Journal of Computer Science 2 (6): 550-557, 2006
ISSN 1549-3636

Operation of an SLA-aware Grid Fabric

Matthias Hovestadt
Paderborn Center for Parallel Computing, University of Paderborn, Germany

Abstract: Research on Grid Computing started under purely technical questions of how to realize the virtualization of distributed resources. Now, the commercial user should be attracted to use the Grid. Future applications of the Next Generation Grid will demand for flexible negotiation mechanisms supporting various ways of Quality-of-Service (QoS) guarantees. In this context a Service Level Agreement (SLA) is a powerful instrument for describing all obligations and expectations within a business partnership. Many research projects already focus on realizing SLAs within the Grid middleware. However this is not sufficient. Resource Management Systems also have to be SLA-aware, since these systems provide their resources to Grid infrastructures. In this study we present the EU-funded project HPC4U (Highly Predictable Clusters for Internet Grids), which aims at realizing such an RMS by means of SLA-negotiation and scheduling and application-transparent fault tolerance.

Key words: SLA, QoS, Grid computing, resource management, fault tolerance

INTRODUCTION

Nowadays, Grid Computing is not only a major research topic, it is also already in use. Many companies recognized the potentials, so that the application of Grid technology is not limited to research only. Companies like IBM, Hewlett Packard and Microsoft currently invest noticeable efforts on research and the support of research communities. Common goal of commercial and academic researchers is to attract commercial users for Grid Computing. In this context, the European Commission convened a group of experts to clarify the demands of future Grid systems and which properties and capabilities are missing in currently existing Grid infrastructures. Their work resulted in the idea of the Next Generation Grid (NGG)[1].

Among the demands on such an NGG are transparency and the guaranteed provision of reliability and a defined level of quality of service (QoS). Compared with the capabilities of current Grid systems, these are challenging demands. When using a Grid infrastructure today, the user gets best-effort services only. Obviously this is not adequate for commercial users, who want to use the Grid for computing business critical jobs. Such a user demands for guarantees, so that he can count on getting the requested and required service level. Assuming a weather service as commercial customer, it must be able to rely on getting the computed weekend weather forecast in time. This underlines the necessity of the NGG, not only focusing on predictable and reliable operation of a single Grid resource, but also on the orchestrated execution of a Grid workflow.

In this context, a Service Level Agreement (SLA) is a powerful instrument for describing all expectations and obligations in the business relationship between customer and service provider[2] (Fig. 1). Such an SLA specifies the QoS requirement profile of a job. At the layer of Grid middleware many research activities already focus on integrating SLA functionality.

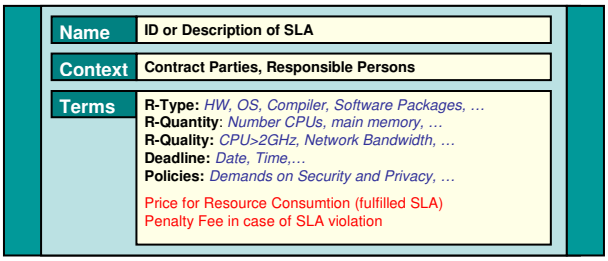

Fig. 1: Elements of an SLA

However, there is a gap between the requirements of an SLA-aware Grid middleware and the capabilities of currently available Resource Management Systems (RMS), offering only best-effort service. Local resource management systems provide their resources to Grid systems, which transfer jobs from Grid users to these provided resources. Hence, SLAs guaranteed by the Grid middleware must be realized by the local RMS [3]. However, if the local RMS operates at best-effort, Grid middleware can not assure specific service levels.

Within the project "Highly Predictable Cluster for Internet-Grids" (HPC4U)[4], which is funded by the EU, an SLA-aware RMS is developed. This RMS will utilize the service of process- storage- and network-subsystem for realizing a guaranteed level of application-transparent fault tolerance. The HPC4U project will allow the Grid to negotiate on Service Level Agreements. It will also feature mechanisms like process and storage checkpointing to realize fault tolerance and to assure the adherence with given SLAs.

