The Virtual Resource Manager: An Architecture for SLA-aware Resource Management

Lars-Olof Burchard[1], Matthias Hovestadt[2], Odej Kao[2], Axel Keller[2] and Barry Linnert[1]

[1]Faculty of Computer Science and Electrical Engineering
Technische Universität Berlin, Germany
[2]Paderborn Center for Parallel Computing (PC[2])
Universität Paderborn, Germany

{baron,linnert}@cs.tu-berlin.de, {maho,okao,kel}@upb.de

Abstract

The next generation Grid will demand the Grid middleware to provide flexibility, transparency, and reliability. This implies the appliance of service level agreements to guarantee a negotiated level of quality of service. These requirements also affect the local resource management systems providing resources for the Grid. At this a gap between these demands and the features of today's resource management systems becomes apparent.

In this paper we present an approach which closes this gap. Introducing the architecture of the Virtual Resource Manager, we highlight its main features of runtime responsibility, resource virtualization, information hiding, autonomy provision, and smooth integration of existing resource management system installations.

1 Introduction

Nowadays, Grid computing is understood as the collaborative usage of distributed resources. In this context, the notion resource is explicitly not limited to hardware resources like compute nodes, network bandwidth, or storage capacity. It also covers the areas of software resources, information resources, or even human resources [6].

From a customer's point of view, usability is a decisive factor [12]. Hence, Grid middleware must provide resource virtualization in a way that the user merely needs to outline the requirement profile of the job. Depending on this information, the Grid middleware is in charge of finding and allocating appropriate resources. Current Grid middleware fulfills most of these requirements, but the future demands for more. According to the visions of the European Commission [1], the *Service Level Agreement* (SLA) technology will be of central importance in building up the *Next Generation Grid* (NGG).

An SLA is the exact statement of all expectations and obligations in the business relationship between the provider and the customer [15]. It does not only cover questions regarding the required resources, but also applies to issues like *Quality of Service* (QoS), fault tolerance, or the measurement of SLA compliance. In addition, it encompasses the price for resource consumption, respectively the penalty fee for breaking the agreement.

Current research activities on SLAs solely focus on Grid middleware which is using local *Resource Management Systems* (RMS). Consequently, any Grid middleware can only agree on an SLA, if the underlying RMS is able to assure the fulfillment, i.e., to guarantee the negotiated QoS. However, current RMSs are at best able to operate with advance reservations, assigning a fixed amount of resources over a given time span to a user. Obviously the potential of an SLA would be very limited, if it had to be implemented with advance reservations [13] only. More sophisticated capabilities such as support of malleable applications, diffuse requests, or resilience mechanisms in case of resource outages have to be added at the RMS level [10]. Furthermore, it is not sufficient to consider only resource management for compute resources. Instead, also the interconnecting networks must be taken into account. Consequently, the considerations described in this document also cover the area of *Network Management Systems* (NMS).

However, simply adding SLA mechanisms to an existing RMS is not sufficient. Likewise, it is essential to consider the requirements of the resource providers which are interested in a smooth integration of existing RMS installations into an SLA-aware environment. Important aspects are the

retention of existing administrative responsibilities, and a fine-grained limitation of information published about the local infrastructure.

In this paper we present the multi-layer architecture of the *Virtual Resource Manager* (VRM) that copes with these requirements and has been designed on top of existing RMSs for clusters and networks. The remainder of the paper is organized as follows: after a brief discussion of related work and the future requirements for RMSs and Grid computing, we outline the architecture of the VRM and its functionality. Section 4 exemplifies the VRM operation modes. The paper concludes with some final remarks and an outlook to future work.

2 Status Quo and Future Requirements

The worldwide research in Grid computing resulted in numerous different Grid packages. Besides many commodity Grid systems, general purpose toolkits exist such as UNICOREpro [16] or Globus [9]. Although Globus represents the de-facto standard for Grid toolkits, all these systems have proprietary designs and interfaces. To ensure a future interoperability of Grid systems as well as the opportunity to customize installations, the OGSA[1] working group within the GGF [7] aims to develop the architecture for an open Grid infrastructure [8].

