Efficient Resource Management using Advance Reservations for Heterogeneous Grids

Claris Castillo, George N. Rouskas, Khaled Harfoush
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
Email: {ccastil,rouskas,kaharfou}@ncsu.edu

***Abstract*—Support for advance reservations of resources plays a key role in Grid resource management as it enables the system to meet user expectations with respect to time requirements and temporal dependence of applications, increases predictability of the system and enables co-allocation of resources. Despite these attractive features, adoption of advance reservations is limited mainly due to the fact that related algorithms are typically complex and fail to scale to large and loaded systems. In this work we consider two aspects of advance reservations. First, we investigate the impact of heterogeneity on Grid resource management when advance reservations are supported. Second, we employ techniques from computational geometry to develop an efficient heterogeneity-aware scheduling algorithm. Our main finding is that Grids may benefit from high levels of resource heterogeneity, independently of the total system capacity. Our results show that our algorithm performs well across several user and system performance and overcome the lack of scalability and adaptability of existing mechanisms.**

I. Introduction

Owing to the advances in technologies such as resource virtualization and network management Grids have experienced enormous growth not only in respect to their adoption–they have became the defacto infrastructure for computing service provisioning in academia and corporate $R\&D$ environments–but also in their functionality, complexity and size. This phenomenon has led to the emergence of a whole new range of applications capable of performing tasks of a complexity not envisioned before. For instance, several scientific workfolk applications [24], [25], [27] involve the orchestration of multiple compute and data transfer stages. These stages normally have strong dependency on completion times; thus the ability to co-schedule and synchronize resources usage is crucial. Furthermore, emerging classes of deadline-driven scientific applications such as severe weather modeling [23] require simultaneous access to multiple resources and predictable completion times. In order to support such temporal dependencies and strict time constraints Grid, middleware needs to offer planning capabilities so users can reserve resources in advance based on resource availability and meet the time requirements of their applications. However, most existing Grids resource schedulers [10], [26], [28], [29], [45], [46] were originally designed to work under best-effort policies. In response to the emerging needs for more sophisticated resource management solutions some Grid resource management'software has evolved to accommodate for advance reservations. Such software includes LSF, PBS-Pro, Maui, Catalina, EASY and COSY (for a comprehensive review of these schedulers refer to [10] and references thereof).

This work was partially supported by NSF grants CAREER ANIR-0347226 and CNS-0434975, Anita Borg Scholarship and Cisco Systems Inc.

Advance reservations have been widely proposed for provisioning for performance predictability, meeting resource requirements and providing guaranteed quality of service to applications [1]–[6], [20], [21], [35], [36], [36]–[38], [40]. GARA [34] is one of the seminal works on advance reservation and defines a basic architecture and simple API for the manipulation of advance reservation of different resources. A theoretical proof that reservations can be used to improve the performance predictability of applications is presented in [38]. A comparison of provisioning models and best-effort mechanisms can be found in [32]. In [37] and [38] the performance and predictability of workflows applications when advance reservations are used is investigated respectively, concluding that it is beneficial for Grids to use advance reservations. A more general study on the usefulness of advance reservation is presented in [35]. In [5], [6] authors investigated the negative impact advance reservations have on system and user performance. In [31], [43], [44] authors show how laxity and fuzziness in the reservation requests may be exploited to address some of the drawbacks of advance reservations. Two of the most recent major works on advance reservations in Grids are [32] and [21]. In [32], the authors propose a multi-objective genetic algorithm formulation for selecting the set of resources to be provisioned that optimizes the application performance while minimizing the resource costs. In [21] a cost-aware resource model is parented in which reservation for each application task is performed separately by negotiating with the resource provider. In [33] the authors present a broker service for the Grid resources that takes into account the fact that deadline and budget are specified, and then optimizes the usage of resources only by considering the current state of the resources but without any planning horizon.

The impact of resource heterogeneity has been investigated in contexts other than Grids. In [41] the authors exploit the heterogeneity found in HPC environment by dividing a task into subtasks and then mapping the latter to resources that best meet their requirements. This work assumes offline scheduling and does not support advance reservations; our work deals with online scheduling and allow users to schedule jobs in advance. In [19] the authors proposed a general framework to quantify the worst-case effect of increasing heterogeneity in models of parallel systems with finite total capacity. An important contribution of this work was a model to characterize resource heterogeneity which we adopt in this paper.

Overall, advance reservation of resources [1]–[6] has generated great interest in the Grid community as a mechanism that Grid providers may employ to offer planning capabilities to application users. Furthermore, it has shown to increase the predictability of the system maximizing the flexibility and adaptability of the system to cope with the dynamic behavior of grid environments [35] [13] [38]. Despite the attractive features of advance reservations, there is great scepticism in the Grid community about their ability to meet their promise; this fact is mainly due to three reasons. First, advance reservations have shown to cause severe performance degradation [5], [6]. Second, typical advance reservation mechanisms lack flexibility as they do not permit graceful degradation in application performance when resource management policies mandate changes in allocations [9]. Third, existing approaches suffer from poor scalability as they are not effective in managing large sets of advance reservations or handling resource fragmentation. Also, most solutions lack of sophistication, and are not able to address the user needs (e.g. for time guarantees) and system requirements (e.g., for high performance/throughput) in an integrated manner. To overcome these challenges, algorithms for advance reservations need to be *efficient* so they can adapt to dynamic changes in resource availability and user demand without hurting system and user performance. Moreover, they must take into account resource heterogeneity since resources in Grid environments are typically highly heterogeneous.

In previous work [40] we developed efficient algorithms for advance reservations of *homogeneous* resources. These algorithms are effective in meeting time requirements (e.g.,deadlines), may be adapted to employ several optimization criteria for scheduling jobs, and their low running times make them practical for large Grid environments. In this paper we address the issue of meeting application time requirements in Grid environments with resources of *heterogeneous* capabilities (e.g, as in the case of compute servers with varying processing power). We consider an environment where users submit jobs dynamically, and these jobs may start at a future time and must be completed within a certain deadline. We first investigate the impact of heterogeneity on the scheduling of resources, and conclude that scheduling algorithms need to be heterogeneity-aware to achieve appropriate system and user performance. Based on this observation, we then develop an efficient heterogeneity-aware scheduling algorithm for advance reservations in this context. We also describe how to apply techniques from computational geometry to develop data structures that allow the service provider to manage efficiently the set of advance reservations and handle effectively the resulting resource fragmentation.

The rest of the paper is organized as follows. In Section II we describe the online scheduling problem we study in this work. In Section III we make a case for heterogeneity-aware algorithms in Grids. By means of a simple experiment we show that resource heterogeneity may have positive impact on performance if heterogeneity-aware algorithms are used. In Section IV we present a novel transformation of the advance reservations problem that exploits techniques from computational geometry. Using insight from this transformation, we then develop a heterogeneity-aware algorithm in Section V, and provide details on its implementation and the associated data structures used to manage the fragmentation of resources. In Section VI we describe several directions for further improving the performance of the scheduling algorithm that are the subject of ongoing research within our group. In Section VII we investigate the performance of our algorithm through simulation, and we conclude the paper in Section VIII.

II. Problem Description

Consider a scheduler $\mathcal{S}$ for a Grid with n servers which may be geographically distributed in a network. We consider a heterogeneous environment in that server i has service rate μ_i, where service rate refers to the amount of work a server can perform per unit of time. We also assume network delays are negligible. A user with job j requiring service submits a request to the scheduler. The request is characterized by a three-parameter tuple (r_j, l_j, d_j), where:

1) r_j is the *ready time* of the job, i.e., the earliest time the job can be made available to the Grid for processing;
2) l_j is the *size* of the job, i.e, the amount of work the job requires; and
3) $d_j (\geq r_j + l_j)$ is the *deadline* of the job, i.e., the latest time by which the job can be completed to provide any utility to the user.