Corresponding Author: Matthias Hovestadt, Paderborn Center for Parallel Computing, University of Paderborn, Germany

Related work: The worldwide research in Grid computing resulted in numerous different Grid packages. Besides many commodity Grid systems, general purpose toolkits exist such as UNICORE[5] or Globus[6]. Although Globus represents the de-facto standard for Grid toolkits, all these systems have proprietary designs and interfaces. To ensure future interoperability of Grid systems as well as the opportunity to customize installations, the OGSA (Open Grid Services Architecture) working group within the GGF[7] aims to develop the archi-tecture for an open Grid infrastructure[8].

In[1], important requirements for the Next Generation Grid (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 QoS level.

An architecture that supports the co-allocation of multiple resource types, such as processors and network bandwidth, was presented in[9]. The Globus Architecture for Reservation and Allocation (GARA) provides "wrapper" functions to enhance a local RMS not capable of supporting advance reservations with this functionality. This is an important step towards an integrated QoS aware resource management. In our study, this approach is enhanced by SLA and monitoring facilities. These enhancements are needed in order to guarantee the compliance with all accepted SLAs. This means, it has to be ensured that the system works as expected at any time, not only at the time a reservation is made. The GARA component of Globus currently does neither support the definition of SLAs or malleable reservations, nor does it support resilience mechanisms to handle resource outages or failures.

The requirements and procedures of a protocol for negotiating SLAs were described in SNAP[10]. 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 this study.

The inadequacy of current systems for emerging Grid requirements has been underlined by MacLaren *et al.*[11]: neither the best-effort approach of batch schedulers nor the inflexible nature of advance reservation is suitable for future RMS. Mechanisms are required which support SLAs, "negotiated between the client (user, superscheduler, or broker) and the scheduler". Unlike other approaches for providing SLA-awareness and service quality guarantees, MacLaren *et al*. propose the development of novel RMS scheduling mechanisms, which are able to negotiate with SLA-requesting customers coming from Grid. Similar ideas and demands also have been presented in[12]. In a subsequent study to[11], Yarmolenko *et al*. evaluated policies for negotiation with resources[13]. The study does not focus on local RMS, but on a central service at Grid middleware, acting as a broker between user requests and available resources. However, solely the RMS has full control over the resources, therefore questions on fault tolerance and QoS provision can not be answered only at level of Grid middleware. Hence, the presented mechanisms form a complementing counterpart to the architecture presented in this study.

The Grid community has identified the need for a standard for SLA description and negotiation. This led to the development of WS-Agreement/-Negotiation[14]. These upcoming standards rely on the new Web Services Resource Framework (WSRF[15]) which will supersede the OSGI specification. We will follow these developments closely and will stick to these standards.

Architecture of HPC4U: The goal of the HPC4U project (Highly Predictable Cluster for Internet Grids) is to provide an application-transparent and software-only solution of a reliable RMS. It will allow the Grid to negotiate on Service Level Agreements and it will also feature mechanisms like process and storage checkpointing to realize Fault Tolerance and to assure the adherence with given SLAs. The HPC4U solution will act as an active Grid component, using available Grid resources for further improving its level of Fault Tolerance.

The HPC4U results will provide Next Generation Grids with the possibility to guarantee the completion of Grid jobs and leverage the larger uptake of Grid environments. The HPC4U software will be customizable and interoperable with other Grids and will open new perspectives to the usage of Grids for additional services as they are today strongly required by the industry. HPC4U will extend well accepted technologies and integrate them with innovative features (such as Grid embedded Fault Tolerance), for all the components required for a dependable Grid (storage, communication, resource management).

The outcomes of HPC4U will be a mix of open source and proprietary software embedded in two outcomes. The SLA-aware and Grid-enabled RMS includes SLA negotiation, multi-site SLA-aware scheduling, security and interfaces for storage, checkpointing and networking support. It will be multi-platform in nature and available as open source. The second HPC4U outcome will be a vertically integrated commercial product with proprietary Linux-specific developments for checkpointing, networking and storage. This outcome will demonstrate the ready-to-use HPC4U functionality (job checkpointing, migration and restart) for Grids based on Linux architectures.

Resource management system: As it has been stated above, the RMS plays a central role within the HPC4U architecture. Since it is an SLA-aware RMS, it has to keep an overview about all SLA-related activities within the system. First, it has to negotiate with customers on new SLAs. These users may be located somewhere in the Grid system, accessing the SLA-negotiation interface of the middleware system. The RMS will first decide on starting a negotiation process (a negotiation request may be rejected due to local policies), then actively negotiating on the contents of the requested agreement. In this negotiation process, the current system condition has to be considered.

