IJPP manuscript No.

List-based Data Structures for Efficient Management of Advance Reservations

Joerg Schneider · Barry Linnert

Received: 01.12.2011 / Accepted: 05.07.2012

Abstract Complex eScience and other sophisticated applications in the field of HPC imply new demands that queuing based resource management systems cannot meet. To guarantee Quality of Service and co-allocation in the Grid, planning based resource management systems implementing advance reservation are needed. These systems face new challenges as a planning based management system has to keep track of the jobs and reservations in the future. Additionally, during the negotiation process of incoming reservations, a good overview of the remaining, not-yet reserved capacity is needed—not only for the current allocation, but also for the whole book-ahead time. Therefore, the resource management problem becomes a two dimensional problem for advance reservations in this field.

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 local resource management system for a computing cluster. The extensive simulations showed a better runtime and

J. Schneider
Technische Universitaet Berlin
Einsteinufer 17 / Sekr. EN 6
10587 Berlin
Tel.: +49-30-314-73388
Fax: +49-30-314-25156
E-mail: komm@cs.tu-berlin.de

B. Linnert
Technische Universitaet Berlin
Einsteinufer 17 / Sekr. EN 6
10587 Berlin
Tel.: +49-30-314-79811
Fax: +49-30-314-25156
E-mail: linnert@cs.tu-berlin.de

higher reservation success rate compared with the currently favored approach of a slotted time and the more sophisticated approach based on AVL trees.

Keywords advance reservation, resource management, data structure, cluster computing, Grid computing

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 a 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 [22]. Hence, a user cannot compare independent resource providers.

A 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

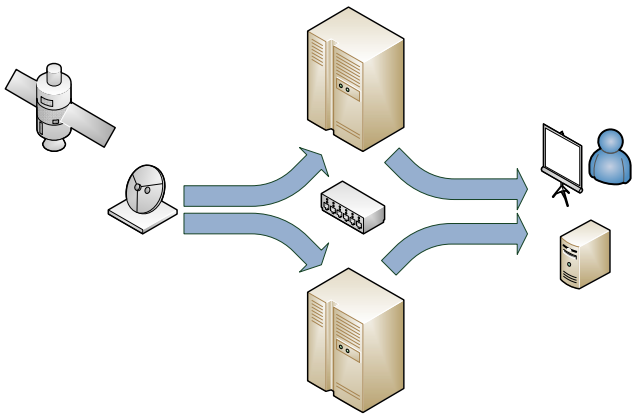


Fig. 1 An example of a complex Grid application, using a number of different resources. Each resource has to be reserved in advance to allow the coordinated execution of the jobs.

now guarantee that the service will be finished by some given deadline [27]. If the *book-ahead time*, the time between the negotiation and the execution of the service, is long enough, the service provider can also adjust the provided resource capacity to meet only the requested capacity, e.g., the amount of raw material stocked or the number of employees working at a given time. On the other hand, the service provider can use the negotiation phase for load balancing, e.g., promoting phases with lower utilization.

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.

For planning in advance, the execution times of the services have to be known. Therefore, either the user or the service provider has to indicate the duration of the service usage. In the offline world mostly the service providers know, how long the different kinds of services will take, in the world of high performance computing the user usually provides a prediction [12,26]. But there are also approaches to predict the execution time based on previous executions of the same parallel program [12,3,20]. However, the prediction of the duration is usually inaccurate.

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 [14] 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 [21]. Sulistio and Buyya measured a utilization drop of 60-80% and increased wait times for the jobs in the queue [24]. 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 [15]. In [17] 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 (see Figure 1) 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 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. In such a case, 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 2). 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,

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

has to hold. If no booking interval is specified, the arrival time $t_{arr}(j)$ and infinity will be used as start and end time, respectively.

