

On Incorporating an On-line Strip Packing Algorithm into Elastic Grid Reservation-based Systems

Anthony Sulistio, Kyong Hoon Kim and Rajkumar Buyya
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
{anthony, jysh, raj}@csse.unimelb.edu.au

Abstract

In Grid systems, users may require assurance for completing their jobs on shared resources. Such guarantees can only be provided by reserving resources in advance. In this paper, we introduce an elastic reservation model, where users can query about a resource availability on a given time interval. They can also provide a reservation duration time and/or number of compute nodes needed as soft constraints to the query. Next, we provide an adapted version of an on-line strip packing algorithm, that takes into a consideration of resource utilization when processing reservation requests. We evaluate our algorithm with a real workload trace and show that the proposed algorithm manages a higher resource utilization and number of acceptance compared to an ad-hoc rigid approach.

1 Introduction

Grid [11] and peer-to-peer (P2P) [20] network technologies enable the aggregation of distributed resources for solving large-scale and computationally-intensive applications. These technologies are well-suited for Bag-of-Tasks (BoT) applications [8], wherein each application consists of independent tasks or jobs [1]. Some projects such as Nimrod-G [6], Gridbus Broker [28], and SETI@home [2] utilize these technologies to schedule compute-intensive parameter-sweep applications on available resources [25].

Managing various resources and applications scheduled in highly dynamic Grid environments is a complex and challenging process. Resource management is not only about scheduling large and compute-intensive applications, but also the manner in which resources are allocated, assigned, and accessed. In most scheduling systems, submitted jobs are initially placed into a queue if there are no available resources. Therefore, there is no guarantee as to when these jobs will be executed. This causes problems in time-critical

or parallel applications, such as task graph, where jobs may have interdependencies.

Advance Reservation (AR) is the process of requesting resources for use at specific times in the future [24]. Common resources that can be reserved or requested are compute nodes and network bandwidth. AR in a scheduling system solves the above problem by allowing users to gain *simultaneous* and *concurrent access* to adequate resources for the execution of such applications [27]. Currently, several Grid systems are able to provide AR functionalities, such as GARA [12] and ICENI [18].

In order to reserve the available resources in such AR systems, a user must first submit a request by specifying a series of parameters such as number of resources needed, and start time and duration of his/her jobs [16]. Then, the system checks for the feasibility of this request. If one or more parameters can not be satisfied, then the request is rejected. Hence, this approach is known as an *inelastic* or *rigid* method, because these parameters are hard constraints that do not permit the system any modifications.

Consequently, the user may resubmit new requests with modified existing parameters, such as a different start time and/or duration until available resources can be found. However, this approach will have a negative impact in increasing the communication overheads between users and the resource. Moreover, it will also degrade the performance of the resource in managing many incoming requests due to previously rejected ones. Finally, if such a solution is found, it might not be a good one since it only looks for the first available resources. As a result, it will cause *fragmentations* of AR jobs, which leave behind many gaps of idle time among them. Hence, resource utilization will be significantly lowered.

To overcome the above problem, we introduce an *elastic* reservation model, which takes into a consideration of resource utilization when processing reservation requests. With this model, users can query about the resource availability on a given time interval. They can also provide a reservation duration time and/or number of compute nodes

(CNs) needed as soft constraints to the query. Then, the resource will give the users an offer and/or a list of alternative ones if these constraints can not be met. This approach allows a flexibility to the users to *self-select* or choose the best option in reserving their jobs according to their Quality of Service (QoS) parameters. We adapted an existing on-line strip packing algorithm [9, 17] for this approach.

The rest of the paper is organized as follows. Section 2 describes an overview of the proposed elastic AR model. Section 3 explains the on-line strip packing algorithm in more details. This algorithm is then evaluated in Section 4 on a real workload trace. Section 5 mentions related work whereas Section 6 concludes the paper and suggests some further work to be done.

2 Elastic Advance Reservation Model

2.1 User

In order to reserve compute nodes (CNs) in a resource, a user needs to submit a reservation request. However, before making a request, the user can query about it to increase the chance of getting accepted. This query operation is defined as $queryReserv(t_s, t_e, dur?, numCN?)$, where t_s denotes the earliest starting time interval, t_e denotes the latest ending time interval, dur denotes the reservation duration time, $numCN$ denotes the number of CNs to be reserved respectively. The “?” sign indicates that this attribute is optional one.

Upon receiving the $queryReserv()$ operation, the resource will find a solution or an offer that satisfies both dur and $numCN$ constraints. Otherwise, these parameters are treated as soft constraints and a list of alternative offers are given. The list is defined as $offerList[] = \{ offer(t_s, t_e, numCN) + \}$, where t_s denotes start time, t_e denotes end time, and $+$ denotes one or more occurrences of this tuple. These offers are temporary results generated from this $queryReserv()$ operation. Hence, the user needs to select an offer and to send a $reserv(t_s, t_e, numCN)$ operation for a guarantee. Note that in this paper, we solely focus on reserving CNs as the type of resource. Moreover, $?$ and $+$ signs are borrowed from a W3C recommendation on XML [4].

