Efficient Implementation of Best-Fit Scheduling for Advance Reservations and QoS in Grids

Claris Castillo, George N. Rouskas and Khaled Harfoush

Computer Science Department, North Carolina State University, Raleigh, NC, USA,
ccastil@ncsu.edu, rouskas@csc.ncsu.edu, harfoush@csc.ncsu.edu

Abstract. In this work we consider the problem of providing QoS guarantees to Grid users by means of advance reservation of resources. Advance reservation mechanisms provide with the ability to allocate resources to users based on agreed-upon QoS requirements and increase the predictability of a Grid system. However, incorporating such mechanisms into current Grid environments has proven to be a challenging task due to the dynamism and heterogeneity of Grid environments. In our previous work we introduced a suite of scheduling algorithms for Grids capable of allocating resource efficiently and providing with QoS guarantee. In this paper we extend our previous work by introducing an efficient implementation of the best-fit scheduling algorithm. We evaluate the performance of the best-fit algorithm by means of extensive simulation and analysis against the scheduling algorithms introduced previously. Our results show that best-fit algorithm performs well across several metrics that reflect both user and system specific goals.

1 Introduction

Providing QoS support for applications has become crucial for the development of Grid technologies and their economic significance. A strong evidence of the urgency for incorporating QoS capabilities in Grids is presented in a recent study performed by a leading technology-market research company [13]. This study shows that computing Grids have achieved 81% of awareness but only 8% of adoption in North American enterprises; with Asia Pacific and Europe exhibiting similar figures. We believe that the lack of QoS support in existing Grid systems is a primary factor for the slow adoption rate of Grid computing in enterprise systems.

This state of affairs has its roots to the fact that Grid technologies emerged in response to the need of uniting resources in an inexpensive and effective manner and not of providing with QoS requirements to users. To make matters worse, providing with QoS in Grids has proved difficult; this is mainly due to the heterogeneity and dynamism of Grid environments. In such complex environments QoS deals with several independent systems which only as a whole can define QoS user demand [9]. Therefore, QoS support needs to be embedded in every layer of the Grid architecture. The scheduler is one low-layer component which plays an important role in the QoS capabilities of a resource management system; since it has direct access to resources and can perform very resource-specific tasks. However, embedding QoS in schedulers has been often overlooked as a design parameter in Grids. Most existing Grid solutions tackle the problem of providing with QoS in Grids by embedding QoS support at higher layers of the architecture *e*xclusively [19]. Examples of this are: service level agreements (SLAs) [10,14,15,20] and monitoring and predictive tools [22], [18], [16,17]. This lack of functional consistency through all the layers of the architecture leads to solutions that exhibit conflict of objectives in between layers; hindering the effective provision of QoS they were designed for in the first place [3].

Advance reservation of resources [1–3, 6, 23–27] has generated great interest in the Grid community as a mechanism that Grid providers may employ to offer specific QoS guarantees to application users when scheduling new jobs. However, the availability of advance reservation mechanisms is currently limited [12], mainly due to two major concerns regarding their performance. First, typical advance reservation mechanisms lack flexibility as they do not permit graceful degradation in application performance when resource management policies mandate changes in allocations [11]. Second, existing approaches suffer from poor scalability as they are not effective in managing large sets of advance reservations or handling resource fragmentation. 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.

We believe that the development of QoS-driven scheduling algorithms is of utmost importance to further development of Grid technology and its economic significance, since without QoS guarantees users will be reluctant to pay for Grid services and service providers will be unable to differentiate themselves from their competitors. In previous work [7] we developed a suite of efficient algorithms for advance reservations and QoS in Grids. These algorithms proved to be effective in handling job deadlines and the resource fragmentation commonly observed when using advance reservations. We also showed that they can be easily adapted to employ several optimization criteria for scheduling jobs and their low running times make them suitable for large Grid environments. Given the great success of the best-fit scheduling algorithm in minimizing resource fragmentation in other contexts (e.g., memory blocks) we have developed an efficient implementation of the algorithm. Similar to our previous work this algorithm reuses concepts of computational geometry to deal effectively with resource fragmentation and deadline requirements. Moreover, its generic design facilitates its adoption to accommodate for both network and computing resources.

