Implementation of Virtual Execution Environments for improving SLA-compliant Job Migration in Grids *

Dominic Battré, Matthias Hovestadt, Odej Kao
Technische Universität Berlin, Germany
{battre,maho,okao}@cs.tu-berlin.de

Axel Keller, Kerstin Voss
Paderborn Center for Parallel Computing (PC^2)
Universität Paderborn, Germany
{kel,kerstinv}@upb.de

Abstract

Commercial Grid users demand for contractually fixed QoS levels. Service Level Agreements (SLAs) are powerful instruments for describing such contracts. SLA-aware resource management is the foundation for realizing SLA contracts within the Grid. OpenCCS is such an SLA-aware RMS, using transparent checkpointing to cope with resource outages. It generates a compatibility profile for each checkpoint dataset, so that the job can be resumed even on resources within the Grid. However, only a small number of Grid resources comply to such a profile. This paper describes the concept of virtual execution environments and how they increase the number of potential migration targets. The paper also describes how these virtual execution environments have been implemented within the OpenCCS resource management system.

1 Introduction

Intensive research made Grid systems become a well accepted working instrument. However, Grids are mostly used in the academic domain. The commercial user demands for contractually fixed quality of service (QoS) levels, expressed by means of Service Level Agreements (SLAs) [1]. An SLA allows the definition of a job requirement profile that has to be fulfilled, e. g. the definition of a deadline that has to be met for successful job completion. This particularly puts high demands on local resource management systems (RMS), which provide their resources to the Grid.

The goal of the EC-funded project HPC4U [2] was the development of an SLA-aware RMS, capable of negotiating with Grid middleware components on new SLAs. Once SLAs have been agreed, the RMS is in charge of ensuring that the SLAs are fulfilled. Mechanisms have been developed for handling resource failures, preventing an SLA violation in such a case. The RMS OpenCCS [3] was used as foundation in this context.

The EC-funded project AssessGrid [4] introduced the notion of risk into all Grid layers. As the reliability of resources cannot be guaranteed at a 100% level, specifying maximally acceptable and providable failure probabilities in SLAs shall increase the trust of service consumers in the Grid and build a decision basis for them and providers about what is possible and to be expected.

The current state of the RMS allows jobs to be transparently checkpointed, which then can be migrated within the cluster, within the administrative domain, or even to arbitrary resources within the Grid. This allows the scheduler to cope with the consequences of a resource failure, or even to react precautionary if failure probabilities of a job increase.

Migration of jobs to remote cluster systems proved to be a valuable instrument for increasing a system's fault tolerance, since it allows the compensation of resource outages also in case of high utilization or severity. However, it presumes the compatibility of source and target resource, which also needs to be ensured in case of migrations over the Grid. Since most Grid systems have a high degree of heterogeneity in their resources, only a small subset of these represents eligible migration target candidates. To cope with these constraints we propose to migrate whole virtual execution environments.

In this paper we outline this idea of virtual execution environments. Instead of establishing a checkpointed execution environment directly in the operating system of a compute node (which is running for instance a RedHat distribution), it is established in a virtual machine (VM) running on the compute node thereby allowing to support arbitrary operating systems within the VM. This will significantly increase the number of potential migration targets and therefore also the overall level of fault tolerance.

In section 2 we will first outline the HPC4U system and describe the migration process of jobs. Afterwards the con-

*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).

1530-2016/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPP-W.2008.40

cept of virtual execution environments will be described in section 3. A presentation of related work follows after Section 4, which details the integration of virtual execution environments in the SLA-aware RMS. The paper will conclude with a short summary.

2 SLA-aware Resource Management

For the realization of an SLA-aware RMS, the project decided to use the planning-based RMS OpenCCS. A new scheduler has been implemented, assigning jobs to resources according to their SLAs [5]. 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. 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 [6].

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. The duration of a checkpoint operation only depends on the used main memory, typically not taking more than some few seconds.

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 communicates only within the cluster.

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 to 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 configured homogeneously.

