# Efficiently Managing Advance Reservations Using Lists of Free Blocks

Joerg Schneider and Barry Linnert
*Technische Universitaet Berlin*
{*komm, linnert*}*@cs.tu-berlin.de*

## Abstract

*Advance reservation was identified as a key technology to enable guaranteed Quality of Service and co-allocation in the Grid. Nonetheless, most Grid and local resource management systems still use the queuing approach because of the additional complexity introduced by advance reservation. A planning based resource management system has to keep track of the reservations in the future and needs a good overview on the available capacity during the negotiation of incoming reservations. For advance reservation, the resource management problem becomes a two dimensional problem. In this paper different data structures are investigated and discussed in order to fit to planning based resource management. As a result the benefits of using lists of resource allocation or free blocks are exposed. This general idea widely used to manage continuous resources is extended to cover not only the resource dimension but also the time dimension. The list of blocks approach is evaluated in a Grid level and a resource level resource management system. The extensive simulations showed a better runtime and higher reservation success rate compared with the currently favored approach of a slotted time.*

## 1. Introduction

As Grid computing is established as the collaborative usage of distributed resources, it faces different challenges. Most of all the resources, such as compute nodes or whole compute systems and network bandwidth, are scarce ones – especially in the field of high performance computing.

Most problems of sharing scarce resources are solved by queuing up the requests. The concept is the same for shopping in a grocery store as for high performance computing. The concept is very simple to realize and, by using a first-come-first-serve-strategy, also provides means to predict the actual start time.

For combined transactions — so called *co-allocations* — with multiple providers, this concept is already too weak. If one wants to travel, it wouldn't be sufficient to queue up at the airline counter and then after arriving at the destination to enter the queue for the hotel without even knowing if there will be a room available on the same day.

If one can choose between service providers, each using an independent queue, it is also hard to select the one which will be available first. This is also a commonly known problem in grocery stores. But in that setup, it is at least possible by earlier experiences and the transparent view on the shopping cart of the other customers to make a prediction of the waiting time. In Grid environment, the queues are usually not transparent and if there is no additional hint by the other users or the service provider, the wait time cannot be predicted [1]. Hence, a user cannot compare independent resource providers.

Another concept also used in the everyday life to avoid the problems of the queuing approach, is to negotiate the actual start time with the service provider. This means that the user contacts the service provider in advance, usually when the need for the service is in sight, and negotiates a start time in the future. Both parties agree that the service will be fulfilled then, i.e., the service provider will have all needed resources available and the user will show up to use the service.

So, this *advance reservation* approach has a number of benefits for the user and the service provider. By negotiating with different service providers, the user can coordinate the execution of co-allocations. In the travel example the usual approach is to book the flight and the hotel room in advance and if the rooms are only available on other dates, to adapt the flight date and vice versa. Another benefit for the user is that he can negotiate with multiple service providers for the same service and can compare the waiting time. The service provider increases the service level with advance reservation, as he can now guarantee that the service will be finished by some given deadline [2].

Using advance reservation also implies some drawbacks. The most obvious one is that it is much more complex to handle then the queuing approach. In the real world a queue is easily formed by the waiting customers and the service provider only has to provide some space for queuing. Even in a compute system, a queue of waiting jobs is just a simple data structure. For advance reservation, someone is needed to negotiate with the user together with some infrastructure to keep track of all the advance reservations and to identify free capacities in the future. Therefore, there is a significant overhead for the resource provider.

In this paper we investigate the impact of different technologies to implement advance reservation in the field of high performance computing. The theoretical discussion is verified by experiments with a cluster manager and a Grid scheduler.

The negotiation of the start time may lead to time frames without resource usage, which are too small to fulfill further incoming requests. Therefore, there is not only a fragmentation in the resource dimension, but also in the time dimension. In [3] this two-dimensional fragmentation problem is discussed in depth.

In a system mixing queued jobs with advance reservations, the utilization drops due to the fixed advance reservation. The reservations obviously disturb the queue reordering (backfilling), often used in queuing based systems to increase the utilization [4]. Sulistio and Buyya measured a utilization drop of 60-80% and increased wait times for the jobs in the queue [5]. However, Heine et.al. showed that by using a native planning based resource management system instead of an enhanced queuing based, the impact of the advance reservations is less dramatic [6]. In [7] a comprehensive overview on resource management systems supporting advance reservation is given.

