VRM: A Failure-Aware Grid Resource Management System

Lars-Olof Burchard, Hans-Ulrich Heiss,
Barry Linnert, Joerg Schneider
Technische Universitaet Berlin, GERMANY
{baron,heiss,linnert,komm}@cs.tu-berlin.de

Cesar A. F. De Rose
PUCRS, Porto Alegre, BRASIL
derose@inf.pucrs.br

Keywords: Grid Computing, Failure Recovery, Advance Reservation

Reference to this paper should be made as follows: L.-O. Burchard, C. A. F. De Rose, H.-U. Heiss, B. Linnert, J. Schneider (xxxx) 'VRM: A Failure-Aware Grid Resource Management System', *Int. J. High Performance Computing and Networking*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Lars-Olof Burchard received his diploma in computer science from Paderborn University, Germany, in 1999. From October 1999 to February 2001, he was a member of the research staff and PhD student at the Paderborn Center for Parallel Computing. In March 2001, he joined the communication and operating systems group at Berlin University of Technology, Germany, where he received his doctorate degree in August 2004. His research interests include distributed multimedia systems, resource management in computer networks and Grid computing.

Cesar De Rose is an Associate Professor in the Computer Science Department at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil. His primary research interests are parallel and distributed computing and parallel architectures. He is currently conducting research on a variety of topics applied to clusters and Grids, including resource management, resource monitoring and distributed allocation strategies. Dr. De Rose received his doctoral degree in Computer Science from the University Karlsruhe, Germany, in 1998. He currently leads the Research Center in High Performance Computing (CPAD - PUCRS/HP) at PUCRS.

Hans-Ulrich Heiss received his diploma and doctorate degrees in computer science from the University of Karlsruhe (Germany) in 1979 and 1987, respectively. 1988-1989 he was a post-doc fellow at IBM T.J. Watson Research in Yorktown Heights (NY), and in 1990 a visiting professor at the University of Helsinki (Finland). After appointments at the universities in Ilmenau and Paderborn (both Germany) he has been a full professor for communication and operating systems at the Berlin University of Technology since 2001. His interests include operating systems, distributed systems, Grid computing, resource management, self-organization, and performance evaluation.

Barry Linnert received his diploma in computer science from the Berlin University of Technology, Germany, in 2000. He works as a research assistant at the communication and operating systems group at the Berlin University of Technology. His interests include operating systems, high performance computing, cluster and Grid computing.

Joerg Schneider received his diploma in computer science from the Berlin University of Technology (Germany) in 2004. His research interests include resource managment in the Grid, complex co-allocations and especially, Grid workflows. He is currently an research assistant at the communication and operating systems group at the Berlin University of Technology.

1 Introduction

Currently, Grid research moves its focus from the basic infrastructure that enables the allocation of resources in dynamic and distributed environments in a transparent way to more advanced management systems that accept and process complex jobs and workflows consisting of numerous sub-tasks and, e.g., provide guarantees for the completion of such jobs. In this context, the introduction of service level agreements (SLA) provides flexible mechanisms for agreeing on the quality-of-service (QoS) provided by various resources, including mechanisms for negotiating SLAs (1). The introduction of SLAs envolves prices for resource usage and also implies fines that must be paid when the assured QoS is not met. Depending on the scenario, this may be, e.g., a missed deadline for the completion of a sub-job in a workflow. Consequently, the definition of SLAs demands for control over each job and its required resources at any stage of the job's life-time from the request negotiation to the completion.

To achive this level of control and reliability next generation Grid management systems have to rely on advance reservation mechnisms. Advance reservations are a way of allocating resources in distributed systems before the resources are actually required, similar to flight or hotel booking. This provides many advantages, such as improved admission probability for sufficiently early reservations and reliable planning for clients and operators of the resources. Grid computing in particular uses advance reservations, which besides reliability of planning simplifies the co-allocation of very different resources and resource types in a coordinated manner. For example, the resource management integrated in the Globus toolkit (2) provides means for advance reservations on top of various local resource management systems.

An example for a resource management framework covering these aspects is the *Virtual Resource Manager* (VRM) architecture described in (3). This planning based Grid management system uses advance reservation mechanisms not only to provide improved admission probability and reliability, but to introduce support for workflows of Grid jobs, too. The functionalities coming with the planning based approach distinguishes the VRM from batch job based management systems.

A typical example for a complex workflow in a Grid is depicted in Fig. 1. The workflow processed in the distributed environment consists of five sub-tasks which are executed one after another in order to produce the final result, in this case the visualization of the data. This includes network transmissions as well as computations on two cluster computers.

One important aspect in this context is the behavior of the management system in case of failures. While current research mainly focused on recovery mechanisms for those jobs that are already active, in advance reservation environments it is also necessary to examine the impact of failures to admitted but not yet started jobs or sub-jobs. In contrast to the sophisticated and difficult mechanisms

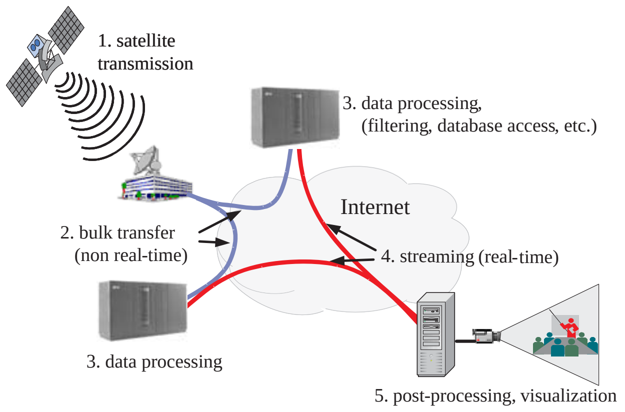
1. satellite transmission
3. data processing, (filtering, database access, etc.)
Internet
2. bulk transfer (non real-time)
4. streaming (real-time)
3. data processing
5. post-processing, visualization

Figure 1: Example: Grid application with time-dependent tasks.

needed to deal with failures for running jobs, e.g., checkpointing and migration mechanisms, jobs not yet started can be dealt with in a transparent manner by remapping those affected jobs to alternative resources.

As previous work in this area showed (4), it is necessary to deal with these affected but not yet active jobs. Consequently, we enhanced the VRM as our prototype for next generation Grid management systems with such failure recovery functionalities.

In this paper, we present the novel, load-based approach that has been developed for our VRM framework and has the advantage of adapting to the actual load situation and remapping affected jobs accordingly. The adaptiveness is an important feature as estimations of the failure duration are unreliable.