When migrating to Grid resources, a dynamic mechanism is mandatory for selecting compatible resources. HPC4U introduced a compatibility profile, holding compatibility requirements of the checkpoint dataset on the target resource, ranging 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 compatibility profile has proven to be a good instrument in finding compatible resources. However, due to its fine grained requirements and the heterogeneous nature of Grid systems, even in large Grid systems only a small number of resources are potential migration targets. Since each difference in the operating system (e. g. the application of a system patch) 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

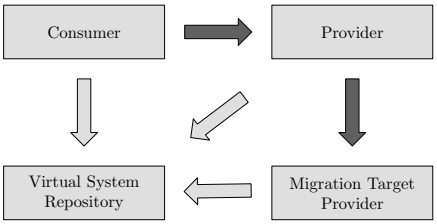

Figure 1. Stakeholders accessing the Virtual System Repository

started first. Within this virtual system arbitrary operating systems can be started. The application is then started within this well-defined virtual system environment.

The central component of this new infrastructure is a system image repository at Grid middleware level, holding system images of different operating systems, e. g. commonly used Linux distributions in different versions and different patchlevels. 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 practice of not knowing whether the job will be executed. The customer can test his job in the specified environment by establishing the image from the system image repository locally.

Figure 1 depicts the role of the virtual system repository in the system architecture. The Grid end-user negotiates with a Grid provider on resources. Since the end-user wants his job to be executed in a specific environment (e. g. Debian Etch), he looks up the Grid middleware virtual system repository, adding the ID of the particular VEE to the SLA negotiation process. For this, the provider has to enhance the SLA template, so that the user not only can request on number and type of resources, earliest start and finish time, but also on the VEE to be established.

If the job is to be executed, the provider looks up the same repository and establishes the user specified VEE. If the job then has to be migrated over the Grid, the provider adds the user-specified ID of the VEE to the SLA-negotiation process with the migration target provider. Since also this provider accesses the same repository, the migrated job checkpoint is restarted in the same user-specified operating system environment.

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 catalog.

At level of Grid middleware the customer requested virtual system ID has to be matched against the supported characteristics of the provider's resources, like it is already common practice 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 accepts 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 therein. 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, where the user also has no knowledge about the system used for executing his job.

At runtime, the virtual system is established on the compute node, so that all job checkpoints can be executed in a well defined and well known environment. As the system ID defines the image and therewith the libraries provided, all dependencies of a checkpointed job are met.

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.

3.1 SLA-Negotiation Aspects

Finding suitable resources and agreeing SLAs is based on WS-Agreement. WS-Agreement [7] is a proposed recommendation by the Open Grid Forum and enables the creation of SLAs in a Grid/Web Service environment. The SLA specifies the context of an agreement (participating parties, expiration, etc.), the definition of the service to be delivered and the guarantees given during the execution. SLAs are represented as WS-Resources and can be queried about their current status. An agreement factory is responsible for negotiating an SLA. It provides so called agreement templates that describe the outline of an agreement and can be filled with information from the consumer. This information comprises requirements about the machine that shall execute the job (number of cpus, speed, memory, etc.) and how to execute the job (path to executable, working directory, staging directions, etc.). This is represented by the Job Submission Description Language (JSDL) and the extension JSDL-SPMD (-Single Program Multiple Data).

Further extensions that did not undergo a standardization procedure specify the virtual system ID and scheduling restrictions. Guarantees need to be monitorable at runtime and comprise for example the memory actually available to the virtual machine.

Agreement factories expose their agreement templates as WS-ResourceProperties. This allows an end-user but also an RMS that needs to migrate a checkpointed job to look for providers that support a desired virtual system. Current Grid information services can support this task. After locating a suitable provider, the consumer requests the creation of an agreement that can be either accepted or declined. In case a provider accepts the agreement it has to establish the specified virtual system on the compute nodes and start the specified job or resume a checkpointed job. The provider guarantees the correct execution of the job and can be held liable in case of failing to do so. Of course a provider is capable of migrating a job to a different provider as well, using the same mechanisms described above.

4 Implementation in OpenCCS

The OpenCCS [3] RMS consists of a number of independent daemons which are interacting over one-to-one communication channels. Each of the daemons has its own specific tasks, either running on the front-end node of the cluster (or even one of the front-end nodes, if multiple nodes are used as front-end) or the compute nodes.

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.