Summarizing, advance reservation is an elaborated technology allowing the coordinated allocation of jobs. However, a lot of additional information is required and a processing overhead is introduced. Nonetheless, it is an important base technology to process complex Grid workflows and to guarantee Quality of Service by using SLAs to ensure deadlines for the job in the Grid environment.

## 2. Formalization

After introducing the basic concept of advance reservation, the following section provides a more formal view on the concept of advance reservation.

A job $j$ is any request for resource usage. It is specified by the resource type it requests $T_j$, how much of this resources capacity will be used $c_j$, and

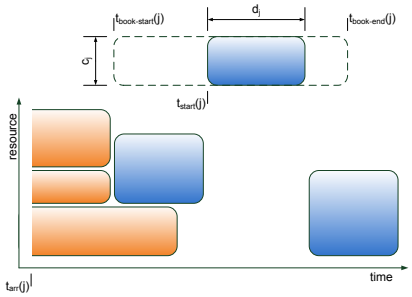



Figure 1. Advance reservation allows planning the execution of jobs in the future.

the duration of the resource usage $d_j$. The unit of $c_j$ depends on the type of the requested resource, e.g., number of CPUs for parallel computer, kbit/s bandwidth for networks, or just 0 or 1 for single-unit resources like 3D visualization systems. The duration is finite and the job will be executed once. Periodic executions are not considered in the resource manager, while the user may periodically submit the same job.

Each resource $r \in R$ is defined by its type $T_r$ and the available capacity $c_r$. Jobs arrive at the resource management system (RMS) before their respective execution starts. This moment is called *arrival time* of the job $t_{arr}(j)$ (see Figure 1). The requester may also restrict the possible execution time by providing a *booking interval* with the start $t_{book-start}(j)$ and end time $t_{book-end}(j)$. In order to get at least one possible start time,

$$t_{book-start}(j) + d_j \leq t_{book-end}$$

has to hold. If no booking interval is specified $t_{book-start}(j) = t_{arr}$ and $t_{book-end}(j) = \infty$ will be used.

The RMS will then determine a matching start time $t_{start}(j)$ for the job, such that

$$\begin{aligned} t_{book-start}(j) &\leq t_{start}(j) \quad \text{and} \\ t_{start}(j) + d_j &\leq t_{book-end} \end{aligned}$$

The difference between the arrival time and the actual start time is called *book-ahead time* $t_{book-ahead}(j) = t_{start}(j) - t_{arr}(j)$. To reduce the complexity of the scheduling process the RMS may impose a maximum book ahead time $\hat{t}_{book-ahead}$.

The RMS has to keep track of all already reserved jobs $J_r$, the respective start times, and resource allocations. In general, only as much capacity can be reserved as is available on the resource:

$$\forall t : c_r \geq \sum_{j \in J_r \wedge t_{start}(j) \leq t \leq t_{start}(j)+d_j} c_j$$

Hence, the scheduling problem can be formulated as: Determine a $t_{start}(j)$ such that

$$\forall t_{start}(j) \leq t \leq t_{start}(j) + d_j :$$
$$c_r \geq c_j + \sum_{j' \in J_r \wedge t_{start}(j') \leq t \leq t_{start}(j') + d_{j'}} c_{j'}$$

This formula ignores the actual mapping of the jobs to the underlying units of the resource. However, this inner-resource mapping has no impact if the job can run on any arbitrary sub-set of resource units.

## 3. Applications

As stated before, in the Grid domain advance reservation was early identified as necessary for co-allocations and guaranteed finish times [8]. Hence, there are a number of articles covering the negotiation and handling of advance reservation [9], [10], [11], [12], [2], [13], [14].

An example for an advance reservation enabled Grid broker is the *Virtual Resource Manager* (VRM) [15]. In the VRM architecture the active domain controller (ADC) plays the role of the Grid broker. It connects to the active interfaces (AI) running on the frontends of the connected resources. The AIs are not only adapter to the specific local resource management system but also enforce the information hiding policy set by the local administrator, i.e., they can be configured to provide only a limited view on the current state and the future reservations. For queuing based resources and unmanaged resources, the AIs can even emulate advance reservation to some degree [16]. The scheduling of the ADC bases its Grid scheduling on advance reservation support in the underlying resources. This way it supports Grid workflows—compositions of multiple dependent jobs including their network requirements—and co-allocations. The VRM needs to keep track of the reservations on the Grid level in the ADC and in case of emulated advance reservation also at the resources in the AI. The VRM in its simulation mode is used as one of the test platforms in the evaluation.