The deadline is a measure of the quality of service required by the user. We assume that deadlines are *hard*, in that a user receives utility only if the job completes service by its deadline. Therefore, if $\mathcal{S}$ determines that the deadline cannot be met, it drops the job and notifies its user accordingly. Note that this restriction may be relaxed with minimal modifications to our algorithm; in

Section VI we describe a set of mechanisms that may be used to re-negotiate and re-plan advance reservations in order to minimize the number of jobs that are dropped.

In our model, the availability of resources is represented by time intervals during which servers are *idle*. We refer to these intervals as *idle periods* in this paper. We say that an idle period is *feasible* for a given job j if it can accommodate j within its deadline d_j. The feasibility of an idle period k for a given job j is determined by both the service rate of the server associated with the idle period and its duration. Therefore, we characterize an idle period k on a server i with service rate μ_i by a three-parameter tuple (st_k, et_k, c_k), where:

- st_k is the starting time of the idle period;
- et_k is the ending time of the idle period; and
- $c_k = \mu_i \times (et_k - st_k)$ is the *nominal capacity* of the idle period, i.e., the amount of work that server i can perform during idle period k.

Note that idle periods in slow (respectively, fast) servers may have a long (respectively, short) duration but small (respectively, large) nominal capacity. Moreover, the nominal capacity c_k of an idle period k represents the *maximum* job size that it can accommodate, assuming that the job is scheduled to start execution exactly at time st_k. As time progresses, the nominal capacity c_k of the idle period decreases at a rate equal to its server's rate μ_i. Consequently, if no job is allocated to the idle period by time $t = st_k$, then the maximum job size that it can accommodate decreases linearly at rate μ_i. Therefore, the nominal capacity of idle periods belonging to fast (respectively, slow) servers expires at a faster (respectively, slower) rate.

We consider the online scheduling problem whereby users submit service requests to $\mathcal{S}$ at random instants. We assume that $\mathcal{S}$ maintains a schedule which records, for each server i, the time periods in the future during which the server is reserved for jobs that have already been accepted to the system. In essence, this schedule represents the set of *advance reservations* that have been made, and it guarantees that server resources will be available to the accepted jobs at specific future times.

Figure 1(a) shows an example schedule for a 2-server system in which server i has rate $\mu_1 = 1$, and server 2 has rate $\mu_2 = 0.5$. The schedule is in the form of a *timetable*, and shows that at the current time (i.e., $t = 0$), there are four jobs scheduled for server 1: the job currently in service which will end at time t_1, job A which has reserved the server from time t_4 to time t_5, job B which has reserved the server from time t_6 until time t_7, and job C which is scheduled from time t_{11} to time t_{12}. Similarly, there are two jobs scheduled for server 2. The figure also shows a new job j requesting service. The job has ready time $r_j = t_3$ and deadline d_j. There are two representations of the new job. The representation at the top has a shorter duration and shows the new job as seen by server 1, while the one below has a longer duration (i.e., double that at the top) and shows the job as seen by server 2.

When a service request (r_j, l_j, d_j) for a new job j arrives, $\mathcal{S}$ immediately runs an algorithm to determine whether it is feasible to schedule the job so as to meet its deadline. If so, then $\mathcal{S}$ uses a set of criteria to select one of the (possibly multiple) servers that can handle this job, updates its schedule, and returns a reference to this server to the user; otherwise, the job is dropped. The scheduling decision impacts the performance perceived by users as reflected by the fraction of jobs meeting (or missing) their deadlines and the response time of the jobs. It also impacts the overall system performance as reflected by the system utilization, which is a measure of how well the overall service capacity of the system is used. The challenge, therefore, is to develop efficient online scheduling algorithms that minimize the fraction of dropped jobs while maximizing utilization.

A. Computational Heterogeneity

To incorporate computational heterogeneity into our framework we use the model introduced in [19]. In this model the authors use *majorization* partial order to compare the imbalance, i.e., heterogeneity, of capacity distributions. The *majorization* partial order, $\succeq$, is defined as follows. Given two nonnegative vectors corresponding to the service rates of two n-servers systems $C = (\mu_1, \mu_2, \mu_3, \cdots, \mu_n)$ and $C' = (\mu'_1, \mu'_2, \mu'_3, \cdots, \mu'_n)$, we have $C' \succeq C$ when

$$\forall k \sum_{i=1}^{k} \mu'_{[i]} \geq \sum_{i=1}^{k} \mu_{[i]} \text{ and } \sum_{i=1}^{n} \mu'_i = \sum_{i=1}^{n} \mu_i \tag{1}$$

where $\mu_{[i]}$ denotes the i-th largest component of C. We say that the computational capacity distribution C_A of a system A is more heterogeneous than the computational capacity C_B of a system B whenever $C_A \succeq C_B$.

We say that a Grid system is (H, n)-heterogeneous, $H \ll n$, if the n servers are partitioned in H groups such that servers in group $h, h = 1, \cdots, H$, have the same service rate μ_h. Note that most existing Grids follow this model as they consist of a collection of clusters of identical processors. Thus, an (H, n)-heterogeneous system has n servers with H different rates. For a given (H, n)-heterogeneous system we may generate a range of service rate distributions that are more or less heterogeneous according to the majorization partial order in expression (1). We let L denote the levels of heterogeneity, i.e., the number of service rate distributions considered for a (H, n)-heterogeneous Grid, labeled in order of increasing heterogeneity:

$$(H, n)_L \succeq \cdots \succeq (H, n)_1 \succeq (H, n)_0 \tag{2}$$

where we use $(H, n)_0$ to denote the completely homogeneous system, i.e., one in which all n servers have the same rate μ. We use this model in the experimental studies we report in Sections III and VII.

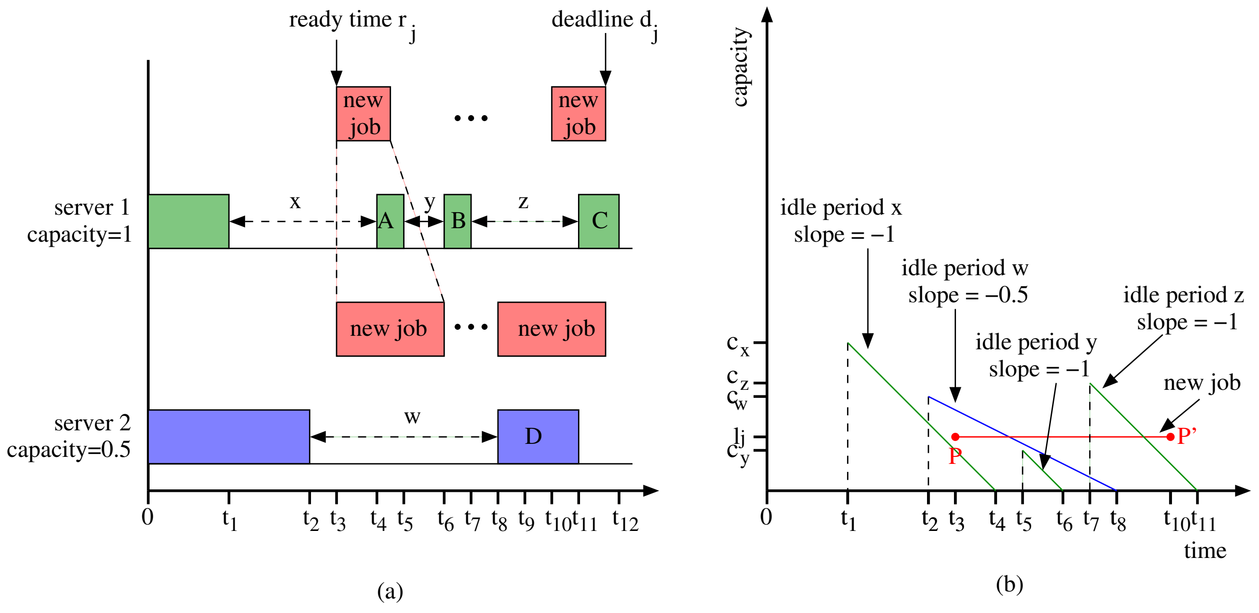


Fig. 1. (a) Schedule of a 2-server system as a timetable, and (b) geometric representation of the idle periods and the new job.

