Grid Checkpointing Service - integration of low-level checkpointing packages with the Grid environment

Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak
{gracjan, radekj, Rafal.Mikolajczak}@man.poznan.pl

Poznan Supercomputing and Networking Center
61-704 Poznan, Noskowskiego 12/14, Poland

Jozsef Kovacs
smith@sztaki.hu
Computer and Automation Research Institute of Hungarian Academy of Sciences
1111 Budapest Kende u. 13-17. Hungary

CoreGRID Technical Report
Number TR-0159
June 16, 2008

Institute on Grid Information, Resource and Workflow Monitoring Services

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265
Abstract

The technology that significantly supports the load-balancing and fault-tolerance capabilities in Grids is the job’s checkpointing mechanism. Nevertheless, contemporary Grid environments distinctly lack the possibility of integration with the low-level processes’ checkpointers. Nowadays, some Grids support the checkpointing of applications which internally implement this functionality and additionally adhere to the imposed interface. On the contrary, the paper describes the Grid Checkpointing Service, the prototype design and implementation of Grid-level service, which makes it possible to utilize low-level or legacy checkpointing packages in Grids. In fact, the presented service is a proof-of-concept implementation of a part of the Grid Checkpointing Architecture (GCA). Nevertheless, the way the GCS was implemented allows it to be installed and utilized independently of the other parts of the GCA.

1 Introduction

The checkpointing mechanism provides the functionality of storing the state of computing processes into images. The data stored in the image can be used to resume the computing process to the same state the process had when the image was created.

To allow the Grid Resource Broker to take advantage of the checkpointing functionality, the checkpointing tools have to be integrated with the Grid environment. The Grid Resource Broker has to be aware of which jobs can be checkpointed while the checkpointing tools have to expose the Grid level interface. Additionally, the storage and accessibility of the created images have to be managed.

In general, there are two most common approaches to integrating the checkpointing functionality with the Grid. The first one is to embed the checkpointing functionality within the computing process and additionally provide it with a well-defined Grid-level interface that would allow external services to utilize this functionality. The second one is to provide the Grid-level service which would act as a kind of a driver to underlying low-level checkpointing

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).
tools, including those based on user- and application-level approach. The former approach is addressed by the OGF’s GridCPR [1] [14] [15] group while the latter is the one that the GCA [2] [3] [16] and the present paper deal with.

The aforementioned GCA architecture has been designed within one of the Virtual Institutes of the CoreGRID Network of Excellence [4]. The architecture specifies the set of Grid Services, design patterns and tools that make it possible to integrate the already existing [5] [6] as well as the future checkpointing packages with the Grid-oriented computing environment. The underlying checkpointing package can be of almost any kind, i.e. cooperation with kernel-level, user-level and application-level check pointers is possible. Cooperation with the VMMes (Virtual Machine Manager) that allows suspending and resuming the state of the whole virtual machine has also been taken into account.

To validate the feasibility and sanity of the GCA’s assumptions, the selected parts of the architecture have been developed in a form of proof-of-concept implementations. The paper presents one of such implementations named Grid Checkpointing Service (GCS). It is important to understand that the GCS is not the exact implementation of a part of the GCA. The GCS validates mainly the key assumptions of the GCA’s component named Checkpoint Translation Service (CTS); however, as such it is implemented in a form of an independent system that can be deployed and utilized within the GRMS [7] and Globus [8] based environments.

Please note that owing to a number of abbreviations and terms used in the paper, the most important ones are collected and briefly defined in the last section at the end of the paper.

2 Architecture

This section discusses the main components of the GCS and their mutual relationships.