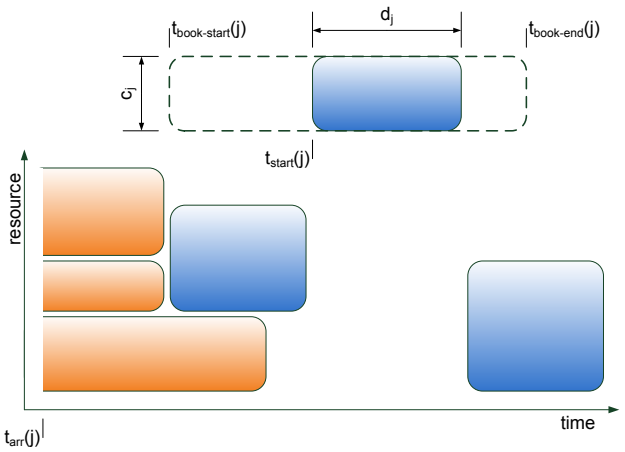


Fig. 2 Advance reservation allows planning the execution of jobs in the future.

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

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

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 identified early as necessary for co-allocations and guaranteed finish times [10]. Hence, there are a number of articles covering the negotiation and handling of advance reservation [18,4,19,30,27,29,9].

An example for an advance reservation enabled Grid broker is the *Virtual Resource Manager* (VRM) [8]. 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 adapters 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 [7]. The scheduling of the ADC bases its Grid scheduling on advance reservation support of 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. If a planning based local resource management system is integrated in the Grid infrastructure controlled by the VRM, this management system is in charge to handle the currently running jobs and all of the reservations mapped to this resource and therefore need efficient data structures as well. The VRM in its simulation mode is used as one of the test platforms in the evaluation and a simulation framework for local compute resource management systems is used to investigate the impact of data structures on the reservation performance.

As stated before most advance reservation Grid broker rely - like the VRM - on the advance reservation support of the local resource management system. For computing cluster some advance reservation enabled schedulers are available like OpenCCS[1], MAUI[2], and Crono[3].

So to speed up the evaluation, we developed a simulation framework for the cluster scheduler. It supports advance reservations for arbitrary cluster topologies and architectures. It is able to compare different scheduling strategies including strategies using restricted information and dynamic application behavior only. 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).

[1] `https://www.openccs.eu/core/`

[2] `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php`

[3] `http://crono.sourceforge.net/`

Advance reservation is also available for other resource types like network bandwidth. How advance reservation can be performed in networks is discussed in detail in [6] and [13].

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 spare 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 Δt. The time slot t_i represents all time stamps t with $i\Delta t \leq t < (i+1)\Delta t$.

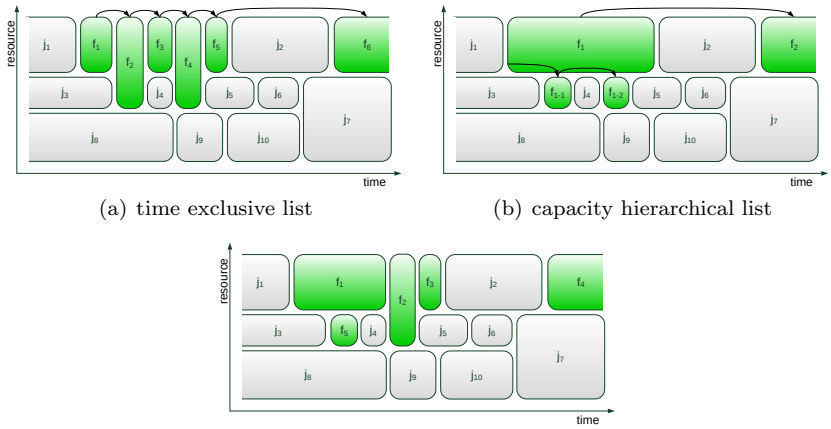
Now for all time slots, the reservations, which are at least partly within the timeslot, are associated with the time slot. If a reservation starts or ends within a time slot, the resource usage is counted for the whole time slot. This makes the algorithm simpler, but also leads to internal fragmentation.

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 Δt.

Furthermore, an array can be used as a ring buffer data structure to manage the time slots [5]. 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. But 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.