The remainder of this document is organized as follows: firstly, the general problem is described, including the properties of the advance reservation environment and the necessary assumptions and conditions to apply our approach. Based on these general remarks, the VRM framework is introduced as it is the target application for the failure recovery mechanism. Following that, the load-based approach for remapping jobs is presented. In Sec. 5, the strategies are evaluated using extensive simulations, showing the superiority of our approach compared to estimations. Before the paper is concluded with some final remarks, related work important for this paper is outlined.

2 Problem Definition

In this section, the general properties of advance reservation systems are described as well as different aspects related to the problem of remapping jobs in case of failures.

Although the VRM architecture is capable of handling various kinds of resources at the same time, this work focuses on computing resources. Comparable kinds of resources, e.g., storage space or bandwidth, can be handled by the mechanism presented in this document as well.

2.1 Properties of the Advance Reservation Environment

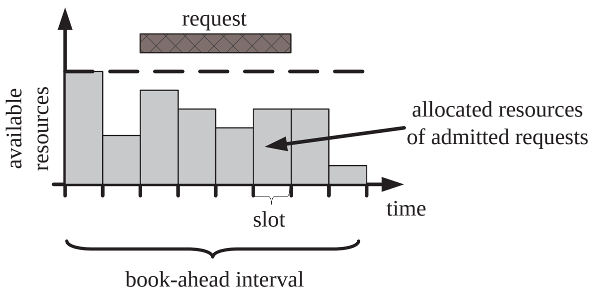

Figure 2: Advance reservations: status about future utilization

Advance reservations are requests for a certain amount of resources during a specified period of time. In general, a reservation can be made for a fixed period of time in the future, called *book-ahead interval* (t_{BA}, see Fig. 2). The book-ahead interval is divided into *slots* of fixed size, e.g., minutes, and reservations can be issued for a consecutive number of slots (2). The time between issuing a request and the start time of the request is called *reservation time* r. The finishing time or the duration for a given request must be specified to ensure reliable admission control, i.e., to determine whether or not sufficient resources can be guaranteed for the requested period. As depicted in Fig. 2, this approach requires to keep the status of each resource, i.e., information about future requests which are already admitted, for the whole book-ahead interval.

In this environment, failure recovery not only has to handle already active jobs, but also those which are admitted but not yet started, so-called *inactive jobs*. This means that the affected inactive jobs have to be *remapped in advance* to another matching resource. As the timing parameters start and stop time were specified during the admission, e.g., as part of a service level agreement (SLA), jobs can only be moved to another resource but not shifted in the temporal dimension.

The general benefits of remapping in advance are shown in (4) where estimations of the failure duration were used. As the end of the failure is usually unknown, it is not easy to decide which active jobs on the resource have to be taken into account for remapping. This paper presents a downtime independent approach to decide this question basing on the actual load situation.

2.2 Implications of the Environment

In order to implement failure recovery mechanisms in advance reservation environments, it is necessary to consider the types of available resources, i.e., resource heterogeneity or homogeneity play an important role. Whenever identical hardware and software infrastructure is available, including a wide range of properties such as processor type, cache sizes, operating system version, or libraries, mapping an inactive job to another resource is relatively simple. It may be even possible to migrate a running job, e.g., with support from checkpointing mechanisms, as some systems provide libraries for that purpose (5). For many resource types, such as cluster systems or parallel computers, such functionality often lacks completely or has to be implemented explicitly by the application. Consequently, in the context of this study active jobs are considered to be not remappable. However, this assumption is not crucial for the usage of our approach or the success of the remapping strategy itself.

Besides migration of active jobs, in a heterogeneous environment even the task of remapping inactive jobs is much more complex and difficult. In order to remap inactive jobs to another resource, e.g., a cluster computer, the differences between the resources often have consequences on the run-time of the respective processes and hence, have to be considered. For example, mapping a job from a 2 GHz processor to a 1 GHz processor may increase the overall execution time. Supposed, the differences between the heterogenous resources can be quantified, e.g., using some kind of benchmarks, these different resources can serve as alternatives for remapping of jobs as well. But in such an environment, the migration of running jobs is even more difficult or often impossible.

The Grid environment, our approach is aimed at, integrates a huge amount of resources of the same and different kind. So, a wide range of applications can be supported, starting from simple non-distributed programs up to complex parallel jobs, e.g., parallel simulations and parallel discretization of differential equations, and even distributed applications. Although the Grid environment, as outlined in the following section, may be capable of supporting distributed applications requesting various different resources, such as several cluster systems and some network links between them, the focus of this presentation will be on singleside parallel or non-parallel long-run applications. This restriction comes only with the actual existing limitations of the Grid environment. The failure recovery approach by itself provides flexibility to handle such difficult tasks as remapping different resources requested by complex distributed applications. Because of the stated problems with multisite applications, we also assume in the following that jobs cannot be split among several resources.

Based on these considerations, we assume a homogeneous environment, at least in the stated way, with resources of compatible hardware and software infrastructure which allows jobs to be executed on any of the available resources.

An example for an actual application environment of our approach is given in the following section.

3 Grid Environment

The approach presented here has been developed in the context of the *Virtual Resource Manager* (VRM) described in (3). Since the VRM supports QoS by means of SLAs, failure recovery is an essential feature. To meet SLAs in terms of, e.g., guaranteed completion times for complex compute jobs, the VRM needs to have complete control over the resources and the jobs during run-time (run-time responsibility). In the following, we briefly describe the architecture of the VRM as the environment of our failure recovery mechanism.

The *Administrative Domain Controller* (ADC) constitutes the central management component of the VRM architecture (see Fig. 3). The ADC is responsible for establishing so called *Administrative Domains* (AD) which consist of a number of underlying local resources and their local *Resource Management Systems* (RMS). These management systems may control arbitrary types of resources, e.g., cluster systems, parallel computers, or networks and are connected to the ADC by *Active Interfaces*. Such an Active Interface is also available to connect the ADC to another ADC responsible for a subdomain. For example, to carry out a gradual information hiding strategy, each department could establish an Administrative Domain and connect these domains to an enclosing domain. Providing these features from SLA negotiation, information hiding and customization up to implementing virtual organisations by setting up hierarchical structures of ADC components the VRM meets next generation Grid requirements.

While the introduction of service level aggreements provides new quality of service functionalities, the support of SLAs comes with new challanges. Now, the demand for failure recovery mechanisms is not only for convenience reasons, but is based on financial implications. As fees have to be payed for a SLA that could not be met, all events affecting the fulfillment of the SLA have to be handled or have to be avoided. As failures are unpredictable in most of the cases, failure recovery has to be an essential element of all SLA-aware resource management systems.