2.1 Architecture’s outline

![Figure 1: Logical perspective of the GCS architecture](image)

The overall logical perspective of the presented system is depicted in Figure 1. The main elements that comprise the GCS are: the WSRF [9] based checkpointing service and execute- and resume-wrapper. These three components form the backbone of the architecture. Both wrappers are intended to be executed by the Grid Resource Broker in place of the original computing tasks. The wrappers perform some preliminary work and then execute or resume the original computing job. Most of the work performed by the wrappers is the same for all types of checkpointers and the checkpointer-dependent scripts are executed at the end of the wrappers. The actual Grid Resource Broker used in the presented implementation is GRMS. The GRMS submits the execute- or resume-wrapper on a destination cluster with the help of the GRAM service, which in turn executes the wrapper on assigned execute-node using the underlying Local Resource Manager (LRM). More information concerning wrappers responsibility is presented in sections 4.1 and 4.3. The component that receives and then forwards the Grid Resource Broker’s checkpoint requests is WSRF-compatible Checkpoint Service (in short Checkpoint Service). The way the GRMS acquires the information about the location of the adequate Checkpoint Service is presented in section 4.1. For now it is important that the service is able to forward the checkpoint-request to the checkpoint-invoker that is dedicated to a particular underlying
low-level checkpointer. After the checkpoint is taken, the Checkpoint Service is in charge of storing the checkpoint image in an external repository and remembering the location of the image with the help of Replica Location Service (RLS) [10]. The aforementioned wrappers and the Checkpoint Service utilize the set of auxiliary tools (see the lowest rectangle in Figure 1) which comprise the ones implemented especially for GCS as well as the common pre-existing Grid-level (e.g. GridFTP [11], OGSA-DAI [12]), and system-level tools (e.g. ssh, sudo). Additionally, the set of supporting components which are required to manage the images repository, access to this repository and the flow of checkpointing related meta-data is assumed to be deployed in the operating environment.

A more detailed deployment perspective of the GCS is shown in Figure 2. Comparing to Figure 1, it is a little more decomposed; however, in order not to lose the readability less important details are still hidden. Since the Grid Resource Brokers often delegate the user’s tasks to computing clusters, the key GCS’s components are meant for operating within the cluster environment. The assumption is that the cluster has the head-node (also known as access node), i.e. the node with routable IP address and set of so called execute-nodes which do not need to have public IP addresses but only a local connection to the head-node. In particular, the head- and execute-nodes can be merged into a one-node cluster with the public IP address. Additionally, we implicitly assume that each individual cluster consists of homogeneous nodes.

As it is shown in Figure 2, the components involved in GCS-related environment can be divided into four categories. The components marked with light grey colour constitute the GCS backbone. They provide a kind of framework that is independent of the actual underlying low-level check pointers but lets them be integrated with the Grid. To make that framework operational, some auxiliary tools and services have to be properly configured. For example, to allow actions delegation and checkpoint images copying between nodes the ssh has to be configured to accept the host-based authentication policy. These auxiliary components are marked with thin mesh in the background in Figure 2. The final integration with the low-level checkpointer takes place by providing the components especially dedicated to this particular checkpointer. In Figure 2, these components are marked with dark grey colour. In order to incorporate the new checkpointer, the integrator has to provide the customized execute- and resume-wrappers and dedicated checkpointer invoker. The last category of GCS components are the pre-existing ones that constitute the operating environment but do not require any special integration effort.

Some of the GCS’ components have to communicate with the ”external world” (e.g. with the Grid Resource Broker or with checkpoint images repository) and these are positioned on the head-node. On the other hand, the execute- and resume-wrappers as well as the actual check pointers and the related checkpointer invokers are positioned on execute-nodes. Since only the head-node has a public IP address, if one of the components from the execute-node requires the action that involves access to any external services, the action is delegated to the proper components on the head-node. Similarly, when the head-node is going to take the action on the execute-node, the action is delegated to it. In both situations the delegation is performed with the help of properly configured standard ssh, scp, and sudo tools.

The functionality of individual components of the GCS is also described in subsequent sections when the control-and data-flow of job submitting, checkpointing, and resuming is presented.

