Virtual Execution Environments and the Negotiation of Service Level Agreements in Grid Systems*

Dominic Battré[1], Matthias Hovestadt[1], Axel Keller[2], Odej Kao[1], and Kerstin Voss[2]

[1] Technische Universität Berlin, Germany
{battre,maho,okao}@cs.tu-berlin.de
[2] Paderborn Center for Parallel Computing, Universität Paderborn, Germany
{kel,kerstinv}@uni-paderborn.de

Abstract. Service Level Agreements (SLAs) have focal importance if the commercial customer should be attracted to the Grid. An SLA-aware resource management system has already been realize, able to fulfill the SLA of jobs even in the case of resource failures. For this, it is able to migrate checkpointed jobs over the Grid. At this, virtual execution environments allow to increase the number of potential migration targets significantly. In this paper we outline the concept of such virtual execution environments and focus on the SLA negotiation aspects.

1 Introduction

At least in the academic domain, Grid systems evolved as a commonly accepted working instrument. Universities are providing their resources to Grid systems, enabling researchers to easily access them, e. g. for executing compute intensive simulations. In the commercial domain however, the Grid systems are not widely used yet, at best limited to small island solutions.

The commercial potential of Grid computing however has been recognized already in the early days of Grid computing. Firstly, companies like IBM, Hewlett Packard, and Microsoft investing noticeable efforts on research and the support of research communities. Secondly, already in 2003 the European Commission (EC) convened a group of experts to clarify the demands of future Grid systems and which properties and capabilities are missing in current existing Grid infrastructures. Their work resulted in the idea of the Next Generation Grid (NGG) [1,2,3]. This work clearly identified that guaranteed provision of reliability, transparency, and Quality of Service (QoS) is an important demand for successfully commercialize future Grid systems. In particular, commercial users will not use a Grid system for computing business critical jobs if it is operating on the best-effort approach only.

* This work has been partially supported by the EU within the 6th Framework Programme under contract IST-031772 “Advanced Risk Assessment and Management for Trustable Grids” (AssessGrid).

Service level agreements (SLAs) are powerful instruments for describing the obligations and expectations of customer and provider within a commercial business relationship [4], e. g. specifying the QoS requirement profile of a job. Modern resource management systems (RMS) are working on the best-effort approach, not giving any guarantees on job completion to the user. Since these RMS are offering their resources to Grid systems, Grid middleware has only limited means in fulfilling all terms of negotiated SLAs.

The EC-funded project HPC4U [5] focused on closing this gap between the requirements of SLA-enabled Grid middleware and the capabilities of RMS, started working on an SLA-aware RMS, utilizing the mechanisms of process-, storage- and network-subsystems for realizing application-transparent fault tolerance. The project finished successfully in 2008, having an SLA-aware Grid fabric as its major outcome. The core of this Grid fabric is an SLA-aware RMS, able to fulfill the terms of negotiated SLAs also in the case of resource failures. For this, the RMS is acting as an active Grid resource, migrating jobs affected by outages transparently to other Grid resources, as long as these Grid providers agree on all terms of the SLA bound to the job to be migrated.

This system is using the Globus toolkit as Grid middleware, fully functional, and available as open source. However, the system currently requires a target resource which is very similar to the source resource, since the project uses kernel-level checkpointing mechanisms. Due to the heterogeneous nature of Grids, only a small subset of resources are eligible migration targets. Increasing the number of potential migration targets would directly increase the level of fault tolerance and reliability.

In this scope we introduced the concept of virtual execution environments. Instead of executing the compute job directly on the compute node of a cluster system, a virtualized environment is established which complies to the requirements of the job to be migrated. In this paper, we will first highlight the architecture of the HPC4U cluster system and outline the concept of virtual execution environments. In the main part we will describe how the requirements on these environments for the particular migration subject can be integrated into the SLA negotiation process. The paper ends with an overview about related work and a short conclusion.

2 SLA-Aware Resource Management

For the realization of an SLA-aware RMS, the project decided to use the planning-based RMS OpenCCS [6]. A new scheduler has been implemented, assigning jobs to resources according to their SLAs. Since the RMS has to guarantee the SLA-compliance also in case of resource outages, the HPC4U system does not only encompass the OpenCCS RMS, but also subsystems providing QoS and fault tolerance on storage, network, and process layer.

