Overbooking in Planning Based Scheduling Systems

Georg Birkenheuer #1, Matthias Hovestadt *2, Odej Kao *3, Kerstin Voss #4
# Paderborn Center for Parallel Computing, Universität Paderborn
Fürstenallee 11, 33102 Paderborn, Germany
1birke@uni-paderborn.de 4kerstinv@uni-paderborn.de

*Technische Universität Berlin
Einsteinufer 17, 10587 Berlin, Germany
2maho@cs.tu-berlin.de 3odej.kao@cs.tu-berlin.de

Abstract

Nowadays cluster Grids encompass many cluster systems with possible thousands of nodes and processors, offering compute power that was inconceivable only a few years ago. For attracting commercial users to use these environments, the resource management systems (RMS) have to be able to negotiate on Service Level Agreements (SLAs), which are defining all service quality requirements of such a job, e.g. deadlines for job completion. Planning-based scheduling seems to be well suited to guarantee the SLA adherence of these jobs, since it builds up a schedule for the entire future resource usage. However, it demands the user to give runtime estimates for his job. Since many users are not able to give exact runtime estimates, it is common practice to overestimate, thus reducing the number of jobs that the system is able to accept. In this paper we describe the potential of overbooking mechanisms for coping with this effect.

Keywords: Grid-Scheduling, Overbooking, Resource Management, SLA

1 Introduction

Grid Computing is providing computing power for scientific and commercial users. Following the common evolution in computer technology, the system and network performance have constantly increased. The latest step in this process was the introduction of multiple cores per processor, making Grid nodes even more powerful. This evolutionary process particularly affects the scheduling components of resource management systems that are used for managing cluster systems. On the one hand, the increasing number of nodes, processors, and cores results in an increased degree of freedom for the scheduler, since the scheduler has more options of placing jobs on the nodes (and cores) of the system. On the other hand, also the requirements of users have changed. Commercial users ask for contractually fixed service quality levels, e.g. the adherence of deadlines. Hence, the scheduler has to respect additional constraints at scheduling time.

Queuing is a technique used in many currently available resource management systems, e.g. PBS [1], LoadLeveler [2], Grid Engine [3], LSF [4], or Condor [5]. Since queueing-based RMS only plan for the present, it is hard to provide guarantees on future QoS aspects.

Planning-based RMS make functionalities like advance reservations trivial to implement. If a new job enters the system, the scheduling component of the RMS tries to place the new job into the current system schedule, taking aspects like project-specific resource usage limits, priorities, or administrative reservations into account. In planning-based systems it is mandatory for the user to specify the runtime of his job. If thinking of the negotiation of Service Level Agreements, this capability is essential for the provider’s decision making process. As an example, using fixed reservations, specific resources can be reserved in a fixed time interval. In addition to plain queuing, the Maui [6] scheduler also provides planning capabilities. Few other RMS like OpenCCS [7] have been developed as planning-based systems from scratch.
However, (fixed) reservations in planning-based systems potentially result in a high level of fragmentation of the system schedule, preventing the achievement of optimal resource utilization and workload. Moreover, users tend to overestimate the runtime of their jobs, since a planning-based RMS would terminate jobs once their user-specified runtime has expired. This termination is mandatory, if succeeding reservations are scheduled for being executed on these resources. Overestimation of job runtime inherits an earlier availability of assigned resources as expected by the scheduler, i.e. at time $t_r$ instead of $t_p$. Currently, mechanisms like backfilling with new jobs or rescheduling (start, if possible, an already planned job earlier) are initiated to fill the gap between $t_r$ and the planned start time $t_s \geq t_p$ of the succeeding reservation. Due to conflicts with the earliest possible execution time, moving of arbitrary succeeding jobs to an earlier starting time might not be possible. In particular the probability to execute a job earlier might be low if users have strict time intervals for the job execution since a planning-based scheduler rejects job requests which cannot be planned according to time and resource constraints in the schedule. An analysis of cluster logfiles revealed that users overestimated the runtime of their jobs by a factor of two to three [8]. For the provider this might result in a bad workload and throughput, since computing power is wasted if backfilling or rescheduling cannot start other jobs earlier as initially planned. To prevent poor utilization and throughput, overbooking has proven its potential in various fields of application for increase the system utilization and provider’s profit. As a matter of fact, overbooking results in resource usage conflicts if the user-specified runtime turns out to be realistic, or if the scheduler is working with an overestimation presumption that is too high. To compensate such situations, the suspension and later restart of jobs are important instruments of the RMS. To suspend a running job and without losing already performed computation steps, the RMS makes a snapshot of the job, i.e. storing the job process environment (including memory, messages, registers, and program counter), to be able to migrate them to another machine or to restart them at a later point of time. In the EC-funded project HPC4U [9] the necessary mechanisms have been integrated in the planning-based RMS OpenCCS to generate checkpoints and migrate jobs. These fault-tolerance mechanisms are the basis for a profitable overbooking approach as presented in this paper since stopping a job does not imply losing computation steps performed. Hence, gaps between advance reservations can be used from jobs, which will be finished before the next reservation has to be started. In the next section we discuss the related work followed by our ideas for overbooking, which are described in Section 3. In Section 4 we conclude the paper with a summary of our ideas and presenting plans for future work.

