A Distributed Load-Based Failure Recovery Mechanism for Advance Reservation Environments

Lars-Olof Burchard, Barry Linnert, Jörg Schneider
{baron,linnert,komm}@cs.tu-berlin.de
Technische Universitaet Berlin, GERMANY

***Abstract*— Resource reservations in advance are a mature concept for the allocation of various resources, particularly in grid environments. Common grid toolkits support advance reservations and assign jobs to resources at admission time. In such a distributed environment, it is necessary to develop carefully tailored failure recovery mechanisms that provide seamless transparent migration of jobs from one resource to another. As the migration of running jobs is difficult, an important issue in advance reservation, i.e., planning based, management infrastructures is to determine the duration of a failure in order to remap jobs that are already allocated to a currently failed resource but not yet active. As shown in previous work, underestimations of the failure duration and as a consequence the remapping of too few jobs results in an increased amount of job terminations. In order to overcome this drawback, in this paper we propose a load-based computation of the jobs to be remapped. A centralized and a distributed version of the strategy are presented, showing it is not necessary to have knowledge beyond the local allocation on the failed resource. The load-based strategy achieves to effectively remap jobs while avoiding - inevitably inaccurate - estimations of the failure duration.**

I. Introduction

The use of advance reservations in Grid computing environments has many advantages, especially for the co-allocation of various resources. Grid research moves its focus from the basic infrastructure that enables the allocation of resources in a dynamic and distributed environment 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 introduces prices for resource usage and also implies fines 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. An example for a resource management framework covering these aspects is the virtual resource manager architecture described in [2]. An application is depicted in Fig. 1.

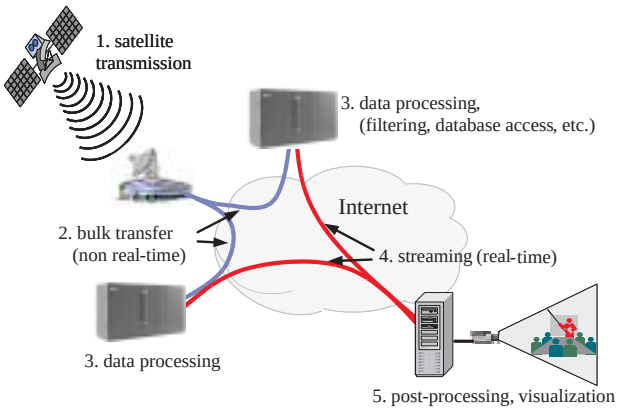

Fig. 1. Example: grid application with time-dependent tasks.

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 onto admitted but not yet started jobs or sub-jobs, i.e., parts of workflows. In contrast to the sophisticated and difficult mechanisms 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.

Based on previous work in this area, which showed it is necessary to deal with these affected but not yet active jobs, in this paper we present a novel, load-based approach that has the advantage of adapting to the actual load situation and remapping affected jobs accordingly. The adaptiveness is the main feature as it relieves the management system from introducing estimations which are unreliable and may result in considerably increased amount of terminated jobs.

We present a centralized and a distributed version of our basic load-based algorithm. As global knowledge about the load situation in the whole Grid may be inaccurate or generally not available, it is necessary to rely on locally available load information in order to remap jobs. The distributed version of the remapping strategy follows

this approach whereas the centralized version requires knowledge about the status of any of the resources in the Grid. It can be shown, that both versions efectively remap the affected flows and achieve similar performance than possible with exact knowledge of the failure durations.

The remainder of this document is organized as follows: firstly, related work important for this paper is outlined. After that, the problem is described, including the properties of the advance reservation environment and the necessary assumptions and conditions to apply our approach. Following that, the load-based approach for remapping jobs is presented. In Sec. V, the strategies are evaluated using extensive simulations, showing the effectiveness of our approaches. Before the paper is concluded with some final remarks, we present the application of the load-based approach in an actual resource management system.