Most advance reservation Grid broker rely like the VRM on the advance reservation support in the local resource management system. For cluster computer some advance reservation enabled scheduler are available like OpenCCS[1], MAUI[2], and Crono[3].

To speed up the evaluation, we developed a simulation framework for cluster scheduler. It supports advance reservations for arbitrary cluster topologies and architectures. It is able to compare different scheduling strategies including strategies using only restricted information and dynamic application behavior. The simulation framework has two levels of schedules, too. First, a global schedule holds the information of the summed up booked capacity while a set of schedules—one for each cluster node—holds the mapping information and at which times the individual nodes are available. The global schedule has both time and resource dimension while the local schedules have only the time dimension (a node is only reserved or free, but not partly reserved).

Advance reservation is also available for other resource types like network bandwidth. How advance reservation can be done in networks is discussed in detail in [17] and [18].

1. https://www.openccs.eu/core/
2. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php
3. http://crono.sourceforge.net/

## 4. Slotted time

As described before, the main problem of an advance reservation scheduler is to keep track of all already reserved jobs and to answer the question: "Is there at least $c_j$ capacity left, for a continuous time range with the width $d_j$".

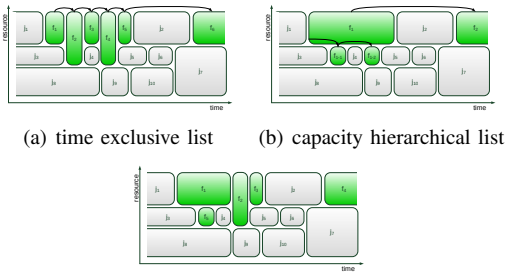
The formal representation of this question (as shown in Section 2) implies a straight forward implementation:

```
1  choose a t_start(j)
2  for all t with t_start(j) ≤ t ≤ t_start(j) + d_j
3     select all reserved jobs in J_r,
          which are planned to run at t
4     sum up the capacity of the selected jobs
5     check if there is enough capacity for the job
6     if not start again with another t_start(j) or
          quit and reject job
7  accept job with t_start(j)
```

If the time $t$ is an arbitrary real value, the loop would be executed endlessly. By discretizing the possible values of the time, this problem can be solved. Therefore, the time is divided in time slots of a fixed length $\Delta t$. The time slot $t_i$ represents all time stamps $t$ with $i\Delta t \leq t < (i+1)\Delta t$.

Now for all time slots, the reservations, which are at least partly within the timeslot, are associated with the time slot. If a reservation starts or ends within a time slot, the resource usage is counted for the whole time slot. This makes the algorithm simpler, but also leads to internal fragmentation.

Using time slots reduces the number of possible start times. Therefore, the runtime performance of the algorithm can be easily adjusted with the time slot width $\Delta t$.



(a) time exclusive list (b) capacity hierarchical list

(c) mixed list

Figure 2. The three different approaches to organize a two-dimensional free block list.

Furthermore, an array can be used as a ring buffer data structure to manage the time slots [19]. The size of the array depends only on the maximum allowed book-ahead time $\hat{t}_{book-ahead}$.

The slotted time approach provides an easy to handle, fast implementation of advance reservations. The runtime as well as the memory footprint of the array is restricted. Both metrics depend linearly on the maximum book-ahead time and the time slot width.

On the other hand, this approach requires selecting a maximum book ahead time and restricts, therefore, the possible reservations. As discussed before, the slot size has to be chosen carefully, as long time slots lead to more internal fragmentation while short time slots increase the runtime and memory footprint. If there is no load in all requested time slots, the algorithm can only detect this by iterating all slots. The same holds for schedules with long running jobs and thus only few changes in the booked capacity over the time. Thus, the algorithm performs more checks than necessary.

## 5. List of free capacity blocks

In order to avoid these unnecessary checks, we developed a new approach to manage two dimensional schedules. We based our approach on a concpet commonly employed to manage continuous range of free resources and adapted it in order to implement advance reservation: A list, where each entry represents a range of free resources.

Hoewver, In order to use this concept to manage multi-unit advance reservations, some additions are needed. Each list item $f \in F_r$ is structured similar to a job and represents a rectangle of unused resources: a time range with the width $d_f$ with always at least $c_f$ unused capacity.