III. The Case for Heterogeneity-Aware Algorithms

To investigate the impact of heterogeneity in resource allocation mechanisms in Grid environments we perform two different experiments. We refer to these experiments as heterogeneity-aware (HA) and heterogeneity-unaware (HU) experiments. In both experiments we consider the problem described in Section II and use the same scheduling algorithm and data structure; the only difference being that in one experiment we adapt the algorithm and data structure to accommodate heterogeneity.

More specifically we consider the well-known first-fit (FF) scheduling algorithm, and we use a linked-list data structure to store idle periods. In the heterogeneity-unaware (HU) experiment, all idle periods over all servers are stored in a single linked list in ascending order of their starting times. To schedule a new job, the FF algorithm searches the linked list and returns the first feasible idle period for the job; we refer to this algorithm as FF-HU. In the heterogeneity-aware (HA) experiment, the idle periods are stored in H linked lists, where H denotes the number of different rates in the system. Specifically, linked list $h, h = 1, \cdots, H$, stores the idle periods over all servers with rate μ_h in ascending order of their starting times. To schedule a new job, the FF algorithm considers the H lists in some order, and searches the first linked list for a feasible idle period; if no such idle period is found, the algorithm continues to search the next list in the order, and so on. The FF-HA algorithm terminates when the first feasible idle period is found, or when all the lists have been searched unsuccessfully. Clearly, the order in which the FF-HA algorithm considers the H linked lists will have an impact on performance.

We used simulation to compare the performance of the FF-HU and FF-HA algorithms; the details of the simulation setup are described in Section VII. Following the model of Section II-A, we consider a (H, n)-heterogeneous Grid with $n = 120$ servers divided into $H = 3$ groups, with the server in each group $h, h = 1, 2, 3$, having the same rate μ_h. We created $L = 4$ (H, n)-heterogeneous systems by selecting the rate μ_h of each server group within each system so that $L = 4$ refers to the most heterogeneous system with respect to expression (1) and $L = 1$ to the least heterogeneous one.

Figure 2 plots the loss rate and utilization against system load, respectively. Each figure shows two sets of four plots, one set for the FF-HU algorithm and one for FF-HA; in this case, FF-HA considers the H lists of idle periods in increasing value of the rate μ_h of the corresponding servers. Each plot within a set corresponds to one of the $L = 4$ levels of heterogeneity, i.e., one of the (H, n)-heterogeneous systems obtained as we described above. We have obtained results for other performance measures, e.g., waiting time, but do not include them here as they exhibit similar trends.

As we can see, for a given level of heterogeneity, the heterogeneity-aware algorithm (FF-HA) outperforms the heterogeneity-unaware one (FF-HU) across the spectrum of system loads. We also observe that the performance of each algorithm improves as the system becomes more heterogeneous, despite the fact that the total service rate is the same for all $L = 4$ heterogeneity levels. This phenomenon is due to the effect of statistical multiplexing, and is discussed in more depth in Section VII. These results, obtained with a basic scheduling algorithm and data structure, suggest that computational heterogeneity

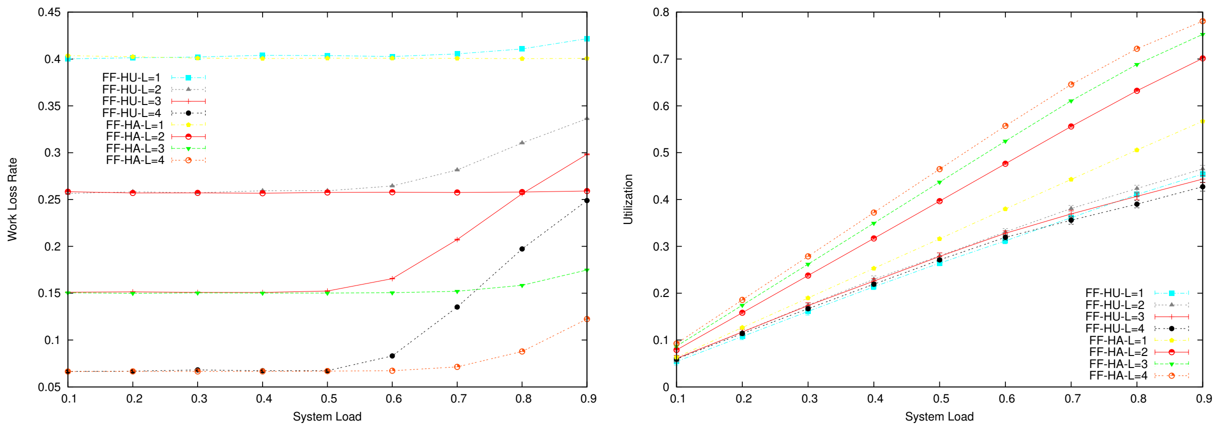


Fig. 2. Comparison of heterogeneous aware and unaware algorithms

may have a significant impact on both user and system performance metrics and should be taken into account when designing scheduling algorithms. Nonetheless, taking heterogeneity into account comes with a price since it adds complexity to the problem and hence to the algorithms. For instance, although the worst-case running time of the FF-HA and FF-HU algorithms is the same (linear in the number of idle periods), the average running time of FF-HA can be significantly longer than that of FF-HU (since it may have to traverse several lists before it finds a feasible period that might be stored near the head of the single list maintained by FF-HU). The challenge, therefore, is to design scheduling algorithms that are both heterogeneity-aware and efficient; this is the subject of the next two sections.

IV. A Geometric Model for Advance Reservations

In this section we employ techniques from computational geometry to model the problem we introduced in Section II. We then use this model to develop an algorithm for advance reservation of resources, along with an associated data structure for storing and accessing efficiently the set of idle periods.

Without loss of generality, in the following discussion we make the assumption that the service rate μ_i of each processor i is such that $0 < \mu_i \leq 1$. This assumption allows us to define the size l_j of a job j as the amount of time for this job to complete on a server of rate $\mu = 1$. Clearly, the duration of the job on a server of rate $\mu_i < 1$ is then equal to l_j/μ_i.

A. Geometric Representation of Idle Periods and Jobs

We represent idle periods and jobs on the first quadrant of a Cartesian coordinate system in which the x axis represents *time* and the y axis represents *nominal capacity*. Figure 1(b) illustrates the geometric representation of the idle periods and new job of Figure 1(a). A job j characterized by the tuple (r_j, l_j, d_j) is represented in this coordinate system as a line segment between two points $P = (r_j, l_j)$ and $P' = (d_j - l_j, l_j)$. Since, in Figure 1(a), the new job is defined by the tuple (t_3, l_j, d_j), the two endpoints of the line segment representation of this job in Figure 2(b) are $P = (t_3, l_j)$ and $P' = (t_{10} = d_j - l_j, l_j)$. As defined, point P represents the *earliest* possible starting time and required capacity for this job if it were scheduled on the fastest server, i.e., one with rate $\mu = 1$; similarly, point P' corresponds the *latest* possible starting time and required capacity for this job to be feasibly completed on the fastest server. Note that although we assume that servers may have different capacities, we use *a single* representation for each job j, namely the line segment with respect to the server of rate $\mu = 1$.