At this, checkpointing of running applications is the core functionality for providing fault tolerance. Without any checkpoints available, the results of a running application are lost in case of resource outages. In the light of long running jobs, potentially running over weeks, using numerous computers of a cluster in parallel, the total number of lost compute hours can be immense. Therefore the RMS will regularly checkpoint the running application (e. g. by creating one checkpoint every 60 minutes). In case of a resource outage, the application can be restarted from the latest checkpointed state.

Since the checkpoint dataset must encompass not only the process memory and state, but also the network and storage, an orchestrated operation of all subsystems is mandatory to ensure consistency. In this process, the process subsystem first stops the running application, so that the state of the application remains fixed, not changing over the execution of the checkpoint operation. Now, the process subsystem creates a kernel-level process checkpoint of the running application. In case of MPI-parallel applications, this checkpoint consists of an image of all parallel running processes as well as the network state. After this, the application's storage partition is saved, so that a consistent image of the application environment has been generated. Finally, the process subsystem unblocks the application, so that its computation resumes.

This mechanism is fully implemented and operational. In the verification and validation tests of the HPC4U project it has proven to be applicable also to commercial applications, where the user may not recompile or relink the code. In fact, the entire checkpointing remains transparent for the user who does not have to modify his application nor the way of starting it.

This way of checkpointing an application might be problematic for applications communicating with cluster-external entities. Since the application remains stopped during the entire checkpointing process, it may not communicate with external processes. Depending on the duration of the checkpoint operation, this may result in timeout errors. However, the typical application targeted with this mechanism only communication within the cluster, e. g. in case of parallel applications.

2.1 Migration and Compatibility Profile

In contrast to application level checkpointing, a kernel-level checkpointed process can not be restarted on arbitrary target systems. Beside high level characteristics like operating system or processor type, the target machine has to be compatible with regard to versions of installed libraries and tools. When a checkpointed job is restarted on an incompatible resource, the job would directly crash at best. In the worst case, the application would resume its computation but return incorrect results.

An obvious way to face this situation and ensure a successful restart on the target machine is to request identical machines. At this, the hardware of a suitable target machine must be identical to the source machine. The same holds for the software installation. Both machines have to have identical operating systems with identical upgrade levels (e. g. RedHat AS4, upgrade 2). These requirements are fulfilled as long as the job is restarted within the same cluster system, since such clusters are usually homogeneously configured.

However, the HPC4U system may not only restart the job on the same cluster, but on any cluster within the same administrative domain or even on any resource within the Grid. In the case of restarting within the same administrative domain, compatibility can be configured statically (e. g. jobs of cluster A may be restarted on cluster B), because administrators know about the particular configuration. When migrating to Grid resources, a dynamic mechanism is mandatory.

The HPC4U project introduced the compatibility profile, holding compatibility requirements of the checkpoint dataset on the target resource. This ranges from high-level requirements like operating systems, processor architecture, or available main memory, up to low-level requirements like version information of loaded system libraries. When migrating over the Grid, this profile is part of the SLA negotiation process. Hence, if the remote system accepts the SLA request, it confirms to provide a resource compatible to the checkpoint dataset.

3 Virtual Execution Environments

The previous section underlined the difficulties of retrieving compatible resources for migration within Grid systems. Only if the target resource matches the compatibility profile, the checkpointed job may be migrated.

It is common practise that Grid resources are operated under a decentralized autonomy of local resource providers. Hence, local administrators decide on the operating system to be installed on compute resources. Moreover, is is the responsibility of local resource administrators to install updates, e. g. applying available patches to the local compute node environment. This local autonomy is the root cause of the difficulty of finding compatible resources. Since each difference in the operating system patchlevel impacts the compatibility for the migration process, it would be beneficial to have commonly accepted and available execution environments, guaranteeing the compatibility regarding paths and libraries.