2 Related Work

The idea of overbooking resources is a standard approach in many fields of application like in flight, hospital, or hotel reservations. Overbooking beds, flights, etc. is a consequence of that a specific percentage of reservations are not occupied, i.e. usually more people reserve hotel rooms [10] or buy flight tickets [11, 12] than actually appearing to use their reservation. The examples of hotels and aeronautical companies show the idea we are following also in the provisioning of compute resources. Overbooking in the context of computing Grids slightly differs from those fields of applications since therein the assumption is made that less customers utilize their reservations than booked. In Grid computing all jobs that have been negotiated will be executed, however, in planning-based systems users significantly overestimate the job duration. Comparing the usage of a compute resource and a seat in an aircraft is not meaningful since generally in computing Grids no fixed intervals for the resource utilization exist whereas a seat in an aircraft will not be occupied after the aircraft had taken off. As a consequence, results and observations from overbooking in the classical fields of applications cannot be reused in Grid scheduling. As a non-classical field of application [13] presents an overbooking approach for web platforms, however, the challenges also differ from the Grid environment.

In the Grid context, consider overbooking approaches is most sensible for planning based scheduling since in queuing based systems even the runtime have to be estimated and thereby an additional uncertainty has to be taken into account. Other work concerning overbooking in Grid or HPC environments is rare. In the context of Grid or HPC scheduling the benefits of using overbooking are pointed out, but no solutions are provided [14, 15]. Overbooking is also foreseen in a three layered protocol for negotiation in Grids [16]. Here, the restriction is made that overbooking is only used for multiple reservations for workflow sub-jobs which were made by the negotiation protocol for optimal workflow planning.

2.1 Planning Approaches

Some work had been done in the scope of planning algorithms. Before showing in Section 3 how overbooking can be integrated, different approaches already developed are described in the following.

2.1.1 Geometry based approaches

Theoretical approaches for planning-based systems identify that the scheduling is a special case of bin packing: the width of a bin is defined as the number of nodes generally available and its height equals the time the resource can be
used. As the total usage time for an arbitrary number of jobs does not end, the height of the bin is infinite. Consequently it is not a bin, rather defined as a strip. Jobs are considered as rectangles having a width equal to the number of required nodes and a height equal to the execution time determined by the user. The rectangles have to be positioned in the strip in such a way that the distances between rectangles in the strip are minimal and the jobs must not overlap themselves. Since strip packing is an NP-hard problem, several algorithms have been developed which work with heuristics and are applicable in practice. Reference [17] gives a good overview of strip packing algorithms. Two kinds, online and offline, of strip packing algorithms are differed. An offline algorithm has information about all jobs to be scheduled a priori whereas an online algorithms cannot estimate which jobs arrive in future. It is obvious that offline algorithms can achieve better utilization results since all jobs are known and can be scheduled by comparing among each other. The approaches could be divided into several main areas: Bottom-left algorithms which try to put a new job as far to the bottom of the strip and as left as possible, level-oriented algorithms [18], split algorithms [18], shelf algorithms [19], and hybrid algorithms which are combinations of the above mentioned ones.