3 Installation and integration issues

The purpose of this section is to highlight the difference between the installation of the GCS’ backbone and the integration of a particular checkpointer with the GCS. The backbone is to be installed only once in the cluster. It performs a lot of work that is common for all installed check pointers, including management of checkpointing-related meta-data and of the checkpointing images. The backbone utilizes the properly configured auxiliary tools like ssh, scp, and sudo on a cluster-level range and OGSA-DAI based interface to Grid-level database of checkpointing-related meta-data. The backbone requires access to services such as MyProxy Server [13], Replica Location Service and GridFTP handled images repository. The location and other properties of all external services are configured with the help of configuration files. Additionally, since a part of the GCS’ backbone is written in perl, this language together with the used modules has to be installed on head- and execute-nodes.

When the GCS backbone is installed, the individual check pointers can be integrated with the environment. The primary prerequisite preceding integration with the actual checkpointer is the installation of that checkpointer on all the execute-nodes. Next, execute- and resume-wrapper and checkpointer invoker that adhere to the GCS-imposed interfaces has to be provided and installed on each execute-node. The relation between those components and the underlying checkpointer has to be indicated in the configuration file. Finally, the related Information Provider has to be updated to inform the Grid Resource Broker about the checkpointer availability and about its properties. In fact, this
last step of integration was found to be the most inconvenient one. Due to the GRMS characteristic, it was impossible to provide a dedicated Information Provider that would advertise to a given GRAM the properties of a newly integrated checkpointer. The only way to relate the integrated checkpointer to the particular GRAM was to add information about the checkpointer directly to the GRAM’s Information Provider. In other words, from the GRMS point of view, the checkpointer itself cannot be a resource but it has to be a property of another resource (i.e. of the GRAM in this case).

4 Usage scenario

Next three subsections present the flow of control between individual components of GCS as it happens during typical actions performed by the system. The job submission, checkpointing, and resuming are described. To help to follow the sequence of the presented steps there is a dedicated figure related to each usage scenario. These figures focus on the flow of control and not on any deployment details. The arrows are labeled with numbers that indicate the order in which the control is passed to the pointed components. Additionally, the description below is decorated with the numbers enclosed in parentheses. These numbers correspond to the ones from the figures and are meant to assist you in tracking the proceeding steps. Each figure contains one box that is filled with gray color. These gray boxes symbolize the components that are to be provided by the checkpointer integrator in order to integrate a particular checkpointing tool with the GCS environment.

4.1 Submitting the task

The components and relationships involved in task submission are depicted in Figure 3. (1) The GRMS allows the end user to submit the jobs to the Grid with the help of the user-provided job descriptor and GRMS’ command line interface. The mentioned job descriptor specifies the submitted job as a set of tasks that are to be executed in the Grid.
environment. Additionally, the job descriptor contains requirements and constraints regarding the computing resource and optionally a workflow definition.

Providing that the user is going to take advantage of the presented GCS, he/she has to modify the related job descriptor (a more advanced GRMS client could do it on the fly, transparently to the end user). The original name of tasks binaries has to be replaced with the name of a special execute-wrapper. This is a general wrapper that has to be installed on each GCS compatible execute-node. The user’s original task binary name is to be specified in the job descriptor with the help of environment variables that are passed to the wrapper. If the original task is defined by the executable file pointed out by the URL, that executable has to be marked in the job descriptor as the input file for the execute-wrapper. Moreover, the requirements regarding the functionality of the checkpointer that is able to deal with the submitted tasks has to be included in the job descriptor. These properties can identify the checkpointer in an unambiguous way or can only specify the desired functionality or the class of the required checkpointer. Finally, the task itself has to be marked as checkpointable.