An idle period k characterized by the tuple (st_k, et_k, c_k) is also represented in the coordinate system as a line segment between two points, $k_1 = (st_k, c_k)$ and $k_2 = (et_k, 0)$. Recall that c_k denotes the nominal capacity of idle period k. Therefore, point k_1 represents the point in time (i.e., starting time) at which the idle period has the largest nominal capacity, and point k_2 the point in time (i.e., ending time) at which the idle period has reached zero capacity. The slope of the line segment representing idle period k is equal to $-\mu_i$, where μ_i is the rate of the server corresponding to this idle period; this representation clearly shows that the nominal capacity of the idle period decreases at rate μ_i. Consider, for example, idle period x in Figure 1(a) with starting time $st_x = t_1$, ending time $et_x = t_4$, and nominal capacity c_x. This idle period is represented in the plane by the line segment between the two points $x_1 = (st_x, c_x)$ and $x_2 = (et_x, 0)$. The slope of the line segment is -1, since the rate of server 1 is $\mu_1 = 1$. Idle periods y, z, and w are similarly represented by the line segments shown in Figure 1(b). Note also that the slope of the line segment corresponding to idle periods y and z is -1, while the one corresponding to w is -0.5 since the latter is on server 2 of rate $\mu_2 = 0.5$.

Feasibility Criteria. We may now use the above geometric representation to determine whether an idle period is feasible for a new job. Consider an idle period k with tuple (st_k, et_k, c_k) represented by the line segment defined by points k_1 and k_2, as explained earlier, and a new job j with tuple (r_j, l_j, d_j) that is represented by a line segment between points P and P'. Idle period k is feasible for job j if and only if both of the following conditions are satisfied.

1) *Starting time feasibility.* Let i be the server corresponding to idle period k, and μ_i be its service rate. For the idle period k to be feasible for the new job j, its starting time st_k has to be sufficiently early for the server to be able to complete the job before its deadline, i.e.:

$$st_k \quad \leq \quad d_j \; - \; \frac{l_j}{\mu_i} \tag{3}$$

Expression (3) is necessary but not sufficient for feasibility, since the idle period k may end early, before job j can complete on server i. Returning to Figure 1, we observe that idle period x satisfies the above condition with respect to the new job. However, the residual capacity of this idle period at the time the new job arrives is not sufficient to accommodate it.

2) *Capacity feasibility.* Assuming that the starting time feasibility is satisfied, an idle period k is feasible for a new job j if the line segment representing k lies above or intersects with, the line segment representing j. Equivalently, this condition is satisfied if the leftmost endpoint of the line segment representing the new job lies below the line segment representing the idle period. In Figure 1(b) we see that idle period y does not satisfy this condition as its line segment lies below the line segment representing the new job; hence, y is not feasible for the new job.

In Figure 1(b), the two conditions are satisfied for both idle periods w and z with respect to the new job represented by the line segment between points P and P'. Consequently, idle period w has enough capacity to accommodate the new job, as long as the latter starts before the time instant at which the corresponding lines intersect; similarly for idle period z.

Our objective is to develop techniques to identify efficiently feasible idle periods for each arriving job request, *without* having to examine all idle periods. As we have shown in [40], we can efficiently find idle periods that meet the starting time feasibility criterion by organizing the idle periods in an appropriate balanced tree structure that can be searched in logarithmic time. However, identifying idle periods that meet the capacity requirement, e.g., determining line segments lying above point P in Figure 1(b), requires that each idle period be examined separately. This is due to the fact that to perform this test the equation representing each line segment needs to be evaluated for the coordinates of the given point.

Next, we employ techniques from computational geometry to obtain an equivalent representation of idle periods and new jobs that allows us to develop an elegant solution to the problem of testing for the capacity feasibility criterion.

B. Duality Transform and Duality Plane.

Geometric duality [17] refers to the direct mapping between a point p (respectively, line l) and a line $p^\star$ (respectively, point $l^\star$). The duality transform maps objects from the *primal* plane to the *dual* plane. We now describe a simple duality transform we use in the remaining of this paper. Let $p := (p_x, p_y)$ be a point in the plane. The dual of p, denoted $p^\star$, is the line defined as

$$p^\star := (y = p_x x - p_y) \tag{4}$$

where p_x and p_y are p's x and y coordinates, respectively. The dual $l^\star$ of a line $l := (y = mx + b)$ is the point p such that $p^\star = l$, that is,

$$l^\star := (m, -b) \tag{5}$$

where m and b are the slope and y-intercept of line l, respectively. One major advantage of this particular duality transform is that it is *order preserving*, that is, point p lies above line l if and only if point $l^\star$ lies above line $p^\star$ [17].

Let us now return to our original problem and the geometric representation of idle periods and jobs shown in Figure 1(b). We transform this primal plane to the dual plane by mapping the line l_k corresponding to an idle period k to a point $l_k^\star$, and the point P corresponding to the earliest time new job j can start execution, to a line $P^\star$. Using basic geometry principles we find that for any idle period k, the value of b in expression (5) is $\mu_i et_k$. Since the slope m of idle period k is $-\mu_i$, where μ_i is the rate of the corresponding server, expression (5) can be written as:

$$l_k^\star := (-\mu_i, -\mu_i et_k). \tag{6}$$

To find $P^\star$, we substitute p_x and p_y in expression (4) with r_j and l_j, respectively:

$$P^\star := (y = r_j x - l_j). \tag{7}$$

Figure 3(b) shows the dual plane corresponding to the primal plane in Figure 3(a); the latter figure is identical to Figure 1(b), and is repeated here for convenience. As we can see, the idle periods are now mapped to points in the dual plane. Specifically, all idle periods on server 1 of rate $\mu_1 = 1$ are now points with y coordinates equal to $-\mu_1 = -1$; similarly, the idle period on server 2 of rate $\mu_2 = 0.5$ has y coordinate equal to $-\mu_2 = -0.5$. Point P, on the other hand, which represents the earliest time the new job can start execution is represented on the dual plane as a line.

Consider now the capacity feasibility criterion we defined above. In the primal plane of Figure 3(a), it is

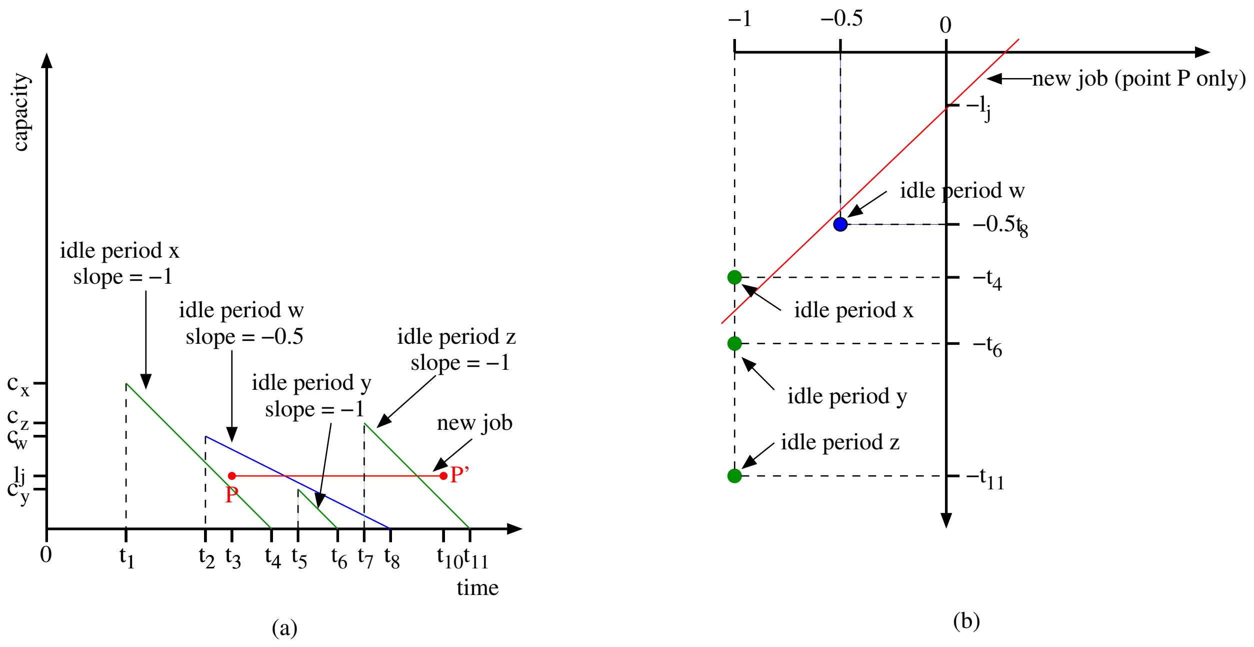


Fig. 3. (a) Primal plane and (b) dual plane representations of the idle periods and new job of Figure 1(a)

