The Gain of Overbooking

Georg Birkenheuer, André Brinkmann, and Holger Karl
{birke, brinkmann, holger.karl}@uni-paderborn.de

Paderborn Center for Parallel Computing (PC^2)
University of Paderborn, Germany

Abstract. This paper analyzes the effect of overbooking for scheduling systems in a commercial environment. In this scenario each job is associated with a release time and a finishing deadline as well as a fee for a successful execution and a penalty for violating the deadline. The core idea is to exploit overestimations of required job execution times, providing an opportunity to aggressively schedule additional jobs. The proposed probabilistic scheduler is based on histories of job execution times, device failure rates, and penalties for SLA service violations. This paper includes a theoretical background and a mathematical model of the overbooking approach and a simulative evaluation with a synthetic workload on a single-processor system.

1 Introduction

The commercial use of grid and cloud infrastructures is steadily growing. Current developments in automatic negotiation of service level agreements (SLAs) and the provision of quality of service (QoS) with fault tolerance mechanisms will further increase its commercial acceptance and adoption [1]. This acceptance allows providers to think about new business cases to foster their competitiveness in the emerging global service economy.

Grid or Cloud contracts will be based on the negotiation of SLAs between the service providers and their customers. During SLA negotiation, the customers reserve the amount of required computing and storage resources for a given time period. If the computation takes longer than expected, jobs are commonly killed at the end of the SLA-defined lifetime. Therefore, users are cautious not to lose their jobs and tend to overestimate their job's execution time and reserve resources accordingly. In practice, this leads to underutilized resources, as jobs finish often much earlier than expected.

Overbooking is used in many commercial fields, where more people buy resources than actually use them. To improve profit, airlines have e.g. become used to sell seat reservations more than once [2, 3]. The number of reservations that will not be used is estimated based on prior experiences and used in the planning processes. This estimated number is only correct with a specific probability. Consequently, if more passengers appear than expected and not enough seats are available in the aircraft, the airline has to pay a penalty to its customers.

Obviously, the objective of overbooking is to improve the expected profit. Instead of selling each seat once, profit can be increased by selling them several times. This

opportunity has to be compared with the risk implied by overbooking, i.e. the compensation for the buyer if no seat is available combined with the probability of that event. The best estimation of risk and opportunity will provide the best profit.

In this paper, we propose different *overbooking* strategies for grid, cloud or HPC infrastructure providers to increase their profit and competitiveness. These strategies differ in many aspects from traditional overbooking strategies for aircraft seats or hotel beds. On the one side, the number of concurrent users of a compute resource is smaller than the number of passengers of an airplane, making it harder to predict expected behavior. On the other side, computing jobs can be started nearly anytime, while a plane only takes off once for each flight.

Conservative scheduling strategies, which do not use overbooking, do not accept a job, if the maximum estimated job duration is even slightly longer than any gap in the current schedule. Applying overbooking, the scheduler can assess the risk to place the job in a gap that is smaller than the estimated execution time. For such an *overbooked job*, the probability of failure is no longer only dependent on machine failure rates (as in conservative scheduling), but it also depends on the likelihood that the real execution time of the job is longer than the gap length.

The proposed strategies are based on an analytical model for overbooking that uses the convolution of the probability density functions of the runtime estimates of the jobs to calculate the probability of failure (PoF) for a SLA. When the calculated risk is acceptably small in comparison to the opportunity, the service provider can accept the SLA.

The experimental evaluation of the strategies is based on real job traces, which show the benefits as well as associated risks of overbooking, especially if the customer base is diverse or unknown to the service provider. The job traces are derived from a one year record of job estimates and actual runtimes on a 400 processor cluster at the Paderborn Center for Parallel Computing (PC^2). To focus on the influence of overbooking, we have reduced the dimension of the job traces from a parallel machine to a single resource. However, we plan to generalize the described approach to overbooking in parallel machines.

The paper is organized as follows: In the next section we discuss the technical foundations of our work, followed by a description of our model and strategies for risk-aware overbooking in Section 3. In Section 4, we evaluate the risk and opportunity of overbooking mechanisms for Grid providers.

2 Related Work

This chapter summarizes relevant related work. Firstly, it discusses scheduling approaches in distributed environments, followed by related work on overbooking.

2.1 Scheduling

Most scheduling strategies for cluster systems are based on a first-come first-serve (FCFS) approach that schedules jobs based on their arrival times. FCFS guarantees

fairness, but leads to a poor system utilization as it might create gaps in the schedule. The gaps can occur, because each job description does not only contain execution time information, but also information about its earliest starting time / release time. Start times in the future are common for interactive jobs or for jobs which are part of a workflow. Interactive jobs are monitored and adopted by the users and the job runtimes have therefore to be known in advance by the user and have to fit to his working times and schedules. Workflow jobs can in principle run at arbitrary times, but dependencies between sub jobs enforce that the start time of a sub job is after the deadline of the preceding one. Gaps can therefore arise, if a new job arrives with a starting time after the end time of the last job in the current schedule. As standard FCFS schedules jobs strictly according to their arrival times, resulting gaps will remain idle and waste resources.

To increase system utilization and throughput in this scenario *Backfilling* has been introduced [4]. Backfilling schedules a new job not necessarily at the end of the plan, but is able to fill gaps if a new job fits in. The additional requirement for the ability to use backfilling instead of simply FCFS is an estimation about the runtime of each job. This allows to determine if the job fits in a gap in the schedule.

The *EASY* (Extensible Argonne Scheduling sYstem) backfilling approach [4] can be used to improve system utilization. Within EASY backfilling putting a job in a gap is acceptable if the first job in the queue is not delayed. This preserves starvation and leads to an increased utilization of the underling system. However, EASY backfilling has to be used with caution in systems guaranteeing QoS aspects, since jobs in the queue might be delayed.

Therefore, [5] introduced the *conservative* backfilling approach which only uses free gaps if no previously accepted job will be delayed. Thus, conservative backfilling still preserves fairness. Additionally, it is possible to plan a job, this means to determine the latest start time for every job. The latest start time is the time a job starts when all its predecessors use their complete estimated runtime.

Simulations show that both backfilling strategies help to increase overall system utilization [6] and reduce the slowdown and waiting time of the scheduling system. The work also shows that the effect of both described backfilling approaches is limited due to the inaccurate user estimation concerning the runtime of their jobs.