(2) When GRMS is determining the proper computing resource for such defined tasks, then, among other parameters, it takes into account the specified checkpointer’s properties and matches them against the ones provided by the GCS-related Information Services. (2) Once the computing resource is selected, the user’s task is passed to the associated GRAM, (3) and further to the underlying LRM. The LRM, according to the local scheduling policy, (4) executes the execute-wrapper on the assigned execute-node. As the execute-node can be equipped with more than one checkpointer, the GRMS passes to the wrapper the lists of the checkpointers that match the user-defined requirements. If the list contains more than one checkpointer, the wrapper has to decide which one would be assigned to the current task. (5) Next, with the help of the OGSA-DAI mechanism, (6) the wrapper stores in the Grid-level database the meta-data required later when the checkpoint is taken (see Section 5). The most important meta-data includes information about the assigned checkpointer, the OS-level wrapper’s owner, address of the current execute-node (can be a cluster-level, non-routable one) and PID of the application. The involved job and task identifiers are used as keys to the meta-data in the database. (7) After that, the wrapper uses the access point registration tool to register in the GRMS the access point to the WSRF-compatible Checkpoint Service. The access point has a form of a URL address and is used by the GRMS as the destination address for the checkpoint requests. (9) Finally, the wrapper executes the customized execute-wrapper associated with the selected checkpointer. The GCS contract says that the purpose of the customized execute-wrapper is to execute the user’s task in a way allowing to do checkpoints later. In case of AltixC/R checkpointer, the provided customized wrapper is simply a bash script which calls the exec command to replace itself with the user’s task. Such a solution ensures that the process constituting user task preserves the wrapper’s PID, so the earlier stored PID now refers to the user task.

Figure 3: Submission related flow of control.

4.2 Triggering the checkpoint

The flow of control as it happens when the checkpoint is being done is depicted in Figure 4. (1) In order to take checkpoint of the task managed by the GRMS, the user issues the suspend task command with the help of GRMS’ command line interface. The only parameters the command depends on are the GRMS’ job and task identifiers which uniquely specify each task. (2) As each checkpointable task has a registered access point, the GRMS sends to it the checkpoint request together with job and task identifiers. In the GCS environment the access point corresponds to the WSRF compatible Checkpoint Service, which is the interface to the checkpointers installed locally in the cluster. (3) This service is just an interface to the checkpointing manager (4) which through the OGSA-DAI derived interface obtains (5) the meta-data about the task. These meta-data help to determine the checkpoint invoker that is to be used, the address of the execute-node on which the checkpoint invoker is to be called, the PID of the process that is to be checkpointed, and the name of a local-scope task owner. (6) Basing on these data, the properly configured ssh and
sudo commands are used to execute the local checkpoint manager. This manager is passed most of the earlier obtained meta-data and resides on the same node as the task that is to be checkpointed. (7) According to parameters passed to the manager the right checkpoint invoker is selected and executed. The aforementioned PID value is propagated to the checkpoint invoker. (8) In case of AltixC/R checkpoint, the checkpoint invoker is just a bash script which with the help of AltixC/R’s chkpnt command does checkpoint of the process indicated by the passed PID. When the checkpoint invoker finishes its work, the local checkpoint manager forces the user task to be terminated (due to GRMS assumptions that task is finished after checkpoint is done) with the help of a standard signal mechanism, and relying on the standard output of the checkpoint invoker (the format of that output is defined by GCS) prepares a single tarball which contains all files constituting the checkpoint image. (10) Then, the tarball is moved to the head-node and the original checkpoint image is removed. (11) Now, as the head-node has access to the “external world”, the tarball file is stored in an external repository and the location of the file is remembered in RLS.

Figure 4: Checkpoint related flow of control.

