processors at resource site z, which are divided between n_z queues. Thus:

$$r_{z0} + r_{z1} + r_{z2} + r_{z3} ... r_{zf} + .. r_{z(n_z-1)} = N_z \quad \text{(A.12)}$$

Since the total number of resource sites is m, the arrival rate of applications at resource site z is given by:

$$\lambda_z = \frac{\lambda}{m} \quad \text{(A.13)}$$

Let u_{zf} be the probability that an application requires processor between $r_{z(f-1)}$ and r_{zf}, and thus is processed by queue f of resource site z. Let C be the probability distribution of processor requirements for an application, this probability is given by:

$$u_{zf} = \int_{a_{z(f-1)}}^{a_{zf}} C(x)dx \quad \text{(A.14)}$$

Thus, the fraction of applications arriving at queue f of the resource site z is given by:

$$\lambda_{zf} = \lambda_z u_{zf} \quad \text{(A.15)}$$

The load shared by each queue, when $E(X)$ is mean service time, is given by:

$$\rho_{zf} = \lambda_{zf} E(X) \quad \text{(A.16)}$$

Since each queue partition has M/G/1 FCFS queue system behavior, thus we can directly use the same results for the average waiting time. This is given by:

$$E(w_{zf}) = \frac{\lambda_{zf}\sigma}{2(1-\rho_{zf})} \quad \text{(A.17)}$$

The total expected waiting time at a resource site is the average of all waiting time at each queue partition, which is given by:

$$E(w_z) = \frac{1}{n_z}\sum_{f=1}^{n_z} E(w_{zf}) \quad \text{(A.18)}$$

Let W_3 and S_3 be the expected waiting time and slowdown of all resource sites, respectively. Thus, they are given by:

$$W_3 = \frac{\lambda}{m}\sum_{z=1}^{m} E(w_z) \quad \text{(A.19)}$$

$$S_3 = W_3 \times E(X^{-1}) \quad \text{(A.20)}$$

The overall expected waiting time and slowdown measures are given by combining waiting time and slowdown of all three queueing system, i.e., Equations A.4-A.5, A.10-A.11 and A.19-A.20:

$$\text{Waiting Time} = E(W) = W_1 + W_2 + W_3 \quad \text{(A.21)}$$

$$\text{Slowdown} = E(S) = S_1 + S_2 + S_3 \quad \text{(A.22)}$$

$$\text{(A.23)}$$

Where W_i and S_i, $i\epsilon[1,3]$ is waiting time and slowdown of i^{th} queueing system.

Thus, the queueing theory based analytical model for our meta-scheduling mechanism is given by the following equations:

$$Minimize(E(W)) \text{ subject to} \quad \text{(A.24)}$$
$$\sum_{k}^{n_z} r_{zk} = N_z, 1 < z < m$$

$$Minimize(E(S)) \text{ subject to} \quad \text{(A.25)}$$
$$\sum_{k}^{n_z} r_{zk} = N_z, 1 < z < m$$

The above model gives an approximation for real meta-scheduling systems. Thus, to predict the performance of our meta-scheduling policy, we can obtain optimal expected waiting time and slowdown. Thus, Equations A.24 and A.25 for expected waiting time and slowdown are needed to be solved for different values of n_z using optimization tools such as Mathematica [Wolfram Research 2008]. These analytical values are used to ascertain the performance of DAM.