Several papers analyzes the effect of bad runtime estimations to the scheduling performance. The interesting effect is that bad estimations can lead to a better performance [7]. However, references [8] approach to improve the scheduling results by adding a fixed factor to the user estimated runtimes shows no advantage while using real job traces. Therefore, effort has been taken to develop methods to cope with the bad runtime estimations and have more accurate estimations. Reference [9–11] tried to automatically predict the application runtimes based on the history of similar jobs. Tsafier et al. present a backfilling approach which uses system-generated runtime predictions instead of given user runtime estimations [12]. The paper presents a scheduling algorithm similar to the EASY approach (called EASY++) that uses system-generated execution time predictions and shows an improved scheduling performance for the jobs' waiting times. The approaches show that automatically runtime prediction can improve back-

filling strategies, but the usability of the automatically runtime predicting approaches lack on wrong decisions according the runtime.

The approaches found in literature are not directly applicable to our work. Aim of the shown algorithms is to improve the system utilization as whole and decrease the slowdown of the single jobs. They assume to run in a queuing based scheduling system which tries best effort and does not deal with strict deadlines, thus SLA are not usable. Our work instead assumes a planning based scheduling scenario with strict SLAs and tries to improve a providers profit by overbooking ressources.

2.2 Overbooking

Overbooking is widely used and analyzed in the context of hotels [13] or airline reservation systems [3, 2]. However, overbooking of grid or cloud resources significantly differs from those fields of applications. The grid does not require fixed starting times for a resource, while e.g. a seat in an airplane cannot be occupied after the aircraft has taken off. As a consequence, results and observations from overbooking in the classical application fields cannot be reused for grid computing.

2.3 Use of Overbooking for Planning and Scheduling

Overbooking for web and internet service platforms is presented in [14]. It is assumed that different web applications are running concurrently on a limited set of nodes. The overbooking approach is based on firstly deriving an accurate estimate of application resource needs by profiling applications on dedicated nodes, and then by using these profiles to guide the placement of application components onto shared nodes. By overbooking cluster resources in a controlled fashion, the approach is able to provide performance guarantees to applications even when overbooked. The difference between this and our approach is that nodes are typically exclusively assigned. Therefore, it is at least difficult to share resources between different applications, while it is possible to use execution time length overestimations, which are not applicable for web hosting.

Overbooking for high-performance computing (HPC), cloud, and grid computing has been proposed in [15] or [16]. However, the references only mention the possibility of overbooking, but do not propose solutions or strategies. In the grid context, overbooking has been integrated in a three-layered negotiation protocol [17]. The approach includes the restriction that overbooking is only used for multiple reservations for workflow sub-jobs, which were queried by the negotiation protocol for optimal workflow planning. Chen et al. [18] use time sharing mechanisms to provide high resources utilization for average system and application loads. At high load, they use priority based queues to ensure responsiveness of the applications. Sulisto et. al [19] try to compensate no shows of jobs with the use of revenue management and overbooking. However they do not deal with the fact that jobs can start later and run shorter than estimated.

Nissimov and Feitelson [20] introduce a probabilistic backfilling approach where user runtime estimations and a probabilistic assumption about the real finish of the job allows to use a gap smaller as the estimated execution time. A job is allowed to be backfilled if the probability that the backfilling postpones the start of the rst job in the queue is less than a given trashold. If the job runs longer than the gap size the

following jobs are delayed. In scope of assessing the success of putting a job in a gap the probabilistic backfilling approach is similar to our overbooking scenario. The difference in the concepts are that the acceptance test of [20] is applied to an already scheduled job with aim to reduce its slowdown while in our approach each job has a deadline and the acceptance test is applied at arrival time to determine if the job could be executed before the deadline.

We propose to combine backfilling with overbooking concerning the acceptance test in a commercial scenario with focus on increasing profit of a grid or could provider. This instrument should increase system utilization and should not affect already planned jobs. To successfully use overbooking strategies, we have to be able to calculate the risk of violating SLAs and therefore we have to know the distribution of job execution time overestimations. The probability of success (PoS) for overbooking can then be calculated based on the likelihood that the job finishes in the given gap and the chance that the resource lives at the beginning of the planned execution and survives the job.

3 Overbooking Model and Algorithms

This section depicts the details of the applied overbooking model and our algorithmic approach, which is presented in Section 3.1. As we want to improve the profit by overbooking certain time slots, we need to estimate the PoF and PoS, which is (1- PoF), for each overbooked schedule. The accuracy of the PoF depends on the quality of the predicted job execution times (see Section 3.2).

Overbooking means to put a job in a gap in a schedule that is smaller than the job's maximum execution time. In fact, this job may actually try to use more time than the available size of the gap, leading either to a loss of this job or to a postponement of the following job. Both cases might lead to a penalty for the service provider. We assume

Table 1: Job scheduling information

Variable	Content	Comment
r	release time	earliest start time
ω	estimated execution time	given by the user
ddl	deadline	given by the user
s	start time	planned start time
f	finish time of the job	planned job end

a system with a single resource that has a failure rate λ and a repair rate μ, which are distributed according to a Poisson distribution. A job j has an earliest release time r, an estimated execution time ω, and a deadline ddl (see also Fig. 1). When the job is placed, the start time s is either its release time or the finish time of the previous job. The finish time f is important if the scheduling strategy follows conservative backfilling, where the job should not delay following jobs. Therefore, the job will be killed at $f = s_{\text{next}}$.

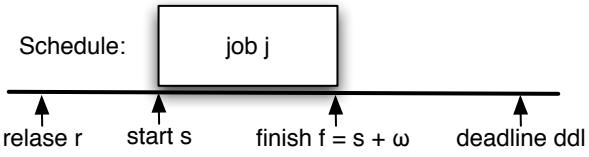

Fig. 1: Example of job information in a schedule.

3.1 Overbooking Algorithm

This paragraph briefly defines possible scheduling strategies for backfilling with overbooking. Generally, the scheduler holds a list of all jobs in the schedule. For each new job j_{new} arriving in the system, the scheduler computes the PoS for the execution of this job in every space free in the schedule where the job might be executed. For the concrete implementation of the scheduling algorithm, several decision strategies could be applied.

- A conservative approach could be chosen, where the job is placed in the gap with the highest PoS.
- A best fit approach uses the gap providing the highest profit, while still ensuring an acceptable PoF.
- A first fit approach, where the job is placed in the first gap with an acceptable PoS.

If the job is not placeable within the schedule, it can be planned as last job, if it is still executable before the user given deadline of the job. The calculation of the PoS is defined in Section 3.3 and decent PoF threshold values are evaluated in Section 4. In this paper we will further investigate an overbooking strategy based on first fit.

