Risk Aware Overbooking for Commercial Grids

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

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

Abstract. The commercial exploitation of the emerging Grid and Cloud markets needs SLAs to sell computing run times. Job traces show that users have a limited ability to estimate the resource needs of their applications. This offers the possibility to apply overbooking to negotiation, but overbooking increases the risk of SLA violations. This work presents an overbooking approach with an integrated risk assessment model. Simulations for this model, which are based on real-world job traces, show that overbooking offers significant opportunities for Grid and Cloud providers.

1 Introduction

Grid, Cloud, and HPC providers need intelligent strategies to optimally utilize their existing resources, while not violating quality of services (QoS) guarantees negotiated with the customers and described through service level agreements (SLAs). For the acceptance test of committing to an SLA, a provider uses runtime estimations as well as a deadline from the customer. Job traces show that the user's ability to estimate runtimes is limited[1]. This leads to a statistical measurable overestimation of runtimes as well as to underutilized resources, as jobs are tending to end earlier than negotiated.

To increase the resource utilization and therefore the profit of a provider, we propose to combine overbooking and backfilling techniques for parallel resources in the acceptance test. This instrument should increase system utilization, while not affecting already planned jobs. To successfully use overbooking strategies, we have to be able to calculate the risk of violating SLAs. Our approach uses a history of the distribution of job execution time estimations and their corresponding real runtimes. The probability of success (PoS) for overbooking can then be calculated based on the likelihood that the job finishes within the given runtime.

Scheduling Model We propose a commercial scenario, where a job execution is negotiated between a Grid customer and a provider. For operation, the grid provider uses a planning based scheduling system. This means that the jobs are not scheduled in a queue, but added to a plan of jobs, where each job has, if accepted, an assigned start time, a number of assigned resources, and a maximum duration. The scenario has four characteristics:

1. the underlying scheduling strategy is FCFS with conservative backfilling
2. the user pay for their submitted jobs proportionally to the computation time they estimate

3. the customers have to receive their jobs' results within a given deadline.
4. the monetary penalty inflicted on the system's owner for missing a job's deadline is equal to the price users pay to for a successful execution

Under these assumptions, the our approach evaluates whether schedulers can exploit automated runtime predictions (along with the fact users typically give inaccurate runtime estimates) in order to *overbook* the gaps within the schedule in a manner that increases the overall profit of the provider.

Technically, the plan of a scheduler has two dimensions, where the width is the number of nodes in the cluster and the time corresponds to the height. When a job request enters the system, the scheduler puts the job in this plan as a rectangle between its release time and deadline. If the job is accepted, it is placed in the schedule and no other jobs can be assigned to this area (see Figure 1). Therein, the latest start is the point in time, where all previous jobs ended consuming their whole estimated runtime. Here the job can start in every case as long the underling resources did not crash. The earliest possible start time is the time where the job can start if all previous jobs needed null runtime. This practically means the start of the latest job starting before or the jobs release time. The job will thus start somewhere in between the earliest and latest possible start. If the scheduling algorithm cannot place the job according to the resource and deadline constraints, the job has to be rejected.

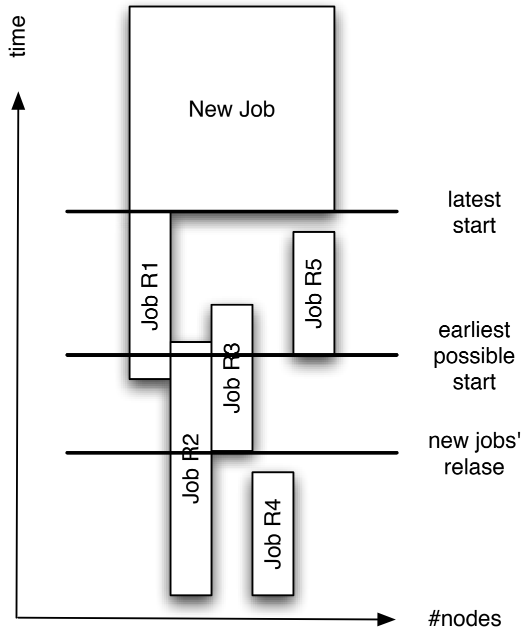

Fig. 1: Exemplary job schedule.

This paper is organized as follows: Section 2 presents the related work on overbooking, scheduling, resource stability assessment, and planning. Section 3 introduces our methods and instruments to measure the inaccuracy of the users' runtime estimates. Section 4 describes how we calculate the probability of success of overbooking a schedule. Section 5 evaluates the proposed methods and algorithms and presents simulation results based on real job traces and the paper finishes with a discussion about the achieved goals.

2 Related Work

This section presents the technical basics and related work in the area of scheduling and overbooking. Firstly, it discusses theoretical approaches for planning, followed by scheduling approaches. Then, the paper introduces related work on machine failures and risk assessment in Grid and Cluster systems. At last, we discuss related work on overbooking and its impact on planning and scheduling.

Planning Theory Planning strategies are also known as *Strip Packing* problem [2]. The aim is to pack jobs in a way that the height of the strip is minimal while the jobs must not overlap themselves. Different strategies have been developed, which should pack as optimal as possible, where optimal packing itself is NP-hard.

Strip Packing distinguishes between offline and online algorithms. Offline algorithms are unusable in our scheduling approach, as jobs are not known in advance. Approaches usable in our online scheduling environment are bottom-left algorithms, which try to put a new job as far to the bottom left of the strip as possible [3]. Level algorithms split the strip horizontally in levels of different sizes [4]. In these levels, appropriate sized jobs can be placed. Shelf algorithms divide the strip vertically in smaller shelves, which could be used for priority based scheduling [5]. Hybrid algorithms are combinations of the above-mentioned algorithms [2]. The disadvantage of the presented scheduling approaches is that jobs in Grid or Cloud environments are connected with an SLA that contains a strict deadline. Therefore, the approach of strip and shelf algorithms of packing jobs earlier or later is impossible. A usable approach here is the simple bottom-left algorithm, where the bottom is given by the earliest start time of a job and a natural ceiling is given by its deadline.

Scheduling Approaches Many scheduling strategies for cluster systems are still based on first-come first-serve (FCFS). FCFS guarantees fairness, but leads to a poor system utilization as it might create gaps in the schedule.

Backfilling, in contrast, is able to increase system utilization and throughput [6]. It has not to schedule a new job at the end of a queue, but is able to fill gaps, if a new job fits in. The additional requirement for the ability to use backfilling is an estimation about the runtime of each job. The runtime estimations are, in our scenario, part of the SLAs. The *EASY* (Extensible Argonne Scheduling sYstem) backfilling approach can be used to further improve system utilization. Within EASY, putting a job in a gap is acceptable if the first job in the queue is not delayed [6]. However, EASY backfilling has to be used with caution in systems guaranteeing QoS aspects, since jobs in the queue might be delayed.

Therefore, Feitelson and Weil introduced the *conservative* backfilling approach, which only uses free gaps if no previously accepted job is delayed [7]. Simulations show that both backfilling strategies help to increase overall system utilization and reduce the slowdown and waiting time of the scheduling system [8]. The work also shows that the effect of the described backfilling approaches is limited due to inaccurate runtime estimations. Several papers analyzes the effect of bad runtime estimations on scheduling performance.

An interesting effect is that bad estimations can lead to a better performance [9]. Tsafrirs shows an approach to improve scheduling results by adding a fixed factor to the user estimated runtimes [10].

Effort has been taken to develop methods to cope with bad runtime estimations. Several approaches tried to automatically predict the application runtimes based on the history of similar jobs [11–13]. Tsafier et al. present a scheduling algorithm similar to the EASY approach (called EASY++) that uses system-generated execution time predictions and shows an improved scheduling performance for jobs' waiting times [14]. The approach shows that automatically runtime prediction can improve backfilling strategies.

