European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Grid Checkpointing Architecture - a revised proposal

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-0036
May 30, 2006

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

Grid Checkpointing Architecture - a revised proposal

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 TR-0036
May 30, 2006

Abstract

Contemporary Grid environments are featured by an increasingly growing virtualization and distribution of resources. Such situations impose greater demands on load-balancing and fault-tolerant capabilities. The checkpoint-restart mechanism seems to be the most intuitive tool that can fulfill the specific requirements. However, as there is still a lack of widely available, production-grade checkpoint-restart tools, the higher level checkpoint-restart services are not well developed yet. One of the goals of the CoreGRID Network of Excellence is to define the high-level checkpoint-restart Grid Service and to locate it among other Grid Services. We aim to define both the abstract model of that service and the lower layer interface that will allow the service to cooperate with diverse existing and future checkpoint-restart tools. The paper is the first step on the road to this goal. It includes the overall sketch of the architecture of the considered service and its connection with the actual checkpoint-restart tools.

1 Introduction

Until now there have been few checkpointing systems that can do computing processes' checkpoints, for instance: psncLibCkpt [1], Altix C/R [2], Condor [3], libCkpt[4] and others. These checkpointing systems always have different capabilities and interfaces, and in most cases are specifically dependent on a particular OS and hardware platform. Therefore, checkpointing systems are not widely used and the existing ones always have some limitations which are different for different systems.

One can try to employ the aforementioned checkpointing systems in the Grid environment. Unfortunately, in contrary to the visions expressed by experts within "Next Generation Grid(s), European Grid Research 2005-2010" [5] and "Next Generation Grids 2, Requirements and Options for European Grids Research 2005-2010 and Beyond" [6], such integration would impose high complexity. Then, if we intend to use the checkpointing functionality in Grids, we have to figure out an abstract Grid Checkpoint Service (GCS) that hides all the complexity and underlying checkpointing systems. Moreover, that service has to fit into the more general architecture which will allow to bring into play the diverse existing and future checkpointing systems. A vision of such GCS and associated Grid Checkpointing Architecture (GCA) is presented in this paper.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

The first version of the Grid Checkpoint Architecture was described in the paper ”Towards the Grid Checkpointing Architecture” [7]. As a result of exchanging experience with partners from SZTAKI and PSNC and feedback from the first paper, a new revised version of the architecture emerged.

2 Architecture

The architecture proposal included in this section is a revised version of GCA presented in the paper ”Towards the Grid Checkpointing Architecture” [7] presented at the PPAM 2005 conference. The proposal defines four layers (Broker, Checkpoint Grid Service, Translation and Core Service) and three interfaces used to exchange information between services.

Architecture layers

The four layers (Figure 1) represent mutual dependencies between different parts of Grid Checkpoint Architecture. Each layer hides all underlying services by providing a set of calls used to perform certain actions. The interaction is allowed only between any services placed on adjoined layers. A brief description of these layers is the following:

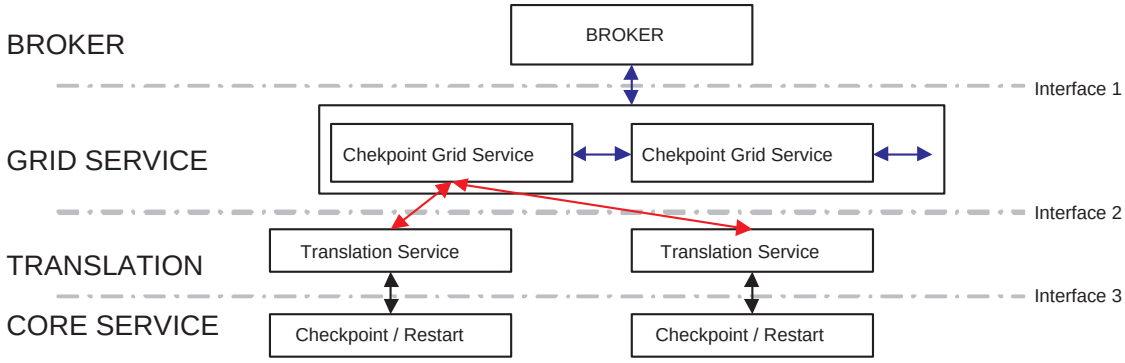

Figure 1: Layers in Grid Checkpointing Architecture

- The **GRID BROKER** layer represents the Grid job manager/scheduler. The first one is triggering checkpoint and restart of the applications. The decision whether to checkpoint/restart an application is made on the basis of information from monitoring services (fault detection) or the scheduling algorithm/resource management policy. The next task is to adjust the job description of application submitted by the user in order to ensure that the application will be checkpointable. This task may require exchange of messages with the Grid Checkpoint Service (GCS),
- The **GRID SERVICE** layer represents a set of Grid Checkpoint Services (GCS). This layer may consist of many independent instances. Each instance is able to forward any request it cannot handle itself (peer-to-peer architecture). A Broker sees the set of GCS instances as a single service with one access point. A single GCS instance may have access to many Translation Services from the Translation layer (relationship ”one to many”) Any service from this layer provides the Broker layer with all checkpointing specific functionality. It manages all metadata related to images of the checkpointed applications (e.g. registers image files in the Grid Storage Service, handles the connections between applications, images and Translation Services etc.). The services from this layer are executing orders passed from the Broker layer using services from the Translation layer. One of the most important tasks of this service is choosing an appropriate Translation Service to execute the task ordered by the upper layer. An example of such task may be finding an Translation Service/*checkpointer* that will be able to handle the application, considering the requirements included in the job description. Accuracy of the *checkpointer* selection depends on how detailed the description of job requirements (regarding the desired functionality of the *checkpointer*) provided by the user/Grid is. The exact policy of the *checkpointer* selection may influence the chance of correct job checkpoint/resume.
- The **TRANSLATION** layer represents a set of Translation Services (TS). The TS acts as a mediator between GCS and actual *checkpointers*. The TS accepts Interface 2 messages and translates them to a format acceptable

by native *checkpointers* (Interface 3). The TS instances are tightly connected with the corresponding *checkpointers* (instances of the Core Service layer). For each Core Service *checkpointer* there should exist at least one Translation Service. The TS maintains information about functionality, requirements and calling semantics of the managed *checkpointer* (the Core Service layer instance). This information is used to execute checkpoint and restart operations and match the application with the *checkpointer* (before the application is submitted). The last function is reporting all checkpoints performed by the underlying *checkpointer*, even if the operation was not triggered by the GCS,

- The **CORE SERVICE** layer represents the ”real” tools used to make the checkpoint. Services placed on this layer will be called *checkpointers* in this paper. In general there is no assumption on what type of checkpoint it should be it may be kernel or user or application level checkpointing. The Core Service may also represent some other software that is able to trigger checkpoints on a given Computing Resource (in such case the corresponding TS can be considered as an interface to interface). A good example may be a local scheduler like the Sun Grid Engine that is capable of issuing checkpoint/restart commands using some third party *checkpointers*. This may lead to the situation when we have more than one ”path of access” to the *checkpointer*. An example of such situation is depicted in Figure 2. On a single computation resource there can be one or more Translation Services. Services placed on this layer will be called *checkpointers* in this paper. In order to avoid problems with the TS selection, while configuring the TS on a computing resource, the administrator should indicate which is the preferred TS for each C/R.

3 Intercommunication Interface

A definition of the messages used to pass information between layers is one of the most important parts in the architecture design. In the GCA there are three singled out sets of messages called **interface 1**, **interface 2**, **interface 3**.

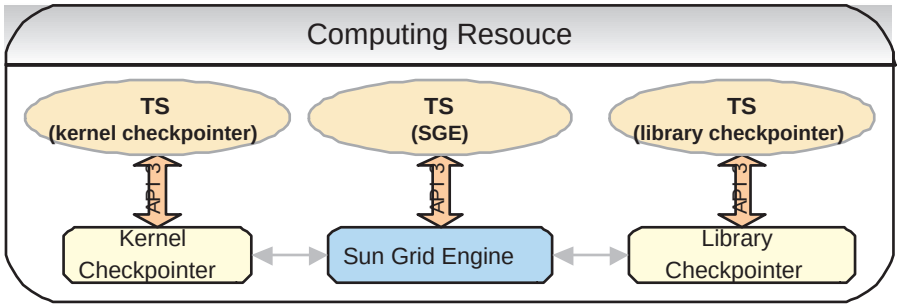

Figure 2: Example of different access paths to checkpointer

3.1 Interface 1

This interface defines communication messages exchanged between the Broker and the GCS. The meaning of the messages are:

- **prepare_job**
  The message is sent by the Broker when the it wants the GCS to provide information about the *checkpointer* that is able to handle the given type of application. Depending on the original job description, the GCS finds the *checkpointer* able to create an image of the job. It may be necessary to modify the job description in order to make it work with the selected *checkpointer*.

- **checkpoint_job**
  This message is sent by the Broker to the GCS in order to save the state image of a job. The Broker has to provide information necessary to identify the job (job id). The message may contain some additional parameters: for

example, a suggested *checkpointer* or/and special parameters passed to the *checkpointer*. The GCS replies with a report including information on where the image was stored, which *checkpointer* was used, etc.