Figure 1 shows an example of existing reservations, represented as a time-space diagram, in a resource. When a new query from *User5* arrives, i.e. $queryReserv(11, 16, 2, 2)$, the resource checks for any available nodes within $[11, 16]$ time interval. It finds a solution, which is $offer(13, 15, 2)$, that satisfies both dur and $numCN$ constraints. Then, the user sends a reservation request, i.e. $reserv(13, 15, 2)$, to accept this offer.

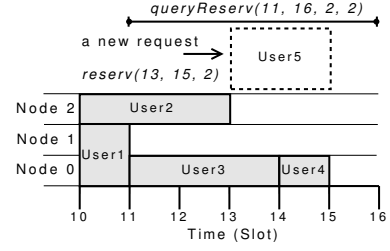


Figure 1. An example of elastic AR with 3 nodes. A dotted box denotes a new request.

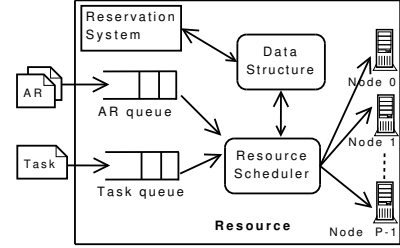


Figure 2. Overall resource model that supports elastic reservation model.

2.2 Resource

Figure 2 shows an open queueing network model of a resource applied to our work. In this model, there are two queues: one is reserved for AR jobs while the other one is for parallel and independent jobs. Each queue stores jobs waiting to be executed by one of P independent CNs. All CNs are connected by a high-speed network. The CNs in the resource can be homogeneous or heterogeneous. In this paper, we assume that a resource has homogeneous CNs, each having same processing power, memory and hard disk.

In this model, each resource has a Reservation System (RS), which is responsible for handling reservation queries and requests. When a resource receives a reservation query or request, it searches for availability. More specifically, the RS checks the data structure for this request. Therefore, the primary role of the data structure is to store and to update the information about CNs' availability as time progresses. Then, a resource scheduler is responsible for managing incoming jobs and assigning them to available CNs.

2.3 Data Structure

A well-designed data structure provides the flexibility and easiness in implementing various algorithms. Hence, some data structures are tailored to specific applications, e.g. a tree-based data structure is commonly used for ad-

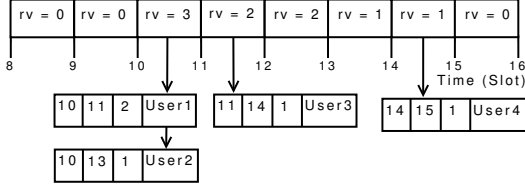


Figure 3. A representation of storing reservations with a sorted queue and $\delta = 1$.

mission control in network bandwidth reservation [5, 29].

For our model, we use an array-based structure for administering reservations efficiently, as shown in Figure 3. It is a *time-slotted* structure, where each slot contains *rv*, the number of already reserved CNs, and a linked list for storing reservations that start at this time. Thus, it partitions *dur* into slots based on a fixed time interval δ . If *dur* spans multiple slots, *rv* on each of them is updated accordingly. Figure 3 shows how reservations are stored with a sorted queue and $\delta = 1$ time interval, by using the example described in Figure 1. For enabling a fast $O(1)$ access to a particular slot, we use the following formula:

$$i = \left\lceil \frac{t}{\delta} \right\rceil \mod M \quad (1)$$

where i is the slot index, t is the request time (in minutes), and M is the number of slots in the data structure. Note that in order not to overlap reservation from different months, we assume that no reservations are made more than one month in advance. As a result, the data structure can be reused for the next month interval. Hence, it is only going to be built once in the beginning.

2.4 Cost

As mentioned previously, in this model we differentiate jobs based on whether they are using AR or not. Therefore, the costs for executing these jobs would also be different. For non-AR jobs, we calculate the running cost as

$$Cost = dur * numCN * bcost \quad (2)$$

where *bcost* is the base cost of running a job at one time unit. Intuitively, the cost for jobs that use AR will incur higher due to the privilege of having guaranteed resources at a future time. Hence, the running cost for AR jobs is charged based on the number of reserved slots in the data structure. More precisely,

$$Cost_{AR} = numSlot * numCN * bcost_{AR} \quad (3)$$

where *numSlot* is the total number of reserved slots, and *bcost_{AR}* is the cost of running the AR job at one time slot. Thus, $bcost_{AR} = \tau * bcost * \delta$, where τ is a constant factor ($\tau \geq 1$) to differentiate the pricing.