First of all, the user interface had to be enhanced, allowing the user to specify a VEE to be established for the job at runtime. At the moment, this repository of VEEs is statically defined by means of a regular configuration file. This can be enhanced at a later time, e. g. by using a Grid-based VEE repository. The user specifies the ID of a VEE, passing it as an additional parameter at job submittion time. This ID is then saved as an additional parameter in the OpenCCS-internal job data structure.

The goal of keeping the existence of VEEs transparent for these components has been achieved, because no front-end component has to evaluate this new parameter. In fact, VEEs provide novel options for the RMS itself, e. g. establishing multiple virtual compute nodes on multi-core processor nodes. These enhancements require new scheduling algorithms, which will be implemented in the next step. In this initial step, we only instantiate a single VEE per node, to avoid implications here.

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 (i. e. the first Xen domain on the host system, which is able to start other virtualized Xen domains) starts the image of the specified VEE as DomU.

Once the DomU (i. e. a virtualized domain) 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+*.

The transparent integration into the OpenCCS architecture is depicted in the communication sequence diagram in Fig. 2, showing the basic message flow for job execution. Other signals (e. g. for executing checkpointing or restart) are handled analogously.

The session manager (SM) is a component running on the frontend-node, responsible for communicating with the compute node. If a compute node should be allocated for a new job, it signals to the NSM that the partition should be arranged (i. e. the node should be prepared that the job can be started afterwards). The sequence diagram shows that this command is received by the NSM.

This message contains the entire job description, in particular the information that the job should be started in a VEE. The NSM evaluates this information, switches to the proxy-mode, and starts the virtual system which is specified in the message. In this virtual system, a NSM+ is automatically started in VM-mode, immediately connecting to his proxy. The proxy-mode NSM+ then sends the *arrange partition* command to the VM-mode NSM+, who signals back as soon as the partition has been prepared. This message is then forwarded to the SM. Since the VM did not take any

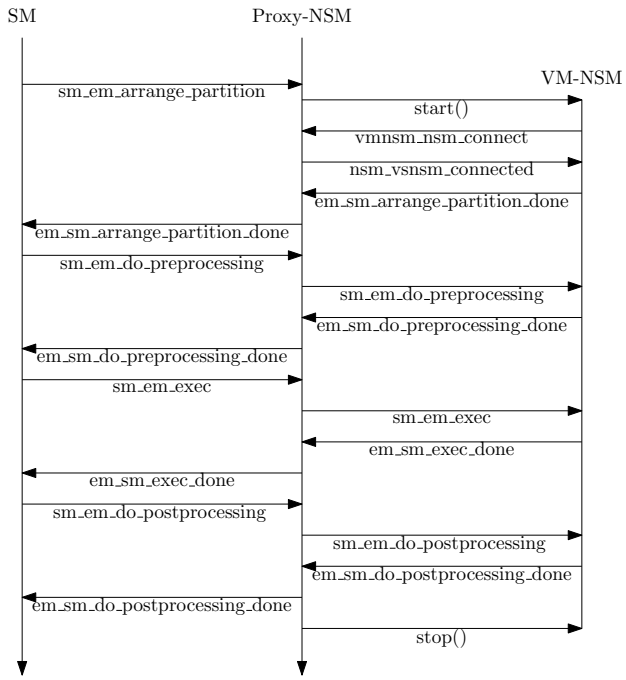

Figure 2. Communication sequence diagram

notice of the communication between the NSM instances, the existence of any virtual systems remains transparent.

Once the (virtual) node partition has been prepared for the job, the SM signals that the NSM should start the preprocessing phase (e. g. initialization of services required for the succeeding job) and finally to start the job itself. These commands are transparently forwarded by the proxy-mode NSM+ to the VM-mode NSM+. After the job has finished, the SM signals that the postprocessing phase should be invoked, which cleans up the node environment. This is also forwarded to the VM-mode NSM+. Since the allocation of the (virtual) node partition ends with the cleanup phase, the virtual system is no longer required, so that the proxy-mode NSM+ stops the virtual system, and turns back to a regular mode NSM.

This system is already implemented and allows users to request specific VEEs for the execution of their jobs. Within the VEE the jobs are then checkpointed, so that the system is able to restart in case of resource outages. In such a case, the RMS first allocates new nodes. Before the EM restarts the job from the latest checkpoint, the proxy-mode NSM+ first initializes the requested VEE, so that the restart can be executed in the same operating system context.