- **resume_job** This message initiates the process of restarting the job from the previously saved image. The GCS replies with a description of the job which should be submitted by the Broker in order to resume the given application from the saved image[1].

3.2 Interface 2

The Interface 2 defines a set of messages exchanged between the GCS and the Translation Service.

- **prepare_submit_job**
  The message is sent by the GCS after it figured out the best *checkpointer*. The target TS, basing on its knowledge about the corresponding *checkpointer*, has to provide a set of changes in a job description (it might be necessary to add some additional parameters) required by the *checkpointer*. In some cases (e.g. kernel level *checkpointer*) there may be no required changes.

- **checkpoint_job**
  The message is used by the GCS to inform the TS that is has to call the corresponding *checkpointer* and make a checkpoint of the selected application. After receiving this message, the TS has to communicate with the native *checkpointer* (directly or using any software that manages access to this *checkpointer*) in order to issue the checkpoint command (Interface 3). The TS should reply with a message consisting of the status of the operation and information about the image.

- **prepare_resume_job**
  The message is sent by the GCS to the TS as a consequence of receiving a **resume_job** message. After the GCS has figured out which *checkpointer* was used, it has to communicate with the corresponding TS (in general it is irrelevant if it is exactly the same TS that created the image). This message indicates that the TS basing on an original job description and its knowledge about the underlying *checkpointer* (and additional parameters specified during the checkpoint) must prepare a description of the job that will resume the application from the selected image.

- **checkpoint_executed** This message is sent by the TS when the controlled *checkpointer* saved the state of an application and the checkpoint was not triggered by the **checkpoint_job** call. This functionality is required in order to support self-checkpointing applications (e.g. the checkpoint is initiated at a certain point of computing). The message must contain information about the image of the application.

3.3 Interface 3

An exact definition of Interface 3 is not a part of the GCA because of a variety of possible methods of triggering functions of the *checkpointer*(e.g. signals, environment variables, executing shell commands etc.).It is up to the Translation Service implementation team to design an appropriate method of communication between the TS and the Checkpoint/Restart tool.

4 Interaction with other grid services

The GCS also utilizes other grid services. Those services were not depicted in Figure 1 because the Interface and functionality of those services are not within the scope of this paper. The dependencies on other grid services are depicted in Figure 2

A short description of functionality required from other grid services is the following:

4.1 Interaction between Broker and GCS

Interaction between the Broker and the GCS is described in sections 3 and 5.

[1]The GSC might reply with the "submit job" command (with the same format as users submit_job); however, it would require delegating the original users rights to the GCS. During the submit phase the Broker acts on behalf of the user anyway, so it is better to let it do all the job.

4.2 Interaction between Information Service and GCS

The Information Service is used to obtain information about jobs and to store information about the executed checkpoints. This repository should be accessible by any GCS instance.

4.3 Interaction between Storage Service and GCS

The Storage Service is utilized to register/store images of the applications. The detailed storage policy (replication, migration etc.) is not in the scope of the GCA.

Transfer of the images (during the migration or restart) is performed by the Scheduler/Broker as a part of the job submission routine (when the broker prepares an environment for a job, it must ensure that files used by this application are accessible; the job images are considered to be files required by the application/job).

4.4 Interaction with other Grid Services

The GCS has to use Authorization Services in order to access files or manage access to information about the stored images. This service, however, is not a part of Figure 2 because it is used only to access other services and provides no additional information to the figure.

5 Scenarios

This section will describe the behavior of services during the operations that involve the use of GCS submission, checkpoint, and restart of job. The migration scenario was omitted because its simplicity it may be considered as a sequence of **job_checkpoint** and **job_restart** commands.

5.1 Job submission

The job submission is a basic functionality of every broker. The ”submit job” scenario is performed each time the user submits an application that should run in an environment managed by the Broker. In order to ensure that every task submitted to the Broker could be theoretically checkpointed and resumed, the GCA introduced an additional phase executed by the Broker during ”job submit”.