clear that the idle period x is not feasible for the new job, as point P lies above the line segment representing x. Due to the order preservation of the duality transform, in Figure 3(b) we see that the point corresponding to idle period x also lies above the line representing point P. Similarly, idle periods y and w are feasible for the new job, and their corresponding points in the dual plane lie below the line representing point P. Therefore, checking for capacity feasibility in the dual plane requires checking whether the points representing idle periods lie below the line representing the new job. This test can be performed efficiently by organizing the idle periods (points) lying on the vertical line $x = -\mu$ (i.e., those corresponding to servers with rate μ) in a search tree structure, and searching for those with a y-coordinate less than that of the point at which the line representing the new job intersects the line $x = -\mu$; this search structure is described in the next section.

The observant reader will have noticed that, in the dual plane of Figure 3(b), the point representing idle period y lies below the line representing point P; however, a look at the primal plane of Figure 3(a) indicates that idle period y is *not* feasible. Note that extending the line segment representing idle period y in the primal plane would result in a line lying above point P, hence the dual plane representation is consistent in this regard. The issue here is that idle period y starts too late to be feasible, therefore it will not pass the starting time feasibility criterion above. Consequently, both the starting time and capacity feasibility criteria must be checked to ensure that an idle period is feasible.

V. Algorithm and Data Structure Description

We now introduce an efficient algorithm for finding a feasible idle period for a new job in a (H, n)-heterogeneous system with advance reservations. The algorithm is derived from the heterogeneity-aware FF-HA algorithm we described in Section III, and will refer to it as FF-HA+. The FF-HA+ algorithm differs from FF-HA in that it maintains H balanced trees, rather than H linked lists, such that balanced tree $T_h, h = 1, \cdots, H$, stores information about the idle periods over all servers with rate μ_h. Similar to FF-HA, when a new job arrives, FF-HA+ searches the balanced tree structures in ascending order of server rate, and returns as soon as it finds a feasible idle period.

A. Balanced Tree Structure

The FF-HA+ algorithm maintains H 2-dimensional binary search trees to organize the idle periods in a (H, n)-heterogeneous system, one such tree $T_h, h = 1, \cdots, H$, for each distinct server rate value μ_h. Whenever the algorithm needs to search the idle periods available in servers associated with rate μ_h, the associated tree T_h is searched.

We will refer to the first and second dimension trees of T_h as T_h^{primal} and T_h^{dual}. As their name indicates, they organize the idle periods according to their parameterizations on the primal and dual planes, respectively. More specifically, tree T_h^{primal} is used to select idle periods that meet the starting time feasibility criterion, and tree T_h^{dual} is used to select among these idle periods the ones that meet the capacity feasibility criterion.

Let us now describe the 2-dimensional tree T_h more in detail. In tree T_h^{primal}, the actual idle periods are in the leaf nodes, arranged in ascending order of their starting time. A leaf node corresponding to idle period k stores the following information:

- the starting time of k;
- the ending time of k; and
- auxiliary data, such as the identity of the corresponding server.

Internal tree nodes store information regarding the idle periods in their subtree. This information is used to navigate the tree and locate idle periods appropriate for the new job. The information at an internal node v consists of:

- the median starting time of the idle periods stored in the subtree of T_h^{primal} rooted at v; and
- a pointer to the secondary priority search tree T_h^{dual} containing idle periods.

Tree T_h^{dual} stores the idle periods sorted in descending order of the y-coordinate of their dual representation, that is, of the corresponding point in the dual plane. Each intermediate node v in T_h^{dual} stores the following information:

- the median y-coordinate of the dual representation of the idle periods stored in the subtree rooted at v; and
- a pointer to the idle period in v's subtree with the maximum nominal capacity.

B. Searching the Balanced Tree Structure

Consider a request to schedule a new job j with parameters (r_j, l_j, d_j). The FF-HA+ algorithm searches the H balanced trees as we explained earlier, and returns the first feasible idle period found. We now describe how the search of balanced tree T_h is performed; this process is identical for all trees $T_h, h = 1, \cdots, H$. Specifically, the search proceeds in two steps:

1) In the first step, the algorithm traverses the tree T_h^{primal} and marks the intermediate nodes v whose subtrees contain idle periods that meet the starting time feasibility criterion.
2) In the second step, the algorithm searches the secondary trees T_v^{dual} at each intermediate node v marked during the first step, to locate the subset of idle periods that meet the capacity feasibility criterion.

Step 1: Search in T_h^{primal}. In this step, the algorithm identifies idle periods that meet the starting time feasibility criterion expressed in (3). To this end, we employ a standard search algorithm which starts at the root node and compares the quantity in the right-hand side of (3) to the median starting time stored at each internal node v. If the median starting time is smaller, then all the idle periods stored in v's left subtree meet the first feasibility criterion; the algorithm marks the left subtree and proceeds to search the right subtree. If the median starting time of the tree rooted at v is larger, then we can safely conclude that all the idle periods in the right subtree are infeasible and proceed recursively to search the left subtree of v. The algorithm returns the set of marked intermediate nodes as soon as it reaches a leaf, and proceeds to Step 2 described below. If no intermediate node is marked, the FF-HA+ strategy continues to search in the 2-dimensional tree T_{h+1} corresponding to the next larger value of server rate.

Step 2: Search in T_v^{dual}. In this step, the algorithm searches the idle periods meeting the starting time feasibility criterion, to identify the ones that also satisfy the capacity feasibility criterion. To this end, the algorithm searches each of the subtrees rooted at the intermediate nodes marked in Step 1 and returns as soon as it finds one feasible idle period (if one exists). We will refer to T_v^{dual} as the secondary tree, i.e., the dual tree, associated with marked node v. The algorithm starts at the root of T_v^{dual} and compares the median y-coordinate stored at each internal node u to the y-coordinate of the point in the dual plane at which the line corresponding to the new job intersects the vertical line $x = -\mu_h$ (refer also to Figure 3(b)). If the latter value is smaller then it can be concluded that all the idle periods in the left subtree are above the line, and hence are infeasible; the algorithm then recursively searches u's right subtree. If the former value is smaller, then all the idle periods in the right subtree of u are feasible, and there may also exist feasible idle periods in its left subtree. In this case, the algorithm accesses the idle period with the *maximum capacity* in the right subtree by following the pointer stored at node u. If this idle period is feasible, the algorithm returns it and assigns it to the new job. Otherwise, the search continues recursively with the left subtree of u. If the algorithm reaches a leaf, then no feasible idle period exists in the given subtree and the algorithm continues searching the next tree marked in Step 1.