4.3 Task’s resuming

The flow of control associated with the task resuming scenario is depicted in Figure 5. The task’s resuming scenario is quite similar to the task submitting. Instead of the original task’s binaries the user has to execute the GCS provided resume-wrapper. The main difference from the submitting scenario comes down to the fact that the task can be resumed only with exactly the same checkpointer type that was previously used to create the checkpoint image, and that during the resume stage the GRMS does not assign new job and task identifiers but uses the previous ones. In the discussed implementation this information has to be provided by the user in a form of command line arguments. To satisfy the first condition, the job descriptor used with the resume command has to unambiguously specify the required checkpointer. Unfortunately, currently there is no support in the GRMS to do it automatically or transparently. The only way to find out what checkpointer was assigned to the task is to query the GCS’ database of checkpointing metadata. To simplify this task, the GCS provides the command line tool that can be used to inquire the related database about the previously assigned checkpointer’s name.

Providing that the user acquired the required checkpointer’s name, he/she has to edit the job descriptor to specify the involved checkpointer’s name and to replace the original user task with the resume-wrapper. In fact, to make things a little easier, the GRMS allows to define the tasks executed on submit and resume stage in the same job descriptor. So the same descriptor can define the wrapper executed on first-submit and on the resume scenario adequately. (1) (2) (3) (4) The way the resume-wrapper reaches the final execute-node is the same as in case of the execute-wraper. Moreover, it should still be possible to do checkpoints of the previously recovered task. Hence, the functionality of the resume- and execute-wrappers partly overlaps. Therefore, to shorten the overall description, the current section covers only the activity essential to resume the task.

In short, the purpose of the resume-wrapper is to fetch the required checkpoint image from the repository and to call the customized execute-wrapper which, basing on the provided checkpoint image and related low-level checkpointer, is finally able to restore the user’s task. The algorithm is as follows. First, as there can be more than one integrated checkpointer on the given execute-note, the wrapper has to find out which one is to be used. As it was already stated, the GRMS passes the list of matching check pointers to the wrapper. Moreover, according to the earlier assumption that during the resume stage the proper checkpointer has to be pointed explicitly, the list passed to the wrapper should contain only one item. So, this time the assignment of the proper checkpointer is trivial. (5) (6) (7) (8) Next, the resume-wrapper asks the image-exporter from the head-node to fetch the checkpoint image from the external repository.
to the local cluster (9) and later to the given execute-node. (10) After that, the associated customized resume-wrapper is executed with the path to the checkpoint image as a passed parameter. (11) That customized wrapper is in charge of using the underlying low-level checkpointer to resume the user task. In case of AltixC/R checkpointer, the customized wrapper ultimately resumes the user job using the associated resume command. As this command returns control only after the resumed user task finishes execution, the customized wrapper is able to remove the checkpoint image that was used to resume the task. Unfortunately, in case of other low-level checkpointers there might not be an opportunity to perform any action after the resumed user task finishes. In such a situation, the additional service that would remove the "utilized" images should be implemented. The mechanism could be based on the GRMS' ability to send event notifications to the registered listeners.

Figure 5: Recovery related control of flow.

5 Pros and Cons

It is hard to compare the GCA specification or GCS implementation with proposals or implementations of other architectures providing similar functionality because there are no alternative initiatives available. Therefore we will try to sum up the pros and cons of the solution presented in this paper in terms of costs introduced by the deployment of the GCA in the "plain" Grid environment and, where possible, to point out the differences in the assumptions with the solutions based only on dedicated application-level checkpointing.

The GCA causes virtually no overhead in terms of latency in the processing of the job by the Grid environment. Additional steps required to enable the checkpointing capabilities of the environment are executed during the submission of the job and in most cases are only necessary to prepare the environment or to store additional information about the application. The majority of the additional costs in terms of the CPU cycles, the storage space or the network bandwidth are imposed by the low-level checkpointer, and if someone wants to employ checkpointing functionality, these costs will occur regardless of the high-level architecture which provides interface to the particular checkpointer.

The major drawback of the presented solutions is the increased complexity of the deployment process. The installation of the Grid environment is a complex and multi-staged process by itself, and the modifications introduced by the GCS (or in general by GCA) make the installation even more complex. The need for additional components that have to be installed and configured properly, and in some cases modifications of the standard configuration files, requires additional knowledge from administrators. Nevertheless, the GCS has been implemented in a way that separates deployment of "backbone" GCS services from the actions required to integrate given low-level checkpointer with the GCS. As a result, the actual integration is significantly easier than the installation of the whole framework that supports this integration.