The rest of the paper is organized as follows. In Section 2 we describe the online scheduling problem considered in this work. Since the design of the best-fit scheduling algorithm reuses some of the concepts for advance reservations introduced in our previous work [21], in Section 3 we provide with the relevant background [21] required for the understanding of this work. In Section 4 we describe an efficient implementation of the best-fit scheduling algorithm and its

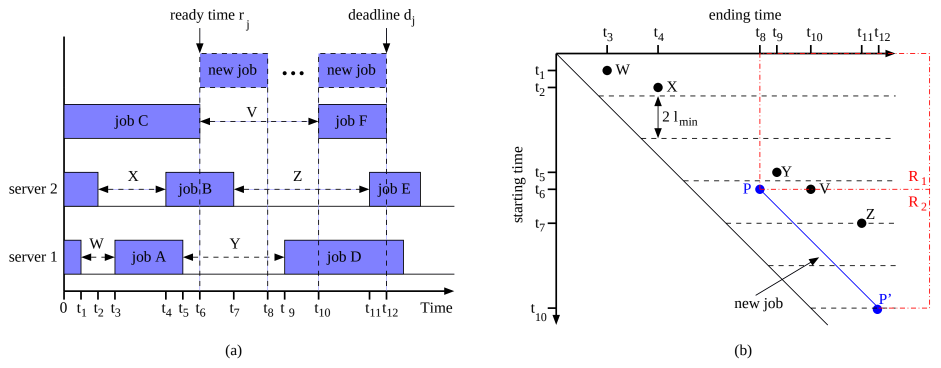

Fig. 1. (a) Jobs scheduled and idle periods in a 2-server system, (b) idle periods as points in the plane, plane partitioned into strips of width $2 \times l_{min}$, and feasible regions R_1, R_2 for the new job

corresponding data structures. In Section 5 we investigate the performance of our algorithm through simulation, and we conclude the paper in Section 6.

2 Problem Description

Following a description of the problem addressed in this paper. Notice that the problem is described in terms of computing resources, but as we mentioned earlier, the scheduling algorithms introduced in [21] and in section 4 can be applied to networking resources as well.

Consider a scheduler $\mathcal{S}$ for a Grid with n servers which may be geographically distributed in a network.We make the assumption that all servers are identical in terms of their processing capacity C; extending the algorithms we present here to non-identical resources is the topic of ongoing research in our group. 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:

- r_j is the *ready time* of the job, i.e., the earliest the job can be made available to the grid for processing;
- l_j is the *length* of the job, i.e, the amount of work the job requires; and
- $d_j (\geq r_j + l_j)$ is the *deadline* of the job, i.e., the latest time by which the job can be completed.

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.

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. The schedule shows that at the current time (i.e., time $t = 0$ in the figure), there are three 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_3 to t_5, and job D which has reserved the server from time t_9 to t_{11}; similarly, three jobs have been scheduled for server 2. The figure also shows a service request for scheduling a new job j with ready time $r_j = t_6$ and length $l_j = t_8 - t_6$.

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 who 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 turnaround times 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.

Several variants of this scheduling problem with advance reservations and/or deadlines have been studied in multiprocessor and Grid systems [26–30]. However, the heuristic solution approaches that have been proposed may not scale well and may not utilize the available system capacity efficiently [2, 31]. In our previous work [21] we developed a framework that applies techniques borrowed from computational geometry to enable the *efficient* scheduling of jobs with deadline constraints. Since the work presented in this paper is an extension of [21], in the next section we provide with the background relevant to the new contribution.