Additionally, 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 at all requested time slots, the algorithm can only detect this by iterating over all slots. The same holds for schedules with long running



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

(c) mixed list

Fig. 3 The three different approaches to organize a two-dimensional free block list.

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 concept 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.

However, 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 resources and must not intersect:

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

Figure 3 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.

```
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'
```

Fig. 4 Determining a reservation candidate in a time exclusive 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 3 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 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 4). 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 split. 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 case of the capacity hierarchical list first a block with a sufficient long time span is selected as depicted in Figure 5.

```
1  initialize capacity c_found := 0
2  initialize stack of used blocks S := ∅
3  for all blocks f ∈ F_r with d_j ≤ d_f
4     put selected block on stack  push(S, f)
5     store available capacity c_found := c_f
6     while S ≠ ∅
7        f_last =top(S)
8        select a not yet tested block f' from the sub list of f_last with d_j ≤ d_f
9        if no block is found
10          use backtracking to change selection of formerly selcted blocks  pop(S)
11          decrease capacity c_found := c_found − c_flast
12       else
13          increase capacity c_found := c_found + c_f'
14          put selected block on stack  push(S, f')
15          if c_found ≥ c_j
16             return reservation candidate
17 return unsuccessful
```

Fig. 5 Determining a reservation candidate in a capacity hierarchical free list.

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 split block has to be split. 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 List of used capacity blocks and AVL tree approach

As described in the former section a list of blocks representing capacity of the managed resource has advantages over the slotted time approach. To meet more sophisticated requests in managing the resource as holding mapping information to show which reservations holds what part of the resource the approach of using a list of free capacity blocks can be inverted to implement a list of blocks representing these mapping information and therefore used capacity. This approach is used for the simulation framework for the local compute resource management system.

In order to optimize the response time of the resource management system an AVL tree data structure is used additionally. In this approach, the leaves hold the used capacity blocks. To optimize the response time of reservation requests the tree has to be balanced - as it is part of the definition of AVL trees. The needed balancing of the tree may come with some impact as new elements of the tree may with some likelihood by integrated on the one side representing the nearer future and lead to an inbalance of the AVL tree.

7 Related Work

As the performance of the management systems heavily depends on the kind of data structure and algorithm used, the question which implementation is the best to hold the schedule information is discussed in the literature.

Singh et al. [19] 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 split, 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 split 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.

A linked list of elements with a variable time span is used by Xiong et al. [28]. However, the authors do not present detailed algorithms for handling their data structure, i.e., how to add reservations or whether reservations spanning multiple list elements where considered or not. Outdated list elements are either not deleted or only deleted in fixed intervals, leading to long lists and therefore to a high search overhead. This problem is addressed by keeping additional pointer to list elements and by booking multiple reservations at once to spare the search overhead. In their experimental evaluation, their implementation of the linked list approach is compared with an implementation of the slotted time model. Their lists performed better only in systems with few reservations and slightly worse in systems with a high utilization. As the details of the used algorithm are unknown, it is not clear whether there are more performance problems like the overhead for outdated elements.

Another list-like data structure is discussed by Kurowski et al. [16]. An element covering a variable time is again called a slot. While in some parts the authors talk about a list of these slots, the implementation details and the complexity discussion is based on an array of such variable slots. Therefore, the data structure is generated when it is needed based on the existing reservations.

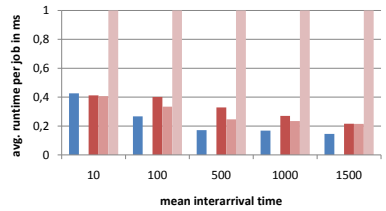
Castillo developed a system to match advance reservations for single-unit resources using techniques from computational geometry [9]. The free list is held as a tree or in case of heterogeneous resources, as a two-dimensional tree to speed up the match making. However, the resource dimension was only partly covered by co-allocating multiple single-unit resources.