Therefore it is necessary, that the RMS has the complete overview about the resources in its own domain (the compute cluster which is operated by the HPC4U system). This encompasses available resources (e.g. number, type and equipment of compute nodes, topology and characteristics of the interconnect between these compute nodes, or characteristics and capabilities of the available storage system). Beside these static aspects, the RMS also has to be aware of dynamic attributes. The validity period of such dynamic attributes normally is really short, as these aspects represent the current condition of the overall system.

The RMS is also responsible for planning not only the current system usage, but also the future. Therefore it holds a schedule of all accepted SLA-jobs. According to this schedule, the general static information, the dynamic information representing the current system condition and the requirements of the new SLA request, such a request will be accepted or rejected. It is important to stress, that the RMS is the only element within the HPC4U architecture which has direct contact with the Grid system. The subsystems of HPC4U are only contacted by the RMS, but not from the Grid user.

To be able to plan requests with assigned SLAs (e.g. a guaranteed minimal bandwidth) an RMS scheduler not only has to count free resources (e.g. compute nodes), it also has to respect system specific constraints like the topology of a high speed network. The RMS used in HPC4U does this by dividing the scheduling process into two parts, a hardware-dependent called Machine Manager (MM) and a hardware-independent part called Planning Manager (PM). This separation allows to consider system specific requirements (e.g. location of I/O-nodes or network topologies) at which the MM part may be adapted to different resource configurations without changing the basic scheduling part (the PM). The MM verifies whether or not a schedule computed by the PM can be realized with the available hardware. The MM checks this by mapping the user given specification with the static (e.g. topology) and dynamic (e.g. node availability) information on the system resources. Information provided by the subsystems are incorporated in this mapping procedure. If the MM is not able to find an SLA conform mapping of the jobs onto the resources at the time scheduled by the PM it tries to find alternatives. The resulting list is sent back to the PM which accepts the schedule or computes a new one based on the schedule given by the MM.

SLA-aware RMSs are responsible for fulfilling the contents of all SLAs, which have been acknowledged. This implies that the RMS has to take appropriate actions in case of resource outages (e.g. power failure of a compute node). Hence, jobs have to be supervised during their whole lifetime. For this purpose, the MM monitors the running jobs and the affected resources. In case of an error (e. g. a resource failure) the MM is able to migrate the job to another matching resource. Since the MM always knows the current schedule this may be done without violating the current schedule. In HPC4U, the RMS will utilize the functionalities of the underlying HPC4U subsystems to provide fault tolerance, thus guaranteeing the adherence with the negotiated SLAs. In situations where such a conflict situation can not be resolved internally (due to resource failures the system does no longer have sufficient resources to match with all agreed deadlines), the HPC4U system can benefit from the available Grid infrastructure by requesting resources from the Grid. By knowing the requirement profile defined in the SLA, the RMS tries to find appropriate resources at a remote Grid site. If such resources were found, the system may start a job migration process to this remote resource, such that the agreed deadline can be held.

The RMS also has interfaces to the three HPC4U subsystems mentioned above. These are located in the MM part of the RMS since the MM controls the execution of jobs. Using these interfaces the RMS may register callback routines which are called in case a subsystem notices an error and is not able to solve the problem alone. For instance we assume a job with an SLA guaranteeing a minimal bandwidth. It may now happen that due to a resource failure (not necessarily used by the concerned application) the networking subsystem is forced to change its routing tables. The subsystem tries to fulfill the SLA since the RMS started the application with this constraint. If the networking subsystem is not able to keep the agreed minimal bandwidth it informs the RMS about the problem. The RMS then has to decide what to do: migrating the concerned application or suspending another one.

The RMS consists of numerous sub-components. The service of all components has to be orchestrated, so that the adherence with agreed SLAs can be ensured. Within this architecture, the Planning Manager is of central importance. In the following, the tasks and interaction between these blocks is to be explained.

In the design of the RMS, a compromise has to be found between two conflicting goals: on the one hand the design of the RMS should utilize available resources (e.g. processors, network and storage) optimally. On the

other hand, the system should not be tailored to one specific configuration or technology, but be able to be deployed and ported to arbitrary usage scenarios.