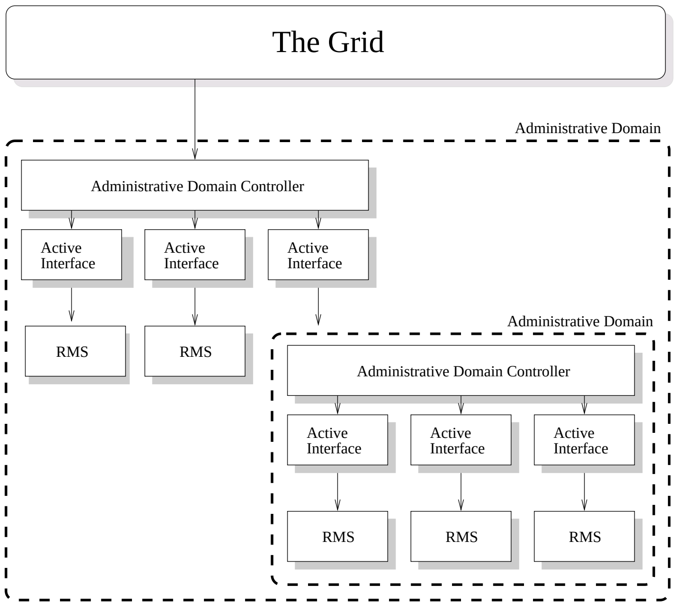

Figure 3: Hierarchical Administrative Domain Structure

The failure recovery mechanism proposed in this paper will be situated in the ADC component. Once any of the underlying compute resources fails partly or entirely, the jobs allocated to the failed resource have to be mapped onto alternative resources within the same domain according to our strategy. Partial failures, e.g., of one or more nodes within a cluster computer, may also require the recovery mechanism to act, as the total capacity of a resource may be exhausted. Within the VRM, inactive jobs can be transparently mapped without further notification to the users which is a major advantage compared to other Grid resource management systems such as Globus (2).

The framework of the VRM provides control over not only compute resources, but also any other resource required for the remapping of jobs, i.e., interconnection networks. For example, remapping a compute job with large amounts of input data requires a reasonable amount of time which must be considered and network transmissions must be planned accordingly which may include the reservation of network bandwidth. Allocation of network bandwidth is often available in dedicated networks for high performance Grid environments, e.g., *LambdaGrids* (6).

3.1 Using Batch Jobs to Improve the Performance

Batch jobs are frequently used also in advance reservation based planning systems (7). These jobs are not provided with fixed start and stop times but will be placed in a queue and are processed as soon as sufficient capacities are available. In our environment, such batch jobs can be placed on a failed resource behind the computed remapping interval as long as the failure persists. The resource management system then places batch jobs onto the currently failed resource and once the remapping interval is extended or the failure persists beyond their anticipated start time, the batch jobs are simply postponed. Otherwise, they can be started. The rationale behind this approach is that no timing guarantees are given for batch jobs and thus, postponing these jobs is less costly compared to terminating planned jobs.

3.2 Discovery of Additional Resources

In case no sufficient resources are available or the accumulation of free resources does not satisfy the additional requirements arising with the remapping of jobs to different computing systems, the VRM is capable of searching for free resources in a peer-to-peer like network of other VRMs. For this purpose, knowledge about the job properties and requirements is essential. The problem of discovering suitable resources in another administrative domain or computing environment can be handled using, e.g., ontology-driven resource discovery as presented in (8). Interfaces to other resource management systems, e.g., Globus, are also conceivable. As the VRM starts the search for alternative resources within its local domain, time-intensive search for suitable resources is only necessary when the local capacities are exhausted.

4 Load-Based Remapping Algorithm

In this section, our load-based remapping approach used in the VRM is presented. In case a failure of a specific resource, e.g., a cluster, is noticed, the management system has to face two different tasks to minimize the impact of the failure. First, the management system has to determine all jobs that have to be taken into account for remapping and, as a second step, these jobs have to be remapped to other resources.

The section starts with considerations how the second step is performed. In the following, the concept of a remapping interval is introduced for selecting the jobs to remap. Two metrics are presented to measure the performance of the remapping. Based on this metrics, we describe two algorithm behaving optimal in terms of one metric and a theoretical algorithms achieving optimal results in both metrics. These algorithms are used as a reference for our approach. After a short overview of the objectives for the remapping interval and some definitions, the algorithm to calculate the remapping interval is presented. The section is concluded with an analysis of the parameter used in the calculation of the remapping interval.

4.1 Remapping Strategies

All jobs chosen by one of the selection algorithms presented in the following have to be considered for remapping. Since usually the termination of at least some jobs may not be circumvented, it is necessary to carefully select the jobs to terminate according to some optimization criterion, e.g., the amount of successfully remapped jobs. Other optimization criteria are also conceivable although not targeted in this paper, e.g., minimizing fines to be paid for terminated jobs. In a distributed architecture, the task of selecting jobs to be remapped or terminated is even more complicated as no information about the amount of free resources at different locations is available.

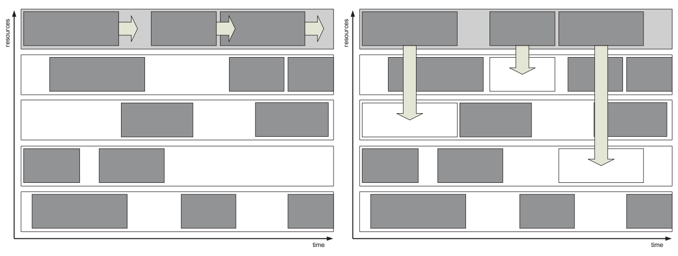

Figure 4: Without timing constraints, rescheduling allows the postponing of jobs (*left*) whereas remapping preserves the guaranteed start and stop times (*right*).

The reservations are fixed in time (see Fig. 2). Thus, it is not possible to shift the jobs to the future on the local system or alternative resources. This differs from scheduling approaches using time as a variable dimension (see Fig. 4). Thus, it is essential to find free resources within exactly the time interval specified during the reservation of the job which may not be possible for any of the jobs to be remapped.

Finding feasible alternative resources for a specified set of jobs is a classical bin packing problem, but in the spatial dimension not in the temporal one (9). Therefore, a simplified algorithm has to be used to solve the generally NP-hard problem online during the failure recovery.

Different strategies for remapping a set of affected jobs were evaluated in (4). As it was shown, the difference among the individual strategies is rather small. This means, it may be possible to select a strategy that matches additional constraints, such as preferring long book-ahead times leading to FCFS strategy as used in this paper.

4.2 Downtime-Independent Remapping

Former work showed, that using predictions of the actual downtime to perform the selction of the jobs to be remapped is critical, as the downtime cannot be accurately anticipated (4). In the case of underestimating the downtime, the number of terminated jobs increases drastically. The mechanism integrated in the VRM does not rely on predictions and is independent of the actual downtime. The general approach used in the VRM is to identify and remap jobs which are unlikely to be safely remapped at any later point in time. For that purpose, in each time slot throughout the duration of the failure a *remapping interval* is calculated. The length of this remapping interval, computed as described in the following sections, is *independent* of the actual downtime, which is usually unknown to the Grid resource management system. As described before, we do not deal with the mechanisms to recover already active jobs on the broken resource. So we assume, that either an adequate migration mechanism is available that handles all jobs that are already running, or–which is more likely–those running jobs are simply terminated.