The approaches found in literature are not directly applicable to our work. The algorithms target queuing based systems and provide best effort. Their aim is to improve system utilization and to decrease the slowdown of single jobs. Our approach is a planning based scheduling scenario with strict deadlines, given by SLAs. We want to provide an acceptance test, where we have to decide if we can successfully accept an additional job and thus improve a provider's profit by overbooking resources.

Machine Failure and Risk Assessment Schroeder [27] and Sahoo [28] have shown that machine crashes in cluster systems are typically busted and correlated and Iosup and Nurmi showed that the failures rates of large clusters follows a Weibull distribution best [31, 32]. The project AssessGrid [29] created instruments for risk assessment and risk management at all Grid layers. This includes risk awareness and consideration in SLA negotiation [30] and self-organisation of fault- tolerant actions. The results allow Grid providers to assess risk and end-users also to know the likelihood of an SLA violation in order to accurately compare providers SLA offers. The motivation of the research presented in this paper has its origin in work done by AssessGrid.

Overbooking Overbooking is widely used and analyzed in the context of hotels [15] or airline reservation systems [16, 17]. However, overbooking of Grid or Cloud resources differs from those fields of applications. A cluster system can always start jobs if enough resources are free, while a free seat in an airplane cannot be occupied after the aircraft has taken off.

Overbooking for web and Internet service platforms is presented in [18]. It is assumed that different web applications are running concurrently on a limited set of nodes. The difference to our approach is that we assign nodes exclusively. Therefore, it is impossible 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 [19, 20]. 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 [21]. The approach includes the restriction that overbooking is only used for multiple reservations for workflow sub-jobs. Chen et al. [22] 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 [23] 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 introduced a probabilistic backfilling approach, where user runtime estimations and a probabilistic assumption about the real end time of the job allow to use a gap smaller than the estimated execution time [24]. In the scope of estimating the PoS of putting a job in a gap, the probabilistic backfilling and our overbooking scenario are similar. The difference is that Nissimovs acceptance test is applied to an already scheduled job and aims to reduce its slowdown, while our approach is used during the acceptance test at arrival time [24].

We have proposed our ideas for overbooking with focus on a single resource [25, 26] and are extending the algorithms and investigations for parallel resources in this work.

3 Probability Density Function

The Probability Density Function (PDF) for a job describes the likelihood that a job ends after exactly x % of its estimated runtime. An example for a PDF is given in Figure 2, which shows this probability distribution for all jobs submitted to a compute cluster in 2007.

Building a PDF We assume that a job has an assigned start time, but it can start anytime after its release time, if the corresponding resources are free earlier. Therefore, all currently running jobs on the assigned resources have an influence on the real start time of a new job. Every such job has its own probability density function that describes its likelihood to end at some point in time. The aim of the following algorithm is to build a joint PDF, which contains information for a set of jobs. This PDF is the basis to calculate the probability that this set of jobs ends before the deadline of the last job.

The challenge is that several jobs j_1 to j_n can end before the start of a new job. The maximum number n of jobs is equal to the amount of resources required by the job. The minimum number is zero, when all resources are free at the job's release time.

The latest point in time, where a job will start is the latest planned finish time of any job planned on the used resources before.

However, a job is allowed to start earlier if possible. The earliest possible start time is either the jobs release time or the time where the job can start if all previous need zero runtime. This is the start time of the latest job starting before. The new job might start directly after this job's start, if it ends directly after dispatching, for example due to a missing input.

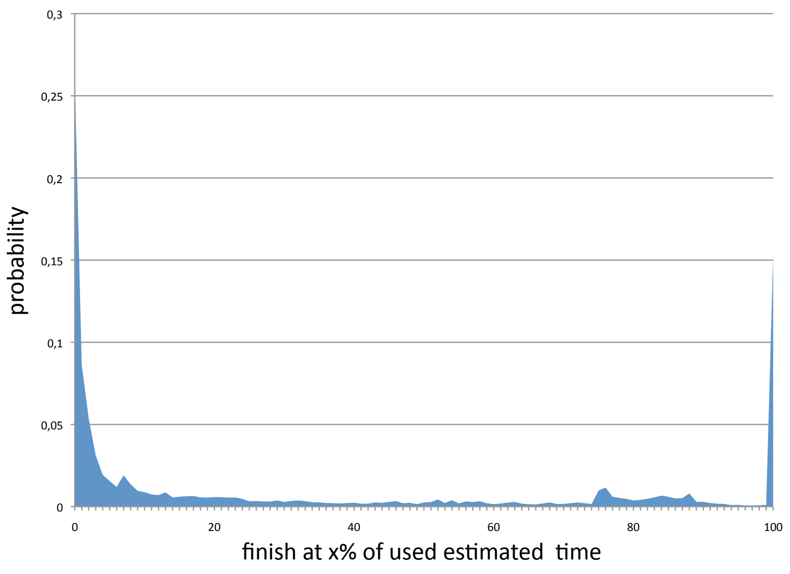

Fig. 2: The PDF derived from all jobs of 2007 in the examined cluster.

An example is given in Figure 1 for a job which requires five resources . We have to calculate the PDF of all jobs that are scheduled before and can possibly run in the time between the release and start of the new job.

Deriving PDFs from Job Traces One way to create PDFs is an analysis of the ratio of real to estimated runtime of historical job traces. Figure 2 shows the Probability Density Function of an exemplary cluster for 2007.

This work assumes that the user's estimation accuracy will not change too much over time. Thus, the past performance of users might be a good estimate for the future. The more job traces are available, the more information the PDF can contain. This work did not only calculate a basic PDF for all jobs, but also different PDFs for different estimated job runtimes.

In Figure 3, we show eight different cumulative distribution functions (CDF) each for an estimated time range, which are integrated over the corresponding PDFs. The figure shows that the estimation quality of the users is best for jobs from three to four hours. As result, we assume that quality of the users' estimations in our planning based scheduling also depends on the estimated length of the runtime.

Calculating the joint PDF for a Job We have derived several PDFs for different runtime estimations. Thus, when a new job arrives in the system, the most appropriate PDF, according to the estimated runtime, is chosen for the job itself. Unfortunately, the PDF

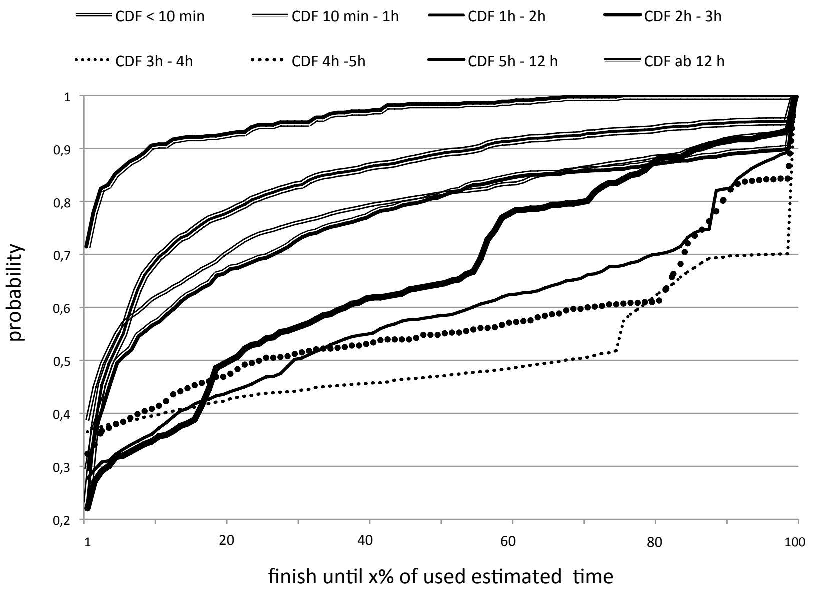

Fig. 3: The CDF for several time slots.

is not a continuous function, but given from traces. In our framework, we have decided to use discrete steps with one value for each percent step.