HPC4U provides novel mechanisms to the RMS. Instead of cancelling a job, the RMS can use the fault tolerance mechanisms of the HPC4U subsystems to ensure the adherence with the accepted SLA.

Process subsystem: The HPC4U process subsystem will include an operating system-driven check-pointing. The process subsystem will create a "virtual bubble" around the process, presenting a virtual environment, e.g. consisting of virtual network devices and virtual process IDs. This virtual environment allows the checkpointing of a process without the necessity of linking additional special purpose libraries into the environment. It is remarkable, that the virtual bubble has only minimal impact on the runtime of a job. Since no recompilation is required, arbitrary applications can be checkpointed and benefit from an increased level of fault tolerance. Hence, also in commercial Grid environments, the RMS can guarantee the adherence with given deadlines.

If a checkpoint has to be created, the process subsystem checkpoints this bubble completely. Since no recompilation of applications is necessary, this checkpoint process is transparent for running applications, so that arbitrary applications can be checkpointed. Even commercial applications (for which the source code is normally not available) can benefit from this mechanism. This is a major advancement compared to already existing solutions. Most of them do not support (MPI-) parallel applications or require that the application is linked against specific libraries (e.g. Condor[16]). This prevents the general applicability as well as the application-transparency of fault tolerance mechanisms. Other solutions do not require relinking, but do not provide full virtualization of resources (e.g. Berkeley Lab Checkpoint/Restart[17]) which hinders the migration of checkpointed applications to other resources.

If checkpointing parallel jobs, the process subsystem will utilize mechanisms of the network subsystem to ensure that also packets currently transmitted over the network are checkpointed.

A major task within the checkpointing subsystem is the retrieval of compatible resources for a given checkpoint. Only if the target system is compatible to the source system, where the checkpoint has been generated, the checkpoint will be able to restart. This question does not only affect issues like processor type or kernel version. Since an application normally accesses shared libraries, also these libraries must be available and compatible. If the RMS tries to resume a checkpointed application on a non-compatible node, at best the application would crash immediately at best. At worst, an incompatibility between two library version would not crash the application, but causing in incorrect results.

Therefore, the checkpointing subsystem will generate a requirement profile of each checkpointed job. In such a profile all relevant characteristics of a job are stated, e.g. CPU type, kernel version and versions of loaded libraries. This profile will then be used by the RMS for querying for compatible resources. These resources may be located on the same cluster system, but also on different cluster systems within the same administrative domain, or even somewhere in the Grid.

Storage subsystem: In the case of resource outages, the HPC4U system will restart affected jobs on suitable spare resources using previously generated process checkpoints. Since a job may constantly change its storage partition, inconsistencies between the resumed process and the storage partition may occur (e.g. data has been written since the last process checkpoint, so the job's storage partition is at a different stage as the restarted job). Therefore the storage subsystem will allow to checkpoint the job's storage partition.

If the job is resumed on a spare resource, the storage partition is restored from the checkpointed state. This way, both checkpoints (checkpoint of the process and checkpoint of the storage partition) are consistent.

The HPC4U project does not only focus on fault tolerance on a single cluster, but also on multiple clusters and even the Internet Grid. Hence, a job may be resumed on resources of a completely different cluster system. At this, not only the data of the checkpointed process has to be transferred to the new system, but also the according checkpoint of the storage partition. As storage partitions may be huge, this might be a time consuming process which would prevent to meet the agreed deadline of a job. Therefore the storage subsystem will provide mechanisms for a background replication of storage data to remote systems.

Since HPC4U will support migration to Grid-resources. As storage partitions may be huge, this might be a time consuming process which would prevent to meet the agreed deadline of a job. Therefore the storage subsystem will provide mechanisms for a background replication of storage data to remote systems.

Network subsystem: The commercial outcome of the HPC4U project will use the SCI network[18] as interconnect between the nodes of a cluster system. The network subsystem will enhance the network management of this SCI network and supports other HPC4U subsystem in providing fault tolerance.

As described above, the process subsystem will create checkpoints of running applications. The network subsystem will ensure that also packets are saved which are transmitted over the network at the time of the checkpoint. This way, the created checkpoint is consistent over all nodes of the job. Furthermore, the

network subsystem will provide sophisticated mechanisms to the process subsystem, which can be used for checkpointing MPI-parallel jobs.