3.2 Job Execution Time Estimations

Evaluations of job execution time estimations ω and their corresponding real execution times x show that the job duration is typically overestimated by a factor of two to three [21]. A closer look on job traces shows that the distribution of the actual to estimated execution times seems to be uniform and has two peaks a the beginning and end of the spectrum forming a bathtub [6]. The first peak can be explained by jobs missing their input data and test jobs. The second peak includes 15% to nearly 30% of the jobs, which *underestimate* their execution time and which are killed after the negotiated execution time. The theoretical examples inside this paper assume the execution time distribution of the jobs to be uniform (see Fig. 2). However, it is important to notice that all results inside this paper do not depend on the shape of the execution time distributions. The simulations shown in Section 4 therefore also use other distributions that are derived from real job run traces.

For the calculation of the probability that the job 2 ends at time t, it is necessary to calculate the expected joint probability density function for the execution time distribution for job 2 and its predecessor job 1, $P_1(t)$ and $P_2(t)$. If jobs are scheduled following to each other, three different cases can be distinguished:

Jobs do not overlap: The jobs are not interfering if $r_2 > f_1$. In this case, the original probability distribution remains unchanged for each job.

Jobs are overlapping and have the same release time: The expected joint execution time distribution of two jobs $P_1(t)$ and $P_2(t)$ with the same release time can be calculated by a convolution of the execution time distributions of the two jobs.

The convolution leads to a distribution as shown in Fig. 3, if $\omega_1 < \omega_2$ and it leads to a simple triangle, if $\omega_1 = \omega_2$. If three or more jobs with the same release time r are convolved, the resulting distribution converges against a Gaussian distribution.

Jobs can overlap: The next job j_2 has a release time r_2 with $r_1 < r_2 < r_1 + \omega_1$. Here the probability distribution of job 2 has only to be convoluted with $P_1(t)$ for $t > r_2$. Furthermore, the probability p to end job 1 before r_2 has to be multiplied with the distribution of the original distribution of the job 2 and added to the convolution.

$$P_2^{new} = p \cdot P_2 + P_2 \circ P_1(t > r_2)$$

In most cases, the convolution has to be calculated numerically, as no (reasonable) closed formula exists.

3.3 PoS for Overbooking

The probability to successfully complete an overbooked job depends on the probability of a machine failure and the probability of the new job to finish in time. To finish in time means that the job has an execution time that fits into a gap between j_{prev} and j_{next}.

For the calculations we will define a job with a tuple $[r, \omega, ddl]$.

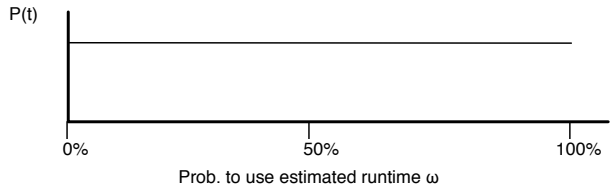

Fig. 2: The probability density function for a single job.

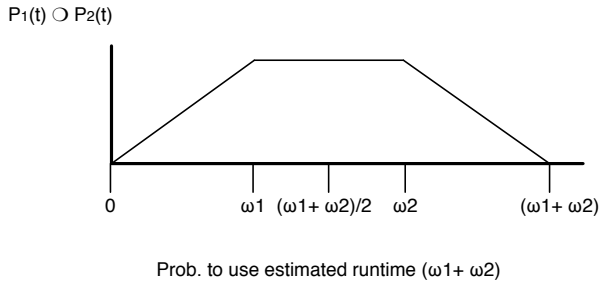

Fig. 3: The probability density function for two jobs with same release time.

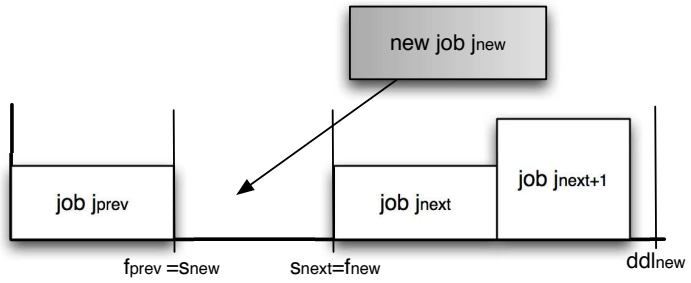

Fig. 4: The job can only be accepted using overbooking.

PoS(j_{new}) The probability $\text{PoS}(j_{\text{new}})$ depends on the probability $P_{\text{available}}(s)$ that the resource is operational at start time s, the probability $P_{\text{executable}}(j_{\text{new}})$ that the job is able to end with its given maximum execution time, and $P_{\text{success}}(j_{\text{new}})$ which is given by the machine failure rate λ and the job's execution time.

$$\text{PoS}(j_{\text{new}}) = P_{\text{available}}(s) \cdot P_{\text{executable}}(j_{\text{new}}) \cdot P_{\text{success}}(j_{\text{new}})$$

$P_{\text{available}}(s)$ The probability that the resource is operational at the start time is

$$P_{\text{available}}(s) = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\frac{1}{\lambda}}{\frac{1}{\lambda} + \frac{1}{\mu}} = \frac{1}{1 + \frac{\lambda}{\mu}}$$

where MTTF is the mean time to failure $\frac{1}{\lambda}$ and MTTR is the mean time to repair $\frac{1}{\mu}$.

$P_{\text{executable}}(j_{\text{new}})$ The calculation of the probability to execute successful $P_{\text{executable}}(j_{\text{new}})$ is described in in detail in Section 3.2. If the job j_{new} has no predecessor it is scheduled at its release time and $P_{\text{executable}}(j_{\text{new}})$ is given by the execution time distribution

and the maximal execution time x of the job. For a uniform distribution it holds that $P_{\text{executable}}(j_{\text{new}}) = \frac{x}{\omega}$. If $x = \omega$ then $P_{\text{executable}}(j_{\text{new}}) = 1$.

If the job j_{new} has a direct predecessor j_{prev} the convolution of the execution time distribution has always to be computed with the previous job's distribution. The reason for calculating the distribution with the previous job results from the fact, that due to overbooking the job j_{new} has no defined start time any more. j_{new} starts at the end of its predecessor j_{prev}. As the distribution of j_{prev} already includes the distributions from all possibly influencing previously planned jobs the convolution has only to be done with j_{prev}. $P_{\text{executable}}(j_{\text{new}}) = 1$ if the job has its full estimated execution time ω available and less if the job is overbooked and $s + \omega < ddl$.