All remaining jobs using the broken resource within the remapping interval are considered for remapping. The management system searches for an alternative resource with sufficient free capacities for every job. If the remapping is not successful, the job remains on the failed system until its start time since the failure may have ended until then, otherwise the job will be terminated. All other jobs, i.e. those outside the remapping interval, are not remapped, even if they are assigned to the currently broken resource.

The failed resource is blocked for the remapping interval only. This means, new reservations for time slots after the remapping interval may be booked on the currently broken resource and may be remapped later if necessary.

Fig. 5 is an illustration of the approach based on the remapping interval. For the first situation at t_1, this means that the dark gray jobs are handled by the failure recovery for active jobs (in the simplest case just terminated) as they were active when the failure occurred. The light gray

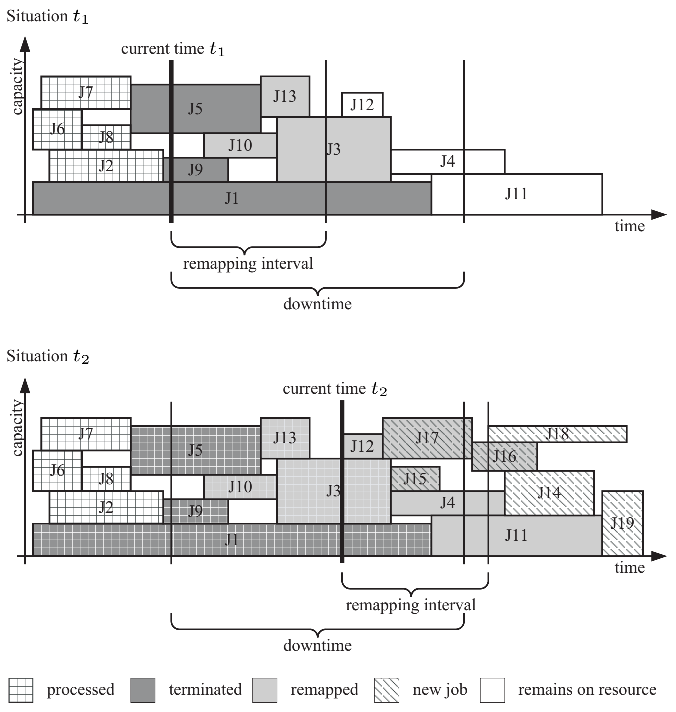

Figure 5: Examples for the usage of the remapping interval during a resource downtime for two different time slots t_1 (above) and t_2 (below).

jobs are considered for remapping and will be assigned to other resources if possible. The white jobs stay on the broken resource also if they reside within the–currently unknown–downtime. The time slots after the remapping interval are available for new jobs.

Due to the recalculation of the remapping interval in each of the following time slots, more and more jobs mapped to the broken resource will be remapped. Jobs submitted after the failure occurs are also remapped if they are within the remapping interval, as can be seen in the second situation at t_2 in Fig. 5. Since the downtime is unknown to the management system, also those jobs are remapped that will start after the downtime (J16). This remapping and the recalculation of the remapping interval is done until the resource is recovered.

4.3 Optimal Remapping Algorithms

For the evaluation of the remapping approach, two metrics are introduced, which reflect the success of the remapping on the one hand and the impact on the system on the other hand. For each metric an algorithm is given achieving optimal results in respect to the metric. This section concludes with a theoretical, optimal algorithm.

The main goal of remapping in advance is to save as many jobs as possible. In order to measure the performance in this respect the *termination ratio* is used, showing the number of unsuccessfully terminated jobs, i.e., as no remapping was possible due to resource shortage, compared to the number of jobs affected by the failure.

Remapping a job to another resource reduces the available capacity of the whole system, as these resources cannot be used for new jobs. While this behavior is not avoidable during the downtime of the resource, each job which is remapped after the resource recovers unnecessarily blocks additional resources. The number of jobs remapped after the end of the downtime is referred to as *overhead*. Reducing the overhead is crucial in SLA-aware environments, i.e., if fees must be paid for the allocation of alternative resources. In addition, failure recovery of sub-jobs in workflows may also affect subsequent sub-jobs – resulting in higher complexity of the remapping task – and should be kept to a minimum.

In order to achieve optimal remapping results, two basic algorithms are conceivable, each optimizing either of the two metrics overhead and termination ratio. In advance reservation environments the probability to successfully allocate resources increases with the reservation time and in the same way the probability to successfully remap with longer distances to the start time (4).

Thus, jobs must be remapped as soon as possible to achieve the optimal termination ratio. The first possible point in time is at failure detection. Hence, the first algorithm A_{all} remaps all jobs at the moment the failure is detected. The algorithm is easy to implement and can even be performed manually by the administrator. However, remapping this way leads to a high overhead.

The algorithm A_1 – achieving the minimal overhead of zero – remaps only those jobs starting in the next time slot as these jobs are surely affected by the failure. As described before, short reservation times lead to a higher probability of rejection and thus, to a high termination ratio. This algorithm can be implemented in an actual resource management system in a straightforward manner.

To achieve the optimal results with respect to both metrics an oracle is needed, which provides the exact length of the downtime in the moment the failure occurs ($A_{optimal}$). Using this information only the jobs within the downtime will be considered for remapping. As the remapping is done at the same time as in A_{all}, the optimal termination ratio is achieved. Using the exact downtime length, no job will be unnecessarily remapped and the overhead is optimal, too. However, it is not possible to use this algorithm in real systems as the downtime of a resource is usually unknown. Substituting the exact knowledge of the downtime by vague estimations leads to a significantly worse performance (4).

The previously described algorithms are used as a reference to assess the performance of our downtime independent approaches.

4.4 Objectives for the Remapping Interval

In order to design a well performing failure recovery strategy with respect to both the amount of terminated jobs and the overhead (see Sec. 4.3), it is essential to carefully choose the length of the remapping interval.

Similar termination ratio as with A_{all} can be theoretically achieved, if the length of the remapping interval is set such that each job is remapped just in the time slot t such that not enough free resources are available one time slot later $t+1$. However, this is not practicable since the calculation of the remapping interval is done for the whole system rather than on a per-job basis, and an accurate prediction of future availability of capacities is usually not possible. Generally, a higher probability for successful remapping is achieved using a longer remapping interval. It was shown in (10) that in advance reservation environments a "critical time" can be determined from where on the probability of successful admission increases significantly. In order to realize a low termination ratio, the remapping interval should be at least as long as the critical time and as long as possible.