II. Related Work

Planning based resource management systems are widely used in the area of grid computing. Advance reservations are an important allocation strategy, widely used, e.g., in grid toolkits such as Globus [3], as they provide simple means for planning of resources and in particular co-allocations of different resources. 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. In contrast to the synchronous usage of several different resources, where also queueing approaches are conceivable, advance reservations have a particular advantage when time-dependent co-allocation is necessary, as shown in Fig. 1. Support for advance reservations has been integrated into several management systems for distributed and parallel computing [4], [5]. In [2], advance reservations have been identified as essential for a number of higher level services, such as SLAs. The focus of this paper is on the requirements for dealing with failures and outages of resources that are reserved in advance.

In the context of grid computing, failure recovery mechanisms are particularly important 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 [4] 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 [6], a framework for handling failures in grid environments was presented, based on 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.

In [7], the basic requirements and opportunities for failure recovery in planning based resource management systems have been examined. In particular, it was shown that remapping admitted but not yet active jobs is essential in order to reduce the number of terminated jobs. It was also shown that when exact knowledge of the actual duration of a failure is available and any jobs commencing during this interval are remapped, the best results in terms of termination probability and overall resource utilization are achieved. 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 this paper, we extend the approach of remapping in advance from [7] and provide strategies that determine the remapping interval depending on the actual load on the available resources in the grid rather than on estimations which cannot be expected to be accurate.

III. Problem Definition

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

A. Properties of the Advance Reservation Environment

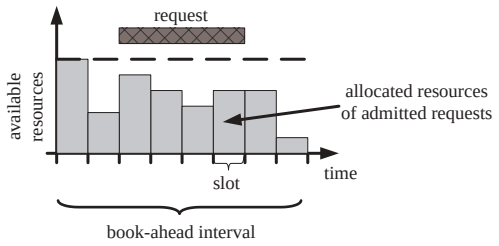

Fig. 2. Advance reservations: status about future utilization

Advance reservations are requests for a certain number 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* (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. The time between issuing a request and the start time of the request is called *reservation time* r. In contrast to immediate reservations which are usually made without specifying the duration, advance reservations require to define the stop time for a given request. This is required to perform reliably admission control, i.e., to determine whether 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 must not only handle 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, jobs can only be moved to another resource but not in the temporal dimension.

The benefits of remapping in advance are shown in [7]. As the end of the failure will be 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 based on the actual load situation.

B. Implications of the Environment

In order to implement failure recovery mechanisms in the advance reservation environment, 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, some systems provide libraries for that purpose [8].

However, in a heterogeneous environment the same task is more complex and difficult, even mapping inactive jobs to a different resource, e.g., cluster, may have consequences on the run-time of the respective processes and hence, must be considered. For example, mapping a job from a 2 GHz processor to a 1 GHz processor will increase the overall execution time. In such an environment, the migration of running jobs is even more difficult. Consequently, in the context of this study active jobs are considered to be not remappable. For many resource types, such as cluster systems or parallel computers, such functionality lacks completely or has to be implemented explicitly by the application. However, this assumption is not crucial for the usage of our approach or the success of the remapping strategy itself.

Furthermore, we consider a homogeneous environment with resources of at least very similar hardware and software infrastructure, such that jobs can be executed on any of the available resources. Because of the known problems with multi-site applications, we assume in the following that jobs cannot be split among several resources. An example for an actual application environment of our approach is given later in Sec. VI.

C. Distributed Environment

A Grid environment is usually seen as the composition of resources owned by different organizations. In such an environment it is unlikely to initiate a centralized control structure or to have all information on the system states available. Besides organizational reasons concerning information hiding and control autonomity also technical reasons to minimize the complexity state for a distributed implementation of such an environment.

In the distributed case the environment is divided into domains. Within each domain an autonomous control system is installed, which handles for example the admission control as well as the failure recovery described in this paper. The number of resources in the domain is as low as it is needed to deploy such a domain-wide control system, even one resource can build up a domain.