The additional checkpointing functionality requires extending job description with fields that were not present in the original version of the Grid environment. Even though in the presented GCS implementation the job descriptor has to be adjusted manually, it is possible that more advanced user interfaces or even the Grid Broker itself will take over this function to make the GCA more transparent from the end user’s point of view. Comparing to the GCA, the opposite approach is proposed by the OGF-lead GridCPR group. The proposal prepared by the OGF is less demanding when it comes to integration with the environment, but there is an assumption that the application is implemented in a special way. Instead, of one-time effort during the deployment of the computational environment (as it is in the GCA case), the work on making the application checkpointable has to be repeated for each application separately. Moreover, thanks to the experience gained with the proof-of-concept implementations, the GCA seems to be more detailed from the technical point of view while the OGF proposal is more conceptual architecture in which some functionality is enclosed into "black boxes". The next important note is that although the GCA is not a checkpointer itself, it makes it possible to use any of the already-existing checkpointing tools regardless of the way they were implemented. This is a huge advantage in comparison with the solution proposed by the OGF where the application-level checkpointing is
the only supported mechanism.

Finally, it cannot be overlooked that the availability of checkpointing functionality in Grids allows the Grid Resource Brokers to perform load balancing by enforcing checkpoint-kill-restart sequence, or achieving a high fault tolerance by periodical checkpoints of the applications.

6 Further Work

Even though the presented GCS implementation is just a proof-of-concept work related to the GCA architecture specification, the obtained end result is an autonomous system which is capable of being deployed within the Grid environment managed by the GRMS Grid Resource Broker. Hence, it is reasonable to consider potential future work extending the current GCS’ functionality or removing any noticed shortcomings.

At the moment the most noticeable drawback of the CTS is a lack of any checkpoint image-related garbage collector. When the user job finishes successfully, the images stored in the repository are not useful anymore. Similarly, there is currently no general mechanism that would remove from the execute-node the images used to resume the user task. The service that would delete such outdated images would be easy to implement thanks to the GRMS’ events notification mechanism.

At the moment, only the most recent checkpoint image can be used to resume the user task. Therefore, the GCS would be extended with the possibility of images versioning. Moreover, GRMS utilizes the checkpointing functionality as a tool to achieve the load-balancing capability. So, the tests in fault-tolerance oriented environment could also be valuable from the GCS point of view.

7 Conclusions

The presented GCS implementation proves the correctness and feasibility of the crucial GCA architecture assumptions. The obtained product is general and flexible enough to allow integrating the Grid with any checkpointer in a relatively simple way. However, the GCS does not implement the GCA specification precisely. Instead, it has been adapted to be operational within the currently available GRMS and Globus-based Grid environments.

Thanks to the performed work, some shortcomings of the GCA design stage and of the employed Grid middleware have been disclosed. The GCA intention is to perceive the functionality and availability of the check pointers being integrated independently of the hosting compute resources. In other words, the functionality and availability of a check pointer should be advertised through the Information Service separately from advertising the related compute resource. However, this functionality will be possible to implement when the related Grid Resource Broker will be capable of somehow associating such an advertising checkpointer with the involved computing resource. Currently the check pointers have to be considered as integral computing resources features (similarly to such features as the amount of available memory or storage space). Other issues, considered as the GCS internal drawbacks are briefly outlined in section 8 entitled “Future Work”.

