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

Saurabh Kumar Garg[1], ***Member, IEEE,***
Srikumar Venugopal[2], ***Member, IEEE,*** *James Broberg*[1], ***Member, IEEE,*** and
Rajkumar Buyya[1], ***Senior Member, IEEE,***
[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{sgarg, brobergj, raj}@csse.unimelb.edu.au
[2]School of Computer Science and Engineering
The University of New South Wales, Sydney, Australia
srikumarv@cse.unsw.edu.au

Abstract

Existing Grid meta-schedulers such as GridWay either target system-centric metrics, (such as utilization or throughput), or prioritize applications based on utility functions provided by the users. The system centric approach gives less importance to users' individual utility, while the user centric may have adverse effects such as poor system performance and unfair treatment of users. Therefore, this paper proposes a novel Continuous Double Auction (CDA)-based meta-scheduler mechanism that schedules parallel applications providing improvements that benefit both users and resources in terms of their effective utilization. We have designed valuation metrics for both user applications and resources that commodify the complex resource requirements of the users and the capabilities of available computational resources. We analytically model the meta-scheduling problem in Grid using queuing theory and compare our heuristic with the analytical model results to ensure its correctness. Then, through simulation, 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 more user requirements than others, but also improves system utilization through load balancing.

Index Terms

Grid Computing, Resource Allocation, Meta-Scheduling, Auction

I. Introduction

Grids are composed of distributed high-performance commodity clusters and supercomputers managed by batch job schedulers such as Portable Batch Scheduler (PBS) [16]. 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 [4], gLite Workload Management System [21], and GridWay [17].

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 to 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 reduce system utilization and lead 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.

Previously, meta-schedulers focused more on improving system-centric performance metrics such as utilization, average load and turnaround time for user applications [32]. They were not designed to cater to the sophisticated QoS needs of an application, particularly when the demand for resources exceeded the supply. In recent years, a number of researchers have explored economy-based models to address user requirements in meta-scheduling. Auctions have been particularly preferred [2], [20], [5] as they provide immense flexibility to participants to specify their valuations for applications and resources. However, these systems have limitations [42]. It is difficult for the users to come up with a valuation corresponding to their utility function. In a grid with changing availability of resources, it is difficult to determine resource and application valuations accurately. Also, users with low budgets and urgent requirements may be starved by those with large budgets. Auctions are applied to commodities that are comparable to one another. However, the parallel applications having rigid processor requirements are not comparable, and cannot be commodified. Hence, the problem is to design valuation metrics that commoditize resource requirements and availability so as to take advantage of the efficiency of auction mechanisms [33]. Thus, we need new scheduling mechanisms that are not only efficient and ensure better performance of Grid resources, but also take into account user interests, resource valuation and demand, and allocate resources fairly to user applications.

In this paper, we present a novel grid meta-scheduling mechanism which uses principles of auctions to allocate resources to parallel applications with competing QoS demands. We have considered parallel applications having multiple communicating processes that are to be allocated on a rigid number of processors available from a single grid site. We have designed valuation metrics that enable trading of slots in different application queues at grid resources among different applications with fixed processor requirements. In this manner, the resource shares are commodified so that an efficient economy-based allocation mechanism can be applied to benefit both resource users and owners alike. We construct an analytical model of meta-scheduling

problem using queueing theory and use it to evaluate the performance of our mechanism. Then, by using simulation on real workload traces of parallel applications from super-computing centers, we show that our scheduler performs better than classical scheduling mechanisms in similar conditions.

The rest of the paper is structured as follows. In the next section, we discuss 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 Section 6 discusses the results. Finally, we conclude the paper and present future steps in this direction in Section 7.

II. Related Work

Currently, meta-schedulers in operation, such as GridWay [17] and gLite Workload Management System [21] use heuristics such as First Come First Serve (FCFS). Moab also has a FCFS batch scheduler with easy backfilling policy [4]. Condor-G [11] uses either FCFS or matchmaking with priority sort [30] as scheduling policies. The application-level schedulers in the Grid (e.g. AppLeS [3]) are tightly coupled with the application itself to achieve high performance. These schedulers focused more on optimizing traditional system metrics such as system utilization and application waiting time without considering market based metrics. Additionally, even though these schedulers give administrator the capability to integrate any mechanism, these schedulers still do not have special mechanisms to handle conflicts between concurrent users with overlapping QoS requirements.

Many market-based scheduling mechanisms have been proposed to handle concurrent user requirements. REXEC [7] and Tycoon [20] 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. [40] proposed a Grid resource allocation model based on Continuous Double Auctions (CDAs) to schedule workflow applications on Grid resources. Vanmechelen et al. [36] have developed centralized and decentralized algorithms for economic resource management using futures market to maximize realized consumer value. LibraSLA [44] prioritizes users on the basis of application deadlines and the user-specified penalties for not meeting them. Bellagio [2] is a system that seeks to allocate resources for distributed computing infrastructure in an economically efficient fashion to maximize aggregate end-user utility. These 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. Also, these systems use only the application valuations provided by the users. However, users cannot be expected to provide accurate valuations as they lack perfect information about resource availability in a dynamic environment such as a grid.

Xiao et al. [43] present an incentive-based scheduling scheme which utilizes a peer-to-peer decentralized scheduling framework to maximize the success rate of application which require one processor for execution, and to minimize fairness deviation among resource providers. We have followed a similar decentralized auction and bidding framework for a market-like computational grid. However, we differ not only in the valuation mechanism used but also in the objectives for resource provider. Our uniqueness particularly lies in the analytical analysis of meta-scheduling problem for scheduling parallel applications which require more than one processor for execution. Also Xiao et al. [43] focus on application with only one processor requirement while we are considering applications with requirements for multiple processor.

Many Genetic Algorithm (GA)-based solutions have been proposed to improve the performance of parallel application scheduling mechanisms [11, 10, 18, 29]. As GAs are computationally expensive to execute, they are not suitable for a dynamic environment such as grids where schedules have to be recomputed regularly since resource availability can change rapidly.

In a previous publication [12], we presented a double auction meta-scheduling mechanism to increase fairness and user satisfaction for Bag-of-Task applications. The work presented here generalizes the application model to parallel applications from a simple bag-of-task model. In this paper, therefore, we focus on designing a meta-scheduling mechanism using auction principles for parallel applications with rigid processor requirements to benefit both the user, by reducing starvation 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: the fixed number of processors required by the application; the dynamic availability of resources with different capabilities in different administrative domains; and continuously arriving applications at the meta-scheduler [45].

Hence, the contribution of this paper is as follows a) a queueing theory based analytical model to analyze the meta-scheduling problem in Grid, and b) a meta-scheduling mechanism for parallel applications that takes advantage of the efficiency of double auctions in order to benefit both users and resources. This is demonstrated via valuation metrics that commodify the resource share available and the users' application requirements so that they can be compared

and matched.