### 2.1.2 Planning for Clusters

In practice, most planning based systems use first-come first-serve (FCFS) approaches. Grid scheduling has to use an online algorithm and consequently planning optimal schedules is not possible. The job is scheduled as soon as possible according to the current schedule (containing all jobs previously scheduled) as well as its resource and time constraints. The FCFS approach might lead to gaps which could be prevented if jobs would have scheduled in a different order. To increase the system utilization, backfilling [20] had been introduced to avoid such problems. Conservative Backfilling follows the objective to fill free gaps in the scheduled produced by FCFS planning without delaying any previously planned job. Simulations show that the overall utilization of systems is increased using backfilling strategies. If effects/delays of not started jobs are acceptable, the more aggressive EASY backfilling [21] can further improve the system utilization. However, [22] shows that for systems with high load the EASY approach is not better than the conservative backfilling. Furthermore, EASY backfilling has to be used with caution in systems guaranteeing QoS aspects since delays of SLA bound jobs might lead to SLA violations implying penalties.

Concluding, there is much work done in the scope of planning based scheduling. Good resource utilization in Grid systems can be achieved by using backfilling. However, applying conservative backfilling does not result in a 100% workload since only gaps can be assigned to jobs whose duration is less than the gap length. The more aggressive EASY backfilling strategy does not necessarily provide a better utilization of the system and implies hazards for SLA provisioning. Combining conservative backfilling with overbooking should further increase the system utilization and does not affect the planned schedule. Consequently, using these strategies combined has no disadvantages for not overbooked jobs and offers the possibility to schedule more jobs than with a simple FCFS approach.

### 3 Planning-Based Scheduling and Overbooking

This chapter explains the basic ideas to use overbooking in planning-based HPC systems. A user sends an SLA bound job request to the system. The SLA defines job type (batch or interactive), number $r$ of resources required, estimated runtime $d$, as well as an execution window $[t_{start}, t_{end}]$, i.e. earliest start-time and latest completion time. The planning-based system can determine before agreeing the SLA whether it is possible to execute the job according to time and resource constraints.

#### Fixed Reservations

Planning-based scheduling is especially beneficial if users are allowed to define time-slots $(r, h)$ in the SLA for interactive sessions, i.e. reserving compute nodes and manually start (multiple) jobs during the valid reservation time. The difference from an interactive session and a usual advance reservation is that the reservation duration equals $t_{end} - t_{start}$. For example $r = 32$ nodes should be reserved from $h = [11.12.08 : 9^{00}, 11.12.08 : 14^{00}]$ for a duration of $h = 5$ hours. Such so called interactive or fixed reservations increase the difficulty of the planning mechanism as these are fixed rectangles in the plan and cannot be moved. This will have worse effects on the system utilization than planning only advance reservations less strict timed. Consequently, supporting fixed reservations step up the demand for additional approaches like overbooking to ensure a good system utilization. However, such fixed reservations appreciate the value of using Grid computing for end-users if these have either interactive applications or need to run simulations exactly on-time, like for example for presentations.

For example, a resource management system (RMS) operates a cluster with 32 nodes and the typical jobs scheduled need 32 nodes and run 5 hours. During the day researchers make two fixed reservations from 9am to 2pm and from 2pm to 7pm. All other jobs are scheduled as batch jobs. In this scenario during the night, in the 14 hours between the fixed reservations, only two 5 hours batch jobs could be scheduled since these could be totally completed. Consequently, the cluster would be idle for 4 hours. To achieve
a better system utilization, either the user has to shift the 
fixed reservations one hour every day. Since this not feasi-
ble because of working-hours, assuming that the batch jobs 
finishes after 4 hours and 30 minutes enables to overbook 
resources and execute the three batch jobs.

3.1 Overbooking

Overbooking benefits from the fact that users overesti-
mate their jobs’ runtime. Consequently their jobs finish be-
fore the jobs’ planned completion time. Taking advantage 
of this observation will increase the system utilization and 
thereby the provider’s profit.