$\boldsymbol{P_{\text{success}}(j_{\text{new}})}$ The probability that the job's resources survive the execution time is given by their failure probability which is determined by the machine failure rate λ and the job's execution time x. $P_{\text{success}}(j_{\text{new}}) = e^{-\lambda \cdot x}$

Decision Making: During SLA negotiation a simple equation can decide whether it is beneficial to accept an SLA with overbooking or not:

$\text{If}(\text{PoS}_{SLA} \cdot \text{Charge}_{SLA} > \text{PoF}_{SLA} \cdot \text{Penalty}_{SLA})$ accept the SLA.
else reject the SLA.

Nevertheless, in the following we will use an overbooking strategy that bases its decision only on a threshold for the probability of failure PoF to investigate the influence of different, more or less aggressive strategies. If the ratio between charges and penalties is constant, then it is possible to derive the results for the presented decision making strategies by simply setting the threshold to the learned best ratio.

4 Evaluation

This section evaluates the possible additional profit that a provider can earn with overbooking. Generally, the expected profit for a job $E[\text{profit}_{\text{job}}]$ for overbooking is:

$$E[\text{profit}_{\text{job}}] = \text{Charge}_{\text{job}} \cdot \text{PoS}_{\text{job}} - \text{Penalty}_{\text{job}} \cdot \text{PoF}_{\text{job}}$$

As this section will show, the additional profit strongly depends on the quality of the prediction of the execution times.

4.1 Simulation Model

Several parameters influence the simulation results:

Simulation Resources: Actually, only a single resource is considered. We plan to extend the simulation to be able to cope with $n \in \text{N}$ resources.

Simulation Metrics For simplification, the simulation considers the relation of charge to penalty as equal, of for each booked time unit. However, Section 4.8 shows that the relation of charges to penalties does not affect PoF values for maximum profit.

Job Creation Model

- The average job length is exponentially distributed with a mean of 72 time units.
- The earliest release time of the jobs also follows an exponential distribution with a mean of 30 time units which is added to the release time of the previously created job.
- After the creation process for the jobs, their release times are monotonously increasing in the job number. But an increasing order of release times would simply favor FCFS and would not be realistic according to our scenario. Therefore, the order is afterwards randomly permuted to create a more realistic release time distribution.
- The deadline of each job is set to its release time plus five times the estimated execution time ($ddl = r + 5 * \omega$).

The mean of the release time distribution compared to the mean of the job length directly describes possible load of the system. If the mean of the release times is bigger as the mean job length more jobs are arriving as feasible in the scheduling system. The chosen simulation parameters enforce that more jobs are submitted than the system could successfully execute. Each simulation ends after the deadline of the last accepted job.

Resources Stability

- The resource's MTTF is 14505 time units or $\lambda = 0.00165$.
- The resource's MTTR is 12 time units or $\mu = 0.0833$.

These assumptions are taken for most of the shown simulation. We will additionally show in Section 4.8 that different MTTF values only lead to an offset in the profit curve for the overbooking strategies.

Simulations Usage: For each test run, the incoming jobs, job length, and release times as well as the up and downtime of the resource have been randomly chosen. Based on this input data, the three strategies have been applied and the results of the strategies have been evaluated and aggregated to the following figures. For each test point, we have performed 10000 runs with 100 incoming jobs per run. Therefore, every result is based on 600000 schedules.

Execution Time Distribution: The required execution time of a job is calculated based on the applied execution time distribution. We have evaluated four different distribution schemes. The first one is the simple uniform distribution. The second one is a bathtub distribution, which seems most realistic if applied to a huge number of jobs and users [6]. This bathtub curve could also be derived from the traces of our local cluster system. The next execution time distributions are derived from traces of two dedicated users of our cluster system. They should depict the fact that given enough information (traces from jobs) of a single user, better overbooking results could be achieved.

Simulations result We calculate a threshold P_{max} that will provide the maximum PoF acceptable by the scheduler for different situations. To evaluate the best threshold for overbooking, we have implemented FCFS and conservative backfilling to be able to compare the overbooking strategies with standard implementations. The overbooking strategy of accepting jobs is based on the PoF given by the convolution of the execution time distribution with the distribution of the previous jobs. A job is placed in the first gap where the calculated PoF is lower than $P_{\max}$.

4.2 Uniform Execution Time Distribution

The evaluation starts with the assumption that the execution time distribution follows an uniform distribution. The simulation results for each test run contain the charge and the penalty of each strategy. The profit of a test run is its charge minus the penalty. Penalties for FCFS and backfilling without overbooking can only occur, if the machine fails during the execution of a job (see Figure 5).

Depicted results shown in the figure 5 are

- y-axis *charge* and *penalty* for each strategy
- x-axis maximal PoF $P_{\mathbf{max}}$ accepted

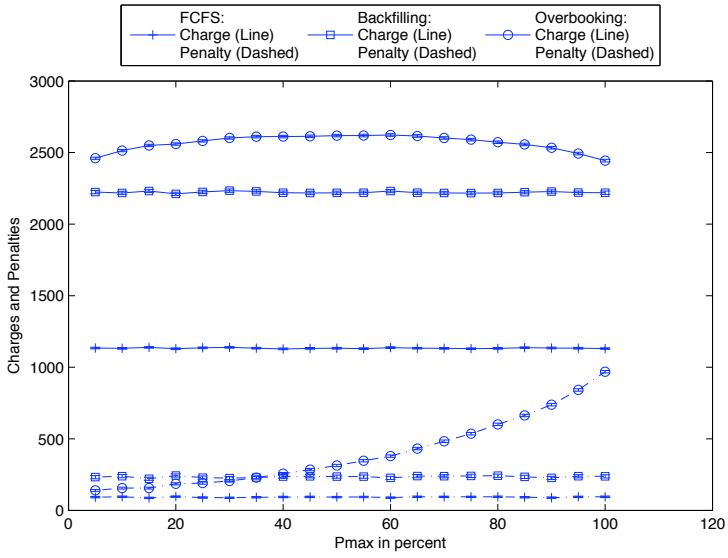

Fig. 5: Charge and penalties with different scheduling strategies.