Burchard [5] compared the approach to manage the free blocks in a tree with the slotted time model and concluded that the slotted time models outperform the tree approach. Therefore, we used the slotted time model as a benchmark for the new list based approach.

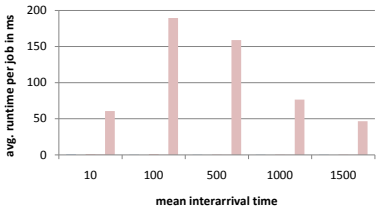
Focusing on the data transfer necessary in Grid environments Andreica and Tapus [2] discussed different approaches such as arrays and trees and introduced a slotted time approach in combination with a tree data structure for multiple dimensions. As these approaches are discussed theoretically an evaluation using resource management environments is not performed and not all features needed to support resource management systems are provided by these approaches. So the analysis of the approaches depends on the special path finding problem coming with the network application and is therefore limited to this examination.

Stevens et al. [23] also deal with reservations of network resources. In contrast to Andreica and Tapus the approaches introduced do include the computer resources as well. For this combined reflection of the topic they present several multi-cost based approaches. These algorithms are used to investigate performance impact of immediate vs. advance reservation. As data structure, they introduce a ,,path capacity availability vector" which is very similar to the slotted time concept. As the limiting factor is the network topology, the examination of different types of data structure is not completely performed.

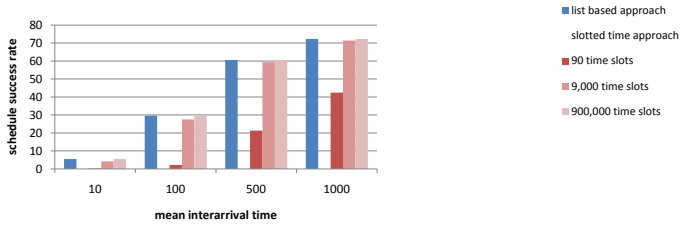
Different types of data structure are investigated by Sulistio et al. [25] all using the slotted time concept as the base. Using different access techniques to handle the extended slotted time structure lead to different performance considering the search, insert, and deletion operations. Their investigations cover all important operations for a advance reservation scheduler and not only the search for reservation candidates as in other paper. Although the enhancement of the focus of the investigation all approaches are based on slotted time, leading to a limitation for resource management systems in productive environments.



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



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



(c) schedule success rate

Fig. 6 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.

8 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.

8.1 Cluster manager

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 [11]. The advance reservation aspects were added like discussed by Aida and Casanova [1] 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.

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

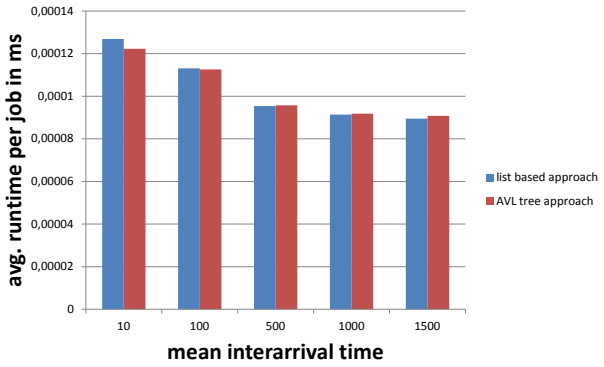


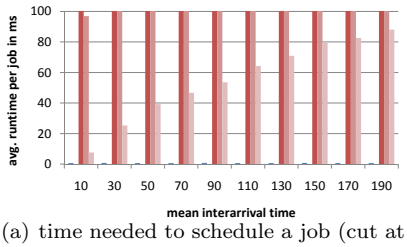
Fig. 7 Enhancing the list model with an additional AVL tree to navigate directly to the start of a booking interval does not decrease the runtime significantly. The success rate was the same for both data structures.

Unlike to the queuing based scheduling, a job not fitting in the schedule during the requested booking time will be rejected. The *success rate* of the reservations depends not only on the already booked capacity. It is also highly influenced by the fragmentation of the schedule. 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 lead 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.