This section shows the process of integrating overbook-
ing in planning-based systems following conservative back-
filling as basic scheduling strategy. At first, aspects are 
highlighted which have to be considered when declaring 
jobs as usable for overbooking. Afterwards the overbooking 
algorithm is described which is followed by remarks con-
cerning fault-tolerance mechanisms, which should prevent 
job losses in case of wrong estimations of actual runtime 
made. An example forms the end of this section.

3.1.1 Runtime-Estimations for Overbooking

The prevention of job losses caused by overbooking is one 
important task of the scheduler. Further, good predictions 
of the overestimated runtime forms the key factors for pro-
fitable overbooking.

On the first glance, users overestimate the job duration 
in average by two to three times of the actual runtime [8]. 
Unfortunately, job traces show that the distribution of the 
overestimation seems to be uniform [22] and depending on 
the trace, 15% to nearly 30% of jobs are underestimated 
and have to be killed in planning-based systems after the 
planned completion time. Obviously, more not completed 
jobs could be killed when using overbooking. For instance, 
using the average value of overestimation from the statisti-
cal measure (which is 150% up to 500%) in scope of over-
booking would lead to conflicts since half of the jobs would 
be killed. Instead of exhausting overestimation to their full 
extend, it will be more profitable to balance between the risk 
of a too high overestimation and the opportunity to schedule 
an additional job. Hence, it might be often beneficial to not 
subtract the average overestimated runtime from the esti-
imated one in order to use this additional time for overbook-
ing. In many cases only using 10% of the overestimation 
can be sufficient. Given a uniform distribution, this would 
force 10% of the overbooked jobs to be lost, but 90% would 
be finished and increase the utilization. These in addition 
to the default strategy executed jobs increase the providers 
profit. To use good predictions of the overestimated run-
time, historical observations on the cluster and of the users 
are necessary. A detailed analysis has to be performed about 
the functional behavior of the runtime since an average or 
median value is not as meaningful as needed for reducing 
the risk for the provider to cause job losses. If enough mon-
itoring data is available, the following question arises: How 
can statistical information about actual job runtime be used 
to effectively overbook machines? The answer is to ana-
lyze several different aspects. First of all, a user-oriented 
analysis has to be performed since users often utilize com-
puting Grids for tasks in their main business/ working area 
which results in submitting same applications with similar 
input again and again [23, 24]. Consequently, analyzing es-
imated and actual runtime should whether and how much 
overestimations are made. If the results show that a user 
usually overestimates the runtime by factor $x$, the scheduler 
can use $x_o < x$ of the overestimated time as a time-frame 
for overbooking. If the statistical analysis shows, that the 
user made accurate estimations, the scheduler should not 
use her jobs for overbooking. If the user underestimates 
the runtime the scheduler might even plan more time to 
avoid job-kills at the end of the planned execution time. An 
application-oriented statistical analysis of monitoring data 
should be also performed in order to identify correlations of 
overestimations and a specific application. Performed stud-
ies show, that automatically determined runtime estimations 
based on historical information (job traces) can be better 
than the user’s estimation [25, 26, 27]. The condition for 
its applicability is that enough data is available. In addition 
to these separated foci, a third analysis should combine the 
user-oriented and application-oriented approach in order to 
identify whether specific users over- or underestimate the 
runtime when using a specific application. This analysis 
should result in valuable predictions.

3.1.2 Algorithmic Approach

This paragraph provides the algorithmic definition of the 
scheduling strategy for conservative backfilling with over-
booking. When a new job $j$ with estimated duration $d$, num-
ber of nodes $n$, and execution window $[t_{start}, t_{end}]$ arrives 
in the system, the following algorithm is used inserting the 
request into the schedule which has anchor point $t_s$ where 
resources become available and points where such slots end 
t_{endslot}:

1. Select, if available, statistical information about the 
runtime of the application and the runtime estimation 
of the user. Compare them with the given runtime for 
the new job $j$. If
   - the estimated runtime $d$ is significant longer than 
     the standard runtime of the application
   - or the user is tending to overestimate the runtime 
of jobs
– then mark the application as promising for overbooking. Assuming a uniform distribution, the duration of the job $d$ can be adjusted to $d' =\frac{d}{1 + \text{maxPoF}}$, where \text{maxPoF} is the maximum acceptable probability of failure. The time interval $o_j = d - d'$ can be used for overbooking.