3 On-line Strip Packing Algorithm

In this section, we describe how to generate suitable offers for AR requests, by using an adapted on-line strip packing (OSP) algorithm for our elastic model. Since no prior knowledge of AR arrivals is given, the proposed OSP algorithm focuses on finding a solution or alternative offers for each request. Hence, OSP aims to increase resource utilization and to reduce fragmentations. This problem is similar to a strip packing problem, where rectangles are coming in and it is trying to minimize the height of rectangles packed on one bin. Applying this problem to our model, an AR request represents a rectangle of which width and height are *numCN* and *dur* respectively.

Algorithm 1 shows the proposed OSP algorithm for each AR request. If the request does not specify any duration time, then OSP sets the *dur* parameter to be δ by default (line 4). Similarly, *numCN* = 1 if the value of *numCN* is not given (line 5). If both parameters are specified in the request, the OSP algorithm considers them as hard constraints and aims to find a solution. Otherwise, *dur* and *numCN* are treated as soft constraints and OSP only gives a list of suitable offers. The boolean variable *needSol* is used to notify such a case (line 1–3).

After getting these constraints, OSP obtains a list of consecutive slots from the data structure within $[t_i, t_e]$ interval (line 8). We define a *consecutive* slot to be a sequence of slots with the same number of *freeCN*, i.e. $maxCN - slot.rv$, where *maxCN* with the maximum number of CNs. Thus, it reduces the total number of slots needed for searching available CNs. Then, OSP ranks them in an increasing order of *freeCN* (line 10), so that *indexRank*[i] indicates the index of a slot with the ($i + 1$)-th low *freeCN* in *slotList*. It follows such that

$$slotList[iA].freeCN \leq slotList[iB].freeCN$$

where $iA = indexRank[i]$, $iB = indexRank[i + 1]$, and $i = 0, \dots, size - 1$.

Since OSP searches for a possible reservation based on *indexRank*[] (line 12–13), it targets at a slot which represents a local minima, such that *freeCN* at this point becomes all busy in a best scenario. However, the local minima still needs to pass the *numCN* parameter (line 14). Then, OSP tests for the *dur* or *ts* (total slots needed) constraint (line 15–42). The current slot is expanded to the left as long as previous slots satisfy the *numCN* constraint (line 19–27). Likewise, the slot is expanded to the right side if more slots are needed (line 28–36). Next, this slot and the expanded ones are grouped to become a new offer. Subsequently, this offer is added to *offerList* (line 37–38).

This offer is within [*head*, *tail*] slot interval in the data structure and has *minCN* free CNs. If the total number of

Algorithm 1: The OSP algorithm for an elastic AR model.

Input: $queryReserv(t_s, t_e, dur, numCN)$
Output: $offerList[]$ or a list of offers, including a solution (if found)

```
1 if ( $dur \neq \phi$ ) and ( $numCN \neq \phi$ ) then
2   |  $needSol \leftarrow true$ ;
3 else  $needSol \leftarrow false$ ;
   // initialize with a default value
4 if ( $dur == \phi$ ) then  $dur \leftarrow \delta$ ;
5 if ( $numCN == \phi$ ) then  $numCN \leftarrow 1$ ;
6  $offerList[] \leftarrow \phi$ ;
7  $ts \leftarrow get\_total\_slot(dur)$ ; // total slots needed
8  $slotList[] \leftarrow find\_consecutive\_slot(t_s, t_e)$ ;
9  $size \leftarrow get\_size(slotList)$ ; // size of list
   // rank  $slotList[]$  in the increasing order
   // of  $freeCN$  and returns its indices
10  $indexRank[] \leftarrow get\_sorted\_index(slotList[])$ ;
   // a loop to search for offers
11 for ( $i = 0$ ) to ( $size - 1$ ) do
12    $index \leftarrow indexRank[i]$ ; // current index
13    $slot \leftarrow slotList[index]$ ; // current slot
   // skips the rest and increments  $i$ 
14   if ( $slot.freeCN < numCN$ ) then continue;
15    $head \leftarrow indexRank[i]$ ; // starting index
16    $tail \leftarrow indexRank[i]$ ; // ending index
17    $totSlot \leftarrow slot.numSlot$ ; // total slot found
18    $minCN \leftarrow slot.freeCN$ ; // lowest  $freeCN$ 
   // look at previous slots (left side)
19   for ( $l = index - 1$ ) to ( $l \geq 0$ ) do // decrement
20      $freeCN \leftarrow slotList[l].freeCN$ ;
21     if ( $freeCN < numCN$ ) or ( $totSlot \geq ts$ ) then
22       | break;
23     end
24      $head \leftarrow l$ ; // starts from this slot
25      $totSlot \leftarrow totSlot + slotList[l].numSlot$ ;
26      $minCN \leftarrow \min(freeCN, minCN)$ ;
27   end
   // look at next slots (right side)
28   for ( $r = index + 1$ ) to ( $r \leq size - 1$ ) do
29      $freeCN \leftarrow slotList[r].freeCN$ ;
30     if ( $freeCN < numCN$ ) or ( $totSlot \geq ts$ ) then
31       | break;
32     end
33      $tail \leftarrow r$ ; // ends until this slot
34      $totSlot \leftarrow totSlot + slotList[r].numSlot$ ;
35      $minCN \leftarrow \min(freeCN, minCN)$ ;
36   end
37    $offer \leftarrow make\_offer(head, tail, totSlot, minCN)$ ;
38    $offerList[] \leftarrow add\_offer(offerList[], offer)$ ;
   // found a solution
39   if ( $totSlot \geq ts$ ) and ( $needSol == true$ ) then
40     |  $offerList[] \leftarrow found\_sol(offer, offerList[])$ ;
41     |  $needSol \leftarrow false$ ; break;
42   end
43 end
44  $offerList[] \leftarrow set\_cost(offerList[])$ ;
45 return  $offerList[]$ ;
```