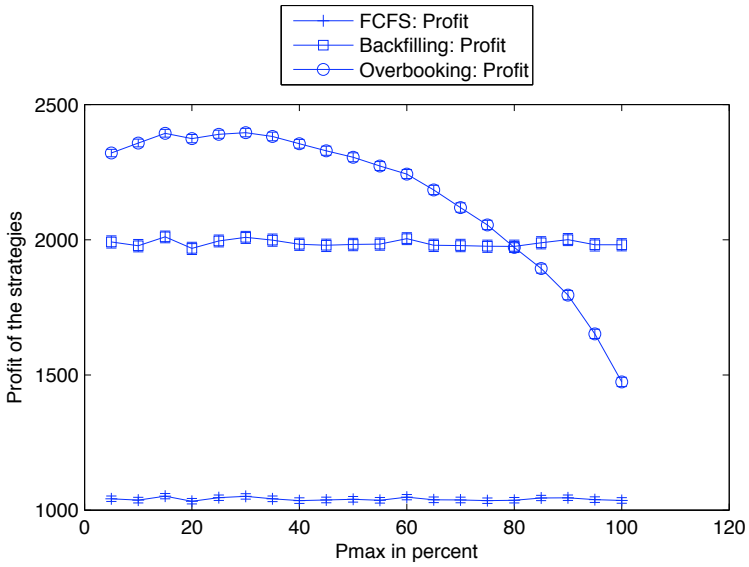

Fig. 6: The accumulated profit for uniform execution time distributions.

In the Figure 6 and following figures showing simulation results are

- y-axis *profit*.
- x-axis maximal PoF $P_{\mathbf{max}}$ accepted

The simulation starts with a maximum acceptable PoF for a job of 0.05 and ends with 1. The x-axis has, besides fluctuations of the randomized measurement parameters, no influence on FCFS and backfilling without overbooking. Markers around the values in the figures show the $95\,\%$ confidence intervals.

For a uniform execution time distribution, the FCFS strategy fills on average 1100 time units per schedule, while the backfilling strategy has a mean planned execution time of 2200 time units per schedule. The overbooking strategy is able to fill on average 2570 time units, depending on the accepted PoF $P_{\max}$ and varying from 2440 to 2630 time units per schedule. It is also shown in Figure 5 that the penalties for overbooking start lower than for conservative backfilling. This is caused by the fact that conservative backfilling is mostly unable to re-start a job after a resource outage, because the remaining time slot is not long enough. In contrast, overbooking still has the opportunity to fill the new, smaller gap in the schedule.

Due to space limitations, we will only present the accumulated profit in the following. The profits with overbooking are, until a $P_{\max} = 0.8$, better than with simple backfilling (see Figure 6). Overbooking strongly depends on $P_{\max}$. The profit is increasing at the beginning due to additionally accepted jobs and is shrinking at the end again due

to the increasing amount of violated SLAs caused by too high accepted values of $P_{\max}$. For these assumptions, $P_{\max} = 0.3$ should be chosen to maximize profit, increasing the profit by 20 % compared to a conservative backfilling strategy.

4.3 Bathtub Distribution

The following simulations are based on a job execution time analysis of the year 2007 for the Arminius HPC cluster system at the Paderborn Center for Parallel Computing (PC^2). Within the analyzed 23286 jobs, 6109 jobs used less than 1 % of the booked execution time, while 3553 jobs used 100 %. For the simulation, we have assumed that 26 % of the jobs have zero execution time, 15 % use 100 % percent of their execution time and 59 % of the jobs are uniformly distributed in between.

Figure 8 shows the profit similar to Section 4.2. FCFS and conservative backfilling do not dependent on the quality of the runtime estimations and we will not discuss their behavior again. The overbooking strategy fills on average 2490 time units for each schedule, varying from 2330 to 2610 time units for different $P_{\max}$. It is shown in Figure 8 that the maximum profit after subtracting the penalties is achieved for $P_{\max} = 0.6$. Starting with $P_{\max} = 0.9$, overbooking induces negative effects. The profit is increased compared to a conservative backfilling strategy by 19 % for $P_{\max} = 0.6$.

4.4 Simulations with precise execution time estimations

This section analyses overbooking approaches for customers with very precise execution time estimations. The simulation uses input data of a user which nearly always uses 88.2 % of the reserved execution time. There are some jobs which have been killed

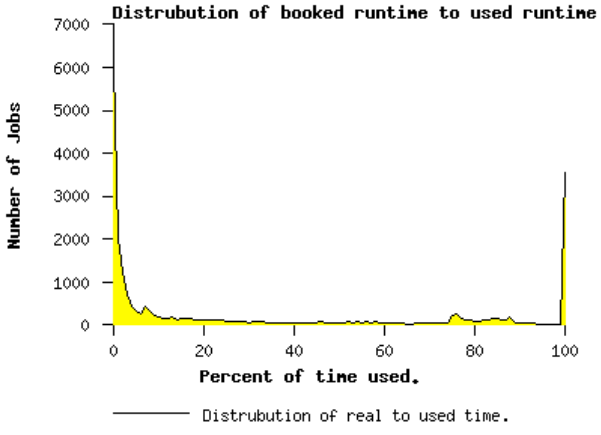

Fig. 7: Execution time distribution on a real cluster system for the year 2007.

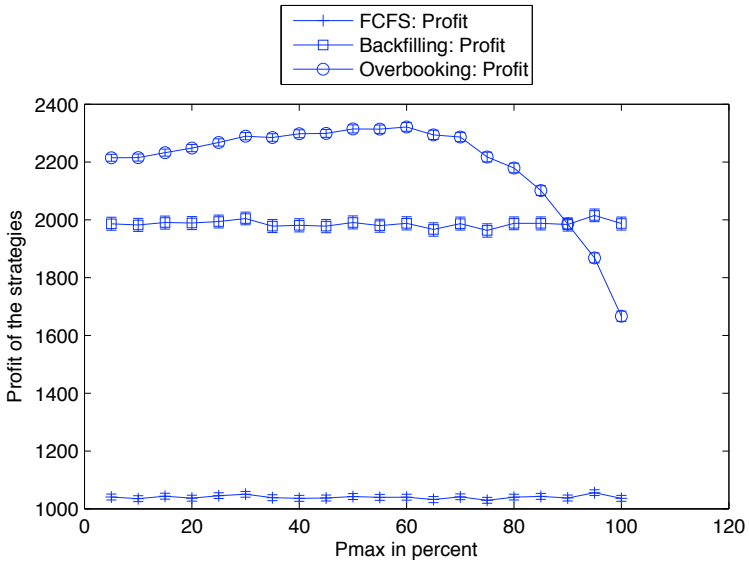

Fig. 8: Profit with different scheduling strategies for a bathtub distribution.

due to missing input data and some which have underestimated their required execution time (see Figure 9).