The algorithm A_1 equals the proposed remapping process with a constant interval length of one. If the remapping interval is longer than one time slot, it is possible that more jobs than necessary are remapped to other resources. This happens even if the resource recovers during the next time slot. In this case, jobs are remapped which could run on the no longer broken resource and thus block free capacities on other resources. As the broken resource is also blocked for the duration of the remapping interval, no new jobs are mapped on the resource for this period. The combination of both effects leads to a reduction of the number of accepted jobs, especially after the failure of the resource has ended. Remapping a job causes extra costs, e.g., for network transmissions of the job and its related data, and these costs are another reason to reduce the number of jobs to remap. Hence, it is necessary to determine the remapping interval as short as possible to keep the overhead low.

As both requirements are contrary, a trade-off must be made between both metrics. In SLA-aware environments this trade-off is designated by the fine to pay for a terminated job and the prices to pay for the alternative resources used for the remapped jobs.

4.5 Definitions

Before we give a formal description of the calculation of the remapping interval some notations are introduced.

As we assume that all resources consist of comparable nodes, the resource usage of jobs, the load and the capacity of the resources is measured in number of nodes. Most values will be normalized by the *total number of nodes* $\bar{c}$ of all resources within the Grid. For other kinds of resources our algorithm is also applicable if an adequate measurement is given.

The load situation in the Grid of a time slot t_0 is described by the *load profile* $l_{t_0}(t)$, with $t \in \mathbb{N}$, which is defined as the normalized total number of allocated nodes on all resources for each future time slot $t_0 + t$.

The capacity lost due to the breakdown of a resource is denoted by c^*, while the load profile consisting only of jobs allocated to the broken resource is denoted by $l^*_{t_0}(t)$. Using this definition, the load profile of the not affected jobs can be defined as $\tilde{l}_{t_0}(t) = l_{t_0}(t) - l^*_{t_0}(t)$. Accordingly, the set of jobs admitted to the system is denoted by J, whereas the set of jobs allocated to the broken resource is denoted by J^*, and the set of unaffected jobs by $\tilde{J} = J \backslash J^*$.

The booking behavior is described by the average booking profile. The *booking profile* $b_{t_x}(t)$, with $t \in \mathbb{N}$, of a time slot t_x denotes the normalized number of booked nodes per future time slot $t_x + t$ of all incoming reservations during this time slot. The average computed over all previous booking profiles is denoted by the *average booking profile* $\bar{b}(t)$. These profiles are illustrated in Fig. 6.

4.6 Calculation of the Remapping Interval

For the calculation of the remapping interval i, the load situation in the Grid is taken into account.

For each time slot t_0 this calculation has to be repeated. Initially, t_0 is the moment the failure occurs, but later on the calculation is repeated for each time slot until the resource recovers.

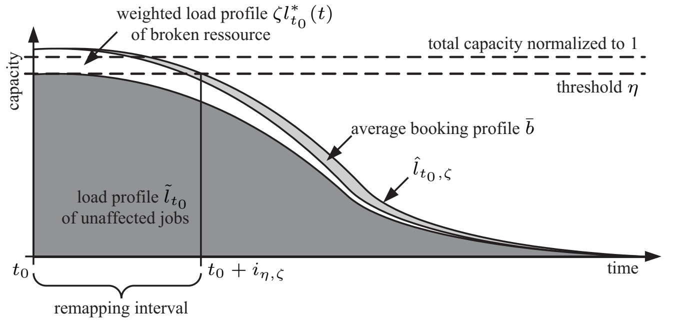
weighted load profile $\zeta l^*_{t_0}(t)$ of broken ressource
total capacity normalized to 1
capacity
threshold η
average booking profile $\bar{b}$
$\hat{l}_{t_0,\zeta}$
load profile $\tilde{l}_{t_0}$ of unaffected jobs
t_0
$t_0 + i_{\eta,\zeta}$
time
remapping interval

Figure 6: Determination of the remapping interval based on the combination of the weighted current load profile and the average booking profile.

First, a *weighted combined profile* $\hat{l}_{t_0,\zeta}(t)$ is created for the current time slot t_0 based on the current load profile of the unaffected jobs $\tilde{l}_{t_0}(t)$ combined with the weighted load profile of the affected jobs $\zeta \cdot l^*_{t_0}(t)$ (with $\zeta \geq 1$) and with the average booking profile $\bar{b}(t)$. This profile is some kind of expectation of the load after this time slot, as it sums up the currently booked load and the average incoming load during one time slot. Weighting the load of the affected jobs higher than the load of the unaffected jobs results in a narrower profile for small or underloaded resources and in a higher profile in case a heavily loaded, large resource is broken. A *threshold* η with $\eta \in [0, 1]$ is used to determine the remapping interval $[t_0, t_0 + i_{\eta,\zeta}(t_0)]$ on the base of the weighted combined profile. The remapping interval is defined by the time after which all values of the weighted combined profile are lower than η (see Fig. 6):

$$
\begin{aligned}
i_{\eta,\zeta}(t_0) \quad &:= \quad \begin{cases} \min\left\{ i \,|\, \forall t > i \,:\, \eta > \hat{l}_{t_0,\zeta}(t) \right\} & \text{if} > 1 \\ 1 & \text{otherwise} \end{cases} \\
\hat{l}_{t_0,\zeta}(t) \quad &:= \quad \tilde{l}_{t_0}(t) + \zeta l^*_{t_0}(t) + \bar{b}(t)
\end{aligned}
$$

The length of the remapping interval has the lower bound of 1 to ensure, that at least all jobs which are supposed to start within the current time slot are remapped; otherwise they will fail. How the parameters η and ζ must be set is analyzed in the following section. An algorithmic overview of this calculation is given in Fig. 7.

```
resource failed at $t_0$
// terminate all active jobs
for each job $j \in J^*$ do
   if $j$.start $< t_0$ do
      terminate $j$
while resource down do
   // calculate the profiles
   init $\hat{l}_{t_0}$
   for each job $j \in J^*$ do
      for each $t \in [j.\text{start}, j.\text{stop}]$ do
         $\hat{l}_{t_0}(t) += \zeta j.\text{numberOfNodes}$
   for each job $j \in \tilde{J}$ do
      for each $t \in [j.\text{start}, j.\text{stop}]$ do
         $\hat{l}_{t_0}(t) += j.\text{numberOfNodes}$
   add average booking profile $\bar{b}$ to profile $\hat{l}_{t_0}$
   // calculate $i$
   for $i = t_{ba}$ to 1 do
      if not $\hat{l}_{t_0}(i) < \eta$ break
   // remap jobs within remapping interval
   for each $j \in J^*$ do
      if $j$.start $< t_0 + i$ do
         remap $j$ if possible
   wait for next time slot: $t_0 = t_0 + 1$
```

Figure 7: Algorithmic overview on the calculation of the remapping interval and the downtime independent remapping.