Running time complexity. In the worst case, the search algorithm marks an intermediate node at each level of the tree T_h^{primal} in Step 1. Given that it has to perform a standard search for each of these trees, the overall complexity is $O(\log^2 V_h)$ for 2-dimensional tree T_h, where V_h is the number of idle periods in the tree. Since the algorithm may have to search all H trees, the worst case complexity for FF-HA+ is $O(H \log^2 V)$, where $V = \max\{V_h\}$. As a comparison, the running time of FF-HA is $O(HV)$, i.e., linear in the number of idle periods, since it has to traverse H linked-list structures. Since H is typically a small constant, whereas the number V of idle periods can be quite large (especially for large systems with thousands of servers and for long time horizons for advance reservations), FF-HA+ is significantly more scalable than FF-HA.

VI. Adaptability: Re-planning Capacity and Maximizing Utilization

As we mentioned earlier in Section I, one of the major concerns regarding the deployment of advance reservation mechanisms has to do with their lack of flexibility that does not permit graceful degradation in application performance when resource management policies mandate changes in allocations. In this section we describe two mechanisms that make it possible to exploit the efficiency of FF-HA+ in order to relax the hard deadline assumption and accommodate changes in resource availability; the implementation of these mechanisms is the subject of ongoing work within our group.

Replanning Capacity. In our work so far we have assumed that deadlines are hard, i.e., jobs are dropped if they can not be allocated within their deadline. It is possible to make the algorithm more flexible and increase the overall ability of the system to meet application QoS requirements by introducing a negotiation process. This process is invoked whenever the scheduler fails to allocate a job and attempts to reschedule existing reservations in order to allocate new incoming jobs whenever possible without affecting the QoS of previously scheduled jobs. This negotiation process may utilize a set of data structures and algorithms similar to the one we described in the previous section to organize, search, and modify existing reservations.

Our algorithm can also be adapted to handle efficiently changes in job demands. Consider, for instance, a job currently running on a server, and assume that it needs to execute for a longer period of time than the one it originally reserved (i.e., the original estimate of its running time was incorrect). In current systems, such jobs are either terminated or preempted and given low priority for scheduling. Given the low running time complexity of our search algorithm, there are several options to handling such situations: one can either invoke the negotiation process to reschedule the job that has reserved the server following the current job, or one can checkpoint the job, invoke the scheduling algorithm to find the next available feasible idle period for it, and then migrate the job to complete execution in another server.

Opportunistic Scheduling. To enable users and Grid administrators to exploit the variations of resource conditions to improve both application and system performance, the FF-HA+ algorithm may be extended to implement opportunistic scheduling. More specifically, new jobs that have no deadline requirements may use resources as they become available, and they may be preempted to accommodate new jobs with deadlines. Such an approach will increase utilization by filling idle periods that might not be used otherwise, and increases the flexibility of the system.

VII. Performance Evaluation

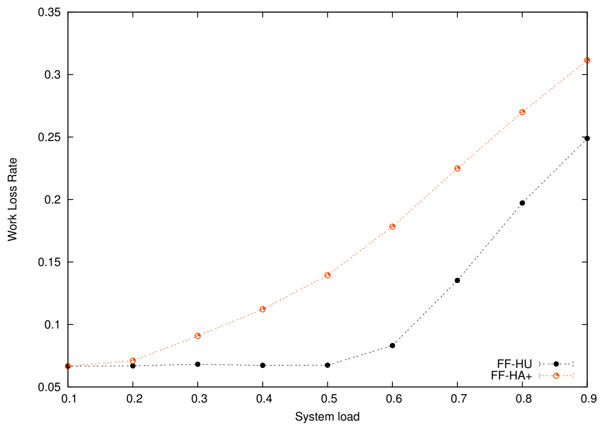
In this section we present simulation results to demonstrate the performance of the FF-HA+ scheduling algorithm. We used the method of batch means to estimate the performance parameters we consider (and which we discuss shortly), with each batch consisting of thirty simulation runs and each run lasting until 10^6 jobs have been submitted to the Grid scheduler. We have also obtained 95% confidence intervals for all the results, which are shown in the figures.

In our simulation, we assume that job requests arrive following a uniform distribution in the range from one minute to 14 days [21]. The duration of each reservation request is randomly selected so that 80% of the incoming jobs are smaller than 4 hours, and 20% are between 4 and 36 hours; the mean job size is 5.6 hours. These values were chosen based on the experience with running real Grid workfolk applications as described in [21], [22]. We let the deadline d_j of job j be uniformly distributed in the interval $(r_j, r_j + q)$, where q corresponds to the "tightness" of the deadline; for most of our experiments we assume $q = 20$ hours unless stated otherwise.

We consider a (H, n)-heterogeneous system with $n = 120$ servers and $H = 3$ distinct service rates. We generated and studied $L = 4$ computational rate distributions such that $L = 1$ refers to the least heterogeneous system and $L = 4$ refers to the most heterogeneous one.

We use four performance metrics in our study. The *work loss rate* is the fraction of work that is dropped due to the fact that the deadline of the corresponding jobs cannot be met. The *system utilization* is the fraction of time the n servers are busy serving jobs. The *waiting time* is the mean amount of time that a job has to wait beyond its ready time until it starts execution; note that dropped jobs do not contribute to the average waiting time. Finally, the *algorithm running time* captures the efficiency of the search algorithm to schedule incoming jobs. To compute the running time we record the CPU time for each simulation corresponding to 10^6 jobs. Work loss rate and waiting time are measures of the QoS perceived by the user, system utilization is a measure of system performance, and running time determines the scalability of the system.

In our first experiment, we compare the FF-HA+ algorithm to the baseline algorithm FF-HU we described in Section III. Recall that FF-HU strategy organizes idle periods in a single linked list ordered in ascending order of their starting time; the algorithm traverses the list and returns the first feasible idle period for a new job, i.e., the one with the earliest starting time. Note that idle periods with early starting times are at risk of expire unused if new jobs are not assigned to them. Therefore, this choice of a feasible idle period is expected to lead to low loss, since assigning a new job to the earliest possible feasible period allows idle periods starting later to be used for future job requests. On the other hand, the running time of the algorithm increases quickly with the



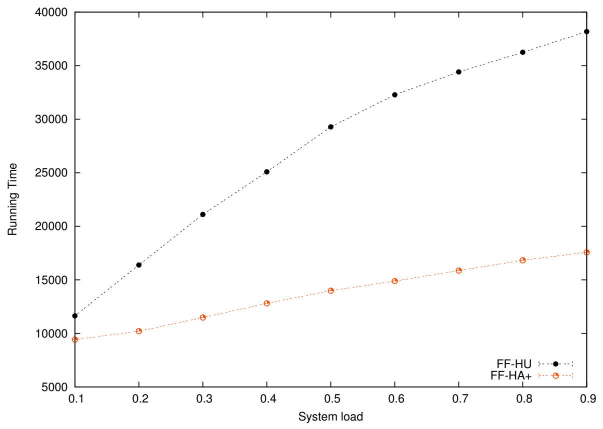
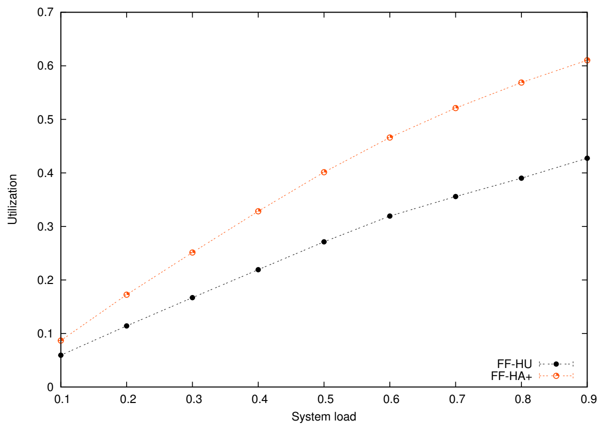


Fig. 4. Comparison of FF-HA+ and FF-HU: (a) work loss rate against load, (b) running time (in milliseconds) against load