Additionally, according to our integration experience, we would like to propose an additional feature related to the Grid Resource Broker’s functionality. We propose to enhance the job description language with the possibility of including the in-line invocations of external Grid Services which would be resolved to the text treated as an ordinary part of the descriptor. Such functionality would facilitate any potential integration with any other systems. In case of GCS, in-line Grid Services invocations would simplify the resume stage by excusing the end user from the duty of finding out what checkpointer was used during the checkpoint stage. This information would be provided to the Grid Resource Broker by placing in a job descriptor the in-line invocation of a specialized Grid Service. However, the Grid Service’s addresses format can be inconvenient to remember for a human being, so a correlated mechanism of logical Grid Service’s names or aliases would also be helpful.

The final remark is that all the tests of the GCS were performed within the cluster or clusters set up on homogenous nodes with the same version of OS and accompanying tools installed. It can mean that in a more diverse environment the constraints pertaining to the jobs resuming could be tighter.

8 List of main terms and abbreviations

This is a reference list of terms and abbreviations used in the paper.
access point
URL address to Grid Service that is able to receive and handle a checkpoint request.

AltixC/R
Low-level checkpointer dedicated to IA64 and Linux platform.

checkpoint invoker
The CTS’ component that utilizes a particular low-level checkpointer to take checkpoints of user’s tasks.

Checkpoint Service (CS)
Checkpoint Service is a GCS’ component implemented in a form of a Grid Service. It is the component that provides the access point.

Checkpoint Translation Service (CTS)
Checkpoint Translation Service is one of components of the GCA. From the GCS point of view, it resembles the CS component.

customized execute-wrapper
The GCS’ component assigned to an individual low-level checkpointer and intended to execute the checkpointable user tasks.

customized resume-wrapper
The GCS’ component assigned to individual low-level checkpointer and intended to resume the user tasks.

execute-node
A cluster’s nodes executing the submitted jobs. This node is not required to have a public IP address.

execute-wrapper
The general wrapper that is executed in place of the original user task.

Grid Checkpointing Architecture (GCA)
Grid Checkpointing Architecture specifies general rules and requirements to integrate low-level checkpointers with the Grid. GCS is a proof-of-concept implementation of GCA.

Grid Checkpointing Service (GCS)
Grid Checkpointing Service is described in the paper, a proof-of-concept implementation of GCA.

Globus
Globus Toolkit is a set of services and tools that constitute the middleware used to develop Grids.

Grid Resource Broker
The general name for a component that manages resource allocation in the Grid environment.

Grid Service
The Web Service compliant with the WSRF standard.

GridCPR
The OGF initiative to define an interface for checkpointable Grid-aware applications.
GridFTP
The file transport protocol for Grid installations.

GRMS
Grid Resource Management Service is the actual implementation of the Grid Resource Broker.

head-node
A cluster’s node with the public IP address.

image-exporter
The GCS’ component installed on the head-node and responsible for storing checkpoint images in the external repository.

image-importer
The GCS’ component installed on the head-node intended to fetch checkpoint images from the external repository.

Information Provider
The component that is able to provide the Information Service with adequate information.

Information Service
The component that advertises to the Grid the information provided by Information Providers.

LRM
Local Resource Manager is a general name for clusters’ managers. Examples of LRM are Torque, PBS and LSF.

MyProxy
Part of a Grid ecosystem allowing credentials to be managed and shared.

OGF
Open Grid Forum is an international community dealing with Grids dissemination and standardization.

OGSA-DAI
The middleware to assist access and integration of data within the Grid.

resume-wrapper
The general wrapper that is executed in place of the original user task in order to resume this task.

RLS
Replica Location Service is a part of the Globus Toolkit, and was designed to support mapping of logical names of files to their physical localization.

VMM
The Virtual Machine Monitor refers to the technology of virtual machines. Example implementations are Xen and VMware.

WSRF
Web Services Resource Framework is a framework for modeling and accessing the stateful resources through the Web Services. The WSRF-compliant Web Service is also named the Grid Service.
References


[10] Replica Location Service Home Page http://www.globus.org/toolkit/docs/4.0/data/rls/


[12] The OGSA-DAI project http://www.ogsadai.org.uk/