For the calculation of the probability that a job ends at time t, it is necessary to calculate the expected joint probability density function for the execution time distribution for the job and its predecessors. In the case that both PDFs are overlapping, the expected joint execution time distribution consists of the convolution of the jobs basic PDF and the calculated PDF of all jobs finishing earlier (see also [26]).

For the simulation, the convolutions are based on discrete values and are stretched or shrank according to the required number. This is given by the number of steps used per time unit and the length of the job. In reality the distributions are continuos functions and the discrete mapping reduces the accuracy. Nevertheless, the convolutions have to be calculated numerically, as no (reasonable) closed formula exists.

4 Risk Assessment for Overbooking

The aim of this section is to define the statistical model used to calculate the probability of success (PoS), which is later assigned to every job. The PoS is calculated based on the statistical runtime overestimations, the estimated runtime for the job, the maximal available runtime inside the gap, as well as the failure rate of the resource. If the gap is smaller than the estimated runtime, there is a chance that the job will still be successful, if it finishes earlier.

The PoS helps the overbooking algorithm to manage and control the underlying scheduling. An acceptance test has to be applied for each new job to decide whether it is beneficial to accept this job even when the underlying resources are overbooked.

Table 1: Job scheduling information.

Variable	Content
r	release time
ω	estimated execution time
ddl	deadline
s	start time
f	finish time of the job
n	number of nodes

We assume that the system consists out of N resources, where each resource has the same failure rate λ and repair rate μ. A job j requests n resources and has an earliest release time r, an estimated execution time ω, and a deadline ddl. When the job is placed, the start time s is either its release time or the finish time of the last 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 $f = s_{\text{next}}$.

Calculating the PoS for Overbooking The probability of successfully completing an overbooked job depends on the probability of resource failures and the probability that the new job finishes in time. To finish in time means that the job has an execution time that fits into a gap between f_{last} and s_{next}. For the calculations, we will define a job j as a tupel $[s, r, \omega, ddl, n]$. The result of the calculation is the probability that a new job is successful in a given gap.

PoS(j_{new}) The probability PoS(j_{new}) depends on the probability $P_{\text{available}}(s)$ that the requested resources are operational at start time s, the probability $P_{\text{executable}}(j_{\text{new}})$ that the job is able to end within 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. Therefore,

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

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

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

where n is the requested number of resources, MTTF is the mean time to failure $\frac{1}{\lambda}$ and MTTR is the mean time to repair $\frac{1}{\mu}$. This model assumes that the node failures are independent, which is a simplification compared to previous work [31, 27]. It has been shown that node failures are bursty and correlated. However, as a job execution is not possible even when one of the planned resources fails, we do not include the amount of other node failures here. In praxis, when the failure rates and behavior of the underling cluster system is known $P_{\text{available}}(s)$ should be analyzed in more detail. However, the failure analysis is not in scope of this work.

$P_{executable}(j_{new})$ The calculation of the probability to successfully execute $P_{\text{executable}}(j_{\text{new}})$ is given by the PDF convolution. The result (between 0 and 1) is the PoS of the job. The higher this value is, the more likely is the success.

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 its own execution time distribution and the maximal execution time t of the job. $P_{\text{executable}}(j_{\text{new}}) = 1$ if the job has its full estimated execution time ω available and less if the job is overbooked.

If the job j_{new} has one or more direct predecessors the convolution of the execution time distribution has always to be computed with the joint distribution of the previous jobs, which already includes the distributions from all possibly influencing previously planned jobs.

$$P_{\text{executable}}(j_{\text{new}}) = \int_0^t (PDF_{\text{jobs before}} \circ PDF_{\text{new job}})$$

$P_{success}(j_{new})$ $P_{\text{success}}(j_{\text{new}})$ describes the probability that the job's resources survive the execution time. It has been shown that crashes in cluster systems are correlated an bursty [27, 28] and the failures rates of large clusters follows a Weibull distribution [31, 32]. Following, the definition of $P_{\text{success}}(j_{\text{new}})$ as $1 - e^{-(\frac{x}{\beta})^k}$ would describe the survival rate. Here x is the execution time, $\beta > 0$ describes the spreading of the distribution, and k describes the failure rate over time. A value of $k < 1$ indicates that the failure rate decreases over time, due to hight infant mortality, $k = 1$ means the failure rate is constant, and a value of $k > 1$ indicates that the failure rate increases with time, e.g. due to some aging process.

However, the Weilbull distribution describes an aging processes of the resources over years while the typical jobs are lasting hours to some days. In addition, the failure rate λ has to be adapted over the day and week/weekend as it is shown that it depends on the load of the system [27, 28]. As the current workload traces do not contain the corresponding machine failure traces, we concentrate on the job traces and simplify the failure rate. We assume a constant failure rate λ for the job execution time x. The constant failure rate allows us to model the probability that the job's resources survive the execution time as the constant failure probability λ, the job's execution time x, times the number of requested resources n, and therefore,

$$P_{\text{success}}(j_{\text{new}}) = e^{-\lambda \cdot x \cdot n}.$$

Risk an Opportunity of Overbooking Mathematically, the opportunity of overbooking would be defined by the PoS of the job and the possible income described as fee of the SLA. On the other hand, there is always a risk accepting an overbooked job. This is defined as the probability occurrence times the impact of this event. The probability of the bad effect is the PoF of accepting the job and the impact is described by the penalty defined in the SLA for a violation.

Accordingly, during SLA negotiation a simple equation can decide whether it is beneficial to accept an SLA with overbooking or not.

- If$(PoS \cdot Charge > PoF \cdot Penalty)$ accept the SLA,
- else reject the SLA.

This term simply says: Do not accept jobs, where the risk is higher than the opportunity.

Possible Planning Strategies 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 free space in the schedule where the job might be executed. For the concrete implementation of the scheduling algorithm, several 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 and a first fit approach places the job in the first gap with acceptable PoS.

Implemented First Fit In this paper we will further investigate an overbooking strategy based on first fit. We check all time-slots starting with the release time of the job where at least the requested amount of resources is available. For each time slot, the algorithm checks, how long the requested resources will be available. If more resources than requested are available, the algorithm chooses the first resources according to their numbers, placing it as left as possible. The algorithm calculates the PoS for placing the new job in this gap based on the chosen resources, the gap length, and the joint PDF. If the PoS is higher than the given threshold, the algorithm places the job in the gap. The approach is thus strongly related to the bottom left first approach in the field of strip packing algorithms [3].

The Overbooking Process Concluding, the overbooking algorithm follows 5 steps:

1. For every new job
2. Detect the possible places for the job
3. Do for all places beginning with the first
   (a) Calculate the joint PDF for the jobs before
   (b) Combine the PDF of the jobs before and the actual job's PDF.
   (c) With this PDF, the resource stability, and number of resources, build the PoF
   (d) If the PoF is smaller than the threshold, accept the job
4. If no place with suitable PoF has been found, reject the job.

5 Evaluation

This section describes the evaluation of the benefit of our overbooking approach. We have used four job traces from the parallel workloads archive[1] as input, namely SDSC SP2, KTH, BLUE, and CTC. The presented simulations evaluate the outcomes for conservative backfilling and two different overbooking approaches. Firstly, we use a basic statistical model with one PDF built from past user-estimations and secondly, we use an extended statistical model with several PDFs for different time slice lengths.

[1] Parallel Workloads Archive:
http://www.cs.huji.ac.il/labs/parallel/workload/

Simulation Model Several parameters influence the simulation results. For each test run, the incoming jobs contain the number of required nodes and an estimated and real job length. The job submission times and their release times as well as the up and downtime of the resources have been randomly chosen (see Table 2). Based on this input data, the strategies have been applied and the results are evaluated.