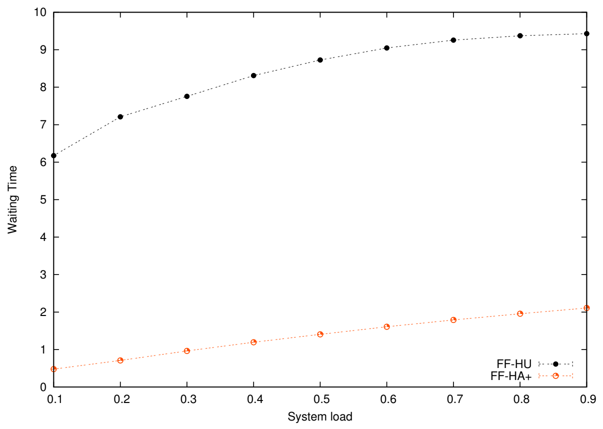


Fig. 5. Comparison of FF-HA+ and FF-HU: (a) utilization against load, (b) waiting time against load

size of the Grid system and the time horizon for making reservations. The FF-HA+ algorithm organizes the idle periods in balanced tree structures, hence it scales well to large Grid systems. However, it does not necessarily return the feasible idle period with the earliest starting time, hence we expect that its work loss rate will be higher than FF-HU. But we emphasize that FF-HA+ will always find a feasible idle period for a new job if one exists.

Figure 4 confirms the above observations. The figure plots the work loss rate and running time of the FF-HU and FF-HA+ algorithms against the system load. As we can see in Figure 4(a), the loss rate increases with the system load for both algorithms. The two strategies exhibit similar loss rates at low loads (when there are sufficient resources to schedule almost all jobs) and high loads (when the issue is the lack of resources, not the particular strategy used). However, the FF-HA+ strategy exhibits a higher loss rate at medium loads, as we expected. A careful examination of our experiments shows that FF-HU incurs less resource fragmentation that FF-HA+. This result is due to the fact that FF-HA+ returns the feasible idle period of maximum capacity among those in its subtree; while this choice was made to speed up the operation of the algorithm, the side effect is higher fragmentation. On the other hand, the running time of FF-HU is significantly higher than that of FF-HA+, especially at medium to high loads; again, this result is consistent with our discussion above.

The system utilization curves in Figure 5(a) suggest that FF-HA+ utilizes better the resources available in the system, i.e., the servers are busy performing work for a longer fraction of time than under FF-HU. However, since the loss rate for FF-HA+ is slightly higher, this results implies that FF-HA+ allocates more jobs to slow processors than FF-HU. A more careful examination of our results reveals that, under FF-HA+, processors with high service rate exhibit a higher fragmentation; since the capacity of processors with high service rate expires faster as time progresses, fragmentation of capacity on high-rate servers has a more detrimental effect on system performance, as exhibited by the higher loss rate of HH-FA+. Figure 5(b) plots the average waiting time that jobs have to wait beyond their ready time. We observe that jobs have to wait significantly longer under FF-HU compared to FF-HA+. In other words, although FF-HU schedules a larger fraction of jobs than FF-HA+, the start time of these jobs is pushed back resulting in longer

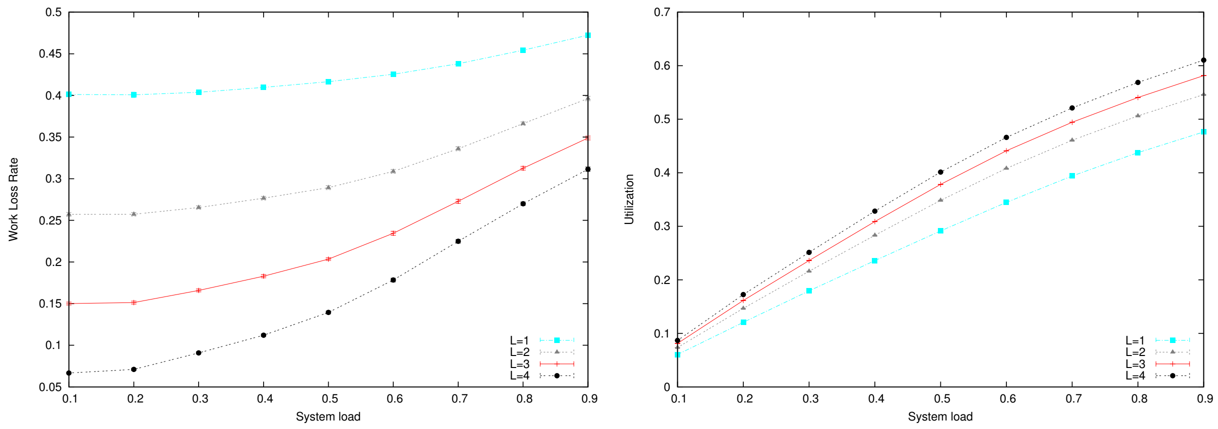


Fig. 6. The impact of heterogeneity: (a) work loss rate against load, (b) utilization against load

waiting times.

Finally, Figure 6 investigates the impact of different levels of heterogeneity on performance. Figure 6 (a) plots the work loss rate against the load for $L = 4$ different levels of heterogeneity, where larger values of L imply higher heterogeneity; Figure 6 (b) is similar but plots system utilization against load. We can see that as resources become more heterogeneous, the loss rate and system utilization both improve, in many cases significantly so. This behavior follows from the fact that to increase resource heterogeneity in a given system while keeping the total service rate constant, as required by expression (2), the service rate of a few fast processors must increase further. In other words, a larger fraction of the total service rate is concentrated on fewer resources. Consequently, making the system more heterogeneous introduces a higher degree of statistical multiplexing, whereby fewer high capacity servers are responsible for serving larger number of customers. The results in Figure 6 then are consistent with the well-known fact from queueing theory that statistical multiplexing improves system performance.

VIII. Concluding Remarks

We have considered the problem of advance reservations for jobs with deadlines in a Grid system with heterogeneous resources. We have developed a geometric representation of idle periods and jobs that provides new insight and allows for efficient organization of the reservations. We have developed a scheduling algorithm with good performance that can scale to large Grid systems and long time horizons. We have also shown that resource heterogeneity may have a positive impact on performance if taken into account in the design of scheduling algorithms.

References

[1] E. Elmroth and J. Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. Lecture Notes in Computer Science, volume 3732, pages 1077–1085. Springer-Verlag, 2005.

[2] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[3] M. Maheswaran K. Krauter, R. Buyya. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, February 2002.

[4] R. Min and M. Maheswaran. Scheduling Advance Reservations with Priorities in Grid Computing systems. In *Proceedings of PDCS'01*, pages 172–176, 2001.

[5] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of IPDPS'00*, pages 127–132, 2000.

[6] A. Sulistio and R. Buyya. A grid simulation infrastructure supporting advance reservation. In *Proceedings of PDCS'04*, pages 1-7, Nov. 2004.

[7] H. Rasheed, M. Dikaiakos, and S. Haridi. Quantification of Grid Resource Heterogeneity Effects on Performance. *Technical Report*, January, 2006.

[8] G. Dasgupta, K. Dasgupta, A. Purohit, and B. Viswanathan. QoS-GRAF: A Framework for QoS based Grid Resource Allocation with Failure provisioning. *Proceedings of the 14th IEEE International Workshop on QoS (IWQOS'06)*, pages 281-283, June 19–21, New Heaven, CT, USA.

[9] I. Foster and A. Roy. Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. *Proceedings of the 8th International Workshop on Quality of Service (IWQOS'2000)*, pages 181–188, June 5-7, 2000.

[10] J. MacLaren. Advance Reservations: State of the Art. http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html.

[11] A. Andrieux, K. Czajkowski,A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specifications WS-Agreement. *Global Grid Forum*, 2004.