The RMS will query the network subsystem for static and dynamic information about the network, e.g. basic network topology or currently available network bandwidth. This information is necessary for SLA negotiation and scheduling. At runtime, the network subsystem will monitor the consumed network bandwidth of a job and enforce bandwidth limits, if specified. Moreover, it will allow the RMS to separate the SCI network into virtual subnets.

Phases of operation: The main goal of HPC4U is to provide future Next Generation Grids with reliable and predictable compute resources. To be interoperable with Grid middleware environments, thus providing an application transparent service to Grid customers, the HPC4U system will provide an interface to Grid middleware.

This interface will enable the Grid to negotiate on an Service Level Agreements. If the HPC4U cluster middleware has agreed to provide a fixed service level, it will realize this level with mechanisms like process checkpointing and storage snapshotting. The HPC4U solution will act as an active Grid component, using available Grid resources to further improve its level of Fault Tolerance.

As the HPC4U system will be integrated into existing Grid middleware systems, adherence to existing standards and protocols is vital. To provide maximum flexibility and compatibility with other research projects and software systems, HPC4U will follow well-defined standards developed by the GRAAP (Grid Resource Allocation Agreement Protocol) working group of the Global Grid Forum (GGF). This will ensure that HPC4U achievements can be integrated into Grid middleware systems like the Globus Toolkit or UNICORE.

Beside the computation of a job on an allocated resource, also other phases of operation can be identified (Fig. 2), where the system handles the job. Within the first phase, both parties (the HPC4U cluster middleware system and the Grid customer) try to agree on the contents of an SLA, which defines the requirements of the customer's job.

In the next phase, the pre-runtime phase, the validity period of the SLA has not actually started, the system has to prepare itself for this new job. This means, that the network has to be configured, the assigned compute nodes have to be initialized and the storage has to be provided.

The main phase of operation is the runtime phase, which starts at the beginning of the validity period of the SLA. First, all input data is transferred from the Grid customer to the compute resource, followed by the computation of the job. At this, the HPC4U cluster

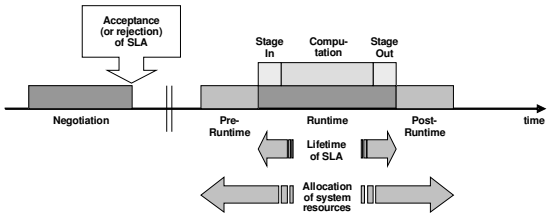

Fig. 2: Phases of operation in HPC4U

middleware has to ensure the compliance with all QoS statements of the SLA. If the computation has finished, result data will be transferred back to the customer (stage out).

After the validity period of the SLA has ended, the job has terminated and all output data has left the system, the post-runtime phase starts. In this phase the system can be reconfigured to "normal" operation.

Negotiation: If the Grid user wants to negotiate on resource usage with the HPC4U system, the user first submits a negotiation request, which reaches the HPC4U system via the upper layer interface. The HPC4U system now decides if it wants to accept this request, thus starting an SLA negotiation with the requesting Grid user. If HPC4U declines the request, no SLA negotiation will start. The reason for this cancellation may be manifold, e.g. if the requestor is not member of a trusted domain or if accounting is not guaranteed. This decision process will be policy driven.

In case the HPC4U system accepts the request, the whole negotiation process is steered by a negotiation module within the RMS, as only the RMS has a complete sight about all resources and resource conditions within the RMS administrative domain (AD). Such an AD may be seen as a pool of resources with the same set of policies, describing which resources are visible and accessible. ADs may be nested. Policies may also be tailored to specific user groups, e.g. providing access only to users from specific domains.

The negotiation module now initiates a new SLA negotiation with the requesting Grid user by creating a Grid service for this specific negotiation instance. This Grid service creates a template for an SLA and returns it to the requestor, together with the endpoint reference of the recently instantiated negotiation Grid service. The SLA template represents a general framework for all further negotiation activities. It gives a formal structure for negotiation as well as Service Description Terms (SDT), which may be topic of the negotiation process.

The requestor now starts the SLA negotiation by utilizing the received SLA template. He creates an SLA request based on this template which specifies all his resource and QoS requirements and transmits it back to the HPC4U system.