In [1], some important requirements for the NGG were described. Among those needs, one of the major goals is to support resource-sharing in virtual organizations all over the world. Thus attracting commercial users to use the Grid, to develop Grid enabled applications, and to offer their resources in the Grid. Mandatory prerequisites are flexibility, transparency, reliability, and the application of SLAs to guarantee a negotiated level of QoS.

These requirements demand for a mechanism of assurance to guarantee the delivery of service which cannot be given using current RMSs and Grid software. This is the motivation for the development of our architecture. Since RMSs are the base of Grid middleware, this implies that RMSs must become SLA-aware. Features like advance reservations, diffuse requests, and negotiation protocols are mandatory to realize SLA-aware RMSs. Furthermore, the RMS has to monitor running applications in order to guarantee the SLA compliance. The originating control cycle comprises the application, the RMS, and the used resources. Today's RMSs do not provide these features and are therefore not SLA-aware.

In [5], an architecture that supports the co-allocation of multiple resource types, such as processors and network bandwidth, was presented. Within the GGF the GRAAP[2] working group addresses this issue. As a result a reservation component called GARA [14] was presented, which can be embedded into Globus. However, this architecture neither allows to define SLAs or malleable reservations, nor does it support fine-granular information hiding or resilience mechanisms to handle resource outages or failures.

The requirements and procedures of a protocol for negotiating SLAs were described in SNAP [4]. However, the important issue of how to map, implement, and assure those SLAs during the whole lifetime of a request on the RMS layer remains to be solved. This issue is also addressed by the architecture presented in the next section .

3 Architecture of the VRM

The main task of the VRM is to provide high level services by combining different local system resources. These services then allow to provide SLAs and to conceal the underlying structures and components of the local systems such that it is possible to provide *virtual resources*. This is done by establishing so called *Administrative Domains* (AD). In ADs, resources for which a special set of policies is valid are pooled. The policies describe which resources are visible and accessible from what kind of entity and how this access can be provided. ADs may be nested and resources within an AD may be combined to form virtual resources (see Fig. 2).

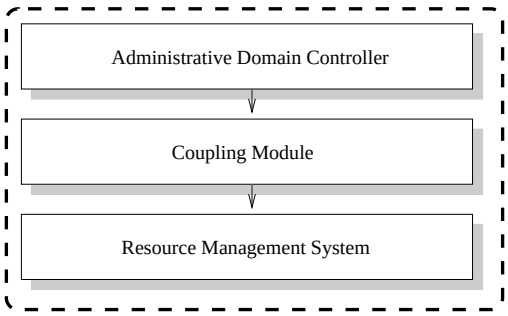

Figure 1. Layers of the VRM Architecture

The VRM architecture comprises three layers as depicted in Fig. 1. These layers are described in the following sections.

3.1 Administrative Domain Controller

The *Administrative Domain Controller* (ADC) is responsible for several different tasks in the field of implementing the VRM. One of the main tasks is to build up the administrative domain and to promote this domain to the Grid or integrate it into other ADs.

[1] Open Grid Services Architecture
[2] Grid Resource Allocation Agreement Protocol