The results of this simulation are shown in Figure 10. The overbooking strategy fills on average 2370 time units, varying from 2320 to 2390 time units per schedule. At $P_{\text{max}} = 0.35$ the maximum profit is available. Then, until $P_{\text{max}} = 0.95$, the additional profit is nearly stable. Starting from P_{max} of 0.95, overbooking jobs for this customer has a negative effect. The profit is increased compared to a conservative backfilling strategy by 13 % for $P_{\text{max}} = 0.35$.

4.5 Execution Time Distributions with Near-Bathtub behavior

The following simulations are based on a user profile, where the user rarely uses more than 60 % of the reserved execution time. There is a peak in the distribution at zero and at 100 % and a high probability that the user consumes between 1 % to 60 %. This interval includes 85 % of all jobs and the interval from 60 % to 99 % only contains 5 % of the jobs (see Figure 11). The overbooking strategy fills on average 2660 time units, varying from 2340 to 2780 time units per schedule. The maximum profit is achieved for $P_{\text{max}} = 0.4$. Then, until $P_{\text{max}} = 0.9$, the additional profit is nearly stable. Starting from $P_{\text{max}} = 0.97$, overbooking jobs of this customer has a negative effect. The profit is increased compared to a conservative backfilling strategy by 22 % for $P_{\text{max}} = 0.4$.

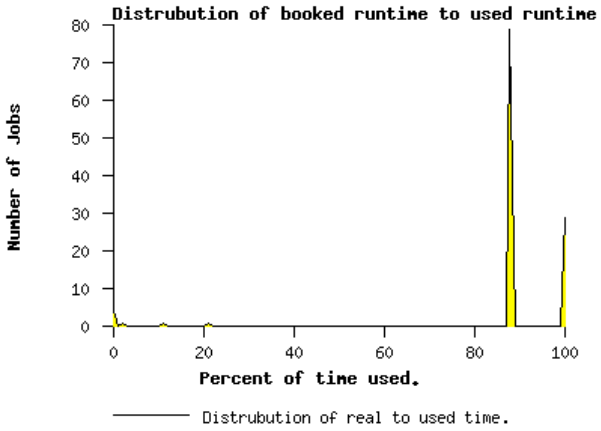

Fig. 9: Execution time distribution with a peak at 88 % for a single user.

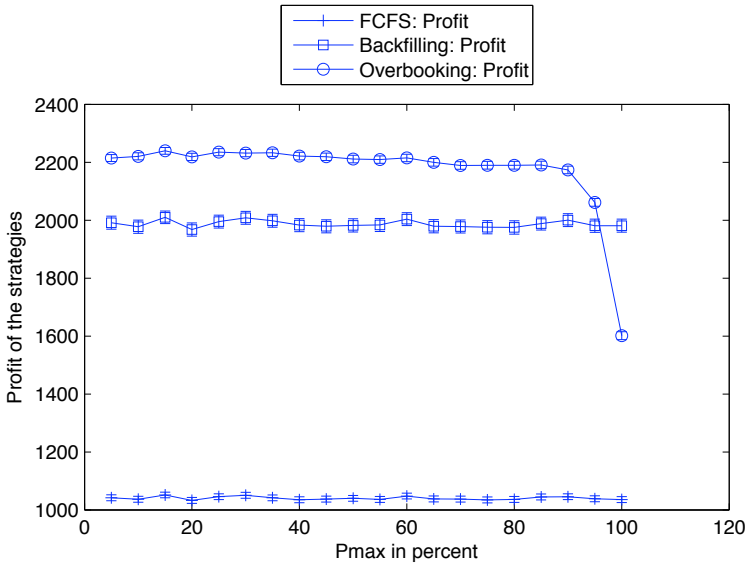

Fig. 10: Profit with different scheduling strategies for precise user estimations.

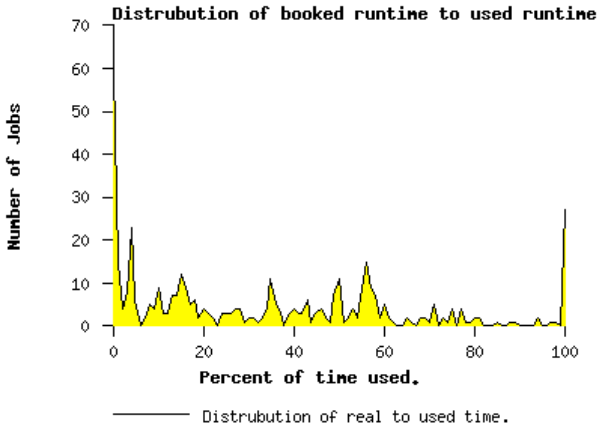

Fig. 11: Job execution time distributions with a near bathtub characteristic.

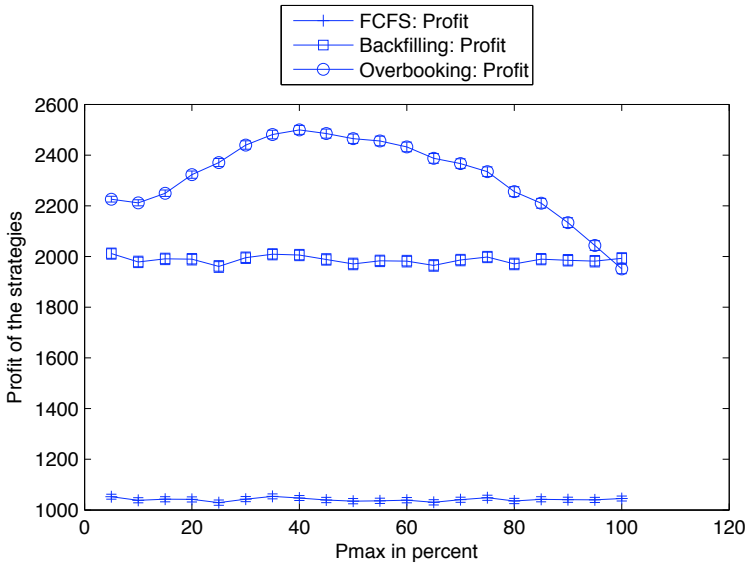

Fig. 12: Simulation results for user with most of the jobs gathered between $1\,\%$ to $60\,\%$ of the execution time.