slots, $totSlot$, from this offer meets the ts and $needSol$ objectives, then it is marked as a solution (line 39–42). Moreover, the $found_sol()$ function moves this offer to the top of the list to become the first choice (line 40). The order of offers in the list can be sorted based on policies of the resource. Once all offers have been created, OSP applies the total cost to them (line 44).

As a result, the OSP algorithm returns a solution and/or a list of offers for the given reservation request. Then, the user decides or *self-selects* one of these offers according to his/her QoS parameters. Moreover, the user is given the flexibility to reduce dur and/or $numCN$ of an offer. Overall, the time complexity for this OSP algorithm is $O(n^2)$ where n is the number of consecutive slots.

4 Performance Evaluation

In order to evaluate the performance of our proposed algorithm, i.e. an On-line Strip Packing (OSP) algorithm, we compare it to a First Fit (FF) algorithm. Moreover, we introduce a Rigid algorithm as a base comparison. The FF algorithm only looks for the first available CNs within a given time interval, whereas the Rigid algorithm treats t_s , dur and $numCN$ as hard constraints. Therefore, if no solution is found, then the Rigid algorithm will reject the request. Note that only the OSP algorithm provides a list of offers for this experiment.

For scheduling and running jobs, we incorporate First Come First Serve (FCFS) and Easy Backfilling (BF) [19] policy into the above algorithms. Hence, we combine FF and Rigid algorithm with both FCFS and BF respectively, and OSP algorithm with BF only. Moreover, all of them use the same data structure with $\delta = 5$ minutes, and has a fixed interval length of 30 days. Finally, we set $bcost = \$0.05$ per minute and $\tau = 4$.

We carried out the performance evaluation by using simulation, because we need to conduct *repeatable* and *controlled* experiments that would otherwise be difficult to perform in real Grid testbeds. Therefore, we use GridSim toolkit [26] by simulating a cluster with $maxCN = 64$.

4.1 Experimental Setup

We use a workload trace of the San Diego Supercomputer Center (SDSC) Blue Horizon obtained from the Parallel Workload Archive [10]. This trace is chosen because it represents a large number of jobs and contains a mixture of single and parallel jobs. Note that we only simulate the first 2-weeks period of the trace, which is approximately 3200 jobs, since the original trace was recorded over a two-year period. We selected 30% of these jobs to use reservation. Several modifications have also been made to this trace, as mentioned below:

- If a job requires more than the total CNs of a resource, we set this job to the maximum number of CNs.
- A request's starting time is rounding up to the nearest time interval. For example, if a job requests to start at time 01:03:05 (hh:mm:ss), then it will be moved to time 01:05:00.
- A job duration time is within [4 minutes, 28 days]. We limit the maximum duration time to prevent overlapping reservations from different months. Hence, the data structure can be reused and built only once.

For the evaluation, we are investigating: (i) the effects of having an elastic AR model, which include the average resource utilization, the average revenue generated for running AR jobs, and the number of rejection for reserving these jobs; (ii) the impact of such model to non-AR jobs, where we measure the average waiting time they spent in a queue; and (iii) the degree of such flexibility offered to AR jobs, where we vary the following parameters:

- book-ahead time, b_t , where it is the time difference between ti_s and the job starting time t_s as stated in the trace. In the experiment, we use $b_t \in \{1, 5, 10\}$ hours prior to t_s .
- search limit time, sl_t , where it is the time added to the ending time interval. In the experiment, we use $sl_t \in \{0, 1, 2, 4, 6, 8, 10, 12\}$ hours.
- time interval, ti , where it is the time period where a job needs to be reserved. We define the starting time interval, $ti_s = b_t + t_s$, and the ending time interval, $ti_e = ti_s + dur + sl_t$.