When a request is submitted to a domain, it will be mapped to a resource in the domain. In the case not enough resources are available in the domain for the requested time, the request will be distributed to other domains. This mechanism is realized in our application framework described in Sec. VI.

The admission control can be done using only domain specific information, how a failure recovery can be done using only the limited information available is analyzed in this paper.

IV. Load-Based Remapping Algorithm

In this section, our novel load-based remapping approach is presented. We show how the algorithm works in an environment with global knowledge and propose a modification of it which can be used in a distributed environment.

A. Downtime-Independent Remapping

As shown in [7], predictions of the actual downtime are critical, as the downtime cannot be accurately anticipated. Underestimations in this case lead to a drastically increased number of terminated jobs. In contrast to a prediction, the general approach employed in this paper 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 is *independent* of the actual downtime, which is unknown in the system.

As described before we do not deal with the mechanisms to recover already active jobs on the broken resource. So we assume, that all jobs which have been running are terminated as the failure occurs.

All jobs using the broken resource within the remapping interval are considered for remapping. In the case of global knowledge the central management system searches for another resource with sufficient free capacities for this 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 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 can be booked on the currently

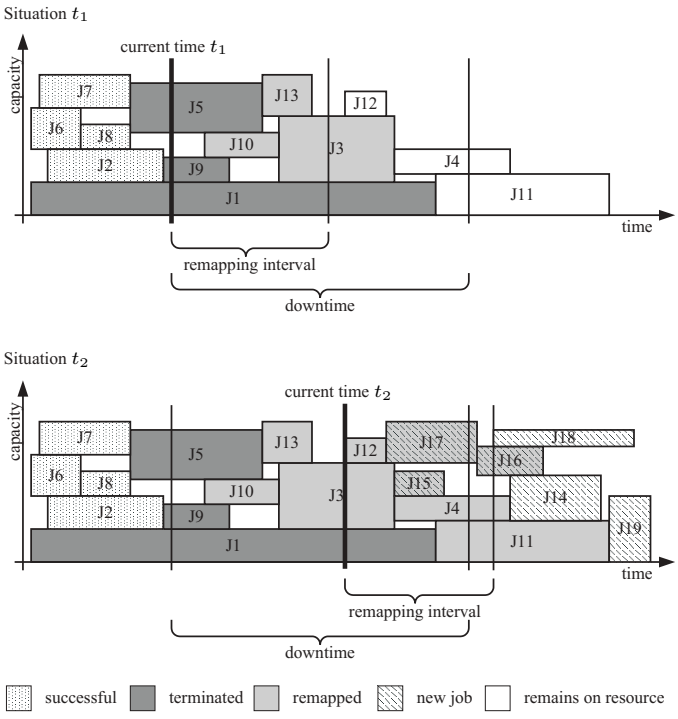

Fig. 3. Examples for the usage of the remapping interval during a resource downtime for two different time slots t_1 (*above*) and t_2 (*below*).

broken resource and are remapped later if necessary. The remapping approach is illustrated in Figure 3. In the first situation at t_1, the dark gray jobs are terminated as they were active when the failure occurred. The light gray jobs are remapped to other resources if possible. The white jobs stay on the broken resource also if they reside within the - currently unknown - downtime. The resource is available for new jobs starting after the remapping interval.

During the following time slots the remapping interval is recalculated and more and more jobs allocated on the broken resource are remapped. As can be seen in the second situation at t_2 in Fig. 3, jobs submitted after the failure occurred are also remapped if they are within the remapping interval. As the downtime is unknown to the management system, also those jobs supposed to start after the downtime, e.g., J16, are remapped. This remapping and the recalculation of the remapping interval is repeated each time slot until the resource is recovered.

B. Objectives for the Remapping Interval

The first requirement is that the remapping interval must be sufficiently long, such that any job affected by a failure can be safely remapped with high probability. In the optimal case, the length of the remapping interval for our load based approach is set such that each job is remapped just in the time slot before there are not enough free resources available in the next time slot. As the calculation of the remapping interval is done for the whole system and not on a per job basis and as an accurate prediction on future reservations is usually not available, a higher probability for successful remapping is achieved using a longer remapping interval.