All items in the list $F_r$ have to cover all unused

```
1  select a block f ∈ F_r with c_f ≥ c_j and
       t_start(j) ≤ t_start(f) ≤ t_book−end(j) − d_j
2  initialize duration d_found := d_f
3  initialize last free block used f_last := f
4  while d_found < d_j
5     go to successor of f_last: f'
6     if t_end(f_last) ≠ t_start(f') (not adjacent)
            or d_f' < d_j (not enough capacity)
7        go back to the first step or quit
8     increase duration d_found := d_found + d_f'
9     set last free block used f_last := f'
```

Figure 3. Determining a reservation candidate in a time exclusive free list.

resources and must not intersect:

$$\forall t : c_r \quad = \sum_{j \in J_r \wedge t_{start}(j) \le t \le t_{start}(j)+d_j} c_j + \sum_{f \in F_r \wedge t_{start}(f) \le t \le t_{start}(f)+d_f} c_f$$

Figure 2 depicts the three options to organize the items of the list:

- *Time exclusive list.* For each point in time there is only one item, representing the current available capacity. The list is ordered by the start time of the blocks, i.e., adjacent blocks follow each other in the free list.
- *Capacity hierarchical list.* Each item spans the whole time span where at least the given capacity is free. During this time span, there may be other sub time spans with more capacity available; these time spans are managed as sub lists of the longer block. Hence, a hierarchical data structure is used.
- *Mixed list.* The splitting of the list items does not follow any rule. The items may be ordered by the start time and the available capacity. The list items should have references to all adjacent free blocks.

Figure 2 also shows another difference to the classical lists of free blocks. In the one-dimensional case, two blocks, which are directly adjacent, would be joined to form a bigger block. In the two-dimensional case this is only possible in the time dimension, if the capacity is the same or in the capacity dimension, if the time span is the same.

In the one-dimensional case, reserving a job requires only iterating the list until a block is found which provides at least the requested amount of resources. This method can be used in the two-dimensional case, too. But it will not necessarily find all options. There

```
1  select a block f ∈ F_r with d_j ≤ d_f
2  initialize capacity c_found := c_f
3  initialize last free block used f_last := f
4  while c_found < c_j
5     select a block f' from the sub list of f_last
         with d_j ≤ d_f
6     if no block is found
7        use backtracking to change selection
            of formerly selected blocks
8     increase capacity c_found := c_found + c_f'
9     set last free block used f_last := f'
```

Figure 4. Determining a reservation candidate in a capacity hierarchical free list.

may be multiple adjacent free blocks necessary to cover an area as big as requested. A simple example are two adjacent blocks which have more capacity left than requested, but can only serve the job together in the time dimension.

Therefore, the allocation process has to be enhanced to include adjacent blocks. In a time exclusive list the approach is similar to the one used for the slotted time (see Figure 3). The main difference to the slotted time approach is the variable length of the analyzed free blocks and that the blocks are not necessarily adjacent which is always the case in the slotted time model.

Changing the time exclusive free list based on a found reservation candidate means to decrease the capacity of all involved blocks. If the job start or end time does not match the start of the first involved block or the end of the last involved block, respectively, these blocks have to be splitted. If the capacity of a block becomes zero the block just gets deleted.

While in the time exclusive list, first a block with sufficient capacity was selected, in the capacity hierarchical case first a block with a sufficient long time span is selected as depicted in Figure 4.

Changing the free list when a job was reserved is much more complex than in the time exclusive list. The algorithm above quickly returns only the answer to whether there is enough capacity. In order to subtract the now reserved capacity from the free list, the hierarchy has to be traversed completely to identify the free blocks intersecting with the reservation in the lowest layer. If the blocks provide less capacity than requested, the whole block will be changed to the remaining capacity. If the reservation starts or ends within the block, a new sub list with one or two blocks representing the remaining capacity before and after the reservation will be added to this block. Additionally, the sub list of the splitted block have to be splitted. If the capacity is not sufficient, the block will be deleted and the not yet allocated capacity will be removed in the same way from the parent block and so on. Therefore, the allocation is a complex splitting operation within the tree-like structure.

In a mixed list it is obviously very hard to find a reservation candidate, as there has to be a search in both time and capacity dimension. The search effort could be eased a bit, if adjacent blocks carry references to each other.