This goal can be achieved by using virtualization technology on the compute resources. Instead of executing applications directly on a compute node, a virtual system is started first. Within this virtual system arbitrary operating systems can be started. The application then is started within this virtual system environment. This way, it is possible to start the application within a well-defined system environment.

The concept of virtual execution environments (VEEs) has been implemented in a first version, allowing to execute a job within a virtualized system environment on a compute node. The realization implies a number of additional requirements for the RMS, particularly for the daemons running on the compute nodes, where the virtual machine is to be established at runtime. To limit the impact on the overall system, a major design principle was to keep the existence of virtualized components as transparent as possible.

On the compute node we have OpenCCS daemons responsible for managing the nodes and executing jobs. The central component here is the Node Session

Manager (NSM), building a linking element between the daemons running on the front-end node and the compute node. It does not only configure and monitor the node, it also invokes the Execution Manager (EM) daemon, if an application needs to be started. This EM then runs with user privileges and executes the commands specified by the user.

For achieving VEE transparency, the tasks of the NSM have been enhanced to handle VEE-related jobs. This VEE-enabled NSM is called NSM+. The first task of the NSM+ in case of VEE-bound jobs is the establishment of the virtual machine on the compute node. For this it invokes commands specified in the OpenCCS configuration file. In the scope of this implementation, the Xen system has been used. Hence, the NSM+ running in Dom0 starts the image of the specified VEE as DomU.

Once the DomU has started, a second NSM+ is started within the DomU, responsible for performing all classic NSM tasks like node configuration or EM invocation. However, the NSM+ running in DomU does not communicate with other OpenCCS daemons directly, but uses the NSM+ running in Dom0 as communication proxy. Hence, the NSM+ running in Dom0 is called *proxy-mode NSM+*, whereas the NSM+ running in DomU is called *VM-mode NSM+*.

4 SLA Negotiation Aspects

The Grid community has identified the need for a standard for SLA description and negotiation. This led to the development of WS-Agreement/-Negotiation [7]. These upcoming standards rely on the Web Services Resource Framework (WSRF).

This WS-Agreement protocol now has been extended to allow flexible SLA negotiation schemes between contractors and service providers. Briefly, modifications consist in the addition of one operation: getQuote(). This is only an extension, which allows to change the original single-round acceptance model to a two-phase acceptance model. This introduce the negotiation possibility, in other words the bargaining capability.

The protocol implemented within the EC-funded AssessGrid project, used for realizing the migration using virtual execution environments, is a two-phase commit negotiation. The user first requests a template from the provider, which gives the provider capabilities. The user then describes his specific needs in a quote request. The quote sent by the provider gives the offer made by the provider, based on the request made by the user. The user is then able to accept this quote, and sign it. The SLA contract is then signed if the provider accepts the user's signed SLA.

4.1 Original WS-Agreement Protocol

The Web Services Agreement Specification (WS-Agreement, or WSAG for short) was produced by the GRAAP working group in the Open Grid Forum (OGF). The version on which is based the Assessgrid negotiation is the draft version of April 2007. The WS-Agreement protocol is based on a single round offer, accept

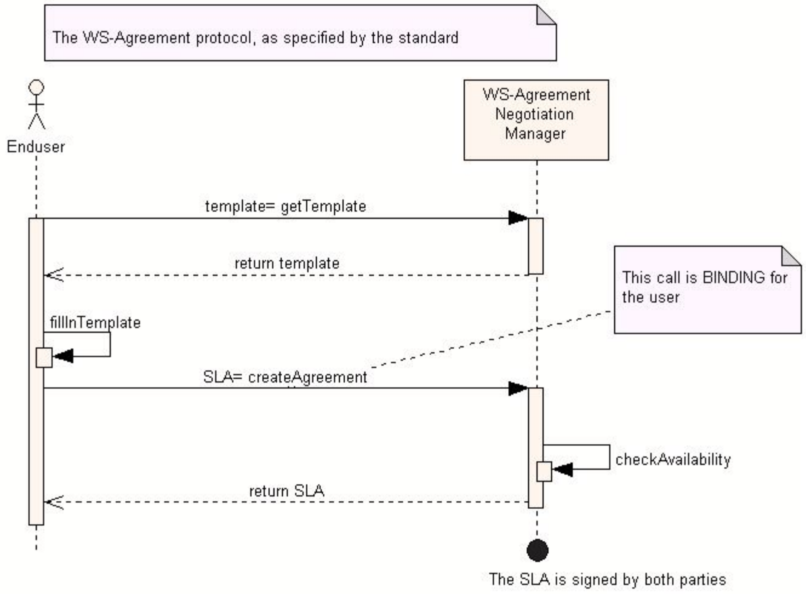
The WS-Agreement protocol, as specified by the standard
Enduser
WS-Agreement Negotiation Manager
template= getTemplate
return template
This call is BINDING for the user
fillInTemplate
SLA= createAgreement
checkAvailability
return SLA
The SLA is signed by both parties