• else the job should not be used for overbooking $d' = d$, $o_j = 0$.

2. Find starting point $t_s$ for job $j$, set $t_s$ as anchor point:

• Scan the current schedule and find the first point $t_s \geq t_{\text{start}}$ where enough processors are available to run this job.

• Starting from this point, check whether $t_s + d \leq t_{\text{end}}$ and if this is valid, continue scanning the schedule to ascertain that these processors remain available until the job’s expected termination $t_s + d \leq t_{\text{endslot}}$.

• If not,
  – check validity of $t_s + d' \leq t_{\text{end}}$ and whether the processors remain available until the job’s expected termination reduced by the time usable for overbooking $t_s + d' \leq t_{\text{endslot}}$. If successful, mark the job as overbooked and set the job duration $d' = t_{\text{endslot}} - t_s$.
  – If not, check, if there are direct predecessors in the plan, which are ending at $t_s$ and are usable for overbooking. Then reduce $t_s$ by the time $a = \min_k\{a_k\}$ of those jobs $k$ and try again. $t_s - a + d' \leq t_{\text{endslot}}$. (In this case, other jobs are also overbooked; nevertheless their runtime is not reduced. If they do not finish earlier than expected, they can still finish and the overbooked job will be started after their initially planned completion.) If successful, mark the job as overbooked and set the job duration $d' = t_{\text{endslot}} - (t_s - a)$.

• If overbooking was not possible, return and continue the scan to find the next possible anchor point.

3. Update the schedule to reflect the allocation of $r$ processors by this job $j$ with the duration $d'$ for the reservation $h = [t_s, \min(t_{\text{endslot}}, t_s + d)]$, starting from its anchor point $t_s$, or earlier $t_s - a$.

4. If the job’s anchor is the current time, start it immediately.

The algorithm defines that a job $k$ which was overbooked by a job $j$ should be resumed until its completion or its planned completion time if it had not finished at time $t_s - a$. Considering SLA bound jobs, this might be doubtful if fulfilling the SLA of job $j$ would be more profitable than of job $k$. However, the reservation duration of job $k$ is only reduced after curtaining the duration of job $j$. Hence, the provider has no guarantee that the SLA of job $j$ would be not violated if stopping the job execution of job $k$. Consequently, the scheduler should act conservatively and provide for job $k$ the resources as required and prevent an SLA violation of job $k$.

### 3.1.3 Checkpointing and Migration

By using overbooking the likelihood of conflicts increases and consequently the need of preventing job losses becomes more important. Which contractor (end-user or provider) has to pay the penalty in case of a not completed job depend on the responsibilities. In conservative planning-based systems, the user is responsible for an underestimated runtime of a job. Hence, if the job is killed after providing resources for the defined runtime, the provider does not care about saving the results. The provider is responsible if it the requested resources have not been available for the requested time. Hence, violating an SLA caused by overbooking results in that the provider has to pay the penalty fee. If the scheduler overbooked a schedule with a job which is planned for less than the user’s estimated runtime and has to be killed/dispatched for another job, the provider is responsible since resources had not been available as agreed. The RMS can prevent such conflicts by using the fault-tolerance mechanisms checkpointing and migration [9]. If the execution time had been shortened by the RMS, at the end of the reservation a checkpoint can be generated of the job, i.e. making a snapshot of its memory, messages, the registers, and program counter. The checkpoint can be stored in a file system available in the network. This allows to restart the not completed job in the next free gap before the job latest completion time. To be efficient, the free gap should be at least as long as the remaining estimated runtime. Note that filling gaps by partly executing jobs should not be a general strategy since checkpointing and migration requires resources and result in additional costs for the job execution. As result, planning with checkpointing and migration allow pre-emptive scheduling of HPC systems.