3 Background

We represent idle periods as points in a Cartesian plane and jobs with non-immediate deadlines as segments, as we explained in [21]. Let us refer to Figure 1, assuming that the current time $t = 0$, Figure 1(a) shows the current schedule of advance reservations for a 2-server system, along with a request to schedule a new job j with the tuple ($r_j = t_6$, $l_j = t_8 - t_6$, $d_j = t_{12}$). Figure 1(b) is the geometric representation of this schedule. The partitioning of the plane in horizontal strips will be explained shortly in this section. An idle period is represented by a point in the Cartesian plane with its x and y coordinate corresponding to its starting and ending time respectively. Since the ending time of an idle period must be greater than its starting time, all points will always be above the diagonal in

Figure 1. Similarly, a job with immediate deadline can be represented as a point $P = (r_j, r_j + l_j)$ in Figure 1 where P represents the earliest the job can start and end execution. The fact that job j has a general deadline is represented in Figure 1(b) by the line segment between points P and P', where $P = (r_j, r_j + l_j)$ (respectively, $P' = (dj\_l_j, d_j)$) corresponds to the earliest (respectively, latest) possible pair of starting and ending times for this job. Notice that an idle period is feasible for a given job if its starting and ending time is smaller and larger than the starting and ending time of the new job, respectively. Following this observation, the scheduler may select any point on this line segment as the starting/ending times of the job, as long as there is an idle period completely containing this point.

In order to enable the design of efficient scheduling algorithms we then partition the area of the plane above the diagonal into strips of width equal to twice the minimum job size l_{min}. Doing so in effect partitions the set of K idle periods into a number H of subsets, where subset $h, h = 1, \cdots, H$, contains the idle periods falling within the h_{th} strip. In each strip the idle periods are stored in a balanced priority search tree. The motivation behind partitioning the plane is that it bounds the number of idle periods per strip. In fact, at most one idle period from each server can be contained in each strip. Consequently, updating the schedule (i.e., adding removing idle periods from the priority search tree associated with a given strip) takes time $O(\log n)$, rather than $O(\log K)$, where typically $n \leq\leq K$. Since each priority search tree structure contains only a subset of the set of idle periods, it may be necessary to search several trees to find a feasible idle period for a new job request. Consider point P in figure representing the earliest time the new job may start execution. In this example, the new job can be scheduled either in the idle period represented by point V or the one represented by Y. Point V can be found by searching the tree structure corresponding to the strip in which point P lies, however, if point V (i.e., the corresponding idle period) did not exist, one would have to continue searching strips above the one in which P lies (i.e., those with starting times earlier than the new job) in order to find an idle period (in this case, point Y) that would not delay the start of the job. On the other hand, if neither V or Y existed, the search would have to continue in strips (e.g., Z) that could accommodate this job at some starting time along the line segment from P to P'.

In addition to allowing the scheduler to handle jobs with general deadlines efficiently, the partition of idle periods into subsets also enables the natural implementation of a variety of strategies for selecting one among multiple feasible idle periods. In [21] we developed a suite of scheduling strategies which make use of the approach we outlined above. These strategies are based on the observation that a job scheduled in an idle period will create at most two new idle periods: one between the start of the original idle period and the start of the job (the *leading idle period*, LIP in short) and one between the end of the job and the end of the original idle period (the *trailing idle period*, TIP in short). The creation of these new smaller idle periods results in further fragmentation of the available capacity, and may prevent future job requests from being accommodated. Therefore, it may be desirable to schedule a new job within the idle period such that the size of either the leading or trailing idle periods created is optimized.

We exploit these observations in [21] and propose three scheduling strategies; min-LIP, min-TIP and first-fit. Min-LIP and min-TIP seek to minimize the leading and trailing idle period respectively. First-fit, in the other hand, does not optimize for the size of the leading and trailing idle periods when scheduling a new job; and returns the first feasible idle period found for the given job. Results obtained from extensive simulations demonstrate that our algorithms perform well across both user and system performance under different conditions. Due to space constraints we are not going to describe these scheduling algorithms in here; we refer the reader to [21] instead. In the next section we present an extension to the suite of algorithms presented in [21], in that, we develop an extended and efficient version of best-fit scheduling algorithm that supports advance reservations.