Simulation Resources: Actually, we chose the number of nodes for the simulation according to the size of the cluster system where the traces were from. Thus, the numbers of nodes in the simulation were 128 nodes for the SDSC trace, 100 for KTH, 144 for the BLUE trace, and 430 nodes for CTC. The stability of the underlying resources is not given in the traces. Therefore, the simulation has set the chance to survive a month for each resource to 95%, which correspondents to $\lambda = 0.000068943$ and lasted the MTTR of 12 hours ($\mu = 0.08333$).

Table 2: Job Creation Model.

Variable	Description
req. job length e	Chosen from job traces
real job length ω	Chosen from job traces
average time between submission o	1 hour
average delay between job submission and release time r	12 hours
deadline ddl	$r + 5 \cdot e$
req. nodes n	Chosen from job traces

Charge and Penalty A very important point for the economical adaptability of overbooking is the ratio of the charge of an SLA to its penalty. The overbooking strategy has to be more careful, if ratio between penalty and charge is higher while the opportunity becomes bigger for higher charges. The simulation assumes the charge and penalty are the same and one hour execution time counts as one virtual money unit.

Job Creation Model The jobs arrival times follow an exponential distribution with given delay to the last job. This delay directly describes the load of the simulation, the faster the jobs are arriving the higher the possible utilization. The chosen simulation parameters enforce that more jobs are submitted than the system could successfully execute. This is done to be able to simulate an environment were overbooking seems to be promising. The release time of the jobs also follows an exponential distribution with a mean of 12 hours which is added to the job submission time. Each simulation ends after the deadline of the last accepted job.

One input parameter of each simulation run is a threshold P_{max} that provides the maximum PoF acceptable by the scheduler for different situations. 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}.

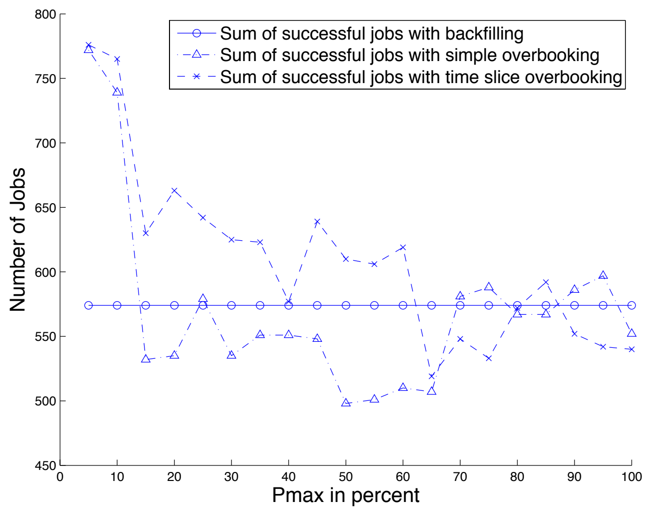

Fig. 4: SDSC: Sum of successful jobs.

Use of Different Job Traces We have removed all jobs that do not contain an estimated runtime as well as a real runtime entry. All jobs except the last $1,000$ trace entries were used for each setting to learn the jobs' runtime behavior. Based on this jobs we created, according to Section 3, a distribution for the simple overbooking approach and several time slice distributions for the time slice overbooking. Thereafter, we have used the last $1,000$ jobs as simulation input.

SDSC Figures 4 to 7 show the results based on the SDSC SP2 trace. Figures 4 and 5 show the accumulated results of Figures 6, and 7. This means for the jobs Figure 4 contains the successful minus the failed jobs and Figure 5 contains the profit minus the penalty. For this simulation, 0.5 hours have been chosen as basic random value for the delay between the jobs. The SDSC cluster system had 128 nodes. From the $60,000$ jobs of SDSC the first $59,000$ were taken to learn the jobs' runtime behavior. The simulation starts with a maximum acceptable PoF for a job of 0.05 and ends with 1. Like in all following simulation runs, 1000 jobs were submitted to the system.

The backfilling strategy always planed 570 jobs with $2,600$ hours execution time. Both overbooking strategies have at the beginning a sum of 770 jobs and a bit more than $2,600$ hours gain. These 770 jobs are the successful jobs minus the failed jobs.

The number of jobs is for both overbooking approaches at the beginning $P_{\max} = 0.05$ much better than backfilling and then rapidly shrinking. This has two reasons. Firstly, for a higher threshold, jobs with more nodes and longer estimated runtimes are accepted. This circumvented the acceptance of some shorter jobs. Secondly, an increasing amount of jobs failed with the increasing $P_{\max}$.

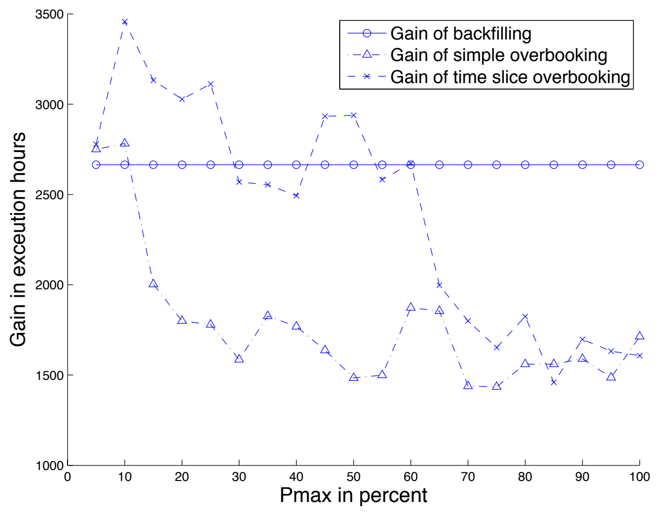

Fig. 5: SDSC: Sum of profit.

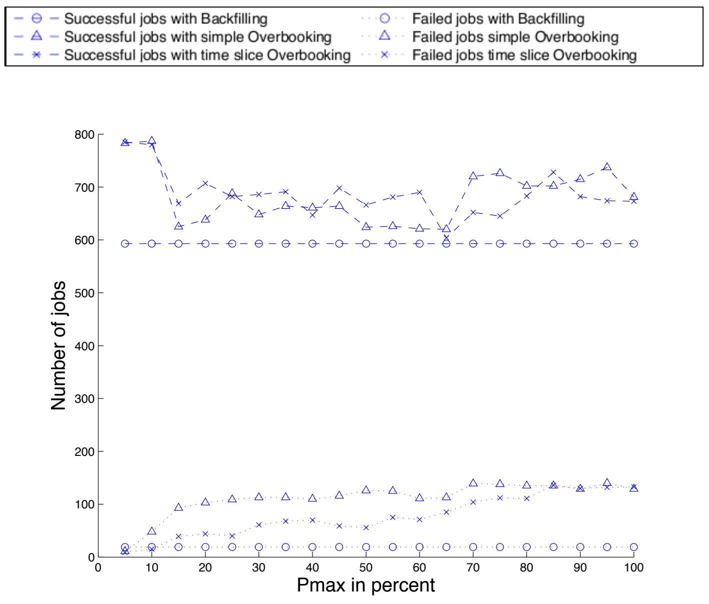

Fig. 6: SDSC: Shown is the number of successful and failed jobs.