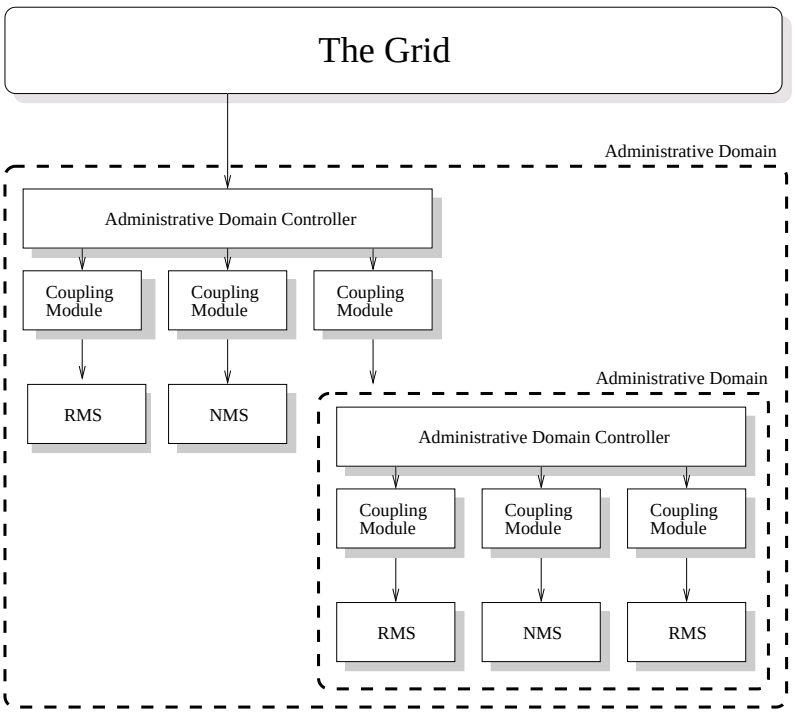

Figure 2. Hierarchical Administrative Domain Structure

Thus, the ADC serves as the gateway between the internal resources and the outside. In this function, the ADC is responsible for publishing information about internal resources and internal network topologies to the outside world. At any time the local administrator of an AD retains the complete autonomy of his own systems. For this purpose, it is possible to specify policies describing which internal resources are available at what time and to what extend. Access policies may be tailored to specific user groups. For security reasons and customization, the ADC allows the administrator to define the granularity of the information that is published. If privacy is not a key issue, it is possible to publish all available information about the internal infrastructure. However, this is not reasonable in most cases. Based on the published information the external requester can decide if the VRM is potentially able to fulfill the requirements of a job.

Additionally, the ADC is able to join the physical resources to so called virtual resources. This *virtualization* is a mechanism which allows to re-combine the physical resources to new resource classes. For example we assume a site operating two clusters. One with 100 Opteron nodes and one with 50 IA32 nodes. The administrator is now able to establish a virtual resource comprising 150 IA32 nodes. This new resource class can be made available to internal and/or external users by means of the policies described above. Furthermore, in case of resource failures such virtual resources are useful to find alternative resources where a job affected by a failure may be resumed in order to keep a given SLA.

Besides virtualization it is also necessary to *combine* resource types in order to support SLAs. For example, the combination of compute power of different nodes and the performance of the node interconnect is necessary to accept SLAs for parallel programs with high communication complexity. For large jobs with a huge amount of data it is also reasonable to combine the performance of the compute nodes at a specified time and the bandwidth of the external network to ensure that all of the data for the job is at place at runtime. Furthermore, it may be desirable to consider disk storage required by an application. An example is a teleimmersion application where visualization information, rendered by a cluster system, must be transferred in real-time over an external network to the visualization device, e.g. a cave. In this case, the availability of the compute resources, the network connection, and the visualization device is guaranteed by an SLA.

Before a new job, i.e., a new SLA, is accepted, the contents of the SLA is negotiated with the ADC. In detail, the negotiation process works at follows: The ADC first checks which resources may be able to comply with the SLA requirements. The ADC then starts a negotiation with the matching internal resources. The results of the internal negotiations are combined and returned to the external re-

quester. The requester, e.g., a Grid based application, has then the opportunity to change its SLA request. The negotiation process iterates until the requester accepts or declines, or the ADC is unable to offer any other resource.