III. System Model

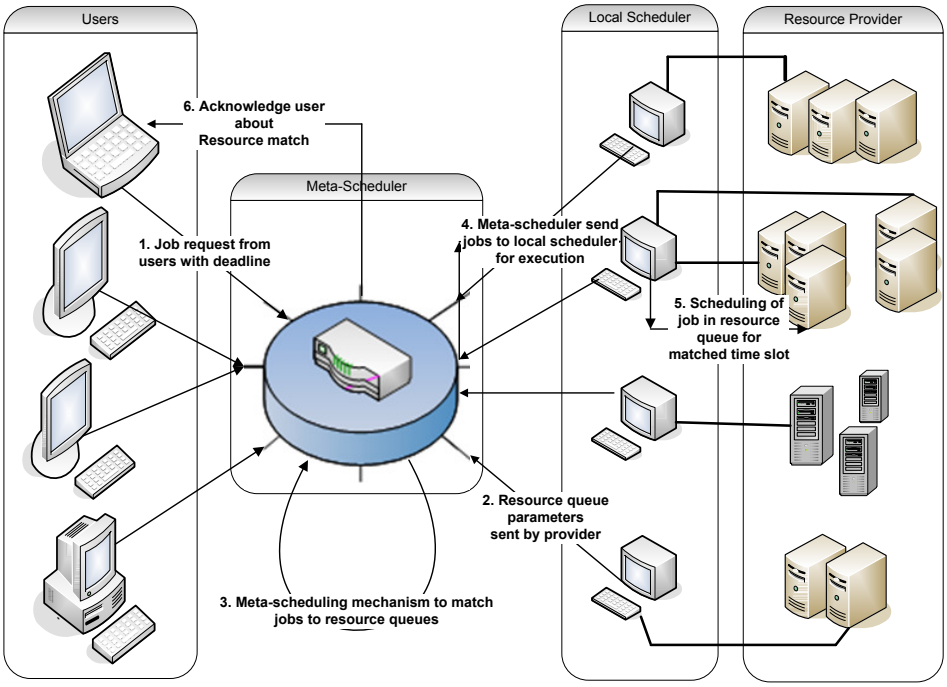


Fig. 1. Meta-Scheduler Interaction with Users and Local Scheduler

The meta-scheduler considered in this work follows the model commonly found in large computing installations across educational and research institutions [24] as shown in Figure 1. In this model, resources are managed at different sites by administrators who have to cater to 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) [29]. The resource 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 [23]. 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 so as to satisfy both the users' and the resources' objectives. It may have control over allocation to some or all 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 transfer 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 a queue on that resource. This initial valuation may be based on the processors provided to the queue. The objective of a resource is to balance load across all queues by assigning as many slots without gaps in the resource allocation (i.e., fragmentation). This can help in maximising utilization.
- **User Jobs:** In this work, we consider the compute intensive parallel application model with multiple communicating processes for user applications. An application has a rigid number of processor requirements that needs to be satisfied at a single resource site in a grid. The reason for this kind of requirement is that the performance of these applications may get severely affected when executed across resource sites with different configurations. 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. To facilitate the comparison between the algorithms described in this work, the estimated execution time of an application provided by the user is considered to be accurate [10].

IV. Double Auction-Based Meta-scheduler (DAM)

Auction-based mechanisms have been the subject of many previous studies. Grosu *et al.* [13] compare resource allocation protocols using First-Price, Second-Price Vickery and Double Auction (DA). They show that DA favors both users and resources, while the first-price auction is biased towards resources and the Vickery auction favors users. Kant *et al.* [18] compared three different DA protocols and concluded that the Continuous Double Auction (CDA) protocol

performs better than the other DA protocols in terms of resource utilization, resource profit and budget spent. Therefore, we have opted for CDA as the basic mechanism for our meta-scheduler.

In a typical CDA, sellers and buyers submit offers (*asks*) and requests (*bids*) respectively to an auctioneer who continually ranks them from highest to lowest in order to generate demand and supply profiles. 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.

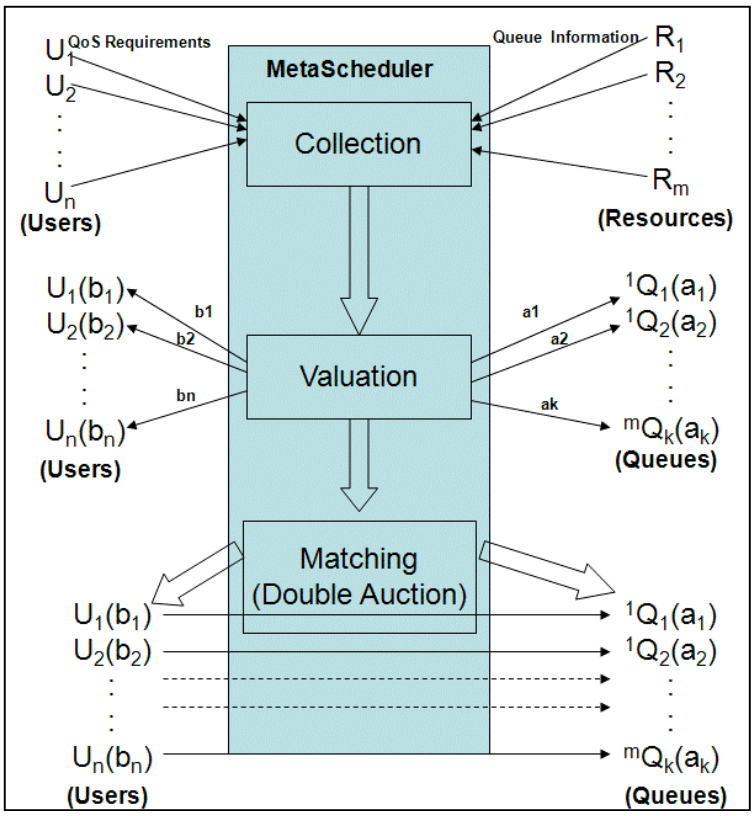


Fig. 2. Double Auction based Meta-scheduler Protocol

The elements of our meta-scheduler, which acts as an auctioneer in this context, can be divided into three parts, as shown in Figure 2 – *collection*: meta-scheduler collects information about queue slot availability and waiting time, *valuation*: assigns values to the user applications and resource queues, and finally, *matching*: matching of applications to resources. Within the meta-scheduler, an application valuation is considered as a *bid* while a resource valuation is considered as an *ask*. In Figure 2, U_n represents the n^{th}user application, a_k and b_n represent the

ask k for resource queue k and the bid corresponding to user n respectively, and ${}^{m}Q_k$ represents the resource queue k at the resource m.