4.2 The User's Selection Policy

As mentioned earlier, a user sends a reservation query to a resource, and receives a list of offers, $offerList[]$. Algorithm 2 shows a selection policy of the user when no solutions are found (line 1–8). The user is willing to accept an offer by reducing the initial dur and $numCN$ values, by up to a half or δ and 1 respectively (line 1–2). Hence, the list is sorted in decreasing order based on the duration time, i.e. from the longest to the shortest duration time (line 3). Then, each offer in the list is checked against $minDur$ and $minCN$ (line 4–7). If no suitable offers are found, then the user rejects all of them (line 8).

Note that this selection policy is overly simplified and might not be feasible in real Grid applications. However, we do this in order to demonstrate the elasticity of the proposed model and the effectiveness of the OSP algorithm.

Algorithm 2: The selection policy of a user.

Input: $offerList[]$ or a list of offers

```

1  $minDur \leftarrow \max(dur / 2, \delta)$ ;
2  $minCN \leftarrow \max(numCN / 2, 1)$ ;
3  $offerList[] \leftarrow \text{sort\_decreasing}(offerList[])$ ;
4 for ( $i = 0$ ) to ( $size - 1$ ) do
5    $offer \leftarrow offerList[i]$ ;
6   if  $is\_suitable(offer, minCN, minDur) == \text{true}$ 
7     then return  $offer$ ;
7 end
8 return  $\phi$ ;           // no suitable offers found

```

4.3 Experimental Results

Figure 4 shows the effects of having an elastic AR model on the average resource utilization. The result of this figure is influenced by the choice of a good scheduling policy, where BF manages to perform much better than FCFS. Moreover, having a degree of flexibility in reservation requests allows an additional improvement in most cases. As expected, Rigid+BF performs worse than FF+BF and OSP+BF overall, because it treats the input parameters as hard constraints. Hence, b_t and sl_t do not have any effects on the Rigid+BF algorithm.

OSP behaves slightly worse to FF for $b_t = 1$ hour since the starting time interval ti_s is too short to make any improvements for the resource utilization. However, when $sl_t \geq 6$ hours, the performance of OSP is improving, and on average OSP performs better than FF for $b_t = \{5, 10\}$.

Figure 5 looks at the resource utilization in more details, as it shows the total consumption of CNs for the entire duration. FF+BF and Rigid+FCFS, as shown in Figure 5 (a) and (c) respectively, fluctuate frequently throughout. This condition can be interpreted as having too many fragmentations. In contrast, OSP+BF manages fragmentations better since AR jobs are assigned to slots within a local minima of free CNs, as displayed in Figure 5 (b).

As a result, jobs that require many CNs have a higher probability of being accepted compare to FF+BF and Rigid+FCFS. Moreover, with an elastic model, users can reduce $numCN$ and/or dur values according to Algorithm 2. Thus, OSP has the lowest number of rejection as sl_t increases, as mentioned in Figure 6.

Figure 7 shows that by allowing users to select an alternative offer if no solutions are found, this approach reduces the total number of rejection around 13.50% ($b_t = 5, sl_t = 0$) to 77.22% ($b_t = 10, sl_t = 12$). Hence, this result translates to more revenue being generated.

Figure 8 shows the average revenue for running AR jobs. The Rigid algorithm has the lowest revenue due to high number of rejections as expected. Since AR jobs is charged higher to run than normal ones, the elastic model has a ma-

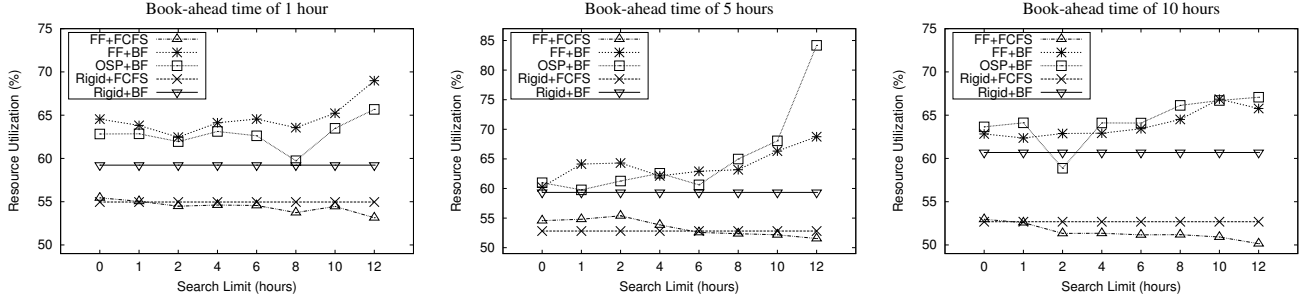


Figure 4. Average resource utilization.