4.7 Analysis of the Parameters

The calculation of the remapping interval depends on two parameters: η and ζ, with η being the threshold used to determine the remapping interval based on the combined profile, whereas ζ determines the weight of the affected load profile within the combined profile. Both parameters are used to balance the behavior and structure of the actual used Grid. In the ideal case both values are 1: The affected load will be included unweighted in the combined profile and only if the combined profile reach the total capacity (normalized to one) a remapping is needed.

In a realistic system there are a number of reasons why this ideal parameter setting does not work. The major reason is the fragmentation: Even if there are sufficient nodes available to handle the job, these may not belong to the same resource and hence, are not usable for the given job as in this work we do not consider multi-side jobs (see Sec. 2.2). There is also another kind of fragmentation in the temporal dimension. As the jobs use the needed number of nodes for a number of time slots, it is not only necessary to have the needed nodes on the same machine but also to have the same nodes for the whole job duration. Besides the fragmentation among resources and temporal dimension, the reliability of the prediction of incoming reservations plays a role. This prediction is based on the average booking behavior in previous time slots and hence cannot predict the arrival of a job size above average.

The effects of the fragmentation in the capacity dimension is mainly handled by the factor ζ. By weighting the load on the broken resources the probability is increased that enough nodes on the same resource are free. The factor ζ is upper bounded by the number of resources, e.g., the number of cluster computers building the Grid. In this case, so many nodes are claimed by the weighted load, that there is at least one resource with sufficient nodes available.

The threshold η deals with the problems of temporal fragmentation and the reliability of the prediction. It is used to decrease the assumed amount of available capacity in the Grid, as on the one hand, the Grid will never be used up to its full capacity and on the other hand the threshold provides a buffer for unpredictably large sized incoming reservations.

In Sec. 5, the behavior of our algorithm with different parameter sets is evaluated and it is shown how to choose η and ζ adequately.

5 Evaluation

In this section, the results of our load-based approach are outlined. In particular, the impact of various choices for η and ζ on the number of unsuccessfully terminated jobs as well as on the overall resource utilization is examined in a simulation environment.

5.1 Simulation Environment

The simulations were made assuming an infrastructure of several cluster and parallel computers with homogeneous nodes, i.e., each job is capable of running on any of the machines involved with exactly the same speed. If an adequate metric to compare the machines is given our algorithm could cover inhomogeneous Grids as well.

The simulations only serve the purpose of showing the general impact of failures and since according to (11) the actual distribution of job sizes, job durations etc. do not affect the general quality of the results generated even when using simple models, the simulations were made using a simple synthetic job and failure model.

The simulations were done using the simulation mode of the VRM framework. A simulated user reserved the resources in advance with the reservation time being exponentially distributed with a mean of 100 slots. The duration of the jobs were uniformly distributed in the interval $[250, 750]$ and each job demanded for 2^k nodes with k uniformly distributed in the interval $[1, 8]$.

Furthermore, a central administrative domain controller (ADC) instance was simulated controlling an infrastructure that consisted of different parallel computers with varying number of compute nodes. In total there were eight machines with 512, 256, 256, 128, 128, 96, 32, and 32 nodes. Obviously, some jobs can only be executed on the larger machines. For each resource a local resource management system capable to support advance reservations was simulated together with an active interface (AI) communicating with the simulated ADC. In our model, all jobs were submitted to the ADC and no additional jobs were submitted to the local resource management systems.

An additional component simulated the failures to occur periodically every 1500 slots with a failure duration of always 500 slots. The resource to break down completely was chosen randomly among the 8 machines. To simulate varying load situations we chose different values for the average reservation time r. Each simulation run had a duration of 20,000 slots and the simulations were repeated until a sufficiently small confidence interval (within $\pm 5\%$ of the mean with 95% confidence) was reached.

In order to assess the performance of the load-based approach, two metrics were chosen that reflect both the amount of jobs that were affected but could not be successfully remapped onto alternative resources and the number of jobs that had to be rejected because resources are blocked due to the failure recovery. The first metric is the *termination ratio*, which is defined as follows:

$$\text{termination ratio} := \frac{|A^*|}{|A|},$$

with A being the set of affected jobs and $A^* \subset A \subset J^*$ (see Sec. 4.5) being the set of unsuccessfully terminated jobs. To measure the overhead a second metric called *request blocking ratio* is defined as

$$\text{request blocking ratio} := \frac{|R|}{|S|},$$

with S denoting the set of all submitted requests and R denoting the rejected requests.

As described in previous sections, for the simulations we assume that running jobs cannot be migrated which is the usual case in a high performance computing scenario. However, our model is general enough to cover also the migration of running jobs, in particular this is discussed in Sec. 5.4.

5.2 Performance of the Load-Based Remapping Approach

In Fig. 8, the termination ratio is depicted for different choices of η and ζ. In general, it can be seen that the impact of η is relatively high, whereas ζ can only be used for fine-tuning. In this setting, the best results are obtained for $\zeta = 2$ with $\eta = 0.775$. As ζ is related to the fragmentation caused by resource boundaries, using larger values of ζ does not improve the overall performance. Consequently, the following examinations were conducted using $\zeta = 2$.

In Fig. 9, the termination ratio of the load-based remapping approach is depicted only in dependence of η. The

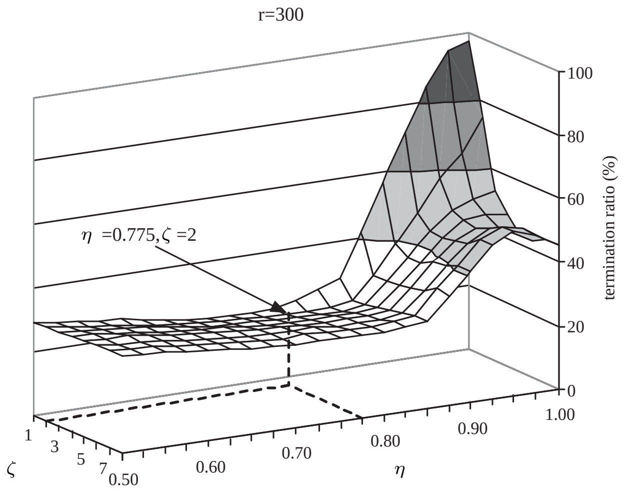

Figure 8: Termination ratio for different choices of η and ζ ($r = 300$).