4 Best-Fit Algorithm Description and Implementation

Consider the new job j and its geometric representation in the plane, as shown in Figure 1(b). The *feasible region* of job j refers to the part of the plane where all idle periods that can accommodate this job may lie. The feasible region is the part of the plane above and to the right of the line segment between P and P', since only any idle period in that region will fully contain *some* point of the line segment. The feasible region can be partitioned into two subregions, R_1 and R_2, as in Figure 1(b). Any idle period lying in R_1 (e.g., idle periods Y an V in the figure) starts at or before the new job's ready time r_j ($= t_6$ in the figure), and ends after the earliest time the job can be completed ($= t_8$ in the figure). Therefore, any idle period in this region can accommodate the new job without delaying its execution, i.e., the job can start execution at its ready time r_j. Any idle period lying in R_2, on the other hand (e.g., idle period Z in Figure 1(b)), starts later than the job's ready time but is large enough for it. Hence, the job may be assigned to any idle period in R_2 at the cost of delaying its execution beyond its ready time.

For the best-fit strategy, we use a 2-dimensional tree T_h to store the idle periods within each strip h, $h = 1, \cdots, H$. In the tree corresponding to T_h's first dimension, t_h^s, idle periods are in the leaf nodes, arranged in ascending order of their starting time. A leaf node corresponding to idle period X stores the following information:

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

The information stored at each of its internal nodes u consists of:

- the median starting time of the idle periods stored in the subtree of t_h^s rooted at u;

– a pointer to a secondary priority search tree t_h^e; and
– a pointer to a secondary regular binary search tree t_h^l.

Trees t_h^e and t_h^l store the idle periods in u's subtree in descending order of their ending time and length, respectively. The information stored at each internal node v of tree t_h^e consists of:

– the median ending time of the idle periods stored in the subtree of t_h^e rooted at v; and;
– a pointer to the idle period with minimum length in v's subtree.

As we explain shortly, the manner in which the data structure is searched depends on the part of the feasible region (R_1 or R_2) in which the corresponding strip lies.

The best-fit algorithm consists of two steps: a search for b_{R_1}, the local best fit in region R_1, followed by a search for b_{R_2}, the best fit in region R_2. After exploring both regions, the algorithm returns the overall best fit for the given job, if one exists. Since in this strategy the algorithm searches for a local best fit in every strip in order to obtain a global optimal, the order in which this search proceeds is irrelevant. However, for the sake of simplicity in our implementation we search both regions in a top-bottom fashion.

Step 1: Search in region R_1. Since the best-fit among a set of feasible idle periods is the idle period with the smallest length, the algorithm first identifies the set of feasible idle periods in the strip, and then retrieves the one with the smallest length. Recall also that all idle periods in R_1 start before r_j (see figure 1) and hence, meet the feasibility requirement in terms of their starting time. However, they may or may not be feasible depending on their ending time. To identify the set of feasible idle periods for a given job j in a strip in R_1, the algorithm searches the secondary tree associated with the strip t_h^e. More specifically, the algorithm visits every internal node v in t_h^e whose subtree contains *exclusively* idle periods with ending time larger than the earliest time the job can be completed; the algorithm stops as soon as it reaches a leaf. To do this, the algorithm starts at the root A of the t_h^e. It compare the earliest ending time of the new job j ($r_j + l_j = t_8$) to the median of the ending times of the idle periods in this tree stored at the root. If the median is larger, that implies that all the idle periods in the right subtree are larger. The algorithm marks the right subtree and continues searching in the left subtree. If the median is smaller then the algorithm discard the right subtree since all the idle periods contained in it have a ending time smaller than the earliest ending time of the job; and hence are not feasible for the given job. In this case the scheduling algorithm continues the search recursively in the left subtree. Since this tree traversal visits each level of the tree at most once and the tree data structure is balanced the cost of visiting the $O(\log n)$ internal nodes is $O(\log n)$ per strip.

For each internal node v visited in tree t_h^e the algorithm computes the local best fit b_v corresponding to the idle periods in v's subtree. Such an idle period is the smallest idle period in v'subtree and can be retrieved by means of the pointer stored at v at a cost of $O(1)$. The algorithm then compares b_v to the most up to date b_{R_1} at that particular point in time; if b_v has a smaller length it updates b_{R_1} with b_v, otherwise, it discards b_v. Recall that retrieving b_v from a given v's subtree costs $O(1)$; therefore, the overall cost for searching b_{R_1} is $O(m \log n)$ where m is the number of strips in R_1 and is at most $m = \lceil \frac{r_j}{2l_{min}} \rceil$.