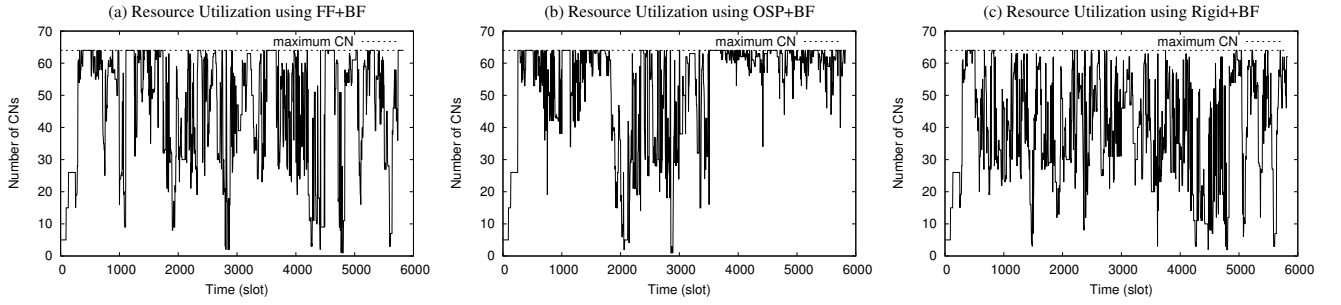


Figure 5. Total number of busy CNs over a two-week period, with $\delta = 5$ minutes, $b_t = 5$ hours and $sl_t = 8$ hours.

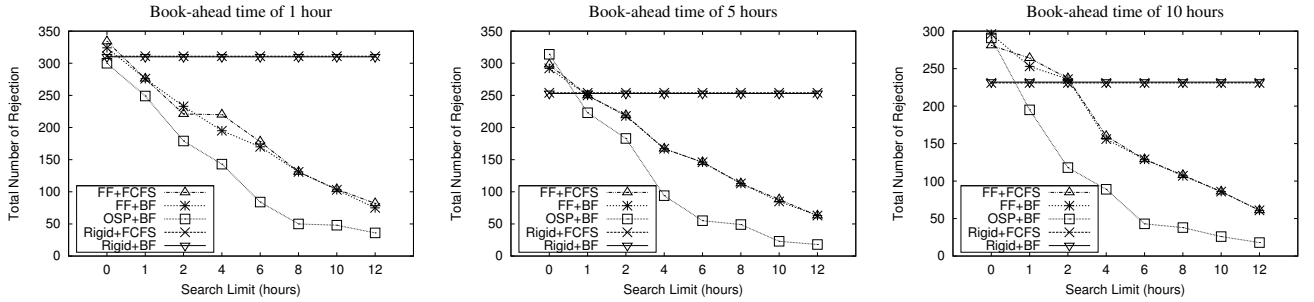


Figure 6. Total Number of Rejection (lower is better).

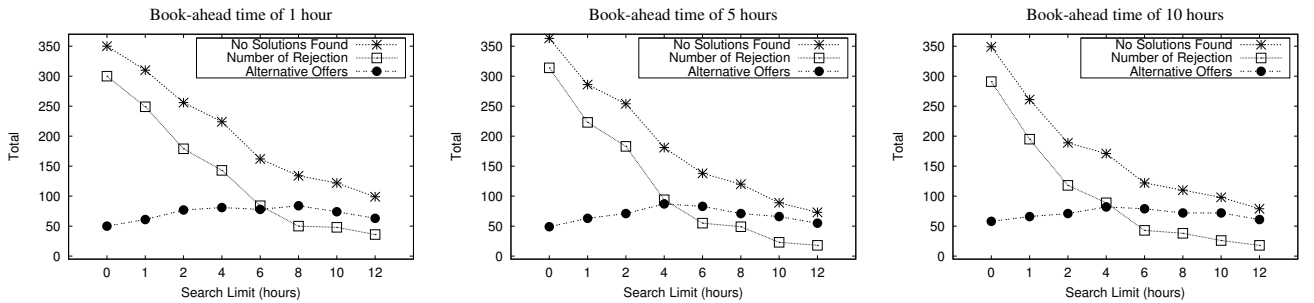


Figure 7. Degree of flexibility in reserving AR jobs for the $OSP + BF$ algorithm.