In our mechanism, the meta-scheduler generates the bids and asks using the initial valuations of the users and resources as input and augmenting them with information about resource availability and user requirements. The valuation methods are described in the next section. At regular scheduling intervals, the meta-scheduler matches the applications (asks) to the resource queues (bids) if the deadline constraint of the application is satisfied. If an application cannot be matched, then it will be reconsidered in the next scheduling interval. Throughout the rest of the paper, we refer to the double auction mechanism as DAM.

A. Valuation Mechanism

One of the most important components of a double auction is to assign valuation to user applications and resources. The Grid services may be valued based on the cost of infrastructure, and economic factors like supply and demand. However, user needs and urgency, and simultaneously, efficient utilization of Grid services must be reflected through valuation of user applications and resources. Therefore, the meta-scheduler must generate a metric for both the users and the resource providers that takes into account all these attributes. This valuation should be dynamic, that is, in each scheduling cycle; it should be updated based on various parameters, and the dynamic demand and supply of system. To design a good valuation metric, we took inspiration from Multi-attribute utility theory (MAUT) [6] [39] [37]. MAUT gives a logical, consistent and tractable approach for quantifying an individual's preferences by consolidating them into a single objective. This allows comparison of many diverse measures via single value. This theory includes first the identification of attributes and desirability function for each attributes and, then aggregation of these desirability functions to a single scalar utility value. The details of our valuation metric are discussed in next section.

a) Resource Valuation (Ask):: The valuation of resources is affected by various attributes such as waiting time, load, initial valuation of the provider, and economic factors such as demand and supply. The load of a queue is defined as the ratio of the number of processors occupied to the total number of processors available in the resource. In order to balance load across independent grid sites, the meta-scheduler tries to submit more applications to the least loaded queues on the resources. Also, the most urgent application must be matched to the fastest queue. Since, in a CDA, the maximum bid is matched to minimum ask, the valuation of resource queues

should be such that the queue with the least waiting time should get the least value. Moreover, the valuation metric should also take into account the initial valuation given by the resource provider, and also the demand and supply of resources in the system (denoted as $Demand$ and $Supply$). $Demand$ is total number of task to allocate and $Supply$ is the total number of processors in 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 $l_{k,t}$. Thus, the resultant valuation metric, which is formed by considering all the attributes, given as following:

$$a_k(t) = O_k \times w_{k,t} \times c_k \times l_{k,t} \times \frac{Demand}{Supply}, \text{ where } O_k \text{ is proportionality constants} \quad (1)$$

b) Job Valuation (Bid):: Similar to resources, a user's application has also many attributes such as the number of CPUs required, deadline and run time. The valuation of an application i at time t is designed in order to provide the maximum value to applications that have an urgent deadline. The urgency can be calculated as $(d_i - t)$, where d_i is the user-supplied deadline for the application. Also, if an application has not been allocated in the previous scheduling cycle, its value should be increased. This is to reduce the possibility of this application getting starved of resources by urgent applications that have arrived in the meanwhile. Let st_i be the submission time of the application. Similar to resource queue valuation, the valuation metric should also take into account the initial valuation of the application given by the user, and also demand and supply of resources in the system (denoted as $Demand$ and $Supply$). Let v_i is the initial application valuation given by the user. Thus, the desirability functions are proportional to v_i, st_i, $Demand$, $Supply$, and $(d_i - t)$. Thus, the resultant valuation metric, which is formed by considering all the attributes, given as following:

$$b_i(t) = H_i \times v_i \times \frac{1}{d_i - t} \times \frac{Demand}{Supply} \times (t+1-st_i), \text{ where } H_i \text{ is proportionality constants, and } d_i \neq t \quad (2)$$

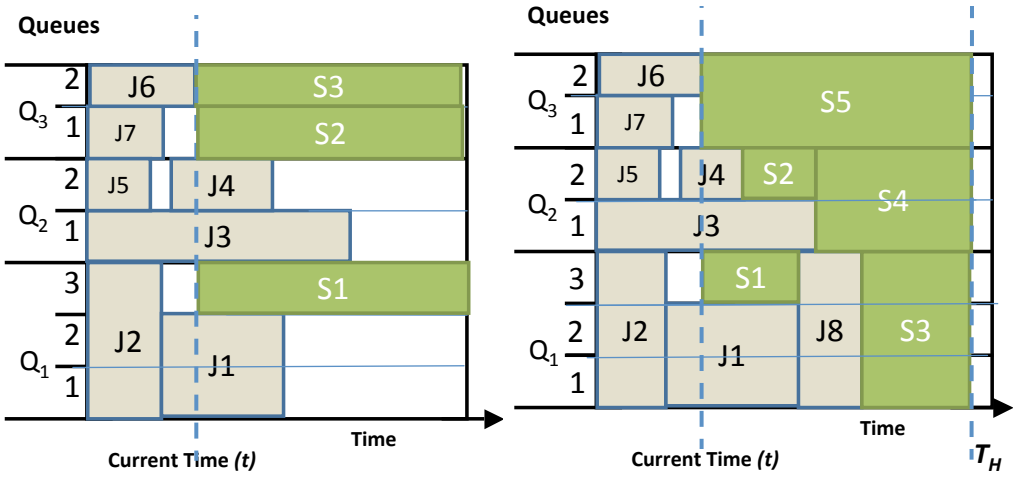
B. The Meta-Scheduling Algorithm

As discussed earlier, the bids and asks generated by the meta-scheduler using the valuation metrics are sorted on the basis of their values. Let following be the ordering of asks and bids after sorting:

$$a_1 < a_2 < \ldots a_j \ldots < a_m$$

$$b_1 > b_2 > \ldots b_i \ldots > b_n$$

A user application is allowed to participate in the match process if $a_m > b_n$. As noted before in Section III, the commodity unit traded on behalf of the resource site is a slot which a set of processors bounded by start and finish times. Figure 3 demonstrates the two methods by which the slots can be generated:



(a) Immediate (b) Packaged

Fig. 3. Available Queue Slots.

Immediate: At time t, the meta-scheduler can provide an *immediate* allocation to applications on the cpus which are free currently. For example in Figure 3(a), on queue Q_3 two processors are available, which can be traded. In this case, the slot start time is the scheduling instant, and the slot is allocated for the estimated run time of the application. In addition, the maximum number of slots available will equal the number of processors. This manner of slots trading can help in reducing fragmentation in scheduling when application's runtime is not precise.

Packaged: Alternatively, the applications can be scheduled so as to fill up the queue up to a specific time horizon T_H. The slots start from the first available time and contains as many processors that are not occupied for a specific duration. In Figure 3(b), the slots S_1 to S_5 are examples of such slots. After current time t, 2 processors are free in Queue Q_3 upto time T_H. Thus available time slot is S_5. While in case of Queue Q_1, one processor is available which constitutes the time slot S_1. In this method slot sizes

can be of different sizes as can be noted from Figure 3(b). This approach, depicted in Figure 3(b), was also used by Singh et. al [34]. In this case, local scheduler at resource can use backfilling to minimise the fragmentation in the schedule such that applications execution does not get delayed.