If the remapping interval is longer than in the optimal case, the broken resource is blocked for a longer period and more jobs than necessary are remapped to other resources. This happens even if the resource will be online 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, for the time after the failure of the resource has ended. The remapping of 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 jobs to remap. Therefore, it is necessary to determine the remapping interval as short as possible.

C. 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* c of all resources within the whole grid. For other kinds of resources our algorithm is also applicable if an adequate metric is given.

The load situation within 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 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^*.

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 requested 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. 4.

D. Calculation of the Remapping Interval

The remapping interval i is calculated based on the load situation within the grid. An outline of this algorithm is given in Fig. 5.

The calculation is done for the current time slot t_0. This may be initially the moment the failure occurs, but for the repeated calculations of the remapping interval for each time slot during the failure, the same calculation applies.

First, a *combined profile* $\hat{l}_{t_0}(t)$ is created for the current time slot t_0 based on the current load profile of the booked jobs $l_{t_0}(t)$ combined with the average booking profile $\bar{b}(t)$. Adding these profiles gives an indication of the expected

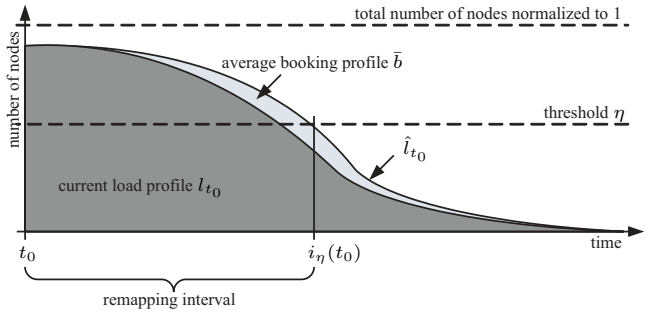

Fig. 4. Calculation of the remapping interval based on the combination of the current load profile and the average booking profile.

load after this time slot as it sums up the currently booked load and the average incoming load during one time slot. The remapping interval $[t_0, t_0 + i_\eta(t_0)]$ is determined using a *threshold* η with $\eta \in [0, 1]$. The remapping interval is defined by the time after which all values of the combined profile are lower than η (see Fig. 4):

$$i_\eta(t_0) := \begin{cases} \min\left\{ i \,|\, \forall t > i \,:\, \eta > \hat{l}_{t_0}(t) \right\} & \text{if } > 1 \\ 1 & \text{otherwise} \end{cases}$$

$$\hat{l}_{t_0}(t) := l_{t_0}(t) + \bar{b}(t)$$

The remapping interval must be at least 1 as all jobs which are supposed to start within the current time slot must be remapped or will fail.

The threshold η deals with the problems of 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.

E. Modification for a distributed environment

The algorithm is heavily based on knowledge about the global state of the grid. It uses not only information, which can be obtained on demand as the failure occurs like the current load profiles, it also needs a permanent access to the booking profiles to calculate a global average booking profile. In a distributed environment both kinds of information may not be available across domain boundaries.

The following modifications to the algorithm are proposed to reflect this limited access on information. In a distributed environment all profiles, the booking profile $\bar{b}(t)$ as well as the load profile $l_{t_0}(t)$, are calculated only using the jobs submitted to the local domain and are normalized by the number of the available resources within the domain to get a comparable value.