[12] A. Leff, J.T. Rayfield, and D.M. Dias. Service-Level Agreements and Commercial Grids. *IEEE Internet Computing*, pages 44–50,volume 7, number 4, July, 2003.

[13] H. Li, and L. Wolters. An Investigation of Grid Performance Predictions Through Statistcal Learning. *1st Workshop on Tackling Computer System Problems with Machine Learning Techniques (SysML), in conjunction with ACM Sigmetrics*, Saint-Malo, France, 2006.

[14] L. Yang, J.M. Schopf, and I. Foster. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. *Proceedings of the 15th ACM/IEEE Conference in Supercomputing (SC'03)*, pages , Phoenix, Arizona, 2003.

[15] I. Foster. What is The Grid? A Three Point Checklist. www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, July 20, 2002.

[16] L. Jin, V. Machiraju, and A. Sahai. Analysis on Service Level Agreement of Web Services. *HP Lab Technical Report HPL-2002-180*, June 21st, 2002.

[17] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[18] A. Sahai, S. Graupner, V. Machiraju, A. Van Moorsel. Specifying and Monitoring Guarantees in Commercial Grids through SLA. *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pages 292–299, May 12–15, Tokyo, Japan.

[19] P. Godfrey, and R. Karp. On the Price of Heterogeneity in Parallel Systems. *Proceedings of 18th ACM Symposium on Parallelism in Algorithms and Architectures*, July 30th to August 2nd, 2006, Cambridge, MA, USA.

[20] L. Dubois, G. Mounie, and D. Trystram. Analysis of Scheduling Algorithms with Reservations. *Proceedins of the 21st IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, March 26–30,2007.

[21] M. Siddiqui, A. Villazon, and T. Fahringe. Grid Capacity Planning with Negotiation-based Advance. Reservation for Optimized QoS. *Proceedings of the 2006 IEEE/ACM Conference in Supercomputing (SC2006)*, pages 103-118, November 15–21, Phoenix, Arizona, 2006.

[22] T. Fahringer, R. Prodan, R. Duan, F. Nerieri,S Podlipnig, J Qin,M Siddiqui,H. Truong,A. Villazon, and M. Wieczorek. ASKALON: A Grid Application Development and Computing Environment. *Proceedings of 6th International Workshop on Grid Computing (Grid 2005)*, IEEE Computer Society Press, Seattle, Washington, USA.

[23] K.K. Drogemeier and R. Wilhelmson. Linked Environments for Atmospheric Discovery (LEAD): A CyberInfrstructure for MEsoscale Meteorology Research and Education. *Proceedings of 20th Conference of Interactive Information Processing Systems for Meteoreology, Oceonography and Hydrology*, 2004,Seattle, Washington, USA.

[24] D.S. Katz, D.S.,J.C. Jacob, E. Deelman,C. Kesselman,G. Singh, M. Su,G.B. Berriman, J. Good,A.C. Laity and T.A. Prince. A Comparison of Two Methods for Building Astronomical Image Mosaics on a Grid. *Proceedings of the 34th International Conference Workshops on Parallel Processing (ICPP'05)*, pages 85–94, June 14–17, Oslo, Norway.

[25] E. Deelman, C. Kesselman, G. Mehta,L. Meshkat, L. Pearlman, K. Blackburn,P. Ehrens, A. Lazzarini,R. Williams and S. Koranda. GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists. *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, pages 225-234, July 23–26, 2002,Edinburgh, Scotland.

[26] R.L. Henderson. Job Scheduling Under the Portable Batch System. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Lectures Notes in Computer Sciences, pages 279–294 , April 25, Santa Barbara, California, US.

[27] P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli,V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall,T. Russ,G. Singh, M. Spraragen,G. Staples and K. Vahi. Simplifying construction of complex workflows for non-expertusers of the Southern California Earthquake Center Community Modeling Environment *Journal of SIGMOD Record*, ACM Press, Volume 34, Issue 3, pages 24–30.

[28] S. Zhou, X. Zheng, J. Wang and P. Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Journal of Software Practica Experience*, John Wiley & Sons, Inc., Volume 23, Number 12, pages 1305–1336, 1993.

[29] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-A Huner of Idle Workstations. *Proceedings of IEEE 8th International Conference on Distributed Computing Systems (ICDCS'88)* , pages 104–111,June 13–17, San Jose, California, US.

[30] M. Hovestadt, O. Kao, A. Keller and A. Streit. Scheduling in HPC Resource Management Systems: Queing vs. Planning. *Proceedings of 9th International Workshop on Job Scheduling for Parallel Processing (JSSPP'03)*, pages 1–20, Lectures Notes in Computer Science, Springer, June, 2003, Seattle, Washington,US.

[31] U. Farooq, S. Majumdar and E.W. Parsons. Impact of Laxity on Scheduling with Advance Reservations in Grids. *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, pages 319–322, September 26, 2005, Atlanta, Georgia, US.

[32] G. Singh, C. Kesselman and E. Deelman. 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*, pages 117–126, June 27–29, 2007, Monterrey, California, US.

[33] R. Buyya and S. Venugopal. The Gridbus toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. *Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models (GECON'04)*, pages 19–36,IEEE Press, April, 2004, Seoul, Korea.

[34] I. Foster, C. Kesselman, C. Lee, R. Lindell,K. Nahrstedt and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. *Proceeings of the 7th International Workshop on Quality of Service (IWQoS'99)*, June 1–4, London, UK.

[35] A.S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer and L. Young. Making the Grid Predictable through Reservation and Performance Modelling. *The Computer Journal*, pages 358–368, Volume 38, Number 3, 2005.

[36] H. Zhao and R. Sakellariou. Advance Reservation Policies for Workflows. *Proceedings of the 12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'06)*, pages 47–67,June 26, 2006,Saint-Malo, France.

[37] G. Singh, C. Kesselman and E. Deelman. Performance Impact of Resource Provisioning on Workflows Applying. *Technical Report* http://www.cs.usc.edu/Research/TechReports/05-850.pdf.

[38] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan and T. Fahringer. Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid. *In Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (E-SCIENCE'06)*, pages 82,December 4–6,2006, Washington, DC, US.

[39] J. Cao and F. Zimmermann Queue Scheduling and Advance Reservations with COSY *Proceedings of the 18th International Parallel and Distributed Symposium (IPDPS'04)*, pages 63–70,April 26–30, 2004, Santa Fe, New Mexico, US.

[40] C. Castillo, G. Rouskas and K. Harfoush. On the Design of Online Scheduling Algorithms for Advance Reservations and QoS in Grids. *Proceedings of the 21th International Parallel and Distributed Symposium (IPDPS'07)*, pages 1–10, March 26–30,2007, Long Beach, California.

[41] H.J. Siegel and S. Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. *Journal of Systems Architecture*, pages 627–639, Volume 46, Elsevier Science, 2000.

[42] A. Andrzejak and A. Reinefeld and F. Schintke and T. Schütt. On Adaptability in Grid Systems. Future Generation Grids, chapter, pages 29–46,Springer Science+Business Media, Inc., January, 2006.

[43] T. Röblitz, F. Schintke and A. Reinefeld. Resource Reservations with Fuzzy Requests. *Journal on Concurrency and Computation: Practice and Experience*, pages 1681–1703,Volume 18, Number 13, November, 2006

[44] T. Röblitz, F. Schintke and J. Wendler. Elastic Grid Reservations with User-Defined Optimization Policies. *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM'04)*, September 30,Antibes Juan-les-Pins, France.

[45] D. Jackson, Q. Snell and M. Clement Core Algorithms of the Maui Scheduler *Proceedings 7th International Workshop of Job Scheduling Strategies for Parallel Processing (JSSPP'01)*, pages 87–103, June 16, 2001, Cambridge, MA, US.

[46] IBM. http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html.