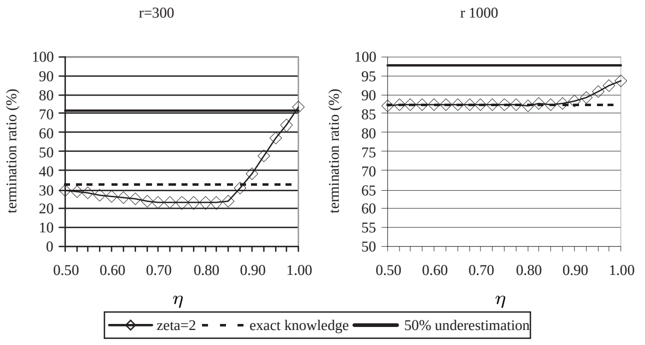

Figure 9: Termination ratio for different average reservation times.

different choices of the *reservation time* r were selected to create load situations where the average reservation time is shorter ($r = 300$) or longer ($r = 1000$) than the failure duration (500). To compare our novel approach with the estimations done in (4) as well as with the reference algorithms defined in Sec. 4.3, in Fig. 9 the termination ratio using the exact failure duration to identify the jobs to be remapped (which is a priori unknown, $A_{optimal}$) and an underestimation of the failure duration by 50% are depicted.

It can be observed, that for $r = 300$, the termination ratio shows a minimum at $\eta \approx 0.8$ and from thereon rises. The curve for $r = 1000$ reaches its minimum earlier and then remains stable. As already described in (4), underestimations of the failure duration result in a considerable increase of the termination ratio. The reason that the exact knowledge performs worse than the load-based approach results from the fact that the failed resource cannot be locked for the whole duration of the failure using the load-based approach. Hence, additional jobs which must be remapped may be placed onto the resource. When this remapping is successful, it contributes positively to the termination ratio. In contrast, when a resource is locked at the occurrence of the failure for the entire failure time, such jobs will never be placed onto the failed resource.

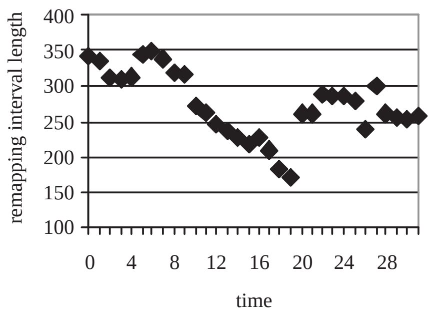

Figure 10: Snapshot of the remapping interval lengths computed during a single failure ($\eta = 0.8$, $r = 300$).

The self-adaptiveness of our approach can be seen more clearly when investigating the development of the actual remapping interval lengths during a single failure. This is depicted in Fig. 10. The diagram shows the length of the computed remapping intervals during a single failure for a sample period of 32 slots using $\eta = 0.8$.

5.3 Impact on the Amount of Accepted Jobs and Utilization

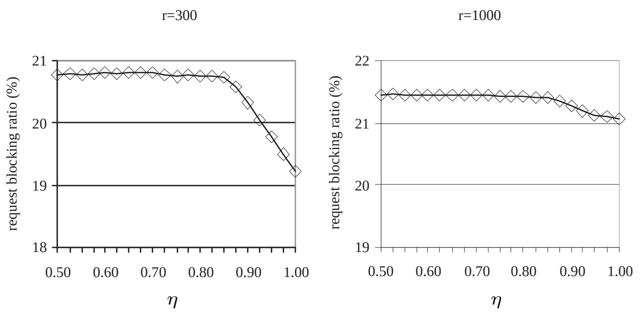

Figure 11: Request blocking ratio for $r = 300$ and $r = 1000$.

The impact of our approach on the amount of rejected jobs can be observed in Fig. 11: The amount of rejected jobs decreases with growing η. Therefore, the request blocking ratio behaves opposite to the termination ratio. Failure situations are considered as exceptions from the normal operation and hence the impact on the overall request blocking ratio is limited. In particular, selecting a small value for η does not impact the request blocking ratio and hence, selecting $\eta \leq 0.6$, as also indicated in the previous section, can be considered as a reasonable choice.

5.4 Comparison with Job Migrations

Although not assumed for the previous examinations, migration of running jobs is an interesting feature, allowing the management system to provide an almost completely transparent failure recovery. In the most extreme case, jobs may be repeatedly migrated throughout their lifetime. One possibility to support job migration is checkpointing. In this case, the failure recovery also needs to take care of the transfer of the checkpointed data to the backup resource and of the correct restart of the job at the new location. Moreover, since checkpointing is only performed at certain intervals, the required rollback to the last checkpoint increases the run-time of the job.

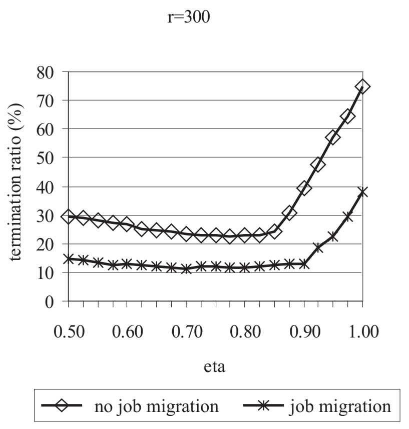

Figure 12: Performance benefit using checkpointing and migration

In Fig. 12, the possible benefit of checkpointing and job migration is outlined. The diagram shows the common approach, i.e., jobs can only be transferred as a whole and active jobs cannot be remapped. In contrast, providing means for migration of running jobs has a significant benefit as the termination ratio can be significantly reduced, i.e., around 50% more jobs can be successfully remapped to other resources. The diagram shows an idealized and unrealistic situation where data transfer over the network and rollbacks to previous checkpoints do not require time. However, the potential of using job migrations is apparent.

6 Related Work

Advance reservations are an important allocation strategy, widely used, e.g., in Grid toolkits such as Globus (12), as they provide simple means for planning of resources and in particular co-allocations of different resources. In (13), several algorithms for supporting advanced reservation of resources in supercomputing scheduling systems are proposed and evaluated. It was shown that the best performance is achieved when applications can be terminated and restarted, backfilling is performed, and relatively accurate run-time predictions are used. In (14) the concept of laxity is used in the reservation window of an advanced reservation to improve the scheduling performance. The paper proposes and analyses an algorithm for the scheduling of advanced reservations with laxities.

Besides flexible and easy support for co-allocations, e.g., in case complex workflows need to be processed, advance reservations also have other advantages such as an increased admission probability when reserving sufficiently early, and reliable planning for users and operators. Support for advance reservations has been integrated into several management systems for distributed and parallel computing (2; 15). In (3), advance reservations have been identified as essential for a number of higher level services, such as SLAs. To implement end-to-end quality of service (QoS) guarantees in emerging network-based applications, (16) proposes the Globus Architecture for Reservation and Allocation (GARA) to address dynamic discovery and advance reservation of resources that will often be heterogeneous in type and implementation and independently controlled and administered.