Algorithm 1: Double Auction Meta-scheduler

```
1  while current_time < next_schedule_time do
2      RecvResourcePublish(P)
       // P contains information about providers
3      RecvJobQoS(Q)
       // Q contains information about users
4  Calculate the Demand and Supply for resources
5  Update the value of bids and asks using eqn. 2 and 1
6  Sort asks in ascending order
7  Sort bids in descending order
8  while all applications are assigned to resource queues do
9      if bid b_i is greater than ask a_j then
10         if QueueWaitingTime(j) + ExecTime(i) < Deadline(i) then
11             if check processor availability on resource j then
12                 Schedule the application i to the resource j
13                 add application with matched resource site in Schedule List (Schd_List
14                 update the value of available time slots with ask a_j
15                 i++
16
17         else
18             add user application to pending application list
19     j++
20 foreach element ∈ Schd_List do
21     notifyuser()
```

Our scheduling algorithm is shown in Algorithm 1. In each scheduling cycle, the meta-scheduler schedules the parallel applications after collecting all user's 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 it assigns valuation to user applications (bids) and resource's queue (asks) using the pricing mechanisms presented in the previous section 4.1 (Line 5).

From the sorted bid list, the bid (b_i) with highest value will be matched to the resource queue with the minimum ask (a_j) that can satisfy the user requirement. Whether the user application i will be scheduled to the resource queue j (corresponding to ask a_j), it will depend on the applications' processor and deadline requirements. Thus, first deadline of application i is checked using waiting time of the resource queue j (Line 10) and then processor availability is checked on resource queue j (Line 11). If there is an ask which satisfies the application's QoS requirements, then the bid is matched to ask, and the user and the resource provider are informed of the match. The application i is then scheduled on to the resource queue j (Line 12) and then added to schedule list (Line 13). The available time slots (commodity units) on resource queue j are updated correspondingly (Line 14). 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 18). Therefore, the application is removed from the bids list in that scheduling cycle. If the required number of processor is not available on the resource queue j, bid b_i is matched with next ask in the sequence (Line 19). Matching for other bids and asks will be repeated until either all bids or asks are matched.

C. Queueing Theory Based Approach Model for Meta-scheduling

In order to know the generalized behavior of any meta-scheduling algorithm, it is important to analyze it with a queueing model which will also help to test the near optimality of the algorithm. Thus, we designed a formal mathematical model to analyze and predict the performance of the DAM. This analytical model which finds the expected (average) metrics for our meta-scheduling heuristic is used to test the correctness and substantiate the strength of DAM in experimental section.

Our analytical model is based on queuing theory which has been used extensively for modelling scheduling problems in distributed systems. Queuing Theory provides a powerful stochastic and probabilistic approach to analytically model the mechanisms of scheduling policies [19]. It can be observed by the queueing view of the system model considered in previous section (as shown in Figure 4) that the system is well suited to be analyzed via queueing theory.

Using this model we can get expected performance metrics such as mean waiting time and mean slowdown of applications, which can be used to directly compare the performance of our proposed meta-scheduling algorithm. 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 [31][15]. For instance, user with small application generally prefer to wait for relatively lesser time than those users who have longer applications.

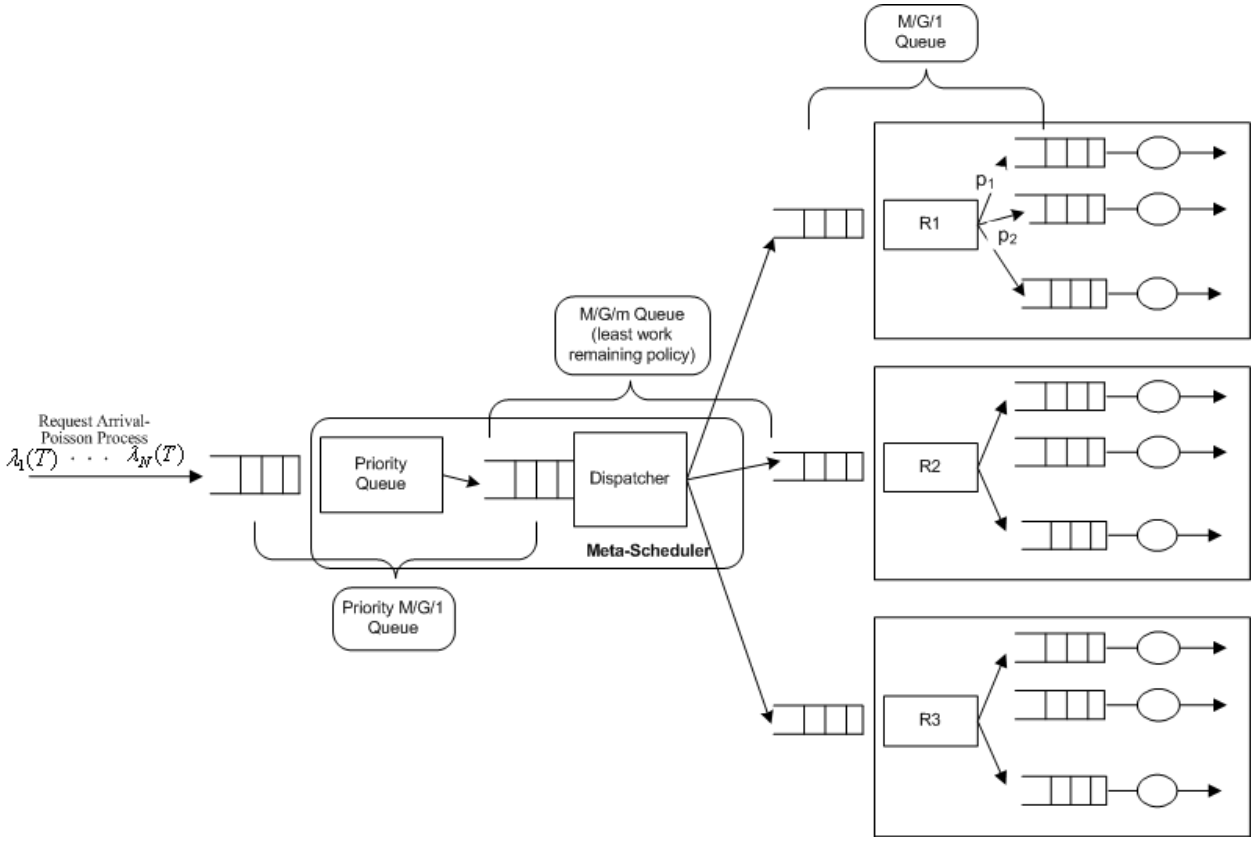


Fig. 4. Queuing Theory View of Meta-scheduling

We can model the system under consideration as a network of three queues 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 meta-scheduler can be modelled by a priority queue system. Since processing time of each application may not be exponential thus we have chosen M/G/1 priority queue system to analyze this part of the meta-scheduler. Then an application is assigned and dispatched to a resource to balance load across all queues in the system. This component of the meta-scheduler is analyzed using a comparable queueing system i.e. central queueing system with least work remaining policy (M/G/m).