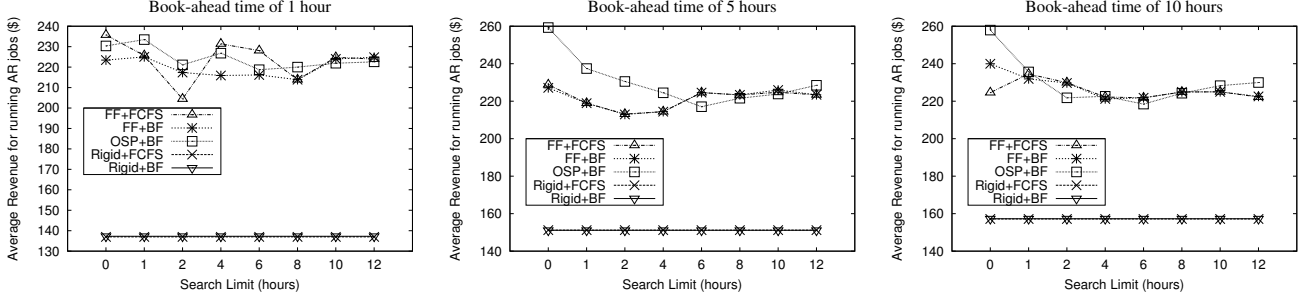


Figure 8. Average revenue for running AR jobs.

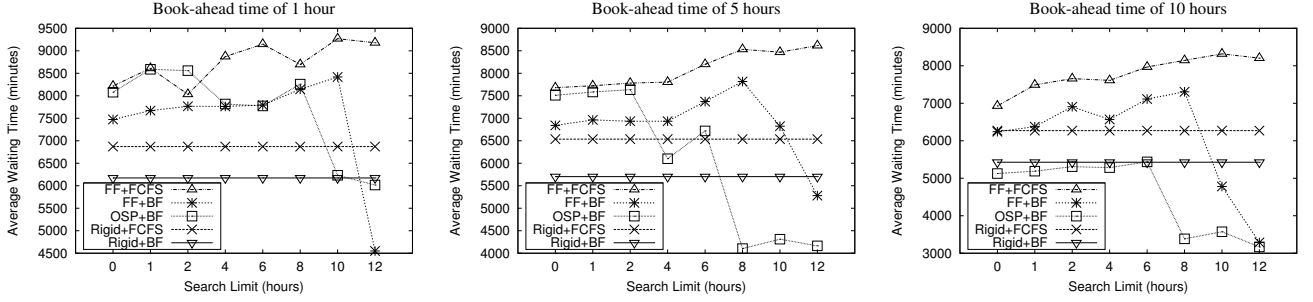


Figure 9. Average waiting time for non-AR jobs (lower is better).

major advantage in generating more revenues, as mentioned earlier. This is because, although a solution might not be found, the user can *self-select* the given offers.

Figure 9 shows the impact of reservation for non-AR jobs, in terms of the average waiting time. The impact is worse when a request has a short time interval, hence no room for flexibility. As b_t becomes larger, OSP+BF manages to minimize the waiting time significantly. If b_t is constant, the waiting time is reduced further if sl_t becomes larger in some cases. However, this result is influenced by the frequency of jobs arrival rate and the choice of a good scheduling policy, where BF performs better than FCFS.

5 Related Work

Strip packing is a generalization of bin packing [15]. Bin packing is an NP-hard problem, where it aims to minimize the number of bins used to store a set of rectangles. Many variants of bin packing have been proposed by several researchers [3, 21] to find a solution.

A flexible method for reserving jobs in Grids has been presented [7, 13, 14], where they talk about extending the reservation time interval or window in order to increase the success rate. However, they do not provide alternative offers if the reservation is rejected. On the other hand, the work done by [22, 23] provides this important functionality.

The fuzzy model introduced by [22] provides a set of pa-

rameters when requesting a reservation and applies speedup models for finding the alternative solutions. However, no optimization on the resource utilization is considered.

The model proposed by [23] uses a 3-layered negotiation protocol, where the allocation layer deals with flexible reservations on a particular Grid resource. In this layer, the authors also used a strip packing method, but the resources are partitioned into different shelves, and each shelf is associated with a fixed time length, number of CNs and cost. Hence, the reservation request is placed or offered into an adjacent shelf that is more suitable. This will lead into higher resource fragmentations. In contrast, our algorithm aims at reducing these fragmentations, as shown in Figure 5.

6 Conclusion and Future Work

Advance Reservation (AR) in Grid computing is an important research area as it allows users to gain concurrent access to resources by allowing their applications to be executed in parallel. It also provides guarantees on the availability of resources at the specified times in the future.

In this paper, we introduce an *elastic* model, which takes into a consideration of resource utilization when processing reservation requests. With this model, users can query about the resource availability on a given time interval. They can also provide a reservation duration time and/or number of compute nodes needed as soft constraints to the query.

Then, the resource will give a solution and/or a list of alternative options if these constraints can not be met, by using an adapted on-line strip packing (OSP) algorithm. This approach enables a degree of flexibility to the users for choosing the best option in reserving their jobs according to their Quality of Service (QoS) parameters.