Now, the negotiation module of the RMS is in charge of verifying the statements of the received SLA

request. This is done in a first step by checking formal requirements, e.g. constraints on available resources. If static boundaries meet existing resource limits, the negotiation module checks in a second step dynamical aspects of the received SLA request. This affects the question, whether or not the specified and requested resource is available at the requested quality at the requested time frame.

According to the specifications of the SLA request, the storage and network subsystem are included in this process. In contrast, the availability is a static property of a cluster node. The cluster configuration of the HPC4U system specifies, whether or not the checkpointing subsystem is available on given nodes. Hence, the RMS can determine without any communication to the checkpointing subsystem, if checkpointing mechanisms can be provided.

It is important to stress, that the specific requests on QoS can not be regarded isolated from each other. In fact, they interfere each other. In case of a deadline bounded job, the storage subsystem has to provide additional storage capacity for saving process checkpoints. QoS requests even interfere within a single subsystem: in case more than one storage snapshots shall be saved, the storage capacity has to be aligned.

Pre-runtime phase: Each negotiated SLA has a unique identifier which is used as reference in further communication. In case of a successful negotiation procedure, the SLA is saved in the SLA management of the HPC4U middleware system.

As the HPC4U system has an SLA-aware scheduler, the contents of all negotiated SLAs are part of the scheduling process. Hence, HPC4U awaits the incoming job and assigns appropriate resources within the HPC4U domain. To utilize these resources, further communication concerning the SLA-bounded job always refers to the unique identifier of the negotiated SLA.

At the moment where the new job enters the system (this does not only imply the executable of the application that should be executed, but also all input data required by this application), the components already have to be fully prepared for this new job. This process of initialization is denoted as pre-runtime phase.

It is important to consider this phase in the scheduling process, as tasks may be time consuming due to their complexity or communication intensity. This task may range from the provision of specific compilers up to an entire node environment.

Comprising, in this phase the RMS initializes the environment for the job execution, i.e. the configuration of local cluster nodes, the network subsystem and the storage subsystem. The emphasis of this process is on establishing a fault tolerant environment, according to the specifications of the SLA.

Runtime phase: After this first phase, the basic environment of the job has been established. Now, the stage-in of process data can proceed. Stage-in is performed by mechanisms at the level of Grid middleware, e.g. the Globus toolkit. Hence, the HPC4U system does not have to provide these mechanisms on its own. These stage-in mechanisms ensure that the process will find its data at runtime. The HPC4U system only has to ensure that storage capacity is available at the agreed quality and quantity.

Once all necessary data has been staged in, the computation of the job may start. The main task of the HPC4U cluster middleware is to ensure that all resources are available as agreed in the SLA during runtime. Therefore monitoring mechanisms have to be used, checking if all resources are operating in normal mode. If resource outages occur (e.g. dropout of a compute node), appropriate failure tolerance mechanisms have to be enforced. Due to adherence reasons with all SLAs, it is of vital importance, that resource failures are detected as soon as possible.

For this purpose, the RMS has to checkpoint the process, respectively snapshot the storage partition in regular intervals. At this, it utilizes the mechanisms of the underlying HCP4U subsystems. Normally, the checkpointing will be executed together with snapshotting, so that consistency between process and its storage is ensured. However, both mechanisms can be executed separately, if required. The time and frequency of checkpoints/snapshots is calculated by the RMS, depending on the contents of the agreed SLA.

These checkpoint and snapshot datasets will be used in case of failures for process migration. Depending on the type of migration (e.g. migration within the same cluster system, migration to a different cluster within the same AD, or migration to a cluster system somewhere in the Grid) specific requirements arise. If migrating within a single cluster system, only the checkpoint dataset has to be moved to a suitable spare resource. In other scenarios, compatibility, security and enforcement of policies have to be observed.

The RMS may use its checkpointing and snapshoting subsystems also in case of anticipated failures. If the monitoring subsystem of the RMS detects such an anticipated failure, the running job may be checkpointed before a resource outage actually occurs. The checkpoint/snapshot dataset then may be transferred to a suitable spare resource, where the job may resume. This way, no computational results are lost, since the job is transferred to spare resources, before a resource outage would have caused the loss of results.