At each resource, a application is executed on more than one machine at the same time. The

local scheduler of resource also uses the backfilling policies and different type of queues to increase utilization [25]. This component of meta-scheduling is difficult to model analytically. Thus, to analyze this part of meta-scheduling, we have taken an approximate model of real local scheduling systems. We have divided CPUs of a resource into multiple partitions, where each partition acts as a M/G/1 queueing system. 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).

Thus, in meta-scheduling, each application goes through three queuing systems before it start 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 i.e. least work remaining (LWR) policy)
- Scheduling at a resource for execution (M/G/1 Queues)

The mean arrival rate of applications is λ. Let the service requirement distribution of applications be any general distribution X. The mean service time of the applications is $E(X)$ and the second moment of service time is σ or E(X^2). The distribution of processor requirement by each application is given by C. Let the number of resource sites be m.

1) Valuation or prioritization (M/G/1 Queue with Priority): In 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 4. 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 application will also be very small. The applications arrival follows a Poisson distribution with mean λ.

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

$$\lambda_j = p_j \times \lambda \tag{3}$$

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

$$\rho_j = \frac{\lambda_j}{\mu_j} \tag{4}$$

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 [8] we obtain the mean waiting time and slowdown ($E(s_j)$) of class-j applications:

$$E(w_j) = \frac{\frac{\lambda \times \sigma'}{2}}{(1 - \sum_{i=1}^{j-1} \rho_i)(1 - \sum_{i=1}^{j} \rho_i)} \tag{5}$$

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

$$\bar{W_1} = \sum_{j=1}^{K} (p_j \times (E(w_j) + \frac{1}{\mu_j})) \tag{6}$$

$$\bar{S_1} = \sum_{j=1}^{K} (p_j \times (E(w_j) + \frac{1}{\mu_j})) \times E(X_j^{-1}) \tag{7}$$

2) Matching or Dispatching (M/G/m Queue): After applications are served by above queuing system based on priority, applications in the out-going queue will be served by the second queuing server on a FCFS basis. It can be considered as third component of meta-scheduler i.e., matching. For simplicity, the applications arriving into the second queueing system are taken to follow a poisson process. 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 [26][27]. 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 [28] and used in several other publications [14] [15]. The approximation for mean queue length is stated as:

$$E(N_{M/G/m}) = E(N_{M/M/m}) \frac{E(X^2)}{E(X)^2}, \text{where X: Service Requirement and N: Queue Length} \tag{8}$$

The load of system is given by:

$$\rho = \lambda \times E(X) \tag{9}$$

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

$$E(N_{M/M/m}) = \frac{P_N \rho}{1-\rho}, \text{where } P_N = [\sum_{z}^{m-1} \frac{(m\rho)^z}{z!} + \frac{m^m \rho^m}{m!1-\rho}]^{-1} \tag{10}$$

$$E(W_{M/M/m}) = \frac{E(N_{M/M/m})}{\lambda} \tag{11}$$

Thus, using Equation 8, 10 and 11, the mean waiting time and slowdown in the queue by using central queue policy is given by,

$$\bar{W_2} = E(W_{M/G/m}) = E(W_{M/M/m}) \frac{\sigma}{E(X)^2} \tag{12}$$

$$\bar{S_2} = \bar{W_2} \times E(X^{-1}) \tag{13}$$

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, local scheduler at a resource uses different backfilling policies to decrease slowdown of applications [25]. 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. Each partition f on resource z is initially assigned r_{zf} 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,

$$r_{z0} + r_{z1} + r_{z2} + r_{z3} ... r_{zf} + .. r_{z(n_z-1)} = N_z \tag{14}$$

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

$$\lambda_z = \frac{\lambda}{m} \tag{15}$$

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

$$u_{zf} = \int_{a_{z(f-1)}}^{a_{zf}} C(x)dx \tag{16}$$

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

$$\lambda_{zf} = \lambda_z u_{zf} \tag{17}$$

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

$$\rho_{zf} = \lambda_{zf} E(X) \tag{18}$$

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

$$E(w_{zf}) = \frac{\lambda_{zf}\sigma}{2(1-\rho_{zf})} \tag{19}$$

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

$$E(w_z) = \frac{1}{n_z} \sum_{f=1}^{n_z} E(w_{zf}) \tag{20}$$

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

$$W_3 = \frac{\lambda}{m} \sum_{z=1}^{m} E(w_z) \tag{21}$$

$$S_3 = W_3 \times E(X^{-1}) \tag{22}$$

The overall expected waiting time and slowdown measures are given by combining waiting time and slowdown of all three queueing system, i.e., Equation 6-7, 12-13 and 21-22:

$$\text{Waiting Time} = E(W) = W_1 + W_2 + W_3 \tag{23}$$

$$\text{Slowdown} = E(S) = S_1 + S_2 + S_3 \tag{24}$$

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

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

$$Minimize(E(S)) \text{ subject to } \sum_{k}^{n_z} r_{zk} = N_l, 1 < z < m \tag{26}$$

The above model gives an approximation for real meta-scheduling systems. Thus, to predict performance of our meta-scheduling policy, we can obtain optimal expected waiting time and slowdown. Thus, Equations 25 and 26 for expected waiting time and slowdown are needed to be solved for different values of n_l using optimization tools such as Mathematica [Wolform Research 2008]. These analytical values are used to ascertain the performance of DAM.

V. Performance Evaluation

A. Experimental Configuration

For our experiments, we use Feitelson's Parallel Workload Archive (PWA) [35] to model the parallel application workload for Grids. 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. Our experiments utilize the applications in the first 1 week of the Lawrence Livermore National Laboratory (LLNL) Thunder (January 2007 to June 2007). The LLNL Thunder trace from the LLNL in USA 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 submission time of parallel application is divided by 1000 to increase the number of applications submitted per schedule interval as per the methodology presented by Sanjay and Vadhiyar [41]. Since the workload trace does not contain any information about the user's deadline and initial valuation, these were generated synthetically. For a user application with a runtime r, the deadline was 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 choosen between 90000 and 160000 currency units, so that it is always greater than application execution cost. 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 [1]. The configurations assigned to the resources in the testbed for the simulation are listed in Table I. The configuration of each resource is decided so that the modelled testbed would reflect the heterogeneity of platforms and capabilities that is normally the characteristic of such installations. Each of the resources were simulated using GridSim [35] as a cluster that employed a multi-partition easy backfilling policy for local resource allocation [9].