If a job is accepted, the ADC is in charge of complying with the terms of the corresponding SLA. Thus, the ADC maintains responsibility during run-time in the case of exceptions like resource failures. If a problem with any local resource management system arises that prevents the compliance with an accepted SLA, the RMS may first try to solve the problem on its own. If this is not possible, a monitoring facility provides feedback to the related ADC. The ADC then tries to find spare resources in other internal RMS systems. If this resource retrieval is not successful, the ADC can either ask an upper layer ADC for assistance or try to retrieve required resources within the Grid.

3.2 Coupling Modules

The VRM is able to span over heterogeneous environments. An integration of existing RMS installations into the VRM framework must be possible in a simple way. For this purpose, *Coupling Modules* (CM) act as brokers between the demands of the ADC and the capabilities of the specific local RMS systems. Each CM is an active interface, that does not simply wrap the underlying RMS but provides additional functionalities. In general, three different types of RMS/NMS must be distinguished:

1. In the simplest case the underlying RMS/NMS system is capable of fully supporting SLAs. This limits the task of the CM to convert between two SLA syntax formats.

2. The RMS/NMS is not SLA-aware, but provides information that may be used to determine certain properties of admitted jobs, such as the begin or end. This for instance can be used by the CM to determine if a certain deadline for the completion of a job can be guaranteed.

3. The RMS/NMS does not provide any information at all or unsufficient information to use for any kind of SLA negotiation. In this case, the RMS/NMS can only be used to process jobs which do not require QoS.

The CM maps a subset of the SLA-mechanisms to the capabilities of the RMS. This is similar to GARAs approach emulating advance reservations on certain systems. In any case, the SLA negotiation between the CM and the ADC is always the same, independent of the underlying RMS. Thus, it is easy to integrate an arbitrary resource management system into this architecture.

The CMs contain not only the SLA negotiation functionality but also the capability to either monitor the underlying components (in case of resource failures) or to accept feedback from an underlying RMS if the RMS supports this. Such a feedback is provided to the ADC which then may react in order to avoid breaching an SLA.

3.3 Local Resource Management Systems

The local resource management systems administered within an AD may provide access to various kinds of physical resources. In the current setting, the ADC allows to access cluster and network management systems. Instead of an RMS or NMS, the management system may also be another VRM. Thus, it is possible to build hierarchical VRM structures.

As described in the previous section, three types of RMS can be distinguished by their ability to support SLA. An example for a classical cluster RMS is CCS [11] which serves as the foundation for the developments presented here. CCS in particular provides interesting functionalities such as reliable planning, i.e., advance reservations, which for instance can be used to guarantee deadlines for job completion in the context of an SLA.

In order to support SLAs, in particular when considering also the opportunity of resource outages or failures, a resource management system also requires support from SLA-aware NMS. This applies while migrating jobs together with their working set or while establishing guaranteed QoS for real-time transmissions, e.g., visualization data, or large files. In this context, the task of the NMS is to control the network resources at admission control, during runtime, or while forwarding a job to another VRM. The integration of the network management into our VRM scheme is done via CMs to one or more NMSs within each AD. In this sense, an NMS is the entry point to access the network functionality, acting as an admission control instance for a *network domain*, i.e., a certain subpart of a possibly large network. On the network level, an SLA as described in this paper does not solely deal with delay, bandwidth, or jitter, but also includes support for e.g. defining deadlines for completion of network transmissions [2].

Technically, it is possible to integrate any network reservation mechanism into our framework. However the full strength of the SLA-based mechanisms can only be exploited using a reservation system that supports advance reservations and more sophisticated reservation mechanisms such as malleable reservations [2] and fault tolerance mechanisms [3].

3.4 Features

With this architecture the VRM fulfills the following requirements:

Responsibility at Runtime: The VRM components (CM, RMS, and NMS) monitor running applications to be able to fulfill an accepted SLA. This may imply actions like job migration within its AD or to higher ADs. An important aspect in this context is the opportunity to deal with resource failures. This is mandatory to guarantee the compliance with a given SLA. Thus, keeping track of the status of any underlying management system during run-time is essential to assure that breaches of an SLA are detected as soon as possible and according measures, such as job forwarding to alternative machines, are taken to guarantee that an SLA can be met even in case of failures or outages.