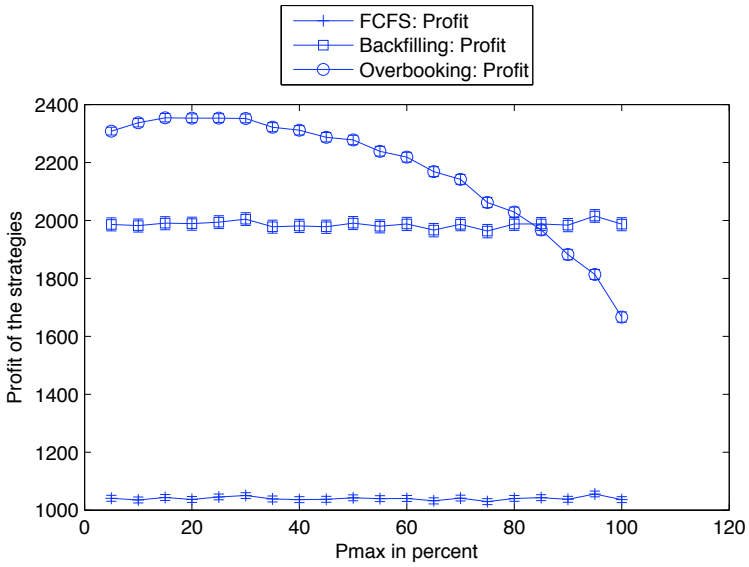

Fig. 13: Here a bathtub execution time distribution occurs while the scheduler assumes a uniform distribution.

4.6 Uniform vs. Bathtub

In the previous sections, the calculation for the overbooking strategies have been based on the same runtime distributions as the simulated jobs. We will investigate in the following sections the influence of imprecise runtime estimations on the quality of the scheduling. Figure 13 shows the case in which the overbooking strategy assumes a uniform job length distribution, while the simulated jobs behave according to bathtub distribution from section 4.3.

The overbooking strategy fills on average 2570 time slots, varying from 2450 to 2610 time units per schedule. The maximum profit can be achieved for $P_{\text{max}} = 0.15$. Starting from $P_{\text{max}} = 0.8$, overbooking jobs of this customer has a negative effect. The profit is increased compared to a conservative backfilling strategy by $19\,\%$ for $P_{\text{max}} = 0.15$. For these comparable probability density functions, overbooking is still able to produce very good results.

4.7 Uniform vs. Peak

Now, the simulation evaluates a very different user behavior compared to the runtime prediction. The simulation still assumes a uniform distribution, while the simulated jobs are created according to the peak distribution from Section 4.4 .

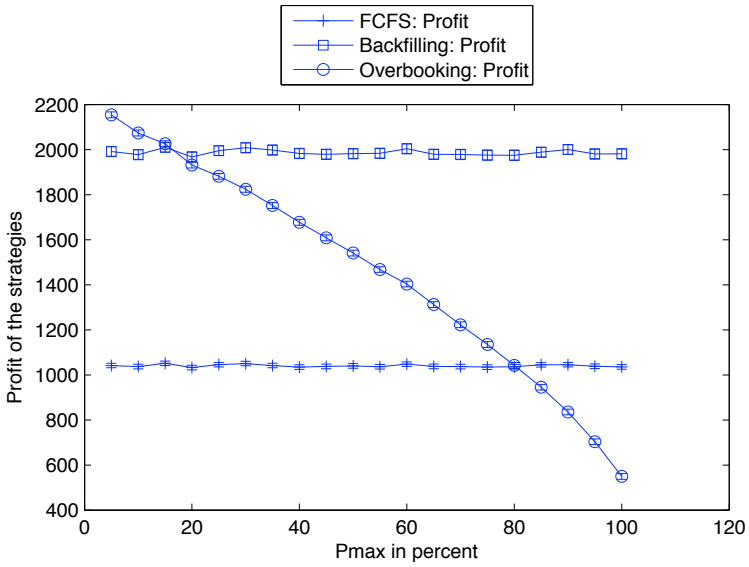

Fig. 14: Here a peak execution time distribution occurs while the scheduler assumes a uniform distribution.

The overbooking strategy fills on average 2210 time units per schedule, varying from 1980 to 2380 time units per schedule. At $P_{\max} = 0.05$ the maximum profit is available there the profit is increased compared to a conservative backfilling strategy by 9 %. Already starting from $P_{\max} = 0.15$, overbooking jobs of this customer has a negative effect and starting from $P_{\max} = 0.8$ the profit is worser than with FCFS.

It is clear that in this case overbooking has a very bad impact on the schedule, as the user does a very exact assessment and the overbooking assumes a uniform distribution. The result shows that for users which are able to accurately predict their job runtimes, overbooking has to be applied very carefully.

4.8 Dependency on Penalty and MTTF

The previous simulation results assumed equal profit and penalty for each time unit. Figure 14 contains results for different ratios of profit and penalty, starting from factor one and going up to a factor of five. As input we have chosen the results of the measurement of Section 4.3. It is clear that an increased penalty decreases the achievable profit. However, the shape of the curves is not affected.

Figure 16 shows additional simulations concerning resource stability. They were performed to evaluate the impact of the machine failure rates on the predictions for overbooking. The result shows that the value of $P_{\max}$ is nearly independent of the machine outage characteristics.

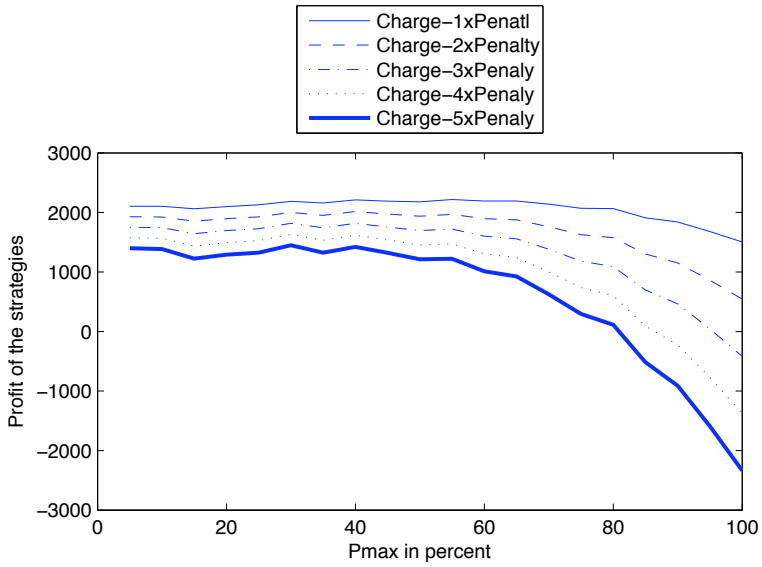

Fig. 15: The resulting profit with different penalty factors

The evaluation shows that the impact of overbooking in Grid, Cloud, or HPC environments is dependent on the accuracy of the underlying assumptions of the relation of booked to real used runtime. With the given data of a real cluster system and assuming SLA negotiations, it is possible to increase the profit of a cluster system by 20 %. Furthermore, the simulations show that assuming accurate assumptions for user's runtime estimations the profit of a cluster system can be further increased. On the other hand, incorrect assumptions can have a negative impact on the profit.