Failure recovery mechanisms are also particularly important in the context of Grid computing, as the distributed nature of the environment requires more sophisticated mechanisms than needed in a setting with only few resources that can be handled by a central management system.

In general, failure detection and recovery mechanisms focus on the requirements to deal with applications that are already active. The Globus heartbeat monitor HBM (2) provides mechanisms to notify applications or users of failures occurring on the used resources. The recovery mechanisms described in this paper can be initiated by the failure detection of the HBM. In (17), a framework for handling failures in Grid environments was presented, based on the workflow structure. The framework allows users to select different failure recovery mechanisms, such as simply restarting jobs, or - more sophisticated - checkpointing and migration to other resources if supported by the application to be recovered. The different recovery mechanisms are discussed and compared. However, the framework can only be used for the recovery of active applications, inactive applications that are already assigned to resources but not yet started are not taken into account. This is the same in (18) where a user-level software designed to provide automatic detection and restart of corrupted or early terminated tasks called ReGS is presented. ReGS allows the automatic detection of job dependencies through a task management language. We consider user-level solutions to failure recovery not as transparent and efficient as system level mechanisms.

In (4), the basic requirements and opportunities for failure recovery in planning based resource management systems were examined. In particular, it was shown that remapping of admitted but not yet active jobs is essential in order to reduce the number of unsuccessfully terminated jobs. It was also shown that the best results in terms of termination probability and overall resource utilization are achieved, when exact knowledge of the actual duration of a failure is available and any jobs commencing during this interval are remapped. However, estimations of the actual downtime are a questionable approach as these estimations are inherently unreliable and underestimations lead to a significantly higher termination ratio than possible with exact knowledge.

In (19), we extended the approach of remapping in advance from (4) and introduced the concept to repeatedly calculate the remapping interval. This approach is extended here again to be applicable in the Virtual Resource Manager (VRM)(3).

7 Conclusion and Future Work

In this paper, the novel load-based failure recovery strategy used in the VRM framework was presented. The mechanism is applicable in any environment where distributed resources must be managed and failures of the system are critical, e.g., SLAs are given for the correct and complete execution of a job. In particular, co-allocation environments such as Grids are target environments for our strategy.

The load-based algorithm is based on previous work on this field which showed, that estimations of the actual downtime of a resource have a particularly negative impact on the termination ratio. Consequently, our approach adapts to the actual load situation and determines a remapping interval accordingly, which diminishes the danger of underestimating failure durations as any job is remapped before it is actually endangered of being terminated. Simulations showed how to select appropriate values for the used parameters to gain the best performance of the failure recovery.

The strategy presented in this paper is generic, i.e., it can easily be applied to almost any resource type and any resource management system. This is particularly important for next generation Grid systems, which essentially need to support higher level quality-of-service guarantees, e.g., specified by SLAs, as in the context of the VRM (3).

Future work will deal with the possibility to integrate checkpointing and migration mechanisms into the load-based approach, which has the potential to dramatically increase the performance of the failure recovery mechanisms as presented before. This approach will allow to more efficiently utilize the available resources as gaps due to temporal fragmentation can be filled. Important issues in this context are the time required for job migration over a network. Moreover, a completely automated selection of both parameters will be developed simplifying the deployment of the system. However, as the simulations conducted for this paper indicate, a static choice is also reasonable and leads to good results.

REFERENCES

[1] Czajkowski, K., I. Foster, C. Kesselman, V. Sander,

and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in *8th Intl. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Edinburgh, Scotland, UK*, ser. Lecture Notes in Computer Science (LNCS), vol. 2537. Springer, January 2002, pp. 153–183.

[2] "The Globus Project," http://www.globus.org/.

[3] Burchard, L.-O., M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The Virtual Resource Manager: An Architecture for SLA-aware Resource Management," in *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid), Chicago, USA*, 2004.

[4] Burchard, L.-O. and B. Linnert, "Failure Recovery in Distributed Environments with Advance Reservation Management Systems," in *15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), Davis, USA*, ser. Lecture Notes in Computer Science (LNCS), vol. 3278. Springer, 2004, to appear.

[5] Litzkow, M., T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," University of Wisconsin - Madison Computer Sciences Department, Tech. Rep. UW-CS-TR-1346, April 1997.

[6] DeFanti, T., C. de Laat, J. Mambretti, K. Neggers, and B. S. Arnaud, "TransLight: A Global-Scale LambdaGrid for E-Science," *Communications of the ACM*, vol. 46, no. 11, pp. 34–41, November 2003.

[7] Keller, A., and A. Reinefeld, "Anatomy of a Resource Management System for HPC Clusters," in *Annual Review of Scalable Computing, vol. 3, Singapore University Press*, 2001, pp. 1–31.

[8] Heine, F., M. Hovestadt, and O. Kao, "Towards Ontology-Driven P2P Grid Resource Discovery," in *5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[9] Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[10] Wischik, D., and A. Greenberg, "Admission Control for Booking Ahead Shared Resources," in *IEEE INFOCOM, San Francisco, USA*, 1998, pp. 873–882.

[11] Lo, V., J. Mache, and K. Windisch, "A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling," in *4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, USA*, ser. Lecture Notes in Computer Science (LNCS), vol. 1459. Springer, 1998, pp. 25–46.

[12] Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," in *7th International Workshop on Quality of Service (IWQoS), London, UK*, 1999, pp. 27–36.

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

[14] U. Farooq, S. Majumdar, and E. Parsons, "Impact of laxity on scheduling with advance reservations in grids," in *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Sept. 2005, pp. 319 – 324.

[15] Snell, D., M. Clement, D. Jackson, and C. Gregory, "The Performance Impact of Advance Reservation Meta-scheduling," in *6th Workshop on Job Scheduling Strategies for Parallel Processing, Cancun, Mexiko*, ser. Lecture Notes in Computer Science (LNCS), vol. 1911. Springer, 2000, pp. 137–153.

[16] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Seventh International Workshop on Quality of Service (IWQoS)*, May 1999, pp. 27 – 36.

[17] Hwang, S. and C. Kesselman, "Grid Workflow: A Flexible Failure Handling Framework for the Grid," in *12th Intl. Symposium on High Performance Distributed computing (HPDC), Seattle, USA*. IEEE, 2003, pp. 126–138.

[18] J. Sanches, P. Veragas, I. Dutra, V. Costa, and C. Geyer, "Regs: User-level reliability in a grid environment," in *5th ACM/IEEE Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, Cardiff, UK, May 2005.

[19] L.-O. Burchard, B. Linnert, and J. Schneider, "A distributed load-based failure recovery mechanism for advance reservation environments," in *5th ACM/IEEE Intl. Symposium on Cluster Computing and the Grid (CCGrid)*, 2005.