In case of a failure within the domain, a domain-specific remapping interval will be calculated. The jobs on the broken resource within the remapping interval will be tried to remap to another resource within the domain first. If

```
resource failed at t_0
// terminate all active jobs
for each job j ∈ J* do
  if j.start < t_0 do
    terminate j
while resource down do
  // calculate the profiles
  init l̂_{t_0}
  for each job j ∈ J do
    for each t ∈ [j.start, j.stop] do
      l̂_{t_0}(t) += j.size
  add average booking profile b̄ to profile l̂_{t_0}
  // calculate i
  for i = t_ba to 1 do
    if not l̂_{t_0}(i) < η break
  // remap jobs within remapping interval
  for each j ∈ J* do
    if j.start < t_0 + i do
      remap j if possible
  wait for next time slot: t_0 = t_0 + 1
```

Fig. 5. Schematic overview of the algorithm for the calculation of the remapping interval and the downtime independent remapping.

there are not enough resources free within the domain, the job will be submitted to another domain within the grid and if this was successful remapped to the newly assigned resources.

V. Evaluation

In this section, the results of our load-based approach are outlined. In particular, we show that the distributed approach achieves the same performance as the centralized one and does not suffer from overhead introduced by gathering status information from other resource which may even impossible in many cases.

A. Simulation Environment

The simulations were made assuming an infrastructure of several cluster and parallel computers with homogeneous node equipment, i.e., each job is capable of running on any of the machines involved.

The simulations only serve the purpose of showing the general impact of failures and since according to [9] the actual distribution of job sizes, job durations etc. do not impact the general quality of the results generated even when using simple models, the simulations were made using a simple synthetic job and failure model. Each job was assumed to be reserved in advance with the reservation time being exponentially distributed with a mean of 100 slots. Job durations were uniformly distributed in the interval $[250, 750]$ and each job demanded for a number of nodes being a power of 2, i.e., $2, 4, 8, \ldots, 256$ nodes with uniform distribution. The assumed hardware infrastructure 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.

In order to assess the performance of the load-based approach, two metrics were chosen. 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. IV-C) being the set of terminated jobs. In addition, the overhead of the respective remapping strategy is defined as

$$\text{remapping overhead} := \frac{|O|}{|A|},$$

where O denotes the set of jobs that were considered for remapping although not actually affected by the failure.

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 opportunity of migrating running jobs.

B. Performance of the Load-Based Remapping Strategies

As the actual load on the resources is crucial for the overall performance of the algorithms, the two extreme situations were simulated: the average global situation is much lower than on the failed resource and on the other hand the global load is higher than on the failed resource. This was modeled by adjusting the average reservation time for the failed resource (r_{local}) and the remaining resources (r_{global}) as depicted in Fig. 6. Furthermore, three failure durations were selected, such that the failure lasts shorter or longer than the average reservation time.

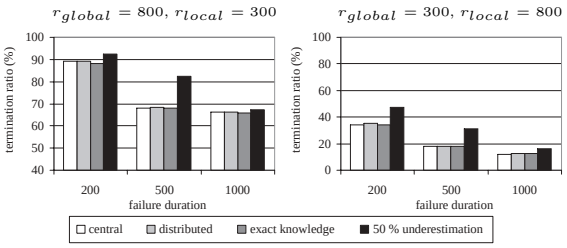

Fig. 6. Termination ratio of the centralized and distributed approaches compared to exact knowledge and 50% underestimation for $r_{global} > r_{local}$ (*left*) and $r_{global} < r_{local}$ (*right*).

In Fig. 6, the minimal termination ratio achieved by the central and distributed approach is depicted. The main result is, that both load-based approaches, i.e., in particular the distributed variant, achieve the same performance as possible with exact knowledge of the failure duration. As described in earlier work, underestimations lead to a significant increase of the amount of terminated flows. The success of the load-based approach is clearly shown in Fig. 6, however, it is also necessary to determine suitable threshold parameters η and assess the remapping overhead caused by the two load-based strategies which is done in the following.

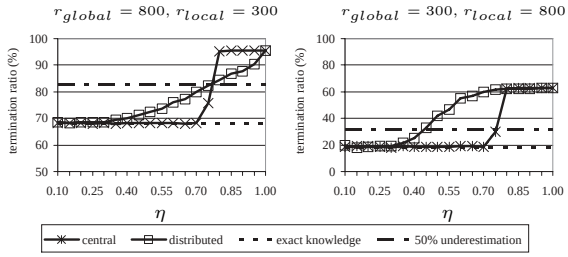

Fig. 7. Termination ratio as a function of the threshold η.