Information Hiding: The embedding of a resource in a Grid environment requires that data about the resource is known in the Grid. However, providers do not always want to publish detailed data about their whole infrastructure to the Grid. The level of visibility should depend on the level of cooperation between provider and customer. The VRM architecture realizes this by means of access policies per AD.

Autonomy of Local Administrative Structures: Providers will only integrate their resources into the Grid if they still have as much local autonomy as possible. A complete control transfer to the Grid would not be accepted. The VRM allows the administrator to specify rules which resources are accessible when, by whom, and to what extent. Rules may apply to either a single RMS or to a sub-AD (refer to Fig. 2).

Providing Virtual Systems: If a provider operates more than one system it is desirable to contribute a virtual system to the Grid comprising parts or all existing systems. The VRM is able to combine resources within its AD to such a virtual resource.

4 Examples

The requirements arising from the Next Generation Grid will demand for local resource management systems with novel functionalities. For this purpose, we presented the architecture of the Virtual Resource Manager. In this section its mode of operation is exemplified by some basic examples.

In the following, we assume an exemplary configuration as depicted in Fig. 3. The administrative domain AD:A-B is spanning over two nested ADs, one in site A and one in site B. The AD in site B comprises only one cluster, whereas the AD in site A again consists of a cluster and another AD. Note that Cluster-2 in AD:A does not form an AD, like Cluster-1 does. While the administrator of the AD of Cluster-1 has a high grade of autonomy of his system, Cluster-2 is directly connected to a CM of the VRM of AD:A. Grid users submitting their jobs to this virtual domain contact a central interface, not knowing or caring where the resources at runtime are located.

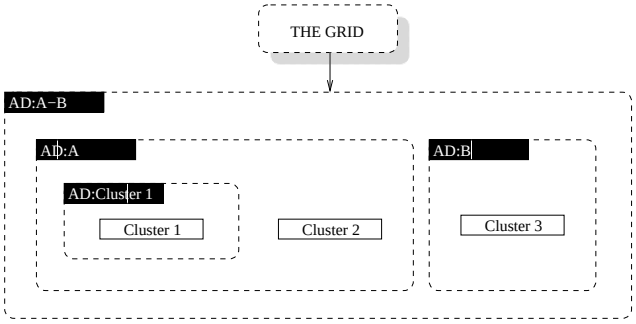

Figure 3. Example: Nested Administrative Domains

By nesting ADs, a fine granularity of administrative autonomy can be established. This means, that the administrator of the AD:A can specify exactly, which information about his AD is known outside and how internal resources may be accessed from outside. If necessary, information about resources in site A can be limited in such a way, that the administrator of the higher level AD:A-B has no knowledge about particular resources within the AD:A.

4.1 Negotiation Process

As described before, the SLA negotiation is managed by an ADC for its domain. The negotiation process starts with a check whether the local AD is potentially capable of fulfilling the SLA. For example, if IA64 hardware is requested and the local AD provides only IA32 processors, the negotiation is useless and will be cancelled.

We consider the environment depicted in Fig. 3 and assume, that AD:Cluster-1 supports SLAs through the functionality of the corresponding ADC, the RMS of Cluster-2 supports only planning (advance reservations) but the corresponding CM is capable of obtaining information about the status of the RMS. It is also assumed that the RMS of Cluster-3 does neither provide feedback about its status nor does it support SLAs. The following example describes a case where a job requires completion up to a certain deadline. Furthermore, it is assumed that only Cluster-2 provides sufficient resources to guarantee job completion in time.