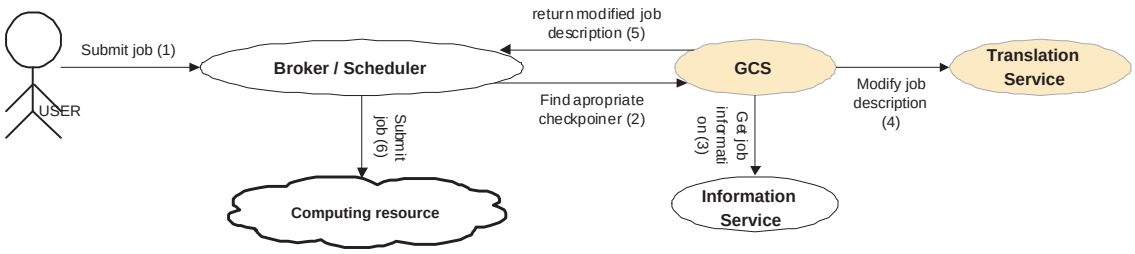

Figure 3: Grid job submission

5.1.1 Job submission (with GCS involvement)

The GCA extends the standard job submission scenario (the user issues the submit command, the Broker finds an appropriate node and runs the job) by inserting an additional ”prepare stage” (steps from 2 to 5 in Figure 3). This phase occurs after the Broker receives the information and before it finds the destination node. This stage requires more changes in brokers, therefore it is optional; however, it increases the chance of running the job on nodes where an appropriate *checkpointer* is installed.

The Broker asks the GCS to modify the job description before it searches for the appropriate computing resource. This operation is performed to ensure that the job will be checkpointable. This may require adding some special options passed to the executable, adding some environment variables, specifying the need of some tool on the destination

node or other changes in the job description. The whole scenario is depicted in Figure 3. A detailed description of all steps is as follows:

1. The user submits a job description to the broker (the job description can also specify the desired *checkpointer* along with other requirements).
2. The Broker forwards the job description to the GCS in order to obtain *checkpointer*-related information [**prepare_job** interface 1].
3. The GCS may need some additional information about the job so it may have to connect to the Information Service to access that information. At this stage the GCS must identify a *checkpointer* (or a set of *checkpointers*) and choose the most appropriate one that should be able to handle the application.
4. The GCS contacts a Translation Service responsible for the communication with the selected *checkpointer*. The Translation Service should be able to provide information on how to modify (or modify on its own) the job description (e.g. by adding some parameters, setting environment variables, adding some requirements on services installed on the computing resource, special queues definition for the local broker etc.) to make it checkpointable with the selected tool [**prepare_submit_job** interface 2].
5. After modifying the job description, the GCS returns it to the Broker.
6. The Broker tries to find a suitable computing resource that fulfills all the requirements and submits the job to the computing resources execution module.

5.1.2 Job submission (without GCS involvement)

If the Broker is not prepared to execute the ”preparation stage”, the workflow is similar to the one presented in the previous scenario with the exception that steps from 2 to 5 are omitted. This scenario represents the existing brokers which in most cases are capable of executing checkpoint and restart commands in some way; however, they are not fully integrated with the GCA.

5.2 Job checkpoint

During the design phase we considered two possible scenarios of checkpoint. The first one was a ”standard” scenario when the checkpoint is triggered by the broker (the Broker wants the job to be migrated or because of the selected checkpointing policy). When the checkpoint image may be created without the GCA involvement (e.g. because the image is created at a fixed point of computation), the second variant of checkpoint is considered.

5.2.1 Broker issued checkpoint

This is the preferred scenario of doing the checkpoint the checkpoint is issued by the broker. Workflow for this scenario is depicted in Figure 4.

1. The Broker issues a checkpoint command [**checkpoint_job** interface 1].
2. The GCS finds the CR where the application is being executed, looks into the job description for the selected *checkpointer* and contacts with the appropriate TS that manages access to this *checkpointer* (if there is no description, the GCS has to find the best *checkpointer* from those installed on the CR) The [**checkpoint_job** interface 2] command is sent to the selected TS.
3. The TS is executing the checkpoint command calling the underlying *checkpointer*.
4. The *checkpointer* reports the status of the checkpoint operation to the TS.
5. The TS returns the information about the checkpoint image and status of the operation.
6. The TS registers information about the image in the Information Service. This information will be used during the restart phase.