Checkpoint/snapshot mechanisms are also valuable instruments for increasing the efficiency of system management. If the administrator of a resource decides

to switch off a compute resource (e.g. due to maintenance reasons), he may manually trigger the checkpoint of the application running on this node. Thus, the administrator benefits from these mechanisms by means of increased flexibility. He neither has to align his maintenance schedule to the current load profile of his machine, nor does he have to block resources for specific maintenance intervals. This results in a higher utilization of the machine, as the machine is not blocked and can be used by arbitrary jobs, even if these jobs can not be finished before the planned maintenance interval. As a matter of fact, maintenance is not the only thinkable reason for manually triggering the checkpoint/snapshot of a running job.

These mechanisms may also be used regularly by the RMS to increase the system utilization. With these mechanisms the RMS may use free slots between reservations for computing other (low priority) jobs, even if this slot is not sufficient for job completion. If such a low priority job is running on a gap timeframe, it may be checkpointed shortly before the end of the gap timeframe. This way, the resource is available for the reserved timeframe. As soon as another gap is available again, the low priority job will be restarted again within this new gap timeframe. This procedure may also be used for running jobs without runtime estimation. Such jobs can be checkpointed and resumed as long as job completion is achieved. However, the effort for checkpointing/snapshotting and resuming has to be kept in mind. The smaller a used timeframe is, the less useful it is. Normal or high priority jobs without runtime estimation, which should be run to completion, should run on reserved timeframes, which have a reasonable size, so that the overall progress of the job can be assured.

The primary goal of all fault tolerance mechanisms is the successful completion of a job. The result of a successfully completed job normally is a result dataset. This dataset has to be transferred back to the Grid customer, the owner of the completed job. This transfer is done in the stage-out process, which concludes this phase. Just like the stage-in process, the stage-out process will be performed by mechanisms of Grid middleware.

Post-runtime phase: The post-runtime phase is the last step of resource consumption. At this point, the computation of the job has finished and all result data has been transferred back to the service client. This phase is the counterpart of the pre-runtime phase, since specific configuration of the cluster system may have to be revoked. This reconfiguration does not only affect the configuration of the compute nodes, but also the configuration of the storage or network subsystem. Furthermore checkpoint/snapshot datasets can be removed, since the job has been completed and these datasets are not required anymore.

Another important task of the post-runtime phase is the analysis of the job runtime. As all monitoring data is available at this point, a concluding analysis of these logs can be accomplished. Goal of these checks is to determine if all specifications of the SLA have been fulfilled. In case of resource outages, it has to be checked if the RMS has reacted as agreed. It is important to emphasize that the HPC4U cluster middleware is not able to agree the completion of a job, since a job may also a fail due to failures within the application. HPC4U can only assure the provision and utilization of certain mechanisms to improve the overall QoS level for the respective job.

CONCLUSION

In this study we have outlined the basic ideas and components of an HPC4U cluster middleware system. HPC4U's main components are the SLA-aware RMS and the subsystems for realizing fault tolerance on process, storage and network.

The goal of the HPC4U project is to provide an application-transparent and software-only solution of a reliable RMS. It will allow the Grid user to negotiate on Service Level Agreements, which will be realized by means of process and storage checkpointing and other sophisticated mechanisms. By this, the HPC4U cluster middleware will be an important building block for realizing future Next Generation Grids.

However, the HPC4U solution will not only passively accept resource requests from Grid users, it will also act as an active Grid component. If the HPC4U system can not compensate resource outages, so that the fulfillment of agreed SLAs is endangered, it may request the Grid for suitable spare resources. If such resources are found, the job will be transparently migrated. This way, available Grid resources are used for further improving the level of Fault Tolerance.

Currently the HPC4U project is within the second of four technological workpackages. This workpackage addresses the first of three major steps in building up this system, namely the realization of the needed fault tolerance extensions to storage, communication and system software in a single node environment. The development and implementation of these basic mechanisms serve as a fundament for merging single nodes into an Intranet Grid and then for including the Intranet Grid into the world wide Grids. The core tasks in this workpackage are related to preparing the building blocks for a grid-wide job migration and have the main goal, to integrate the job checkpointing with the storage and resource management component. Furthermore, a RMS monitoring mechanism will collect information about the available resources and their status and publish this via RMS.