Fig. 1. Negotiation Sequence Diagram

message exchange between the negotiating parties. The basic synchronous wsag: CreateAgreement() operation directly captures this exchange, as can be seen in the following diagram:

The WS-Agreement specification supports additional asynchronous patterns, but the semantics of this abstract exchange is always the same: the initiator sends an obligating offer, with explicit terms of agreement, which the responder may accept or reject by unilateral decision. The agreement relationship is in place as soon as the responder decides to accept.

The agreement initiator can operate either on behalf of a service consumer or a service provider. As depicted in the sequence diagram in figure 1 , the initiator first requests the SLA template from the agreement responder querying a property of the AgreementFactory WS resource - calling getResourceProperty(). Then the initiator fills in the template taking into account the corresponding creation constraints. The result is the SLA Offer, that is sent by the initiator to the responder by calling createAgreement() operation. Now the responder is free to accept the offer (by returning an EPR to an Agreement resource) or to reject it by sending a fault response. The initiator is now bound to the agreement and he can inspect its properties. Considering the most usual case in which the initiator is the service consumer, a user could start using the service once the service state is ready.

4.2 Protocol Extension

The effort made by the AssessGrid team to develop a flexible and robust SLA negotiation protocol, can be seen as a continuation of work within the

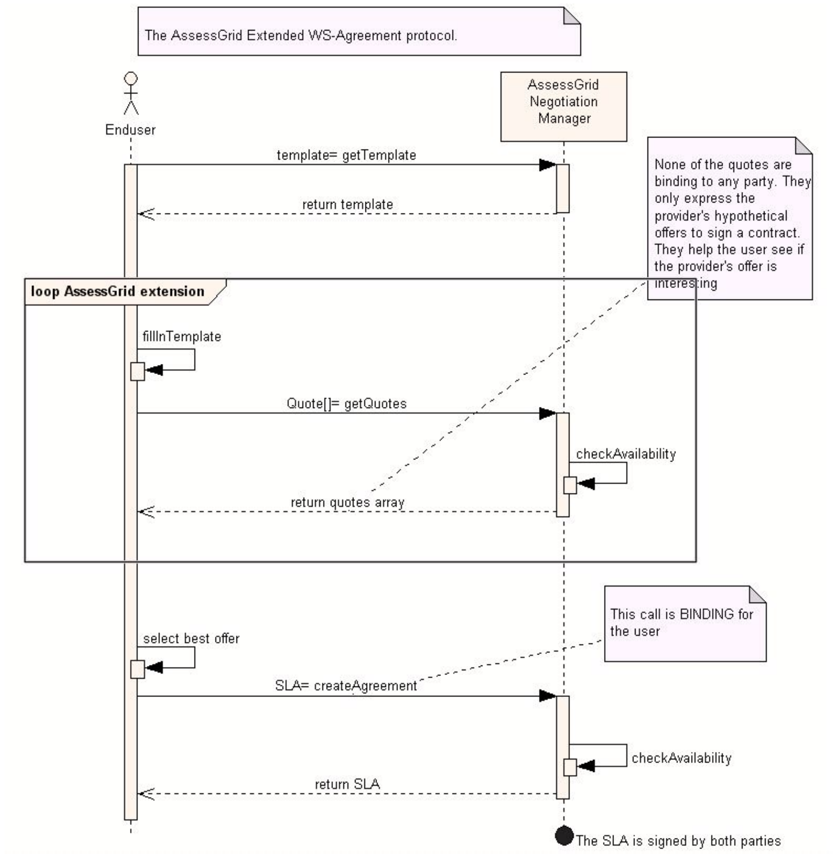
The AssessGrid Extended WS-Agreement protocol.
Enduser
AssessGrid Negotiation Manager
template= getTemplate
return template
None of the quotes are binding to any party. They only express the provider's hypothetical offers to sign a contract. They help the user see if the provider's offer is interesting
loop AssessGrid extension
fillInTemplate
Quote[]= getQuotes
checkAvailability
return quotes array
This call is BINDING for the user
select best offer
SLA= createAgreement
checkAvailability
return SLA
The SLA is signed by both parties