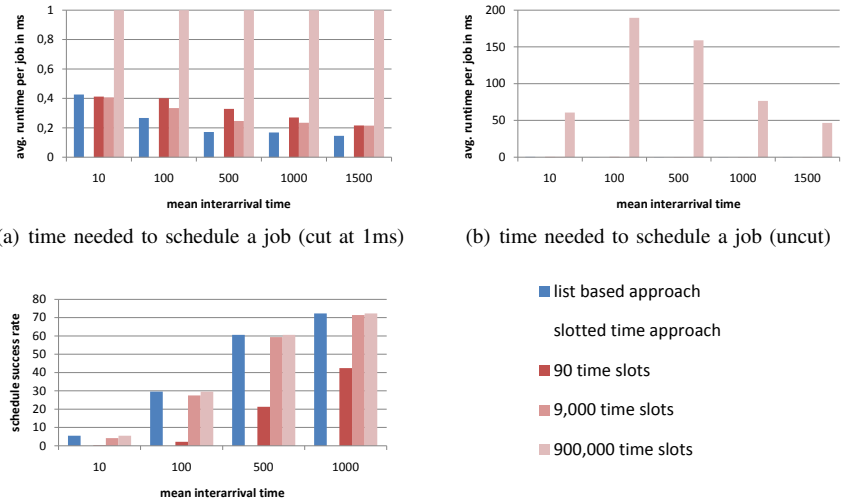
Summarizing, free block lists provide a mean to save reservations with arbitrary start and end times without internal fragmentation coming with the data structure. In an empty or low loaded system, the free lists contain only a small number of list items. Still there is no lower limit as in the slotted time approach. If there are a lot of small reservations resulting in high fragmentation, the free block lists become very long and have to be processed linearly. In the worst case, there is a list item for every distinguishable point in time, e.g., every millisecond. Furthermore, in all three variants, the free block lists need more complex implementations than the slotted time approach.

## 6. Related Work

The question which implementation is the best to hold the schedule information is also discussed in the literature. Singh et al. [11] used a mixed free list where they called each block a slot. If there are adjacent reservations in the time dimension, the free blocks are called inner slots. Inner slots cannot be splitted, i.e., the user has to book the whole free block, regardless of the actually requested size. On the other hand, reservations had to fit in a single block as combinations were not considered. In their model, only blocks which mark the end of the list in the time dimension, could be splitted in both dimensions. This procedure reduces the fragmentation, keeps the free block list small, and provides a simple allocation algorithm. However, the user is forced to reserve (and pay) for a larger block than actually required.

Castillo developed a system to match advance reservations for single-unit resources using techniques from computational geometry [14]. The free list is held as a tree or in case of heterogeneous resources, as a two-dimensional tree to speed up the match making. However, the resource dimension was only partly covered by co-allocating multiple single-unit resources.

Burchard [19] compared the approach to manage the free blocks in a tree with the slotted time model and concluded that the slotted time models outperform




(a) time needed to schedule a job (cut at 1ms)

(b) time needed to schedule a job (uncut)

(c) schedule success rate

Figure 5. In the simulated cluster, the list based approach is always better then the best slotted time configuration in respect to the success rate and the runtime. Furthermore, in the slotted time approach a compromise between both metrics has to be made.

the tree approach. Therefore, we used the slotted time model as a benchmark for the new list based approach.

## 7. Evaluation

To compare both implementation models, two resource managers – a Grid broker and a cluster scheduler – were enhanced to support both implementations. In both cases the time exclusive free list was used.

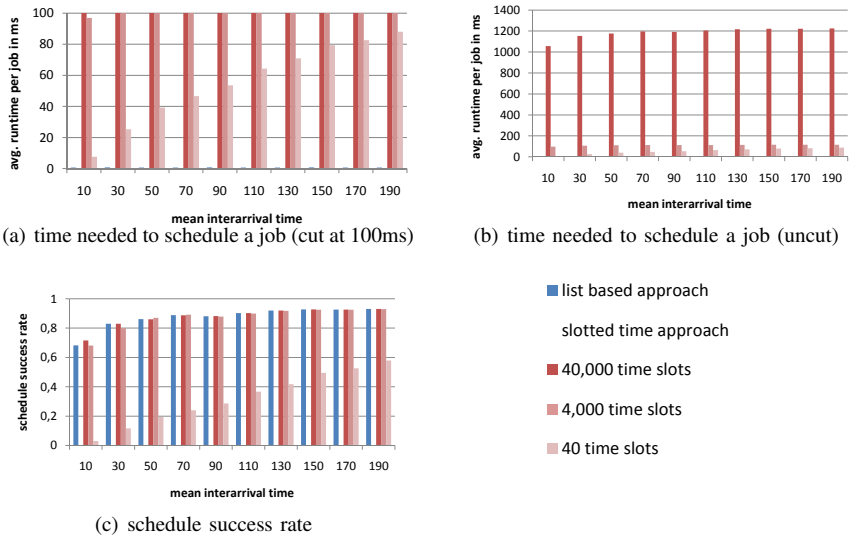
First, the simulation framework for cluster scheduler as presented in Section 3 was used as the evaluation platform. A cluster with 128 processors was simulated (experiments with 512, 2048, and 4096 CPUs showed similar results). The synthetic workload was generated with Feitelson's workload generator for batch systems [20]. The advance reservation aspects were added like discussed by Aida and Casanova [21] which based their analysis on the Grid Workloads Archive[4] and the traces of the French Grid5000 test bed[5]. This workload was then processed using time-exclusive lists and slotted time. In the slotted time model, the maximum available book-ahead time was divided in $90$, $9,000$, and $900,000$ time slots. The experiments were repeated until a sufficiently small confidence interval was reached.