Step2: Search in region R_2. After the algorithm has searched for b_{R_1} it proceeds to search b_{R_2} in R_2. Notice that as an idle period in R_2 moves further up (down) from the line segment between P and P' its length increases (decreases), until it reaches the line segment itself where the length of the idle period is l_j. It follows that the best fit in a strip in R_2 is the closest idle period to the line segment between P and P'. To find such idle period the algorithm performs a simple binary search on tree t_h^l. More specifically, it searches for the idle period with the minimum length larger than the length of the job, l_j. Since in R_2 there are at most $k = \lceil \frac{d_j}{2l_{min}} \rceil$ strips, the overall complexity for searching b_{R_2} in Step 2 is $O(k \log n)$.

5 Performance Evaluation

We use simulation to evaluate the performance of best-fist scheduling strategy. We use 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 as a Poisson process with rate λ. Job sizes are distributed according to a bounded Pareto distribution. The minimum job size is set equal to 1, and is taken as the unit of time. The maximum job size is set to 50 time units, and we vary the mean job size $\bar{x}$ by changing the value of the parameters of the Pareto distribution. We let L denote the amount of time that the scheduler $\mathcal{S}$ can look "into the future"; in other words, a job may request to be scheduled at most L units of time in the future. We let the deadline d_j of job j be uniformly distributed in the interval $(r_j + l_j, r_j + l_j + q(L - r_j - l_j))$, where $q, 0 \leq q \leq 1$ is a parameter that controls the "tightness" of the job deadlines. In our simulations, we let $L = 200$.

We use three performance metrics in our study. The *loss rate* is the fraction of jobs that are dropped due to the fact that their deadline cannot be met. The *system utilization* is the fraction of time the n servers are busy serving jobs. Finally, the *average delay* 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 delay.

To evaluate best-fit's performance we include the results of the scheduling strategies min-LIP and min-TIP introduced in [21]. The reasons for this being that first, their performance ia well understood [21] and second, they share similar optimization objectives with best-fit. This in turns provides us with a more solid comparative framework; enhancing the quality of our study and results.