When the job is processed by the VRM of AD:A-B, the corresponding ADC starts a negotiation with the underlying local RMS and sub-ADs[3]. Thus, a negotiation is started

[3] An NMS is not required in this case.

with AD:A and AD:B. As described before, AD:B cannot provide any information that allows to reliably predict that the job will be completed on Cluster-3 in time. Hence, the corresponding ADC cancels the negotiation process immediately. In contrast, AD:A negotiates with AD:Cluster-1 and the CM connected to Cluster 2. While AD:Cluster-1 is not able to guarantee the requested deadline at all, the CM attached to Cluster-2 determines that the deadline can be guaranteed. Consequently, the negotiation of AD:A with AD:Cluster-1 is cancelled and the job is assigned to Cluster-2. AD:A reports this result to the ADC of AD:A-B. This means, no further negotiation between AD:A-B and AD:A is required and the SLA can be accepted.

In this case, the ADCs of both AD:A and AD:A-B need to monitor the job during runtime in order to comply with the SLA. For this purpose, the CM of Cluster-2 monitors the RMS status and in case of a failure reports this to the ADC of AD:A. This ADC may then try to re-map the job onto the available resources within its domain. This can include a new negotiation with AD:Cluster-1. In case, the problem cannot be solved within AD:A, it is reported to the next higher AD, i.e., AD:A-B. The corresponding ADC then tries to solve the problem within its domain.

4.2 Assured Job Completion

Many applications need a runtime that is longer than anything allowed or available on a single resource. Such jobs often checkpoint their state and are resubmitted by the user when they have been aborted by the RMS at the end of the requested duration. The check-pointing is done cyclic either driven by a timer or after a specific amount of computing steps.

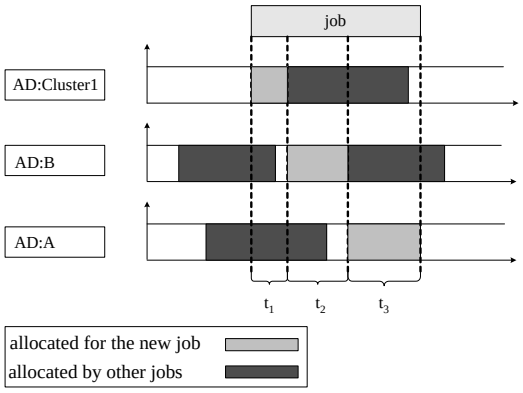

Figure 4. Example: Job cannot be completed at only a single site. Hence, it is running on different sites at different times.

As described above the VRM determines qualified resources while negotiating an incoming SLA-request. We now assume that the incoming job is managed by the AD:A-B (see Fig. 3) and is able to run in site A as well as in site B. The VRM of the AD:A-B plans slots in site A and in site B. This planning is done in collaboration with the ADCs of AD:A and AD:B. When the slot at site A ends, the job migrates to site B and back again. This is shown in Fig. 4[4].

Computing the schedule may become complex since constraints like the cost of migration in relation to the length of a used time slot, the availability and cost of resources, a given deadline, or the overall system utilization have to be considered. The client may provide additional information such as "the runtime should be x full hours for the application and n minutes are required for each check-pointing".

If the application is able to catch signals, the user may additionally specify a signal (e. g. USR1) which initiates the check-pointing. The VRM is then able to enforce a checkpoint by sending the given checkpoint signal in time. After waiting for the given time (n minutes in our example) the VRM stops the application. This allows to utilize time slots shorter than a regular checkpoint cycle e.g. once per hour. Thus, gaps in the schedule (e.g., night times) can be utilized more efficiently.

Depending on the negotiated priority, the VRM may also use other mechanisms like cycle stealing, runtime extension, or ruling out other jobs to ensure the job completion [10].

The migration of data includes the whole working set, i.e., the program together with its status and the input and output data. This may comprise a considerable amount of information. Hence it should be supported by network reservations in order to guarantee the timely arrival of the data at the respective site. In addition to the time required for check-pointing, the time needed for the migration of the data itself must be considered in the schedule.