### 3.1.4 Example

To exemplify the approach, in the following a possible overbooking procedure is explained. Assume a Grid cluster with $X$ nodes, each job on the cluster will be assigned to the same fixed reservations with a duration of 5 hours $[7^{000} - 12^{000}, 12^{000} - 17^{000}]$. Assume further that the fixed and some usual advance reservations are already scheduled,
directly beneath each other. Thus, the resources are occupied for 20 hours of the schedule: $[7^{00} - 12^{00}, 12^{00} - 17^{00}, 17^{00} - 21^{00}, 21^{00} - 3^{00}]$. This schedule is the same every day in the week considered. Then another job $j$ with $h = 5$ hour reservation for $X$ nodes should be inserted in the schedule in the next two days. However, the resources are only free for 4 hours $[3^{00} - 7^{00}]$. Consequently, the scheduler has to reject the job request, in case it cannot overbook the schedule $3^{00} + 5$ hours $= 8^{00} \leq 7^{00}$.

We assume that the scheduler has statistics about the estimated and actual runtimes of applications. and users that propound an overestimation by 40%. Assume the scheduler can take $d = d'$ of the statistic over-estimated runtime of a job for overbooking, let the maximum PoF be $maxPoF = 13\%$. For a five-hour job these are 34 minutes. (As $h = 5$ hours $= 300$ minutes and $maxPoF = 0, 13 \rightarrow d' = \frac{d}{1 + maxPoF} = \frac{300}{103} \approx 265.5$ minutes $\rightarrow o = d - d' = 300$ minutes $- 266$ minutes $= 34$ minutes.) If we overbook the advance reservation $h$ for 34 minutes, the schedule is still not feasible ($3^{00} + 4:26$ hours $= 7^{26} - \leq 7^{00}$) since the gap is only 4 hours and $j$ would be given a runtime of 4 hours and 26 minutes. If the predecessor is also overbooked by 34 minutes, each job is reduced by half an hour and reservation $h$ can be accepted ($3^{00} - 0:34$ hours) + 4:26 hours $= 6^{52} \leq 7^{00}$. Thus the job $j$ with an user estimated runtime $d = 5$ hours has a duration $d' = 4:26$ hours and an estimated earliest start-time from $t_s = 2^{26}$ in the overbooked time-slot $[3^{00} - 7^{00}]$. The complete schedule is $[7^{00} - 12^{00}, 12^{00} - 17^{00}, 17^{00} - 21^{00}, 21^{00} - 3^{00}, 3^{00} - 7^{00}]$.

Note that, overbooking is possible only, if neither $j$ itself nor the predecessor (in case of its overbooking) is a fixed reservation. Hence, in our example avoiding an idle time of 4 hours can be achieved by using overbooking only, if the job $j$ and the reservations before $[2^{100} - 3^{00}]$ are not fixed. For all reservations which are not fixed, a greater flexibility exists if the start time $t_s$ of all those jobs could be moved forward to their earliest start time $t_{start}$, if the jobs before are ending earlier than planned. In this case, if the other execution times are dynamically shifted, any overbooked reservation from the schedule could be straighten out before execution. This approach has the big advantage that an 1 hour overbooked reservation $h$ could be finished even it would use the estimated time, if in total the predecessor reservations in the schedule require all in all 1 hour less time than estimated.

4 Conclusion and Future Work

The paper first motivated the need for Grid systems and in common with the management of supercomputers the advantages of planning-based scheduling for SLA provisioning and fixed reservations. However, advance reservations have the disadvantage to decrease the utilization of the computer system. Using overbooking might be a powerful instrument to re-increase the system utilization and the provider’s profit. The paper presented the concept and algorithm to use overbooking. Since overbooking might lead to conflicts because of providing resources for a shorter time than required, fault-tolerance mechanisms are crucial. Checkpointing and migration can be used for preventing job losses and SLA violations and support the applicability of overbooking in Grid systems.

Future work focus on statistical analysis of runtime and implementing the overbooking algorithm in the backfilling scheduling algorithms. At last we will develop a simulation process for testing and evaluating how much overbooking increases the system utilization and provider’s profit.

References