Fig. 2. Enhanced Negotiation Sequence Diagram

WS-Agreement specification. The extended protocol answers the requirements where a negotiation before a final agreement is needed.

The agreement mechanism within the WS-Agreement draft specification does not meet the negotiation requirement. The main drawback comes from the single round offer, accept agreement mechanism.This has an important consequence: there is no possibility for a service consumer to request offers from different providers so that he can choose the best one among them. In order to do so, he would have to become the agreement initiator and would have to call createAgreement() from several providers to propose some SLA to each. The problem is that he would then be bound to every provider that decides to accept. The concept of SLA quote does not exist in the WS-Agreement draft specification, it is not possible for a consumer to simply ask a provider what his terms would be without being committed to the provider by this action. In the real word, negotiation processes usually begin by the initiator asking non-committing questions to the other party.

The solution proposed has been to introduce the concept of SLA quote into the agreement mechanism. The concept of this enhancement is depicted in figure 2. The getQuotes() methods offers the end-user the possibilty to have a first evaluation of a request for service. Based on this first quote, the user can then decide to accept it using the createAgreement() method. If the provider's quote is not satisfactory, a new quote can be requested by entering a new quote request, with slightly different parameters.

4.3 Negotiation on Virtual Execution Environments

The central component of this new infrastructure is a central system image repository, holding system images of different operating systems, e. g. commonly used Linux distributions in different versions and different patchlevels (cf. figure 3). Each of these images has a unique ID. The contents of this image repository can be published within the Grid using well established mechanisms like resource information catalogues. Since these information services are public, not only a single provider is able to access the contents, but all Grid stakeholders.

Already at SLA negotiation time, the service customer now has the opportunity to specify a system image to be established at execution time, instead of the current practise of not knowing if the job will be executed on an up-to-date Debian system or an outdated SuSE. Hence, the customer can test his job in the specified environment by establishing the image from the system image repository locally, being sure that the job will succeed and return the expected results.

Providers will typically only support a subset of images from the image repository, e. g. images of distributions that are known by the system administrators, or that have proven to run stable on the compute resources. Providers are free in the selection of supported system images, which are then published in the Grid resource information catalogue.

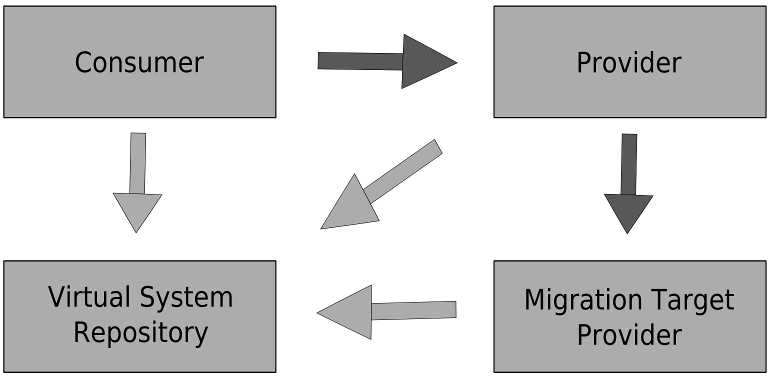

Fig. 3. Virtual System Repository

At level of Grid middleware the customer requested virtual system ID has to be matched against the provider supported characteristics of their resources, like it is already common practise with all other system parameters like number of nodes or amount of main memory. Hence, currently existing matching mechanisms, e. g. used by Grid broker systems, can be used for also matching the virtual system IDs.