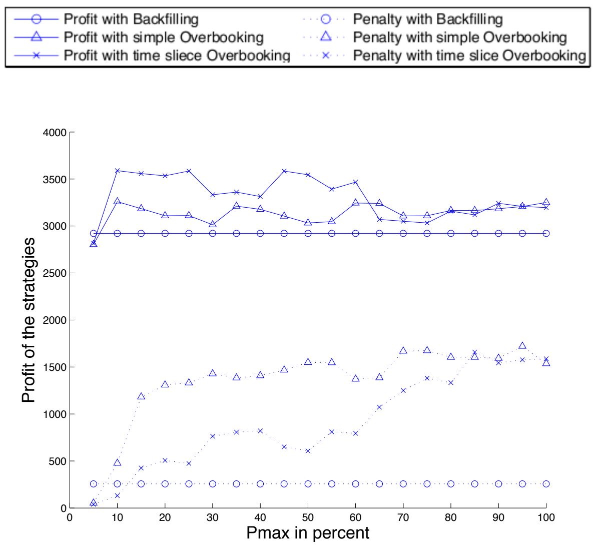

Fig. 7: SDSC: Shown is the profit and penalties of successful and failed jobs.

For the gain, which reflects the successful utilization of the resources, the behavior is a little different. The gain of simple overbooking is at the beginning just a little bit better than the backfilling approach and shrinks for higher P_{max}. This shows that the quality of the underlying statistical analysis is paramount for a successful overbooking approach. The profit for the time slice approach increases from $P_{\text{max}} = 0.05$ to 0.1 where the sum of successful jobs is falling. This is caused by the fact that the simulation has accepted some longer jobs including more nodes, which were not chosen in the run with a lower threshold (0.05). Thereafter, the gain of time slice overbooking is falling as more resource consuming jobs are accepted. Some of this jobs are failing hand in hand with the higher accepted risk. This shows that the threshold choice is very important for successfully applying overbooking. The gain of simple overbooking is worse than backfilling from a P_{max} of 0.1. The time slice overbooking performs better until a P_{max} of 0.3.

For this simulation, $P_{\text{max}} = 0.1$ should be chosen to maximize profit. With the SDSC traces it is possible to increase the profit by $30\,\%$ compared to a conservative backfilling strategy.

All in all, there are many peaks in the figures. This is caused by the fact that with little higher PoF threshold an additional job can be accepted that prohibits the acceptance of some following jobs and vice versa.

KTH Figures 8, 9,10, and 11 show the results based on the KTH trace. For this simulation, 0.1 hours have been chosen as basic random value for the delay between the jobs.

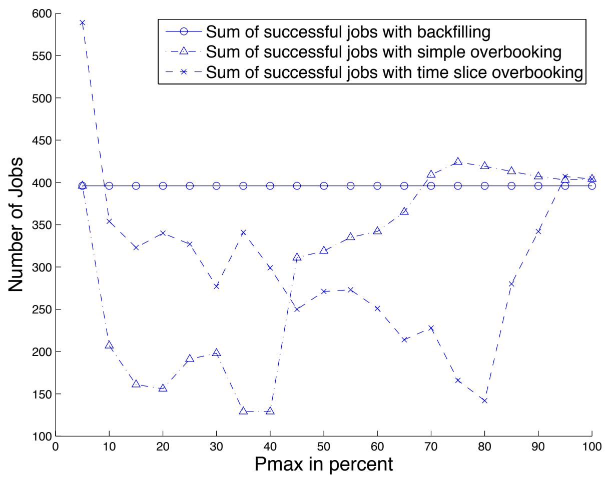

Fig. 8: KTH: Sum of successful jobs.

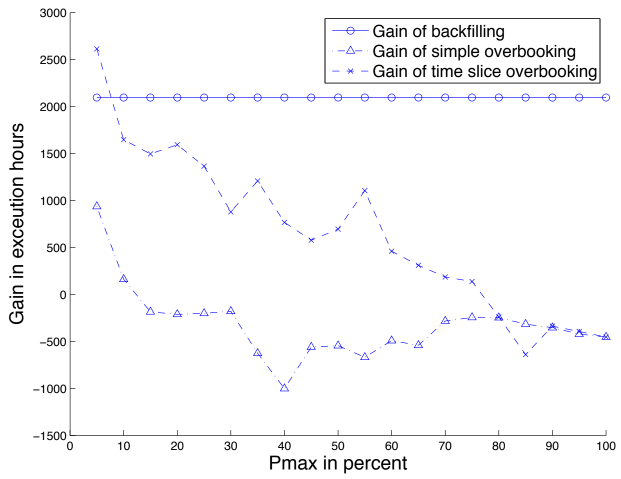

Fig. 9: KTH: Sum of profit.

The KTH cluster had 100 nodes. From the $28,500$ jobs of KTH the first $27,500$ were taken to learn the jobs' runtime behavior.

The backfilling strategy always planed 400 jobs with $2,100$ hours execution time. The simple overbooking strategy has at the beginning also a sum of 400 successful jobs

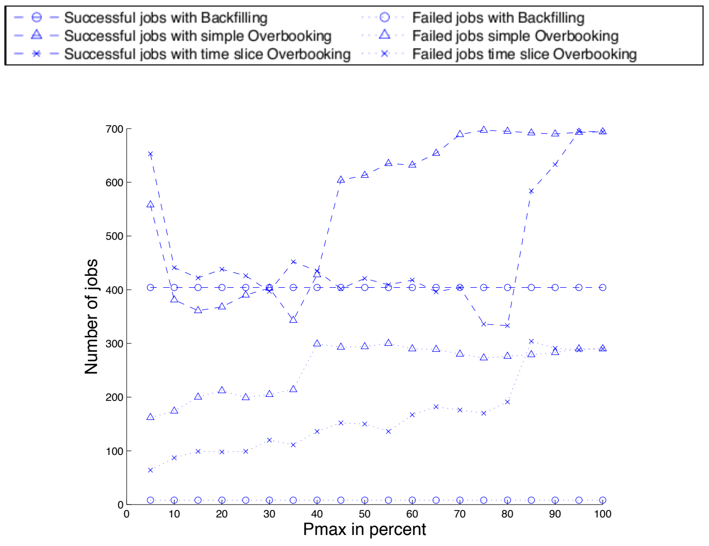

Fig. 10: KTH: Shown is the number of successful and failed jobs.

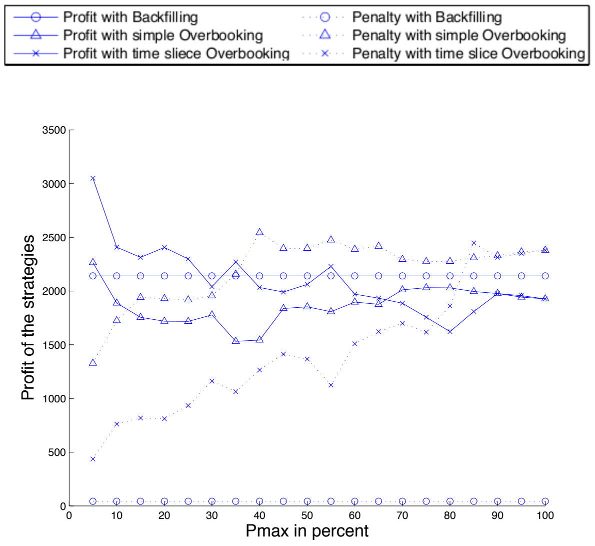

Fig. 11: KTH: Shown is the profit and penalties of successful and failed jobs.