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

TABLE I

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

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

- **Shortest Job First (SJF):** In this algorithm, the 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 [17].
- **Earliest Deadline First (EDF-FQ):** In this algorithm, the applications with the earliest deadline are scheduled on to the resource queue slot with the least waiting time.
- **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 [38] and Bellagio [2].
- **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 [7].

The following criteria were 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:

$$u = \frac{deadline - start\_time}{execution\_time} - 1 \tag{27}$$

  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 $1 > u > 0.75$ and very relaxed when $u > 1$. 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.

B. Analysis of Results

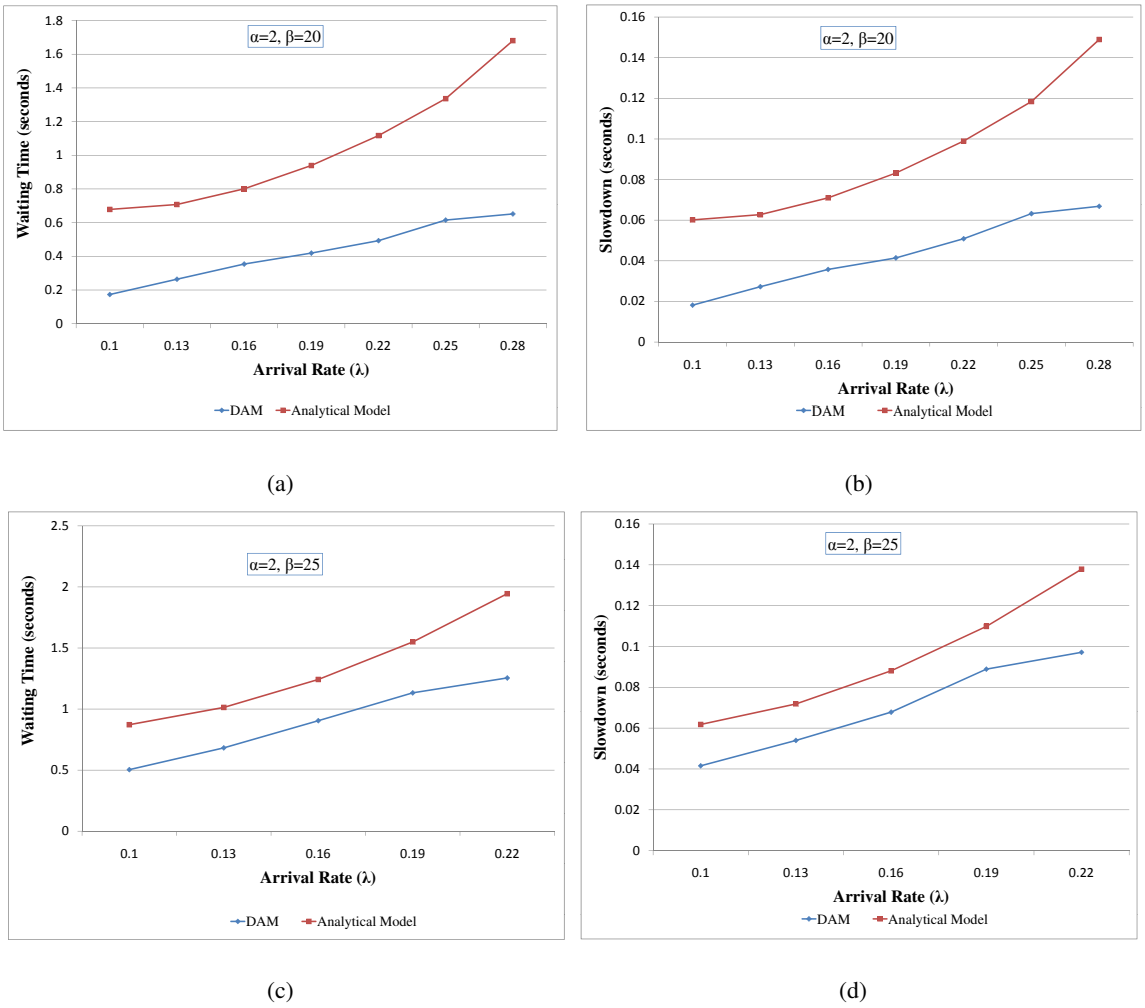


Fig. 5. Comparison of DAM with analytical results

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

1) Comparison of DAM with Theoretical Results: In Section IV-C we designed an analytical model for meta-scheduling problem in order to ascertain the behavior and substantiate the strength of our proposed Double auction based meta-scheduling (DAM) algorithm. The analytical model gives an near-optimal solution which can be compared with DAM and evaluated under

a variety of conditions using important metrics such as mean waiting time and mean slowdown of applications.

The optimal mean waiting time and mean slowdown is calculated using the approximate queuing model for meta-scheduling, by solving Equation 25 and 26 using the NMinimize function in Mathematica [Wolform Research, 2003]. 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 Section IV-C is an approximation to the meta-scheduling problem, thus the optimal solution obtained is actually a near-optimal solution for the meta-scheduling problem at steady state.

Experimental Methodology: Li et al. [22] analyzed various Grid Workloads and found that the Weibull distribution is best fitted to model runtime of applications. Thus, the application runtime is generated using a Weibull distribution (α, β) [22]. The arrival rate application is assumed to be Poisson distribution with parameter λ. Since our aim is to compare both analytical and simulation results, thus the probability distributions for arrival rate and runtime of applications are same in both analytical and simulation experiments. However, since the analytical model is an approximation of the real systems, thus the r_i values can differ between the analytical model (where they are numerically solved) and the simulated real systems (where the scheduling is done 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 were considered demonstrating a wide spectrum of load and arrival rate. The performance metrics were computed for these arrival rates each with different mean application runtime (represented by the combination of values of α and β). In order to obtain steady state results, we ignored scheduling of first 5000 applications and measure mean waiting time and mean slowdown for next 5000 applications. For each value of λ, experiment is repeated 30 times and average of results from these repeated experiments is used for comparison.