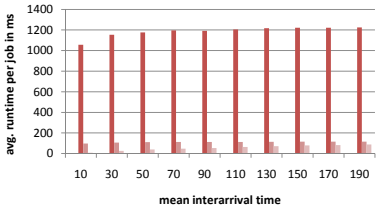
The results of the experiments as depicted in Figure 6 clearly show the properties of the slotted time approach: many short time slots lead to a higher runtime while few long time slots have a reduced acceptance rate due to internal fragmentation. When comparing the list based approach and the slotted 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.

8.2 AVL trees in the cluster manager

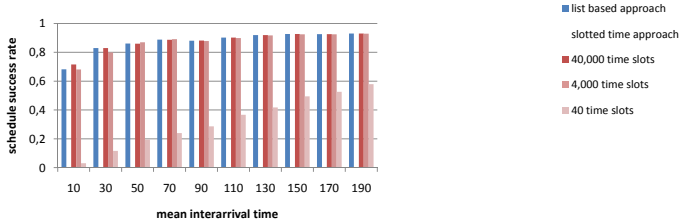
We also investigated the impact of the optimization using an AVL tree data structure handling the blocks of capacity. As depicted in Figure 7, the AVL tree does perform on the same level as the list based approach. The similiar performance can be explained by the overhead of balancing the tree which



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



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



(c) schedule success rate

Fig. 8 In the Grid scheduler, the list based approach outperforms the slotted based approach, too.

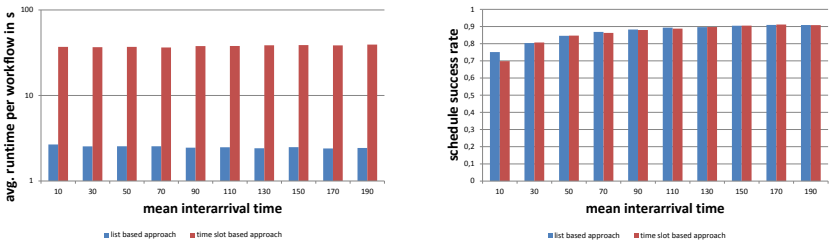
voids the performance benefits of a faster access compared to the lists. Implementing the tree based approach without balancing the tree, i.e., ignoring the AVL tree requirements, leads to a degenerated tree which equals a list data structure. The tree will degenerate as new reservations will likely be added in the rightmost subtree or at least in one of the right subtrees as they represent the further future.

8.3 Grid manager

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

The result of the simplified Grid experiment is depicted in Figure 8. The runtime of the scheduler with many slots was again drastically higher than the runtime of the list based scheduler. 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 connected by a network topology based on the wide area network



(a) time needed to schedule a workflow (b) schedule success rate

Fig. 9 Repeating the experiment with a realistic Grid setup and a workload of complex Grid workflows shows again the shorter runtime of the list based approach.

network of an internet service provider. Each resource had its own advance reservation enabled resource manager and a Grid wide workflow scheduler used also the same data structures to keep book of its reservations. Instead of single reservations, we submitted complex workflows with multiple reservations together with dependencies and data transfers between them. We used a time slot configuration with an comparable acceptance rate to the list approach (see Figure 9(b)). Figure 9(a) shows the average runtime to schedule a whole workflow including the whole negotiation and scheduling overhead. Due to this overhead, the runtimes are not as sensible to changes in the load situation (the interarrival time) as in the previous experiments. However, the large overhead of the slotted time approach is still clearly visible.

Therefore, experiments with both applications show that the free list based approach always provides 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.

9 Conclusion and Outlook

Advance reservation resource management systems can be used to provide a higher service level, e.g., the guaranteed execution time is needed to negotiate co-allocations. However, in contrast to queuing based 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 capacity 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. Even the potential optimization by using more complex data structures like AVL trees shows no better performance in our experiments.