In this particular example, the functionality to migrate a job during its runtime is applied to a job with finite run-time in order to assure job completion. In addition, it is possible to apply the same mechanism also to jobs that need to be constantly accessible within the Grid, e.g., certain server processes. Then, the task is to assure availability of the server rather than job completion.

4.3 Fault Tolerance

If an accepted SLA cannot be fulfilled with the available resources within the related AD (e.g., due to a hardware failure), it has to be migrated to a different resource if possible. The same holds for a link failure which impacts a real-time transmission within an AD, e.g., in a teleimmersion application.

[4]This is a simplified sketch which does not include the time required for check-pointing and migration.

In such a case, compute resources must be determined that suffice the corresponding SLA and are currently unused. Furthermore, the transfer of such a job requires network transmission capacity for both the program itself as well as its working set and the data that are used by the program. This can be a considerable amount of information and it is essential that sufficient network transmission capacity is available to transfer all the required data to the new location.

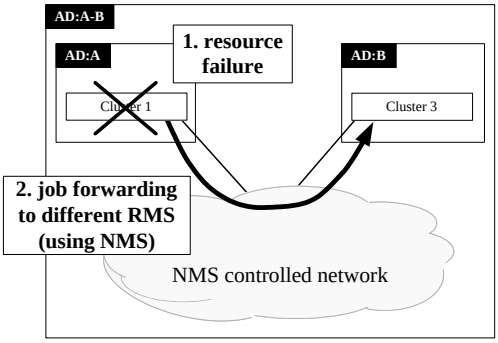

Figure 5. Example: Job Forwarding within an AD.

In case, a machine within AD:A fails and as a consequence a job running on that machine can no longer be executed, it is possible that an SLA given for the particular job can no longer be guaranteed. The ADC of AD:A will be notified of this failure, determine the possible breach of the SLA, and hence try to locate an alternative machine within the same administrative domain. In case this procedure is successful, the transmission of the job to the alternative machine (e.g., a dedicated backup cluster) is initiated and the job is restarted with the last saved status. For that purpose, some kind of check-pointing mechanisms is required to obtain the status before the failure. Otherwise, the job is restarted from the beginning.

When the search for a suitable alternative machine within the same administrative domain fails, the VRM of the next higher administrative domain will be contacted to try to reschedule the broken job. In the example depicted in Fig. 5, the VRM of domain AD:A was unable to relocate the job within its domain. Hence, the next higher AD of domain AD:A-B takes responsibility and forwards the job to domain AD:B.

In the same manner, the job forwarding component can become active in case of a link failure between two machines that are currently transmitting data such as real-time visualization information (see Fig. 6). At first, the NMS which is responsible for the network connection may try to reroute the traffic. For example, a bandwidth broker as described in [3] will try to reroute the connection on an alternative network path. However it may be possible that rerouting cannot be performed due to exhausted resources.

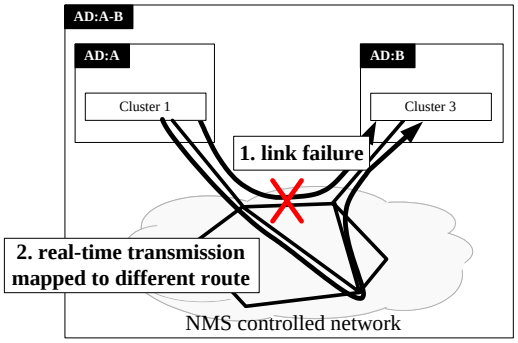

Figure 6. Example: Link failure and remapping of real-time transmission to an alternative path.

In this case, the VRM is required to map the related job(s) onto alternative machines with sufficient amount of interconnecting capacity. In the example depicted in Fig. 6, the NMS is able to deal with the link failure on its own.