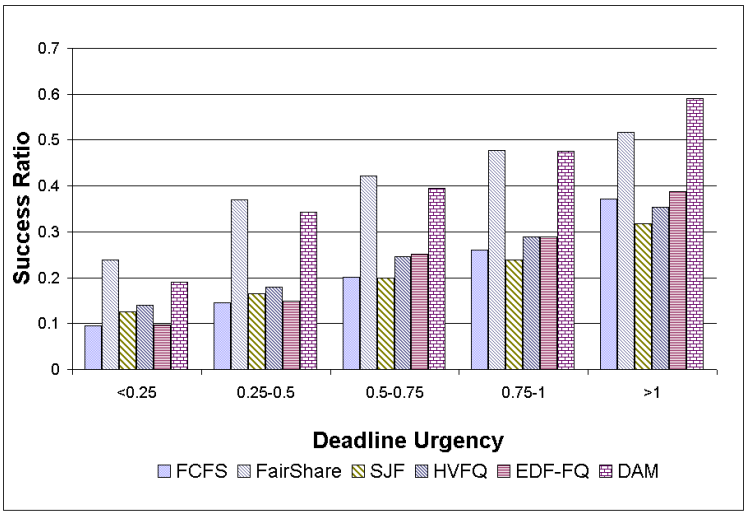
Discussion of Results: Figure 5 shows that the simulation results of DAM follows similar increasing trend as the analytical model. This not only validates the correctness of DAM heuristic but also indicates that the performance of DAM is near optimal in terms of metrics such as mean

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 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 reduces the slowdown and waiting time of applications more than the optimal solution of analytical model.

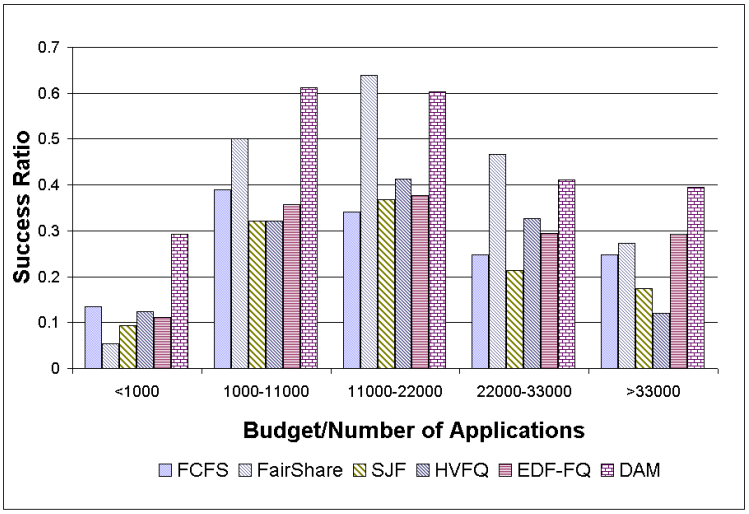
2) Benefit for Users: This section shows how our meta-scheduler is more fair to users by not only completing the most number of applications with different QoS needs but also benefiting every user in different urgency and budget groups.

a) Effect of User Urgency: Figure 6(a) shows the percentage of total applications completed successfully against the users' urgency values. Figure 6(a) shows that DAM and FairShare 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 and FairShare scheduled 15% more applications than their closest competitor (EDF-FQ) . 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. FairShare performed very similar to DAM and even scheduled about 5% more applications than 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 was greater than 1, then DAM scheduled about 8% 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.

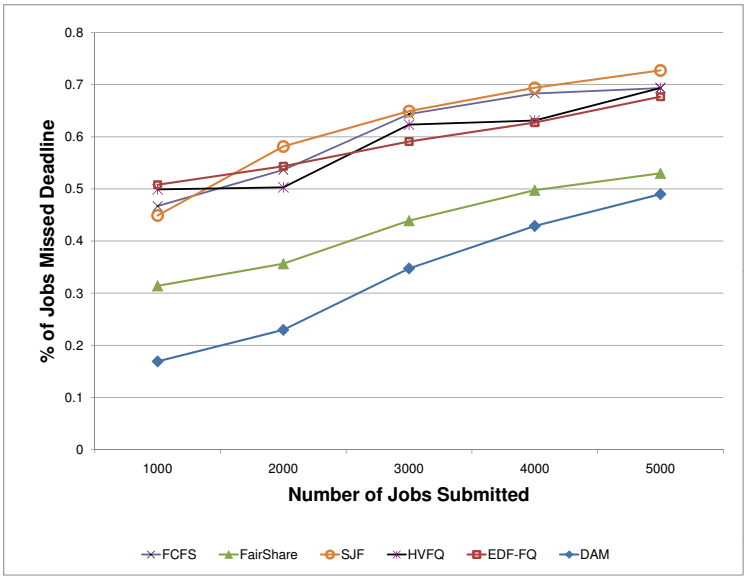
b) Effect of User Valuation: From Figure 6(b), we can see that DAM completes more number of applications across almost all valuations than the other algorithms. Even though Fareshare performed slightly better than DAM for medium valuation groups, DAM outperform FairShare for all other groups by scheduling atleast 12% more applications. For applications with very low valuation (< 1000), the DAM managed to schedule about 30% of the applications as



(a) Effect of user urgency



(b) Effect of user valuation



(c) Number of Deadlines Missed

compared to 5% for FairShare which perform 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 was supposed to favour users with high budget, has scheduled almost $10\% - 25\%$ less number of applications than DAM. This is because HVFQ does not consider other requirements of applications such as deadline.

c) Number of deadline missed: From Figure 6(c), we can clearly see as the demand for resources (number of applications) increases, the number of applications that missed their deadline also correspondingly increases due to the scarcity of resources. But DAM is able to complete around 4% to 15% more applications than other algorithms. EDF-FQ performed relatively better than the other algorithms such as SJF, HVFQ and FCFS, while SJF performs the worst as it does not consider the effect of deadlines.

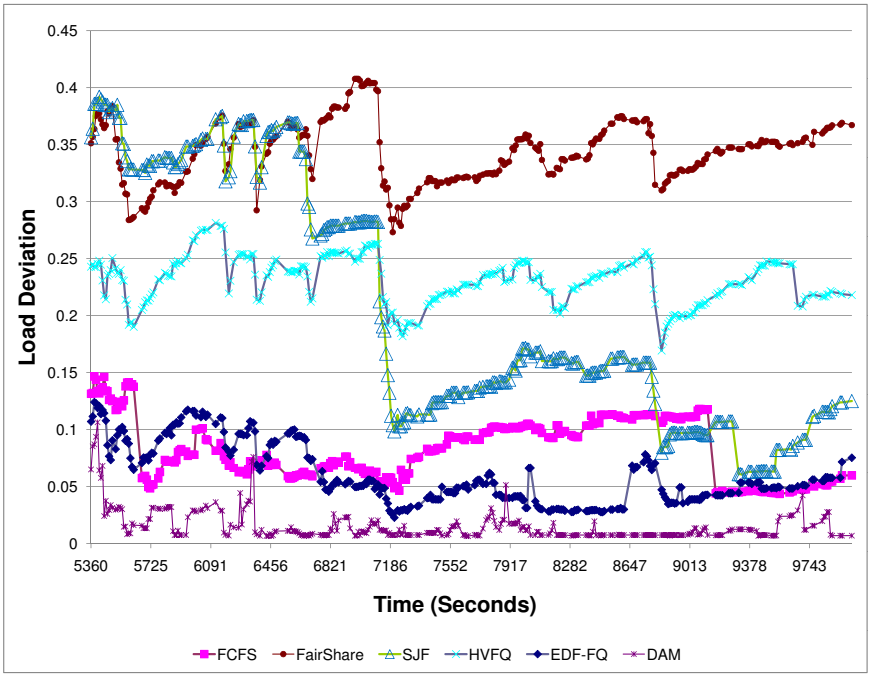


Fig. 7. Variation in Load across Resources.