For the two load-based strategies, the threshold parameter η has very different impact on the actual termination ratio. This is outlined in Fig. 7 for a failure duration of 500 slots[1]. The central approach achieves satisfactory results with relatively high choice of η and the change from maximal to minimal termination ratio happens with only small change of η. In contrast, gradual adjustments η in the distributed setting have only small effects and the minimum is reached with small η.

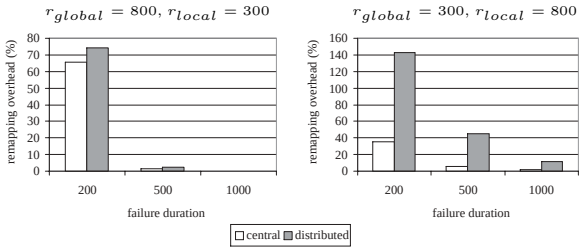

Fig. 8. Remapping overhead of the load-based strategies.

The overhead caused by the remapping strategies is depicted in Fig. 8. The diagrams show the remapping overhead resulting from the choice of η such that the termination ratio is minimal. For example, in the setting with $r_{global} < r_{local}$ and failure duration of 200, the central approach reaches its minimal termination ratio with $\eta = 0.7$ which results in approximately 36% remapping overhead, whereas the distributed approach leads to 140% overhead when the minimal termination ratio is reached, i.e., with $\eta = 0.3$.

The general result is that the remapping overhead of the distributed strategy is similar to the results of the central approach for $r_{global} > r_{local}$. This cannot be expected for the opposite case, i.e., $r_{global} < r_{local}$. In this situation, the distributed approach suffers from the inaccurate knowledge about the global load situation.

In Fig. 9, the remapping overhead is depicted for varying threshold η. The failure duration was 200 slots in order to cover an extreme situation where the duration of the failure is short compared to the load situation on the network.

In order to assess the impact of the global load situation on the distributed and central approach, a number of

[1]The results are similar for the other failure durations.

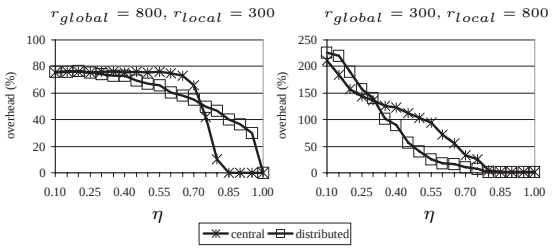

Fig. 9. Remapping overhead as a function of the threshold η (200 slots failure duration).

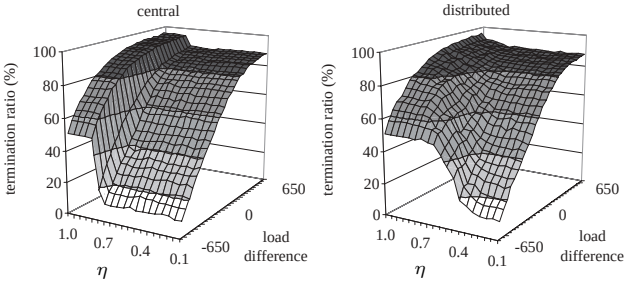

Fig. 10. Termination ratio for varying load difference and threshold η. The average failure duration was 500 slots.

different load situations were generated, with fixed load on the failed resource and varying load on the remaining resources. The termination ratio for varying η and varying load difference, i.e., $r_{global} - r_{local}$, is given in Fig. 10. The figures reflect the situation also shown in Fig. 7. The distributed approach requires small choice of η whereas the central approach reaches its best performance earlier, i.e., with higher values of η. The load difference itself, which describes the degree of inaccuracy of the knowledge in the distributed case, does not play an important role.