Using advance reservations in this context has a significant advantage in case of failures. Whenever the duration of the resource outage or failure can be estimated, it is possible to re-map allocations that are most likely affected by the failure onto other resources *before* the usage phase of the particular resources commences. This reduces the preemption probability for jobs, but requires an architecture like ours that keeps control over the whole system during runtime.

A bandwidth broker [3] can be easily adopted to support such functionality. In case other network reservation mechanisms are used, such as RSVP, it is required to integrate a monitoring component in the VRM for the network transmissions of the jobs that are managed within an administrative domain.

5 Conclusion and Future Work

In this paper, we presented an architecture for the SLA-aware management of different resource types that closes the gap between future demands and the capabilities of today's RMSs. The key features of our system are the support of SLAs and information hiding by building administrative domains. In order to support SLAs, it is of importance to establish runtime responsibility throughout the lifetime of a job. Thus, it is possible to cope with failures or outages of resources. As outlined in [10], planning based RMSs are well suited for SLA-aware resource management. Hence, the considerations presented here were based on a planning based RMS for clusters and a network management system supporting advance reservations. Although in the examples described here the focus was on cluster and network management, the architecture can be extended to deal with any

other resource type.

A prototype implementation of our architecture is in progress. It is based on the planning based RMS CCS [11] and a network management system supporting advance reservations and malleable requests described in [2]. Future work will deal with the integration of our VRM system into existing Grid architectures.

References

[1] H. Bal, C. de Laat, S. Haridi, K. Jeffery, J. Labarta, D. Laforenza, P. Maccallum, J. Mass, L. Matyska, T. Priol, A. Reinefeld, A. Reuter, M. Riguidel, D. Snelling, and M. van Steen. Next Generation Grid(s), European Grid Research 2005-2010. Technical report, Expert Group Report for the EU, Brussel, 2003.

[2] L.-O. Burchard. On the Performance of Computer Networks with Advance Reservation Mechanisms. In *11th IEEE International Conference on Networks (ICON), Sydney, Australia*, 2003.

[3] L.-O. Burchard and M. Droste-Franke. Fault Tolerance in Networks with an Advance Reservation Service. In *11th International Workshop on Quality of Service (IWQoS), Monterey, USA*, volume 2707 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2003.

[4] K. Czajkowski, 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 U. S. E. D.G. Feitelson, L. Rudolph, editor, *Job Scheduling Strategies for Parallel Processing, 8th InternationalWorkshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002. Revised Papers*, July 2002.

[5] I. Foster, 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.

[6] I. Foster and C. Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc. San Fransisco, 1999.

[7] Global Grid Forum. `http://www.ggf.org`.

[8] GGF Open Grid Services Architecture Working Group (OGSA WG). Open Grid Services Architecture: A Roadmap, April 2003.

[9] The Globus Alliance: Globus Toolkit. `http://www.globus.org`.

[10] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003 Seattle, WA, USA, June 24, 2003 Revised Papers*, 2003.

[11] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In *Annual Review of Scalable Computing, vol. 3, Singapore University Press*, 2001.

[12] H. Kishimoto, A. Savva, and D. Snelling. OGSA Fundamental Services: Requirements for Commercial GRID Systems. Technical report, Open Grid Services Architecture Working Group (OGSA WG), `http://www.gridforum.org/Documents/Drafts/default_b.htm`, April 2003.

[13] J. MacLaren, V. Sander, and W. Ziegler. Advanced Reservations - State of the Art. Technical report, Grid Resource Allocation Agreement Protocol Working Group, Global Grid Forum, `http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html`, April 2003.

[14] A. Roy and L. Winkler. GARA: An Architecture for Advanced Reservations. Presentation, July 1999.

[15] A. Sahai, S. Graupner, V. Machiraju, and A. v. Moorsel. Specifying and Monitoring Guarantees in Commercial Grids trough SLA. Technical Report HPL-2002-324, Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, November 2002.

[16] UNICORE Forum e.V.: UNICOREpro. `http://www.unicore.org`.