In case of job migrations to other cluster systems, the user requested VEE ID is part of the SLA negotiation process, so that the target RMS is also able to establish the same operating system context on its machines. However, due to the current lack of a Grid middleware image repository, the VEE configuration files of all cluster systems have to be synchronized manually.

5 Related Work

In [8], 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 [9]. 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 [10]. Their implementation uses VM-based workspaces configured with Xen hypervisors [11], 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 [12]. The deployment can be realized by using pre-configured VM images or refining configuration which influences the flexibility and deployment time. In [13] 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

The EC-funded project HPC4U introduced SLA-awareness to resource management system (RMS). Transparent kernel-level checkpointing mechanisms are applicable to arbitrary (even commercial) codes. If an SLA is in danger to be violated (e. g. a resource failed, but the job has to be completed until a given deadline), the RMS is not only able to restart the job from the latest checkpointed state on the same cluster system, but also to migrate it to other cluster systems in the same administrative domain, or even to any other ressource in a Grid system.

Since kernel level checkpointing mechanisms put strong requirements on the compatibility of the target system, the HPC4U system introduced a compatibility profile, describing all requirements of the checkpoint dataset on the target resource. This profile proved its ability to help selecting a compatible resource. However, due to its low-level requirements, even in large Grid systems only a small number of resources qualified as potential migration targets.

Introducing virtual execution environments (VEEs) on the compute nodes allows addressing the question of resource compatibility in a novel way. Instead of describing the requirements of a compatible environment, the compatible environment itself is established on a compute resource at resume time of a checkpointed job.

This paper describes the initial step in realizing such VEEs on the basis of the RMS OpenCCS and the Xen virtual machine monitor. Even if it does not yet unleash the full potential of VEEs, it already increased the number of potential migration targets. The implementation is transparent for most OpenCCS components and only requires a small uncritical enhancement in the job description.

The implementation of a Grid middleware image repository is subject of current work. Moreover, the compute node based mechanism of establishing virtual working environments is to be enhanced, supporting multiple virtual systems to be started on a single node. This enhancement then will also have implications on the system scheduling.

References

[1] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel, "Specifying and Monitoring Guarantees in Commercial Grids through SLA," Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, Tech. Rep. HPL-2002-324, 2002.

[2] "Highly Predictable Cluster for Internet-Grids (HPC4U), EU-funded project IST-511531," http://www.hpc4u.org.

[3] "Open Computing Center Software (OpenCCS)," http://www.openccs.eu.

[4] "Advanced Risk Assessment and Management for Grids (AssessGrid), EU-funded project IST-031772," http://www.assessgrid.eu.

[5] D. Battré, M. Hovestadt, O. Kao, and A. Keller, "Planning-based Scheduling for SLA-awareness and Grid Integration," in *26th Workshop of the UK PLANNING AND SCHEDULING Special Interest Group (PlanSIG2007)*, 2007.

[6] M. Hovestadt, "Fault Tolerance Mechanisms for SLA-aware Resource Management," in *Workshop on Reliability and Autonomic Management of Parallel and Distributed Systems (RAMPDS-05), at 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005), Fukuoka, Japan*, 2005.

[7] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement)," http://www.ogf.org/documents/GFD.107.pdf, 2007.

[8] K. Jeffery *(edt.)*, "Next Generation Grids 2: Requirements and Options for European Grids Research 2005-2010 and Beyond," ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf, 2004.

[9] N. Fallenbeck, H.-J. Picht, M. Smith, and B. Freisleben, "Xen and the Art of Cluster Scheduling," in *Second International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, 2006.

[10] T. Freeman, K. Keahey, I. T. Foster, A. Rana, B. Sotomoayor, and F. Wuerthwein, "Division of labor: Tools for growing and scaling grids." in *ICSOC*, ser. Lecture Notes in Computer Science, A. Dan and W. Lamersdorf, Eds., vol. 4294. Springer, 2006, pp. 40–51. [Online]. Available: http://dblp.uni-trier.de/db/conf/icsoc/icsoc2006.html#FreemanKFRSW06

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[12] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the grid," in *11th International Euro-Par Conference*, September 2005.

[13] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Sci. Program.*, vol. 13, no. 4, pp. 265–275, 2005.