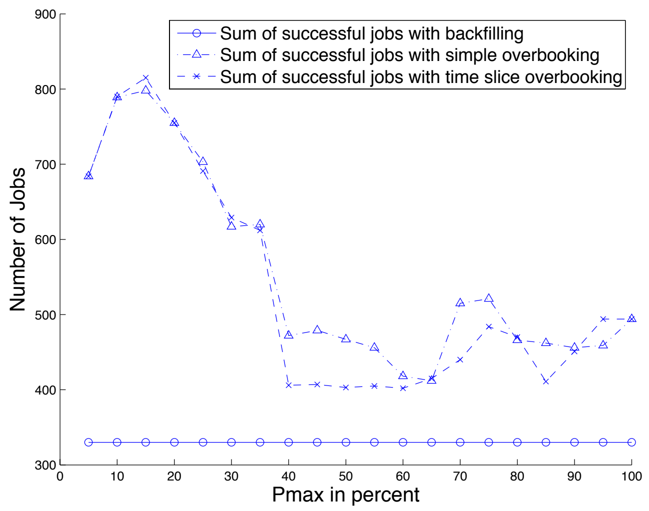

Fig. 12: BLUE: Sum of successful jobs.

with $1,200$ hours gain, while the time slice overbooking has a sum of 600 successful jobs and $2,600$ hours gain.

The gain of simple overbooking is nearly always worse than the backfilling strategy. This shows that applying overbooking with a simple statistical analysis can have a severe impact on the providers profit. The sum of successful jobs and gain is falling rapidly under the backfilling level. Interesting is that the number of successful jobs and profit is rapidly shrinking from 0.05 to 0.1 and all in all less jobs are accepted. This means, due to a little higher accepted PoF, jobs with more resource requirements are accepted and fail. With higher P_{max} the amount of successful jobs is increasing again. This has little effect on the sum of successful jobs as simultaneously the number of failing jobs is also increasing. However, with the use of an improved statistical analysis even with a varying behavior of jobs the overbooking can, carefully adapted, increase the profit. With $P_{\text{max}} = 0.05$ the gain of time slice overbooking is better than the backfilling approach. It is increased by about $23\,\%$. This trace shows that for some user behaviors on clusters an enhanced statistical analysis should be adapted, to further improve the overbooking result. Using statistical analysis based on applications or users basis serve this purpose.

BLUE Figures 12, 13,14, and 15 show the results based on the BLUE trace. For this simulation, 1 hour has been chosen as basic random value for the delay between the jobs. The BLUE cluster had 144 nodes. From the $243,000$ jobs of BLUE the first $242,000$ were taken to learn the jobs' runtime behavior.

The backfilling strategy always planed 330 successful jobs with an execution time gain of $2,500$ hours. Both overbooking strategies have at the beginning a sum of 690

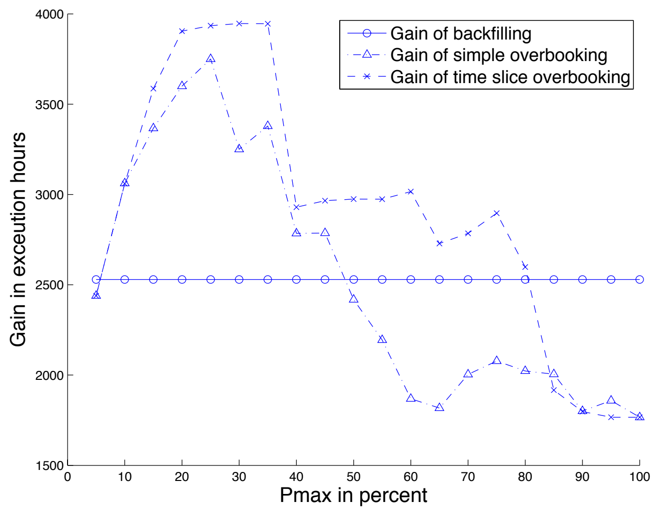

Fig. 13: BLUE: Sum of profit.

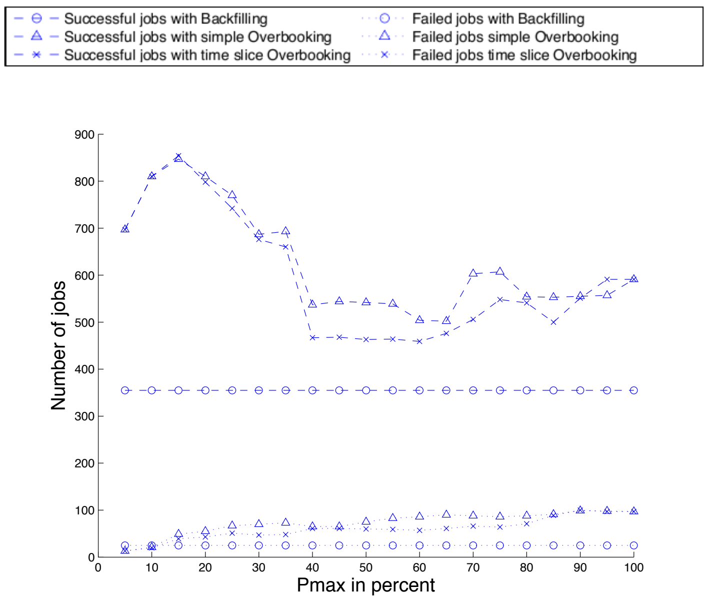

Fig. 14: BLUE: Shown is the number of successful and failed jobs.

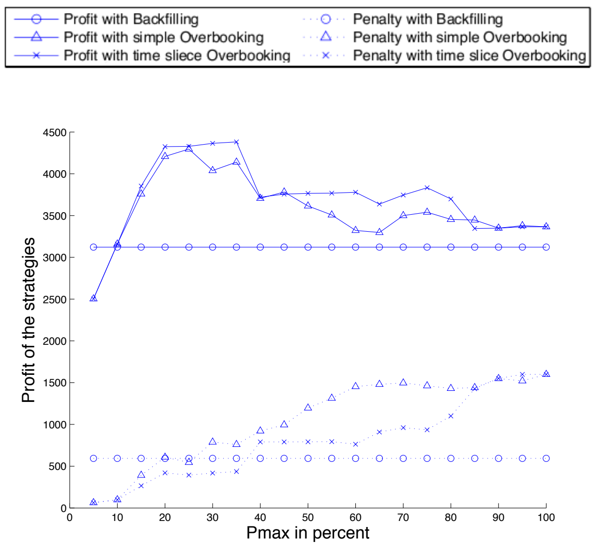

Fig. 15: BLUE: Shown is the profit and penalties of successful and failed jobs.

successful jobs and $2,500$ hours gain. Overbooking strongly depends on $P_{\max}$. For the first simulation runs with low $P_{\max}$ the profit and jobs improves with the increasing $P_{\max}$. From a $P_{\max}$ of 0.2 the sum of successful jobs falls due to less accepted but larger jobs and from a $P_{\max}$ of 0.4 also the gain is falling due to the continuous increasing amount of violated SLAs. Backfilling has more gain than simple overbooking from $P_{\max} = 0.5$ and is better than time slice overbooking from a $P_{\max} = 0.8$. For the BLUE trace and a $P_{\max} = 0.25$, the simple overbooking strategy can increase the gain by 50 % and the time slice overbooking can increase the gain by 55 % .

CTC Figures 16, 17, 18, and 19 show the results based on the CTC trace. For this simulation, 0.1 hours have been chosen as basic random value for the delay between the jobs. The CTC cluster had 430 nodes. From the $67,000$ jobs of CTC the first $59,000$ were taken to learn the jobs' runtime behavior.

The backfilling strategy always planed 840 jobs with $7,700$ hours execution time. Both overbooking strategies have at the beginning a sum of 930 successful jobs and also $7,700$ hours gain. The gain of the simple overbooking approach is maximal for $P_{\max} = 0.1$ and falls under the gain of backfilling from $P_{\max} = 0.15$. The time slice approach produces a maximal gain for $P_{\max} = 0.15$ and is falls under the backfillings' gain from a $P_{\max} = 0.3$. For the CTC trace and a $P_{\max} = 0.1$, the simple and time slice overbooking strategy can increase the gain by 4 %.