The consistent realization of this vertical approach is based on existing software solutions of the HPC4U partners. A first prototype implementation of our architecture has already been finished. It enables the user to request for a fault tolerant handling of his single-node running jobs. The HPC4U system starts such a job within a virtual bubble, using the subsystems for transparent checkpoint and migration within the same cluster system. Ongoing work within HPC4U focuses on providing checkpointing and migration also to parallel-node jobs and the realization of inter-cluster Grid migration.

Within the scope of the succeeding workpackage the existing FT mechanisms of the HPC4U system will be extended to multi-node/Intranet Grid environments on the one hand and to distributed running multi-node jobs on the other hand. The extension to Intranet Grids means, that the RMS must find suitable resources within this domain as a target for the job migration. This includes the suitability of the software and hardware architecture, the availability of the required resources and the compliance with the existing SLAs. Thus, for each migrated job, a start time for resumed processing will be assigned in a way that the given deadline can be reached, if the process duration information supplied by the user is correct.

The second extension to multi-node jobs is a large scientific challenge, as already existing mechanisms are limited to single-node jobs. The migration of multi-node jobs affects the checkpointing, migration and restart mechanisms on job-, storage- and communication-level, which must be able to deal with the specific characteristics of a multi-node job. The RMS has to be capable of handling multi-node jobs, since new requirements arise for compatibility, portability and migration. The HPC4U system resulting of this workpackage will be capable of cross-border migration, allowing an RMS to migrate jobs on resources within the own administrative domain or over multiple administrative domains. This will further increase the Fault Tolerance, as (temporarily) HW/SW/Network failures can be compensated with a higher probability, as the pool of appropriate resources is significantly enlarged by all cross-border resources.

REFERENCES

1. Bal, H. *et al*., 2004. Next generation grids 2: Requirements and options for european grids research 2005-2010 and beyond. ftp.cordis.lu/pub/ist/doc/ngg2_eg_final.pdf.
2. Sahai, A. *et al*., 2002. Specifying and monitoring guarantees in commercial grids through SLA. Internet Systems and Storage Laboratory. HP Laboratories Palo Alto, Tech. Rep. HPL 2002-324.
3. Burchard, L.-O. *et al*., 2004. The virtual resource manager: An architecture for SLA-aware resource management. 4th Intl. IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGrid), Chicago, USA.
4. Highly Predictable Cluster for Internet Grids (HPC4). EU-funded project IST-511531, http://www.hpc4u.org.
5. UNICORE Forum e.V. http://www.unicore.org.
6. Globus Alliance: Globus Toolkit. http://www.globus.org.
7. Global Grid Forum. http://www.ggf.org.
8. GGF Open Grid Services Architecture Working Group (OGSA WG), 2003. Open Grid Services Architecture: A Roadmap. http://www.ggf.org/ogsa-wg.
9. Foster, I. *et al*., 1999. A distributed resource management architecture that supports advance reservations and co-allocations. 7th Intl. Workshop on Quality of Service (IWQoS), London, UK.
10. Czajkowski, K. *et al*., 2002. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. Job Scheduling Strategies for Parallel Processing. 8th Intl. Workshop, Edinburgh, U.S.E.D.G. Feitelson, L. Rudolph, Ed..
11. MacLaren, J. *et al.*, 2002. Towards service level agreement based scheduling on the grid. 14th Intl. Conf. on Automated Planning and Scheduling (ICAPS04), Whistler, B.C., Canada.
12. Heine, F., M. Hovestadt and O. Kao, 2004. HPC4U: Providing highly predictable and SLA-aware clusters for the next generation grid. 4th Cracow Grid Workshop, Cracow, Poland.
13. Yarmolenko, V. *et al*., 2005. SLA based job scheduling: A case study on policies for negotiation with resources. 4th All Hans Meeting, Nottingham, UK.
14. Andrieux, A. *et al.*, 2004. Web services agreement specification (WS-Agreement). http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf.
15. Czajkowski, K. *et al.*, 2004. The WS-resource framework. http://www.globus.org/wsrf/specs/ ws-wsrf.pdf.
16. Condow Project. http://www.cs.wisc.edu/condor.
17. Berkeley Lab Checkpoint/Restart. http://ftg.lbl.gov/CheckpointRestart.shtml.
18. Dolphin Interconnect Solutions Inc., 2006. http://www. dolphinics.com/products/.