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. Aida, K., Casanova, H.: Scheduling mixed-parallel applications with advance reservations. In: HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing, pp. 65–74. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1383422.1383432. URL `http://www.alab.ip.titech.ac.jp/papers/aida_hpdc2008.pdf`
2. Andreica, M., Tapus, N.: Efficient data structures for online qos-constrained data transfer scheduling. In: Parallel and Distributed Computing, 2008. ISPDC '08. International Symposium on, pp. 285 –292 (2008). DOI 10.1109/ISPDC.2008.36
3. Anglano, C.: Predicting parallel applications performance on non-dedicated cluster platforms. In: ICS '98: Proceedings of the 12th international conference on Supercomputing, pp. 172–179. ACM, New York, NY, USA (1998). DOI http://doi.acm.org/10.1145/277830.277866
4. Brandic, I., Benkner, S., Engelbrecht, G., Schmidt, R.: 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). URL `http://ieeexplore.ieee.org/iel5/10501/33262/01572215.pdf`
5. Burchard, L.O.: Analysis of data structures for admission control of advance reservation requests. IEEE Transactions on Knowledge and Data Engineering **17**(3), 413–424 (2005). DOI 10.1109/TKDE.2005.40. URL `http://kbs.cs.tu-berlin.de/publications/res_mgnt/Bur04c.pdf`
6. Burchard, L.O.: Networks with advance reservations: Applications, architecture, and performance. Journal of Network and Systems Management **13**(4), 429–449 (2005). DOI http://dx.doi.org/10.1007/s10922-005-9004-7
7. Burchard, L.O., Heiss, H.U., Linnert, B., Schneider, J., Kao, O., Hovestadt, M., Heine, F., Keller, A.: The virtual resource manager: Local autonomy versus QoS guarantees for grid applications. In: V. Getov, D. Laforenza, A. Reinefeld (eds.) Future Generation Grids, *CoreGrid*, vol. 2 (2006). URL `http://www.user.tu-berlin.de/komm/paper/FGG-local-autonomy-vs-SLA.pdf`
8. Burchard, L.O., Hovestadt, M., Keller, O.K.A., Linnert, B.: 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). DOI 10.1109/CCGrid.2004.1336558
9. Castillo, C.: On the design of efficient resource allocation mechanisms for grids. Ph.D. thesis, North Carolina State University (2008). URL `http://www.lib.ncsu.edu/theses/available/etd-04292008-003344/`
10. Czajkowski, K., Foster, I., Kesselman, C.: Resource Co-Allocation in Computational Grids. Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8) pp. 219–228 (1999). URL `http://globus.org/alliance/publications/papers/paper3.pdf`