3) 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 be noted that while the deviation across resources for other

algorithms are steadily increasing, on average DAM has kept it, consistently, almost close to 0. This implies that the DAM was able to successfully balance the load across all the resources. 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 resulted in on average 5 times more Load Deviation than DAM. Moreover, from Figures 6(a) and 6(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 benefitted the users in similar way as DAM, but FairShare resulted in 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.

VI. Conclusion

In this paper, we have presented a meta-scheduler for allocating parallel applications on distributed resources within a grid. The resource sites are organised as a collection of queues with different capabilities and waiting times. The goal of the scheduler is 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. Also, the scheduler has to be fair to all users and prevent starvation of applications that have relaxed deadlines or low valuation.

Given the many objectives of the scheduler, we have employed the efficient continuous double auction protocol as 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 traded and exchanged in the double auction. We analyzed the meta-scheduling problem in Grid using queuing theory and proposed an approximate analytical model which is used to analyze the performance of our Double auction-based meta-scheduling (DAM) heuristic. Based on analytical analysis, we show that the proposed DAM heuristic provides the near-optimal solution of mean waiting time and slowdown metric in wide variety problem instances.

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 resources across all the target metrics. The double auction

mechanism is not only able to schedule about 10% more user applications but also has the highest success ratio in almost all the groups for applications with different deadlines and different valuations. For the users with lowest budget (< 1000), the success ratio of their applications increased by almost 18%. Similarly, DAM also benefitted resource side by equally distributing the workload according to capacity of resource. Thus, DAM is able to improve the balancing of load across the constituent resources of the grid with almost zero load deviation. In benefiting resources, DAM even outperformed the FairShare algorithm which also benefited users similar to DAM. The load deviation in case of FairShare was almost 35% more than DAM. The key reason here is that the valuation metrics were able to capture information that was important to both users and resources, and therefore schedule applications effectively through the double auction mechanism. Thus, we demonstrate that by inclusion of both system metrics and market-parameters we can get more effective scheduling which will benefit both users and resources. We have also demonstrated how classical economic mechanisms, adapted suitably, are able to deal with multiple QoS requirements of the users more effectively then then 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 real meta-scheduler such as GridWay and run in real grid systems. Moreover, this paper presents a first step in designing valuation metrics by considering a quite simple and intuitive model. We also intend to experiment with different valuation methods, and examine the applicability of the mechanism to other distributed application models.

ACKNOWLEDGEMENTS

We would like to thank 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 Australia Research Council (ARC).

REFERENCES

[1] European data grid project. `http://eu-datagrid.web.cern.ch/eu-datagrid/default.htm`.

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

[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, et al. Adaptive computing on the grid using AppLeS. *IEEE transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

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

[5] J. Broberg, S. Venugopal, and R. Buyya. Market-oriented Grids and Utility Computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[6] J. Butler, D. Morrice, and P. Mullarkey. A multiple attribute utility theory approach to ranking and selection. *Management Science*, 47(6):800–816, 2001.

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

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

[9] M. de Assunção and R. Buyya. Performance analysis of allocation policies for interGrid resource provisioning. *Information and Software Technology*, 51(1):42–55, 2009.

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

[11] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

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

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

[14] M. Harchol-Balter, M. Crovella, and M. C.D. On choosing a task assignment policy for a distributed server system. *Journal of parallel and distributed computing(Print)*, 59(2):204–228, 1999.

[15] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3):253–285, 1997.

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

[17] E. Huedo, R. Montero, and I. Llorente. A framework for adaptive execution in grids. *Software Practice and Experience*, 34(7):631–651, 2004.

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

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

[20] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1(3):169–182, 2005.

[21] E. Laure et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[22] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In *Job Scheduling Strategies for Parallel Processing, New York, NY*, 2004.

[23] D. Lifka. The ANL/IBM SP Scheduling System. *Lecture Notes In Computer Science*, pages 295–295, 1995.

[24] R. Montero, E. Huedo, and I. M. Llorente. Grid Scheduling Infrastructures with the GridWay Metascheduler, 2006.

[25] A. Mu'alem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimatesin scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[26] R. Nelson and T. Philips. An approximation to the response time for shortest queue routing. *ACM SIGMETRICS Performance Evaluation Review*, 17(1):181–189, 1989.

[27] R. Nelson and T. Philips. An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation*, 17(2):123–139, 1993.

[28] S. Nozaki and S. Ross. Approximations in finite-capacity multi-server queues with Poisson arrivals. *Journal of Applied Probability*, 15(4):826–834, 1978.

[29] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance Management for Web Services. *Research Report RC22676, IBM TJ Watson Research Center, Yorktown Heights, NY*, 10598, 2003.

[30] R. Raman, M. Livny, and M. Solomon. Resource Management through Multilateral Matchmaking. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania*, 2000.

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

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

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

[34] G. Singh, C. Kesselman, and E. Deelman. A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th international symposium on High performance distributed computing, Monterey, CA*, 2007.

[35] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice and Experience*, 20(13):1591–1609, 2008.

[36] K. Vanmechelen, W. Depoorter, and J. Broeckhove. Economic Grid Resource Management for CPU Bound Applications with Hard Deadlines. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, France*, 2008.

[37] A. vd Kuijl, M. Emmerich, and H. Li. A novel multi-objective optimization scheme for grid resource allocation. In *Proceedings of the 6th international workshop on Middleware for grid computing*. ACM New York, NY, USA, 2008.

[38] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kep hart, and W. S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[39] J. Wallenius, J. Dyer, P. Fishburn, R. Steuer, S. Zionts, and K. Deb. Multiple Criteria Decision Making, Multiattribute Utility Theory: Recent Accomplishments and What Lies Ahead. *Management Science*, 54(7):1336, 2008.

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

[41] Y. Wiseman and D. Feitelson. Paired Gang Scheduling. *IEEE Transcatiosn On Parallel and Distributed Systems*, pages 581–592, 2003.

[42] R. Wolski, J. Plank, J. Brevik, and T. Bryan. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. *International Journal of High Performance Computing Applications*, 15(3):258, 2001.

[43] L. Xiao, Y. Zhu, L. Ni, and Z. Xu. Incentive-based scheduling for market-like computational grids. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):903–913, 2008.

[44] C. Yeo and R. Buyya. Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In *Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, USA*, 2005.

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