Unlike to the queuing based scheduling, a job not fitting in the schedule during the requested booking time will be rejected. Due to fragmentation, not only the already booked capacity has an influence on the *success rate* of the reservations. We used the success rate together with the average *runtime to reserve a job* as metrics to assess the performance of both implementation models. To see the behavior of both models, we simulated various load situations by adjusting the arrival rate of new jobs in the system. In the experiments, a low *mean interarrival time* resulted in a very highly loaded system while a higher value leaded to more space in the schedule. However, we only used interarrival times resulting in an overload situation, i.e., even a perfect system could not reach $100\%$ success rate. In lower loaded system with much space in the schedule, we could not measure the effect of the fragmentation as the scheduler would find a reservation slot even in a highly fragmented schedule.

The results of the experiments as depicted in Figure 5 clearly show the properties of the slotted time approach: many short time slots lead to a higher runtime while few long time slots have a reduced acceptance rate due to internal fragmentation. When comparing the list based approach and the slotted time approach, one sees that the list based implementation has a better runtime than the configuration with few time slots. At the same time, the list based implementation provides the same acceptance rate as the configuration with $900,000$ slots.

To verify the results, two Grid setups were made using the simulation mode of the Grid broker framework

4. http://gwa.ewi.tudelft.nl/
5. http://www.grid5000.org/




(a) time needed to schedule a job (cut at 100ms)

(b) time needed to schedule a job (uncut)

(c) schedule success rate

Figure 6. In the Grid scheduler, the list based approach outperforms the slotted based approach, too.

VRM (see Section 3). For the first experiment, a simplified Grid was used which consisted of a simulated client and a single resource only. The workload model used was similarly generated as on the cluster level. The same sequence of atomic jobs was submitted to a resource manager using a slotted time model (with $400$, $40,000$, and $4,000$ slots) and to a resource manager using a time exclusive list of free capacity.

In result of the simplified Grid experiment is depicted in Figure 6. The runtime of the scheduler with many slots was again drastically higher than. However, only using so many slots the slotted approach can come close to the schedule success rate of the free list based approach.

We made another experiment with a realistic Grid setup of multiple resources and a workload of complex Grid workflows consisting of multiple jobs each. The results were comparable to the results shown here, while the impact of the longer runtime of the slotted time model was more visible. The higher impact can be explained by the higher number of schedule operations in such a complex distributed setup.

Therefore, both experiments show that the free list based approach provides always a similar runtime and acceptance rate as the respectively best configuration for the slotted time model. However, as the slotted time implementation has to be tuned for either a good runtime or a high acceptance rate, the free list based implementation is the better choice for an advance reservation scheduler.

## 8. Conclusion and Outlook

Using advance reservation resource management systems are enabled to provide a higher service level, e.g., the guaranteed execution time can be used to negotiate co-allocations. However, in contrast to queuing bases systems, advance reservation based resource management systems need to keep track of all reservation for future time spans and elaborated techniques to identify free capacity during the admission of new jobs.

In this paper, the approach of discretizing the time by introducing time slots is compared with our adaption of list of free blocks. This general idea widely used to manage continuous resources is extended to cover not only the resource dimension but also the time dimension. We identified three options to realize this two-dimensional lists and discussed the benefits and drawbacks in comparision to the slotted time model. We implemented the list based approach for two application domains — Grid management and Cluster scheduling. Extensive simulations of both implementations showed drastical enhancement in both runtime and acceptance ratio. The slotted time approach needs much more time to find a suitable reservation candidate and to mark the allocated capacity as booked. The list based approach eliminates internal fragmentation. Therefore, the number of accepted jobs rises, too.