CTC with low load Figures 20 and 21 show the results based on the CTC trace with a low load. For this simulation, 1 hour has been chosen as basic random value for the de-

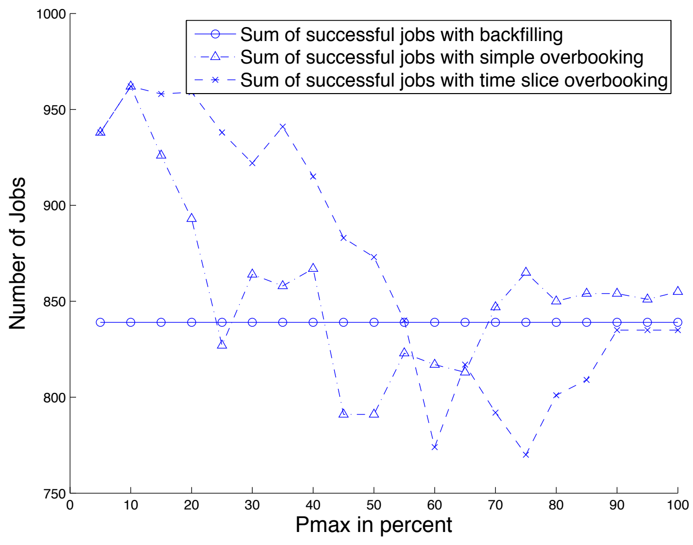

Fig. 16: CTC: Sum of successful jobs.

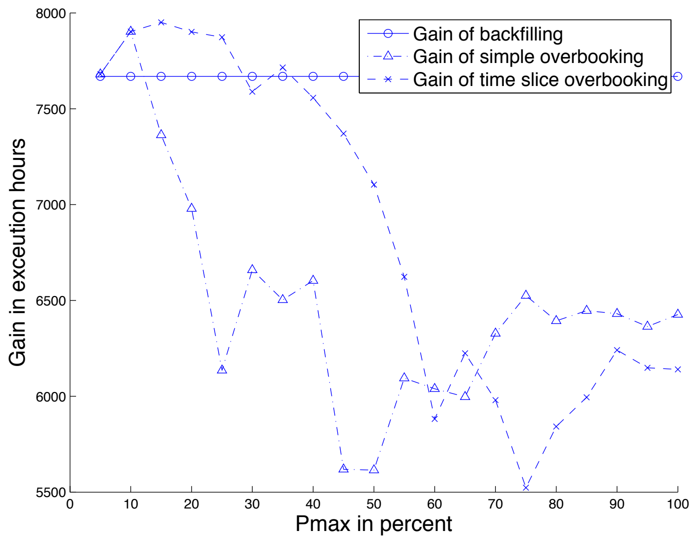

Fig. 17: CTC: Sum of profit.

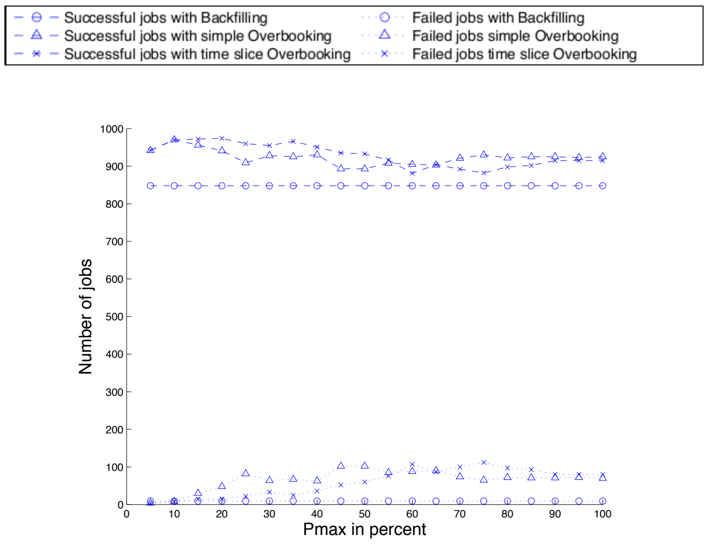

Fig. 18: CTC: Shown is the number of successful and failed jobs.

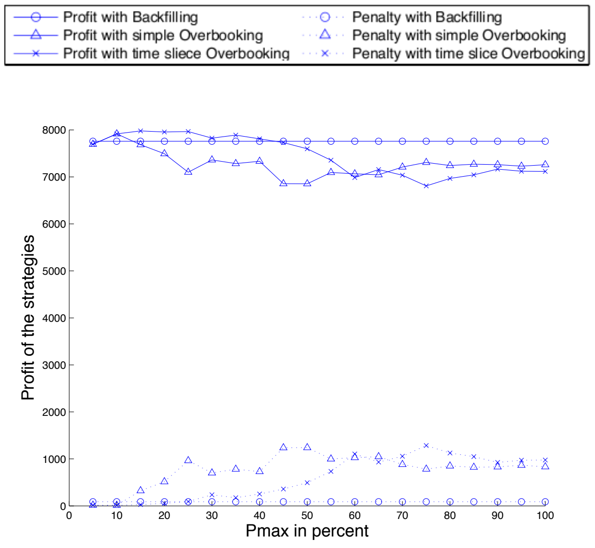

Fig. 19: CTC: Shown is the profit and penalties of successful and failed jobs.

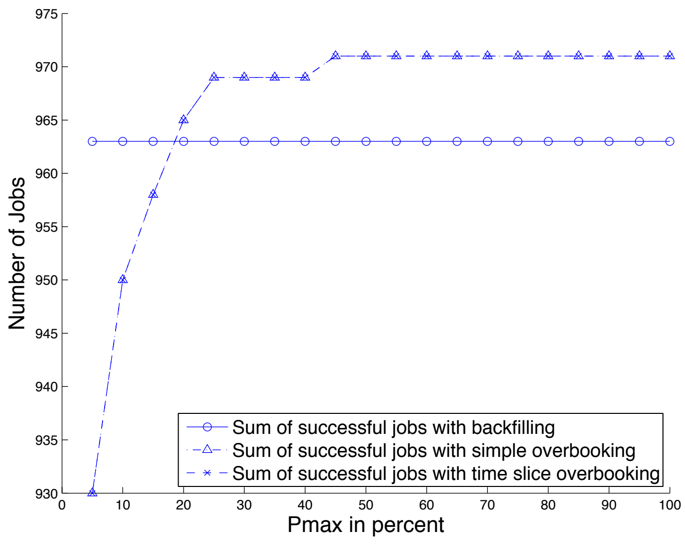

Fig. 20: CTC with low load: Sum of successful jobs.

lay between the jobs. The backfilling strategy always planed 978 jobs with $8,000$ hours execution time, thus nearly every incoming SLA. We skip the figures for profit/penalty and success/failed jobs here as nothing happens.

The overbooking approaches accept at the beginning less jobs than the backfilling approach. The reason for this behavior is that the risk of machine outages is also calculated in the PoF calculation; this means that for long running jobs including many cores there is a chance that the job might fail due to a machine outage. When the threshold is very low the machine does not accept some of this jobs even if the machine is empty. With a threshold of more than 0.1 the overbooking is similar to backfilling. As no more jobs can be accepted, even accepting high risk, the overbooking profit does not decrease.

6 Discussion

This work aims to increase a providers profit in a commercial scenario, by applying overbooking to resource planning. In the evaluation section we simulated the approach based on job traces from the parallel workload archive.

The simulation underlines that overbooking, carefully applied, provides a good opportunity for a grid provider to further increase its profit. For instance:

– With the SDSC traces and a threshold of $P_{\text{max}} = 0.1$ the profit is increased by 30 % compared to a conservative backfilling strategy.