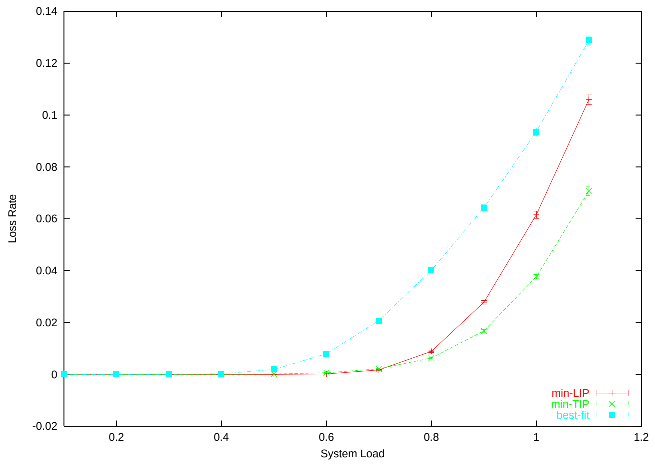

Fig. 2. Loss rate vs. system load ρ, $n = 20, \bar{x} = 3.28, q = 0.1$

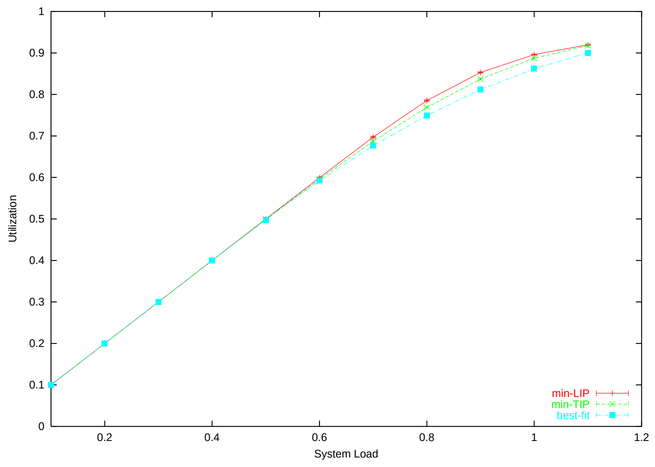

Fig. 3. Utilization vs. system load ρ, $n = 20, \bar{x} = 3.28, q = 0.1$

Figures 2-4 plot the loss rate, utilization and average delay, respectively, for the three scheduling strategies against the system load ρ. The system load is calculated using the familiar from queueing theory expression $\rho = (\lambda\bar{x})/n$. For the results shown in these figures, we let the number of servers $n = 20$, the mean job size $\bar{x} = 3.28$, and the tightness of the job deadlines $q = 0.1$. Note that the load values in the figures range from low ($\rho = 0.1$) to very high ($\rho = 1.1$) at which the system is more than 100% loaded. Also, the 95% confidence intervals are quite narrow for all curves shown.

From Figure 2 we can see that the loss rate increases with the system load for the three scheduling strategies, as expected. We observe that for low loads

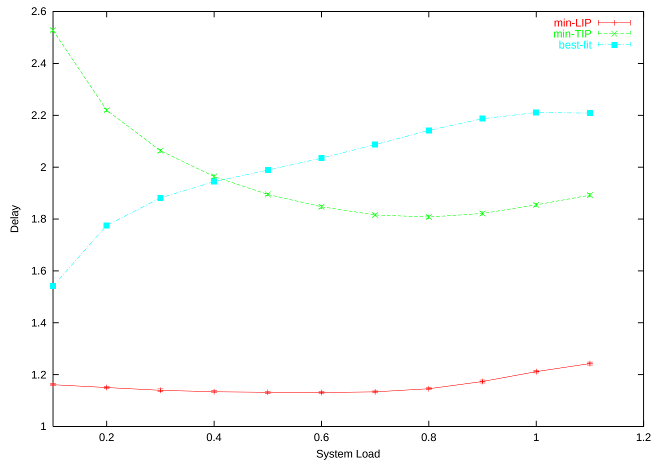

Fig. 4. Average delay vs. system load ρ, $n = 20, \bar{x} = 3.28, q = 0.1$

the three scheduling strategies exhibit almost identical performance. However as the load increases, the curve corresponding to best-fit diverges and becomes worse. This fact can be explained as follows; for min-LIP (respectively, min-TIP) the leading idle period (respectively, trailing idle period) is being minimized, therefore, the level of resource fragmentation is minimum. For best-fit, on the other hand, it is more likely that two idle periods, i.e., the leading and trailing idle period, will be created for every job scheduled; incurring in further fragmentation of the resources and therefore in higher loss rate.

Figure 3, which plots the system utilization versus the load, confirms our observations regarding the relative performance for the three different scheduling algorithms. As expected, utilization increases with the system load initially, but at some point the curves level off. These results are consistent with our observations for Figure 2. Best-fit exhibits the worst performance among the three scheduling algorithms since best-fit seeks to minimize the total amount of unutilized resources left per schedule, without seeking to accommodate for the time dynamics observed in advance reservations. More specifically, to maximize utilization in a planning system, i.e., with advance reservation support, the scheduler needs to guarantee that any resource remaining unused at certain point in time will eventually (i.e., some time in the future) be claimed for a given job. This is difficult to achieve under the best-fit strategy since it tends to create two most likely small trailing and leading idle periods for every new job scheduled. Min-LIP and min-TIP in the other hand are mostly like to create one single idle period for every scheduled job. This result indicates all three algorithms are capable of identifying and using idle periods to schedule jobs, thus ensuring that fragmentation of system capacity does not compromise overall performance.

Let us now turn to Figure 4 which plots the average job delay against the system load. As we can see, jobs experience the highest delay under best-fit. This result is consistent with the plots in Figure 2, in that as the load increases the resources become more fragmented and the scheduler is forced to allocate idle periods that start far in the future, i.e., $>> st_i$ to new incoming jobs. Overall, the average delay values in Figure 4 are relatively low, and correspond to a fraction of the mean job size $\bar{x} = 3.28$ for the three different scheduling algorithms.