In future work, we plan to add the capacity hierarchical list to the comparision. To verify the simulation based results, we also plan to deploy the list based approach in real setups.

## References


[1] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson and L. Rudolph, Eds. Springer Berlin / Heidelberg, 1999, vol. 1659, pp. 202–219.

[2] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan, and T. Fahringer, "Applying advance reservation to increase predictability of workflow execution on the grid," in *Second IEEE International Conference on e-Science and Grid Computing, 2006. e-Science '06.*, Dec. 2006, p. 82.

[3] J. Gehr and J. Schneider, "Measuring fragmentation of two-dimensional resources applied to advance reservation grid scheduling," in *Proceedings of 9th International Symposium on Cluster Computing and the Grid (CCGrid 09)*, 2009.

[4] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 127–132.

[5] A. Sulistio and R. Buyya, "A grid simulation infrastructure supporting advance reservation," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004*. Cambridge, Boston, USA: ACTA Press, Anaheim, California, 2004.

[6] F. Heine, M. Hovestadt, O. Kao, and A. Streit, "On the impact of reservations from the grid on planning-based resource management," in *Workshop on Grid Computing Security and Resource Management*, ser. Lecture Notes in Computer Science, vol. 3516/2005. Springer Berlin / Heidelberg, 2005.

[7] J. MacLaren, "Advance reservation: State of the art," Open Grid Forum - GRAAP working group, Tech. Rep. doc6097, June 2003.

[8] K. Czajkowski, I. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pp. 219–228, 1999.

[9] T. Röblitz, F. Schintke, and A. Reinefeld, "Resource Reservations with Fuzzy Requests," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 13, pp. 1681–1703, November 2006.

[10] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, "QoS Support for Time-Critical Grid Workflow Applications," *Proceedings of the 1st International Conference on e-Science and Grid Computing (e-Science 2005)*, pp. 108–115, 2005.

[11] G. Singh, C. Kesselman, and E. Deelman, "A provisioning model and its comparison with best-effort for performance-cost optimization in grids," in *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2007, pp. 117–126.

[12] H. Zhao and R. Sakellariou, "Advance Reservation Policies for Workflows," in *12th International Workshop on Job Scheduling Strategies for Parallel Processing*, ser. LECTURE NOTES IN COMPUTER SCIENCE, vol. 4376. Saint-Malo, France: Springer, June 2006, p. 47.

[13] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming Journal*, vol. 14, no. 3-4, pp. 217 – 230, Nov 2006.

[14] C. Castillo, "On the design of efficient resource allocation mechanisms for grids," Ph.D. dissertation, North Carolina State University, 04 2008. [Online]. Available: http://www.lib.ncsu.edu/theses/available/etd-04292008-003344/

[15] L.-O. Burchard, M. Hovestadt, O. K. A. Keller, and B. Linnert, "The virtual resource manager: An architecture for SLA-aware resource management," in *4th Intl. IEEE/ACM Intl. Symposium on ClusterComputing and the Grid (CCGrid) 2004, Chicago, USA*, 2004.

[16] L.-O. Burchard, H.-U. Heiss, B. Linnert, J. Schneider, O. Kao, M. Hovestadt, F. Heine, and A. Keller, "The virtual resource manager: Local autonomy versus QoS guarantees for grid applications," in *Future Generation Grids*, ser. CoreGrid, V. Getov, D. Laforenza, and A. Reinefeld, Eds., vol. 2, 2006.

[17] L.-O. Burchard, "Networks with advance reservations: Applications, architecture, and performance," *Journal of Network and Systems Management*, vol. 13, no. 4, pp. 429–449, 2005.

[18] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, "End-to-end quality of service for high-end applications," *Computer Communications*, vol. 27, no. 14, pp. 1375–1388, 2004.

[19] L.-O. Burchard, "Analysis of data structures for admission control of advance reservation requests," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 413–424, March 2005.

[20] D. G. Feitelson, "Packing schemes for gang scheduling," in *IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1996, pp. 89–110.

[21] K. Aida and H. Casanova, "Scheduling mixed-parallel applications with advance reservations," in *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2008, pp. 65–74.