If a provider agrees on an SLA specifying a virtual system image ID, it has to establish the specified system image at runtime on the compute node and then execute the user job within the specified environment. If the customer did not specify any image ID, the provider may choose a default virtual system to be executed on the compute node. In this case, the customer does not have any knowledge about the system that is used for executing his job, but this only corresponds to the current situation.

At runtime, the virtual system is then established on the compute node, so that all job checkpoints can be executed in a well defined and well known environment. Hence, all dependencies of the checkpointed job on the system environment are known, since the system has been executed in the specified virtual environment.

This is particularly beneficial for the migration process, since it is no longer mandatory to generate a compatibility profile, as explained in the previous section. Instead of querying for libraries or library versions, the resource provider is able to query for resources that can execute the particular virtual system ID. Hence, instead of describing requirements on a compatible environment, now the compatible environment can be requested directly.

In case of a resource failure, the provider first tries to identify job to be migrated. The goal is to release them from the current schedule, so that the remaining resources suffice the requirements of the remaining jobs, so that their SLAs can be fulfilled. Regarding the SLA-bound jobs to be migrated, the terms of their SLAs are subject of the SLA negotiation process between the migrating provider and the provider taking over the job.

Particularly this virtual system ID is part of the SLA negotiation process of the source and the target resource provider, which is providing the migration target resources. If the target resource provider agrees on the SLA, it agrees on establishing the specified virtual system on the compute node at runtime, where the checkpointed job is to be resumed. This mechanism also applies to all further migration operations of this job, e. g. if the new resource provider again has to query the Grid for backup resources due to resource failures. At the bottom line, the job is only migrated, if the target provider agrees on all terms, so that - thanks to the migration capabilities - all SLAs of all jobs are fulfilled.

In practise, resource failures in a cluster system typically affect more than only one single job, making the handling of multiple affected jobs necessary. However, it is not necessarily the best choice to migrate the job that has been directly affected by the resource outage (e. g. the time between job completion and SLA-specified deadline does not allow migration). Instead, the system should pay attention to factors like revenue or remaining time between job completion and deadline.

The scheduler of the RMS takes these factors as heuristic for selecting an order to remove jobs from the system, which overlap with the jobs affected by the outage (including the job itself). As long as the revenue loss caused by migration is smaller than the penalty which has to be paid for violating the SLA of the failure affected job, the job qualifies for the migration candidate set, if the RMS is also able to get an SLA offer of some other provider.

If a set has been identified, which migration allows the fulfillment of the SLA of the affected job, the RMS tries to get SLA offers from other providers. If this is successful, the migration can be initiated. Otherwise the RMS has to remove the respective jobs from the set, continuing with the set generation. If no set can be found, the already agreed SLA offers are canceled.

5 Related Work

In [2], important requirements for the Next Generation Grid (NGG) were described, i. e. Grids to be successfully deployed in commercial environments. Mandatory prerequisites of such Grids are flexibility, transparency, reliability, and the application of SLAs to guarantee a negotiated QoS level. The Grid community has identified the need for a standard for SLA description and negotiation. This led to the development of WS-Agreement [7].

The usage of virtualization technology at provider level within RMSs has been described in [8]. Here the Xen virtual machine monitor is used for increasing system utilization and response time of a cluster system operated with the Sun Grid Engine RMS. Long running sequential jobs are executed in a virtualized environment and suspended for executing short running parallel jobs. This work underlined the general applicability of virtualization technology on compute nodes, but did not address fault tolerance nor the execution of parallel applications within virtual environments.

The benefits of using virtual machines for running applications in the Grid had been addressed from the perspective of virtual organizations in [9]. Their implementation uses VM-based workspaces configured with Xen hypervisors [10], which offer a client to create and manage other virtual machines. VM images are configured for specific applications and deployed as Edge Services. By using the VOMS credentials of a VO administrator, deployed VM images can be accessed and applications of the end-user can be executed without additional configuration effort. This work has a similar approach as our developments since it aims to increase the flexibility of the resource usage and also considers the negotiation of resource requirements. However, the capability of using fault-tolerance mechanisms for the provisioning of SLAs is left out of consideration.