Concluding, it can be stated, that under various load situations, both strategies are able to effectively reduce the termination ratio, achieving similar results as possible with exact knowledge about the failure duration. However, the centralized approach leads to significantly lower overhead in cases where the load on the failed resources is much higher than the global load, i.e., $r_{global} < r_{local}$. In such a setting, the centralized approach benefits from the global knowledge.

VI. Application Environment

Due to its generic approach, the previously described failure recovery mechanism is useful for many different types of resources. A particular target environment is the *Virtual Resource Manager* (VRM) described in [2]. This grid resource management system was designed to provide services able to meet new requirements such as support of *service level agreements* (SLA). These capabilities are crucial for the acceptance and usability of next generation grid management systems. Support of SLA implies control over resources and jobs during the run-time in a reliable way. Because of complexity of new grid applications, such as collaborated jobs and workflow jobs, allocation of a lot of different resources is needed which can only provide by local advance reservation systems.

So the VRM is up to managing the different local resources to implement an environment able to meet these new requirements.

In the following, the application of our failure recovery approach within the VRM architecture is briefly described.

An instance of the VRM consists of different *local resource management systems* (RMS) representing and managing one resource, *active interfaces* dealing with different levels of services provided by the RMS and, as the management component of the VRM instance, the *administrative domain controller* (ADC).

The local resource management systems may control arbitrary types of resources, e.g., cluster systems, parallel computers, storage servers, or networks, while the active interfaces provide more reliable and transparent services needed to support SLAs on different RMS. The ADC itself combines all of the different resources, negotiates the SLA and has to ensure that the SLA is to be meet.

Because different ADCs can be organized in a hierarchically way one ADC can be seen as a resource itself. Despite of building hierarchical domains ADCs can also communicate in a peer-to-peer like manner to overcome the limitations of a domain structure and adapt the structure in focus to a Grid like behavior.

The failure recovery mechanism proposed in this paper is situated in the ADC component. Once any of the underlying resources fails partly or entirely, the jobs allocated to the failed resource are 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 one ADC domain, inactive jobs can be transparently mapped without further notification to users which is a major advantage compared to other grid resource management systems such as Globus [4]. This holds also for ADC overlapping failure recovery.

For this the framework of the VRM provides control over compute resources as well as over any other resource required for the migration of jobs, i.e., also interconnection networks. For example, migrating a compute job with large amounts of input data requires a considerable amount of time that 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* [10].

Because of the possibly huge number of different resources managed by the ADC and the complexity of jobs handled by the VRM environment, while the migration of large jobs implicates such complex jobs by itself, failure recovery within such an distributed and dynamic environment turn out to be an challenge for management components. The failure recovery strategy has to meet different constrains especially time constrains in order to reduce termination ratio of allocated jobs. This leads to

the demand for minimizing the time consumption of all tasks in order to meet most of the SLA by remapping these jobs to another resource.

As the calculation of global current and future resource load and booking profiles is one of the time consuming tasks besides the migration of the job itself this overhead has to be minimized.

VII. Conclusion and Future Work

In this paper, a novel load-based failure recovery strategy was presented for management systems with advance reservations, with both a centralized and a distributed version. The mechanism is applicable any environment were 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, underestimations of the actual downtime of a resource have a particularly negative impact on the termination ratio as to few jobs are remapped. 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. Selecting an appropriate value for η is simplified by the fact that smaller values of η do not impact the performance and the lower η is chosen, the lower the termination probability.

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, either centralized or distributed. 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 [2].

Future work will deal with the possibility to integrate checkpointing and migration mechanisms into the load-based approach. Important issues in this context are the time required for job migration over a network and also scheduling batch jobs with best-effort treatment during the failures which is expected to improve the utilization during the actual failure period. Moreover, the dependence of η on the failed resource capacity will be further investigated, as this may lead to a completely automated selection of η. However, the simulations conducted for this paper indicate, that a relatively 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] 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, pp. 126–133.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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, pp. 112–123.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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