We evaluate the performance of the OSP algorithm using a workload trace from San Diego Supercomputer Center. The results showed that OSP accepts more reservations than First Fit and Rigid, which translate to generating more revenue for the resource. In addition, OSP maintains a good level of resource utilization, which is not too far from First Fit. Finally, the impact of reservations to local jobs can be minimized significantly by using OSP and receiving queries with a larger time interval or a higher degree of flexibility.

An extension to this work is to consider a runtime estimation and heterogeneous resources with fault tolerance in the model. In addition, a further work is needed to implement and to evaluate our algorithm in a real Grid system.

References

- [1] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid? In *Proc. of the 14th Intl. Symposium on Parallel and Distributed Processing*, Mexico, May 1–5 2000.
- [2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
- [3] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *Operational Research Society*, 38(5):423–429, 1987.
- [4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, editors. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C, 2006. <http://www.w3c.org/TR/xml>.
- [5] A. Brodnik and A. Nilsson. A static data structure for discrete advance bandwidth reservations on the internet. In *Proc. of Swedish National Computer Networking Workshop (SNCNW)*, Stockholm, Sweden, September 2003.
- [6] R. Buyya, D. Abramson, and J. Giddy. Nimrod-G: An architecture for a resource management and scheduling system in a global computational grid. In *Proc. of the 4th Intl. Conference & Exhibition on High Performance Computing in Asia-Pacific Region (HPC Asia)*, Beijing, China, May 2000.
- [7] M. Caramia, S. Giordani, and A. Iovanella. Grid scheduling by on-line rectangle packing. *Network*, 44(2):106–119, 2004.
- [8] W. Cirne, F. Brasileiro, J. Sauve, N. Andrade, D. Paranhos, E. Santos-Neto, and R. Medeiros. Grid computing for bag of tasks applications. In *Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and E-Government*, Sep 2003.
- [9] W. F. de la Vega and V. Zissimopoulos. An approximation scheme for strip packing of rectangles with bounded dimensions. *Discrete Appl. Math.*, 82(1-3):93–101, 1998.
- [10] D. Feitelson. Parallel workloads archive. <http://www.cs.huji.ac.il/labs/parallel/workload>, 2007.
- [11] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.
- [12] 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. In *Proc. of the 7th IWQoS*, London, UK, 1999.
- [13] C. Hu, J. Huai, and T. Wo. Flexible resource reservation using slack time for service grid. In *Proc. of the 12th Intl. Conference on Parallel and Distributed Systems (ICPADS)*, Minneapolis, USA, July 12–15 2006.
- [14] N. R. Kaushik, S. M. Figueira, and S. A. Chiappari. Flexible time-windows for advance reservation scheduling. In *Proc. of the 14th Intl. Symposium on Modeling, Analysis, and Simulation (MASCOTS)*, California, USA, Sep. 11–13 2006.
- [15] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985.
- [16] J. MacLaren, editor. *Advance Reservations: State of the Art (draft)*. GWD-I, Global Grid Forum (GGF), June 2003.
- [17] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS J. on Computing*, 15(3):310–319, 2003.
- [18] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow enactment in ICENI. *UK e-Science All Hands Meeting*, pages 894–900, September 2004.
- [19] A. W. Mu’alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.
- [20] A. Oram, editor. *Peer-to-peer: Harnessing the Power of Disruptive Technologies*. O’Reilly Press, 2001.
- [21] W. T. Rhee. Optimal bin packing with items of random sizes. *Math. Oper. Res.*, 13(1):140–151, 1988.
- [22] T. Roelblitz, F. Schintke, and A. Reinefeld. Resource reservations with fuzzy requests. *Concurr. Comput. : Pract. Exper.*, 18(13):1681–1703, 2006.
- [23] M. Siddiqui, A. Villazon, and T. Fahringer. Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *Proc. of the 2006 ACM/IEEE conference on Supercomputing (SC’06)*, Florida, USA, November 2006.
- [24] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proc. of the Intl. Parallel and Distributed Processing Symposium*, Cancun, Mexico, May 1–5 2000.
- [25] A. Sulistio and R. Buyya. A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems. In *Proc. of the 17th Intl. Symposium on Computer Architecture on High Performance Computing (SBAC-PAD)*, Rio de Janeiro, Brazil, October 24–27 2005.
- [26] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. On incorporating differentiated levels of network service into GridSim. *FGCS*, 23(4):606–615, May 2007.
- [27] A. Sulistio, W. Schiffmann, and R. Buyya. Advanced reservation-based scheduling of task graphs on clusters. In *Proc. of the 13th Intl. Conference on High Performance Computing (HiPC)*, Bangalore, India, Dec. 18–21 2006.
- [28] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience (CCPE)*, 18(6):685–699, 2006.
- [29] T. Wang and J. Chen. Bandwidth tree – a data structure for routing in networks with advanced reservations. In *Proc. of the 21st Intl. Performance, Computing, and Communications Conference*, pages 37–44, Phoenix, USA, 2002.