In addition to providing insight into the relative behavior of the three strategies due to the different optimization objectives considered, Figures 2-3 illustrate that properly designed scheduling algorithms can effectively overcome the obstacles of capacity fragmentation to deliver high performance in terms of metrics that reflect the requirements of both users and service providers.

6 Concluding Remarks

We have developed an efficient implementation of the best-fit scheduling algorithm for advance reservations in Grids that uses techniques from computational geometry. We have also presented results from extensive simulation experiments to demonstrate that the algorithm is simultaneously user and system centric, that is, it is able to meet deadline requirements imposed by the users while maximizing system utilization. Our results challenge the common belief about optimal performance of the best-fit scheduling algorithm. The fact that best-fit algorithm is outperformed by both min-LIP and min-TIP algorithms suggests that although its performance is optimal in contexts such as in memory architectures, its suitability is questionable in more dynamic contexts such as the one considered in this work. Our work provides a practical and efficient solution to the problem of scheduling computing and network resources in dynamic Grid environments.

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. Reference removed to preserve the anonymity of the authors.
8. H. Rasheed, M. Dikaiakos, and S. Haridi Quantification of Grid Resource Heterogeneity Effects on Performance *Technical Report*, January, 2006.

9. L. Burchard, B. Linnert, F. Heine, M. Hovestadt, O. Kao, and A. Keller A Quality-of-Service Architecture for Future Grid Computing Applications *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, April 3–8, Denver, Colorado, 2005.
10. G. Dasgupta, K. Dasgupta, A. Purohit, and B. Viswanathan *Proceedings of the 14th IEEE International Workshop on QoS (IWQOS'06)*, pages 281-283, June 19–21, New Heaven, CT, USA.
11. 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.
12. J. MacLaren Advance Reservations: State of the Art http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html
13. F. Gillett Global Compute Grid Adoption Is Nearly Flat *Forrester–Business Technographics*, October 13, 2006.
14. 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
15. 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
16. 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.
17. 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.
18. R. Wolski Experiences with Predicting Resource Performance On-line in Computational Grid Settings *Proceedings of ACM SIGMETRICS Performnace Evaluation Review*,Volume 30, Number 4, pp 44–49, March, 2003.
19. I. Foster What is The Grid? A Three Point Checklist. `www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`, July 20, 2002.
20. 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.
21. C. Castillo, G. N. Rouskas, and K. Harfoush On the Design of Online Scheduling Algorithms for Advance Reservations and QoS in Grids *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*, March 26-30, 2007, Long Beach, California, US.
22. 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.
23. L. Dubois, G. Mounie, and D. Trystram Analysis of Scheduling Algorithms with Reservations *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, March 26–30,2007.
24. 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.

25. 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.
26. W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of IPDPS'00*, pages 127–132, 2000.
27. R. Min and M. Maheswaran. Scheduling Advance Reservations with Priorities in Grid Computing systems. In *Proceedings of PDCS'01*, pages 172–176, 2001.
28. E. Caronand, P. K. Chouhan, and F. Desprez. Deadline scheduling with priority for client-server systems on the grid. *IEEE/ACM International Workshop on Grid Computing*, pages 410–414, 2004.
29. H-L. Chan, T. Lam, and K. To Nonmigratory online deadline scheduling on multiprocessors. *SIAM Journal on Computing*, Volume 3, Number 4, pages 669–682, 2005.
30. A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A study of deadline scheduling for client-server systems on the computational grid. In *Proceedings of IEEE Symposium on High Performance and Distributed Computing (HPDC'01)*, pages 406–415, August 3–6, Redondo Beach, CA, 2001.
31. R. J. Al-Ali, K. Amin, G. von Laszewsky, O. F. Rana, D. Walker, M. Hategan, and N. Zaluzec Analysis and provision of QoS for distributed grid applications. *Journal of Grid Computing*, Volume 2, Number 2, pages 163–182, June 2004.