11. Feitelson, D.G.: Packing schemes for gang scheduling. In: IPPS '96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, pp. 89–110. Springer-Verlag, London, UK (1996). DOI 10.1007/BFb0022289. URL http://www.cse.huji.ac.il/~feit/parsched/jsspp96/p-96-6.pdf
12. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: D.G. Feitelson, L. Rudolph (eds.) Job Scheduling Strategies for Parallel Processing, pp. 1–34. Springer Verlag (1997). URL www.cs.huji.ac.il/~feit/parsched/jsspp97/p-97-1.ps.gz
13. Foster, I., Fidler, M., Roy, A., Sander, V., Winkler, L.: End-to-end quality of service for high-end applications. Computer Communications **27**(14), 1375–1388 (2004). DOI 10.1016/j.comcom.2004.02.014
14. Gehr, J., Schneider, J.: 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). DOI 10.1109/CCGRID.2009.81. URL http://www.user.tu-berlin.de/komm/paper/2009-measure-2D-fragmentation.pdf
15. Heine, F., Hovestadt, M., Kao, O., Streit, A.: On the impact of reservations from the grid on planning-based resource management. In: Workshop on Grid Computing Security and Resource Management, *Lecture Notes in Computer Science*, vol. 3516/2005. Springer Berlin / Heidelberg (2005). URL http://www.fz-juelich.de/jsc/vsgc/pub/heine-2005-OIR.pdf
16. Kurowski, K., Oleksiak, A., Piatek, W., Weglarz, J.: Hierarchical scheduling strategies for parallel tasks and advance reservations in grids. Journal of Scheduling pp. 1–20 (2011). URL http://dx.doi.org/10.1007/s10951-011-0254-9. 10.1007/s10951-011-0254-9
17. MacLaren, J.: Advance reservation: State of the art. Tech. Rep. doc6097, Open Grid Forum - GRAAP working group (2003). URL http://forge.ogf.org/sf/sfmain/do/go/doc6097
18. Röblitz, T., Schintke, F., Reinefeld, A.: Resource Reservations with Fuzzy Requests. Concurrency and Computation: Practice and Experience **18**(13), 1681–1703 (2006). URL http://www.zib.de/reinefeld/Publications/2005_roeblitz_cpe.pdf
19. Singh, G., Kesselman, C., Deelman, E.: 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, pp. 117–126. ACM, New York, NY, USA (2007). DOI 10.1145/1272366.1272382
20. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. In: The 4th Workshop on Job Scheduling Strategies for Parallel Processing (1998). DOI 10.1007/BFb0053984
21. Smith, W., Foster, I., Taylor, V.: Scheduling with advanced reservations. In: Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, pp. 127–132 (2000). DOI 10.1109/IPDPS.2000.845974
22. Smith, W., Taylor, V., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: D. Feitelson, L. Rudolph (eds.) Job Scheduling Strategies for Parallel Processing, *Lecture Notes in Computer Science*, vol. 1659, pp. 202–219. Springer Berlin / Heidelberg (1999). DOI 10.1007/3-540-47954-6_11
23. Stevens, T., Leenheer, M.D., Develder, C., Dhoedt, B., Christodoulopoulos, K., Kokkinos, P., Varvarigos, E.: Multi-cost job routing and scheduling in grid networks. Future Generation Computer Systems **25**(8), 912 – 925 (2009). DOI 10.1016/j.future.2008.08.004. URL http://www.sciencedirect.com/science/article/pii/S0167739X08001234
24. Sulistio, A., Buyya, R.: A grid simulation infrastructure supporting advance reservation. In: Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004. ACTA Press, Anaheim, California, Cambridge, Boston, USA (2004). URL http://gridbus.org/papers/gridsim-ar-pdcs04.pdf
25. Sulistio, A., Cibej, U., Prasad, S.K., Buyya, R.: Garq: An efficient scheduling data structure for advance reservations of grid resources. International Journal of Parallel, Emergent and Distributed Systems **24**(1), 1–19 (2009). DOI 10.1080/17445760801988979. URL http://www.tandfonline.com/doi/abs/10.1080/17445760801988979

26. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Modeling user runtime estimates. In: 11th International Workshop on Job Scheduling Strategies for Parallel Processing, *Lecture Notes in Computer Science*, vol. 3834/2005. Springer Berlin / Heidelberg, Cambridge, USA (2005). URL `http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir05/Est05JSSPP.pdf`
27. Wieczorek, M., Siddiqui, M., Villazon, A., Prodan, R., Fahringer, T.: 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., p. 82 (2006). DOI 10.1109/E-SCIENCE.2006.261166
28. Xiong, Q., Wu, C., Xing, J., Wu, L., Zhang, H.: A linked-list data structure for advance reservation admission control. In: X. Lu, W. Zhao (eds.) Networking and Mobile Computing, *Lecture Notes in Computer Science*, vol. 3619, pp. 901–910. Springer Berlin / Heidelberg (2005). DOI 10.1007/11534310_95
29. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Scientific Programming Journal **14**(3-4), 217 – 230 (2006). URL `http://gridbus.org/papers/Workflow_JSP_2005.pdf`
30. Zhao, H., Sakellariou, R.: Advance Reservation Policies for Workflows. In: 12th International Workshop on Job Scheduling Strategies for Parallel Processing, *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4376, p. 47. Springer, Saint-Malo, France (2006). URL `http://www.cs.man.ac.uk/~rizos/papers/jsspp2006.pdf`