5 Conclusion and Future Work

This paper has motivated the need for overbooking in Grid, Cloud or HPC environments and outlined the limitations of current scheduling algorithms. Thereafter, the idea of using overbooking to increase the ability to accept more SLAs has been shown. As overbooking increases the risk of SLA violations, mechanisms for determining whether or not it is worthy to use overbooking have been shown followed by an evaluation of the impact from the proposed methods on the ability to successfully accept additional SLAs. Therefore, a threshold P_{max} has been defined with which a provider can maximize the profit for a overbooking strategy. The evaluation shows that the additional profit depends on the accuracy of the underlying runtime estimations and can be given real runtime distributions around 20 % of additional utilisation.

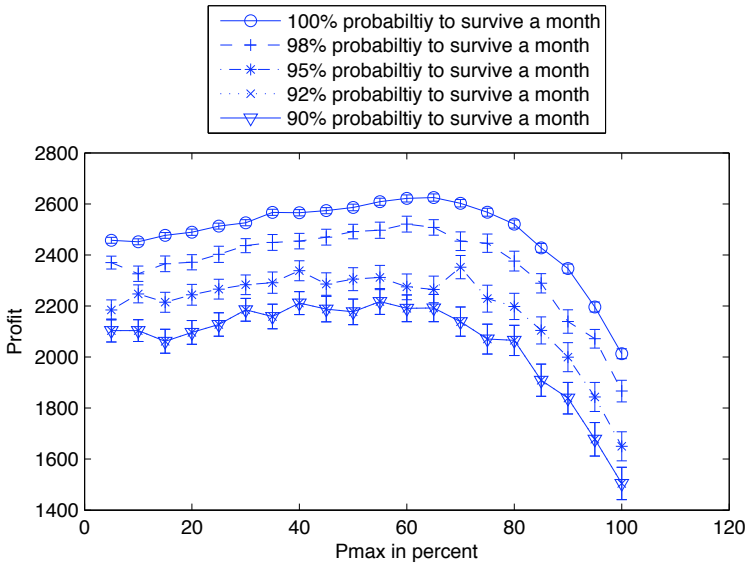

Fig. 16: The resulting profit with different penalty factors

An interesting point of future work will be to define overbooking strategies which will be able to combine multiple resources and that can be used for parallel jobs. It might also be interesting to determine if there are user and application specific distributions which would allow to increase the quality of the risk estimations for overbooking. With this knowledge, the quality of the estimations could be further increased.

Acknowledgment

The authors would like to thank the EU for partially supporting this work within the 6th Framework Programme under contract IST-031772 "Advanced Risk Assessment and Management for Trustable Grids" (AssessGrid).

References

1. Battre, D., Hovestadt, M., Kao, O., Keller, A., Voss, K.: Increasing fault tolerance by introducing virtual execution environments. 1. GI/ITG KuVS Fachgespräch ãVirtualisierung Ò (2007) 49
2. Rothstein, M.: Or and the airline overbooking problem. Operations Research **33**(2) (1985) 237–248
3. Subramanian, J., Jr, S.S., Lautenbacher, C.: Airline yield management with overbooking, cancellations, and no-shows. Transportation Science **33**(2) (1999) 147–167

4. Feitelson, D., Jette, M.: Improved utilization and responsiveness with gang scheduling. Job Scheduling Strategies for Parallel Processing: IPPS'97 Workshop, Geneva, Switzerland, April 5, 1997: Proceedings (1997)
5. Feitelson, D., Weil, A.: Utilization and predictability in scheduling the ibm sp2 with backfilling. 12th International Parallel Processing Symposium (Jan 1998)
6. Mu'alem, A., Feitelson, D.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp 2 with backfilling. IEEE Transactions on Parallel and Distributed Systems **12**(6) (2001) 529–543
7. Zotkin, D., Keleher, P.: Job-length estimation and performance in backfilling schedulers. High Performance Distributed Computing (Jan 1999)
8. Tsafrir, D., Feitelson, D.: The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. IEEE International Symposium on Workload Characterization (Jan 2006)
9. Gibbons, R.: A historical application profiler for use by parallel schedulers. Job Scheduling Strategies for Parallel Processing: IPPS'97 Workshop, Geneva, Switzerland, April 5, 1997: Proceedings (1997)
10. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. Lecture Notes in Computer Science (Jan 1998)
11. Tsafrir, D., Etsion, Y., Feitelson, D.: Modeling user runtime estimates. Lecture Notes in Computer Science (Jan 2005)
12. Tsafrir, D., Etsion, Y., Feitelson, D.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (2007) 789–803
13. Liberman, V., Yechiali, U.: On the hotel overbooking problem-an inventory system with stochastic cancellations. Management Science **24**(11) (1978) 1117–1126
14. Urgaonkar, B., Shenoy, P., Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms. ACM SIGOPS Operating Systems Review **36**(si) (2002) 239
15. Andrieux, A., Berry, D., Garibaldi, J., Jarvis, S., MacLaren, J., Ouelhadj, D., Snelling, D.: Open issues in grid scheduling. UK e-Science Report UKeS-2004-03, April 2004
16. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in hpc resource management systems: Queuing vs. planning. Job Scheduling Strategies for Parallel Processing: 9th International Workshop, Jsspp 2003, Seattle, Wa, Usa, June 24, 2003: Revised Papers (2003)
17. Siddiqui, M., Villazon, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized qos. (2006) 21
18. Chen, Y.W.M., Liu, X.: Efficiently rationing resources for grid and p2p computing. Lecture Notes in Computer Science. **Volume 3222/2004. Network and Parallel Computing** (2004) 133–136
19. Sulistio, A., Kim, K., Buyya, R.: Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. (2008) 267–276
20. Nissimov, A., Feitelson, D.: Probabilistic backfilling. JSSPP 2007 (Jan 2007)
21. Streit, A.: Self-tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids. PhD thesis (2003)