The deployment of a configured VM on a hypervisor has no significant overhead since it can be performed in less than a second [11] which is comparable to the overhead of Grid tools. The deployment can be realized by using preconfigured VM images or refining configuration which influences the flexibility and deployment time. In [12] an XML description is presented which should balance both aspects. Here, the flexibility is pointed out as an important issue if brokers configure the workspaces according to end-users requirements. Our system uses pre-configured VM images. However, generally no constraints exist concerning the approach for the description and deployment of the workspaces.

6 Conclusion

Introducing SLA mechanisms to all levels of the Grid is a prerequisite for attracting the commercial user to use Grid environments. For the level of Grid fabric, the project HPC4U realized an RMS able to negotiate on SLAs, ensuring their fulfillment also in the case of resource outages by migrating jobs even to resources over the Grid. Since kernel-level checkpointing is used for realizing application transparent fault tolerance mechanisms, the target migration resource has to be highly compatible to the source resource. Due to the heterogeneous nature of the Grid, this requirement significantly limits the number of available spare resources.

The concept of virtual execution environments allows to execute a job in an environment, which complies to all requirements of a job. As long as a provider is able to execute a given virtual environment on the compute nodes of his cluster system, the system qualified as potential migration target. This increases the number of spare resources, thus also increasing the overall level of fault tolerance and reliability.

The mechanisms presented in this paper preserve the local autonomy of resource administrators, who still can decide which virtual system should be supported. Moreover already available resource querying mechanisms within the Grid can be used for finding resource providers supporting a specific virtual execution environment. SLA negotiation mechanisms and resource scheduling presented in this paper ensure that the migrated job successfully resumes at the target resource, so that no SLA is violated.

The system described here has been implemented in the scope of the Assess-Grid project. It uses the Globus toolkit and the OpenCCS resource management system. Currently, virtual execution environments are defined statically at provider level. In the next step, this static configuration will be superseded by establishing image repositories at Grid middleware level.

References

1. Priol, T., Snelling, D.: Next Generation Grids: European Grids Research 2005-2010 (2003), `ftp://ftp.cordis.lu/pub/ist/docs/ngg_eg_final.pdf`
2. Jeffery, K.(ed.): Next Generation Grids 2: Requirements and Options for European Grids Research 2005-2010 and Beyond (2004), `ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf`
3. De Roure, D.(ed.): Future for European Grids: GRIDs and Service Oriented Knowledge Utilities. Technical report, Expert Group Report for the European Commission, Brussel (2006)

4. Sahai, A., Graupner, S., Machiraju, V., van Moorsel, A.: Specifying and Monitoring Guarantees in Commercial Grids through SLA. Technical Report HPL-2002-324, Internet Systems and Storage Laboratory, HP Laboratories Palo Alto (2002)
5. Highly Predictable Cluster for Internet-Grids (HPC4U), EU-funded project IST-511531, `http://www.hpc4u.org`
6. Open Computing Center Software (OpenCCS), `http://www.openccs.eu`
7. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement) (2007), `http://www.ogf.org/documents/GFD.107.pdf`
8. Fallenbeck, N., Picht, H.J., Smith, M., Freisleben, B.: Xen and the Art of Cluster Scheduling. In: Second International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006) (2006)
9. Freeman, T., Keahey, K., Foster, I.T., Rana, A., Sotomoayor, B., Wuerthwein, F.: Division of labor: Tools for growing and scaling grids. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 40–51. Springer, Heidelberg (2006)
10. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM, New York (2003)
11. Keahey, K., Foster, I., Freeman, T., Zhang, X., Galron, D.: Virtual workspaces in the grid. In: 11th International Euro-Par Conference (2005)
12. Keahey, K., Foster, I., Freeman, T., Zhang, X.: Virtual workspaces: Achieving quality of service and quality of life in the grid. Sci. Program. 13(4), 265–275 (2005)