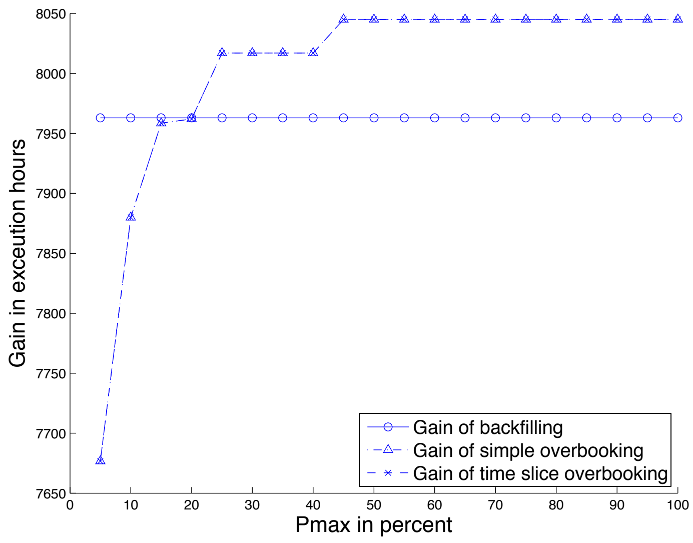

Fig. 21: CTC with low load: Sum of profit.

- With KTH and $P_{\text{max}} = 0.05$ the profit is increased by $23\,\%$.
- With BLUE and $P_{\text{max}} = 0.3$ the profit is increased by $55\,\%$.
- With CTC and a $P_{\text{max}} = 0.15$ the profit is increasing by $4\,\%$

In addition, the evaluation shows that the performance of the time slice overbooking is nearly always better than the simple overbooking. This shows that the quality of the underlying statistical analysis is paramount for a successful overbooking approach.

Where with some traces the profit is increasable by over $50\,\%$, for others only very little additional profit is possible. An improved statistical analysis might still allow to increase the profit, however when the jobs of users in a cluster system (nearly) always fully use then estimated runtime the application of overbooking is not profitable. In addition, the last evaluation shows that the application of overbooking makes sense in cluster systems with high load only.

7 Conclusion

This paper has motivated the idea of using overbooking to increase the ability to accept more SLAs in Grid, Cloud or HPC environments. As overbooking increases the probability of SLA violations, mechanisms for assessing the risk have been shown. The evaluation shows that the additional profit depends on the load of the system, the accuracy of the underlying runtime estimations, and the given real runtime distributions. The additional profit varies depending on the accuracy of the statistical analysis and the load of the system up to over $50\,\%$ of additional gain.

For future work it is interesting to determine if there are user and application specific distributions that would allow to increase the quality of the risk estimations for overbooking. Additionally we plan to examine the abilities of using virtualization techniques. This would allow to migrate jobs that took more time as their original gap length allows. If enough other resources are available at the end of a job's gap, the job is moved to these resources, thus an SLA violation might be prevented. Finally, we want to find a heuristic, which can estimate the PoF for a job without the CPU time-consuming convolution of PDF distributions.

References

1. Cirne, W., Berman, F.: A comprehensive model of the supercomputer workload. In: 4th Workshop on Workload Characterization, Citeseer (2001)
2. Hopper, E., Turton, B.: A review of the application of meta-heuristic algorithms to 2D strip packing problems. Artificial Intelligence Review **16**(4) (2001) 257–300
3. Berkey, J., Wang, P.: Two-dimensional finite bin-packing algorithms. Journal of the Operational Research Society (1987) 423–429
4. Ntene, N., van Vuuren, J.: A survey and comparison of level heuristics for the 2D oriented strip packing problem. Discrete Optimization (2006)
5. Baker, B., Schwarz, J.: Shelf algorithms for two-dimensional packing problems. SIAM Journal on Computing **12** (1983) 508
6. Feitelson, D., Jette, M.: Improved utilization and responsiveness with gang scheduling. Proceedings of the Job Scheduling Strategies for Parallel Processing: IPPS'97 Workshop, Geneva, Switzerland, April 5 (1997)
7. Feitelson, D., Weil, A.: Utilization and predictability in scheduling the ibm sp2 with backfilling. Proceedings of the 12th International Parallel Processing Symposium (Jan 1998)
8. 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
9. Zotkin, D., Keleher, P.: Job-length estimation and performance in backfilling schedulers. In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC) (Jan 1999)
10. Tsafrir, D., Feitelson, D.: The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. Proceedings of the IEEE International Symposium on Workload Characterization (2006)
11. Gibbons, R.: A historical application profiler for use by parallel schedulers. Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP): IPPS'97 Workshop (1997)
12. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP) (Jan 1998)
13. Tsafrir, D., Etsion, Y., Feitelson, D.: Modeling user runtime estimates. Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP) (2005)
14. Tsafrir, D., Etsion, Y., Feitelson, D.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems (TPDS) (2007) 789–803
15. Liberman, V., Yechiali, U.: On the hotel overbooking problem-an inventory system with stochastic cancellations. Management Science **24**(11) (1978) 1117–1126
16. Subramanian, J., Jr, S.S., Lautenbacher, C.: Airline yield management with overbooking, cancellations, and no-shows. Transportation Science **33**(2) (1999) 147–167

17. Rothstein, M.: Or and the airline overbooking problem. Operations Research **33**(2) (1985) 237–248
18. Urgaonkar, B., Shenoy, P.J., Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms. In: Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI). (2002)
19. 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 (2004)
20. Hovestadt, M., O.Kao, Keller, A., Streit, A.: Scheduling in hpc resource management systems: Queuing vs. planning. Proceedings of the Job Scheduling Strategies for Parallel Processing (JSSPP) (2003)
21. Siddiqui, M., Villazón, A., Fahringer, T.: Grid allocation and reservation - grid capacity planning with negotiation-based advance reservation for optimized qos. In: Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing. (2006) 103
22. Chen, M., Wu, Y., Yang, G., Liu, X.: Efficiently rationing resources for grid and p2p computing. In: Proceedings of the IFIP International Conference Network and Parallel Computing (NPC). (2004) 133–136
23. Sulistio, A., Kim, K.H., Buyya, R.: Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. In: Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid). (2008) 267–276
24. Nissimov, A., Feitelson, D.: Probabilistic backfilling. JSSPP 2007 (Jan 2007)
25. Birkenheuer, G., Hovestadt, M., Kao, O., Voss, K.: Overbooking in planning based scheduling systems. In: Proceedings of the 2008 International Conference on Grid Computing and Applications (GCA), Las Vegas, Nevada, USA (may 2008)
26. Birkenheuer, G., Brinkmann, A., Karl, H.: The gain of overbooking. In: Proceedings of the 14th Workshops on Job Scheduling Strategies for Parallel Processing (JSSPP), Rome, Italy (may 2009)
27. Schroeder, B., Gibson, G.: A large-scale study of failures in high-performance computing systems. In: Proc. of the 2006 international Conference on Dependable Systems and Networks (DSN06), Citeseer (2006)
28. Sahoo, R., Squillante, M., Sivasubramaniam, A., Zhang, Y.: Failure data analysis of a large-scale heterogeneous server environment. In: Dependable Systems and Networks, 2004 International Conference on. (2004) 772–781
29. Birkenheuer, G., Djemame, K., Gourlay, I., Hovestadt, M., Kao, O., Padgett, J., Voss., K.: Introducing risk management into the grid. In: in Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (e-Science'06) Amsterdam, Netherlands: IEEE Computer Society: p. 28. (2006)
30. Djemame, K., Padgett, J., Gourlay, I., Voss, K., Battre, D., Kao, O.: Economically enhanced risk-aware grid sla management. In: in Proceedings of eChallenges e-2008 Conference, Stockolm, Sweden. (2008)
31. Iosup, A., Jan, M., Sonmez, O., Epema, D.: On the dynamic resource availability in grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, IEEE Computer Society (2007) 26–33
32. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. Lecture Notes in Computer Science **3648** (2005) 432