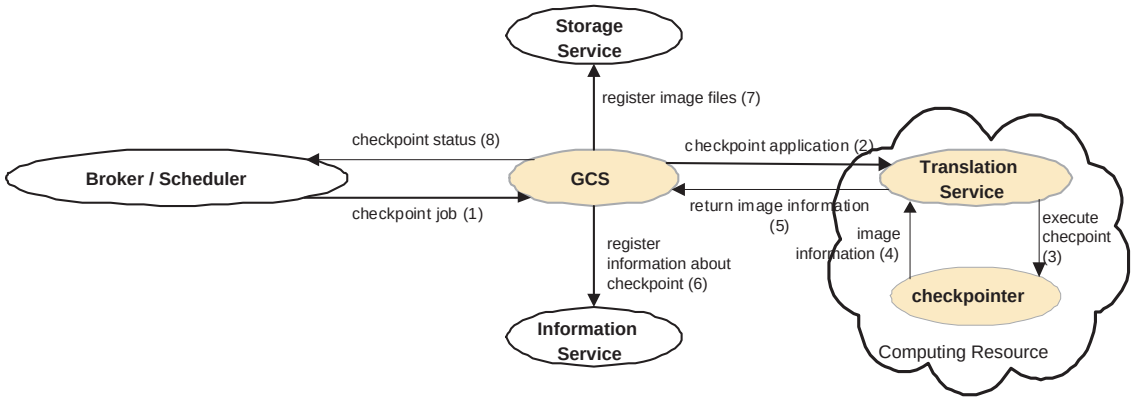

Figure 4: Checkpoint issued by the broker

7. The files containing the checkpoint image are registered in the Storage Service to make them accessible by other Grid Services.

8. The status of the whole checkpoint operation is returned to the Broker.

5.2.2 Independent checkpoint

This is the second variant of a checkpoint scenario if the *checkpointer* does not support ”triggered” checkpoint or performs checkpointing after a certain period of time or at fixed point of computation. The Translation Service provides the GSC with information about image placement, date and time of image creation (according to possibilities of obtaining this information). In order to allow the application to be restarted from the image created in that way, the TS has to report every checkpoint that is executed by the underlying *checkpointer*. For each information about such ”independent” checkpoint the GCS has to try to find an appropriate job issued by the Broker. Only if such mapping is possible, the information about a checkpoint for the given image is stored in the IS. The steps in this scenario are the following:

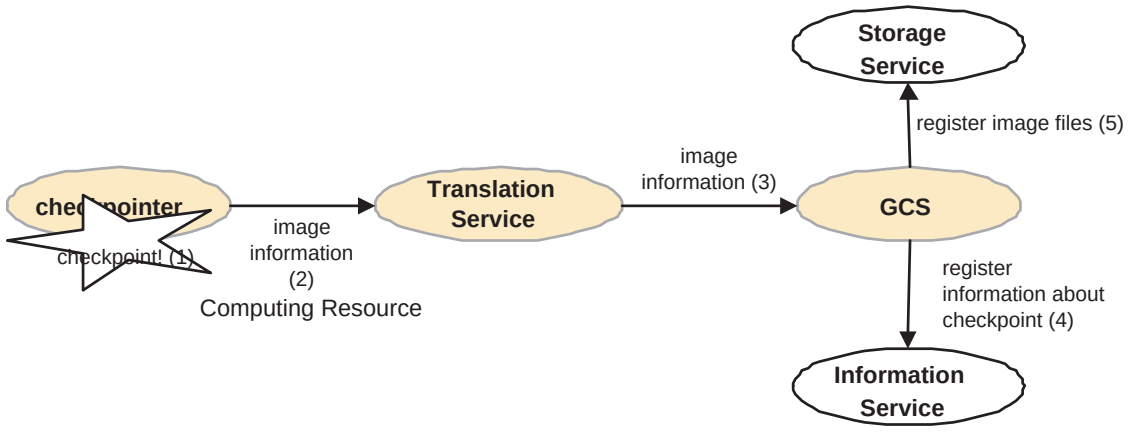

Figure 5: Checkpoint triggered by checkpointer

1. At some specified point of time the checkpoint is executed (the event that triggered the checkpoint is outside the GCA).

2. The TS has to intercept information about the checkpoint (4).

3. Information about the checkpoint is sent to the GCS. The GCS has to check if there is a Grid application with the local id equal to the one passed with the message. The type of the local id may depend on the TS (process id, parent process id, id of the job in the local queue etc.) that sends the information [**checkpoint_executed** interface 2] (5).

4. If the GCS can match the local id returned by the TS in the previous step with any grid application, the information about the image is stored in the Information Service (6).

5. files with the application image are registered in the Storage Service (7).

Numbers in the brackets refer to stages depicted in Figure 5.

5.3 Job start

In the current version of the GCA the restart of the application is divided into two stages. The first stage of the restart procedure is initiated by the grid broker by calling the GCS. During this stage a description of a special job which will be used by the Broker to trigger the restart is prepared. A special job description is based on the description and requirements of the original job because the node where the resumed job will run has to fulfill all the requirements of the original job. Depending on the *checkpointer* that was used to create the images, some additional requirements may be added or changed. The second stage is performed by the broker and is identical with the normal job submission scenario. The resume of the application is performed by the execution module of the Grid according to a description provided by the GCS and TS.

The workflow during the restart of a job is depicted in Figure 6

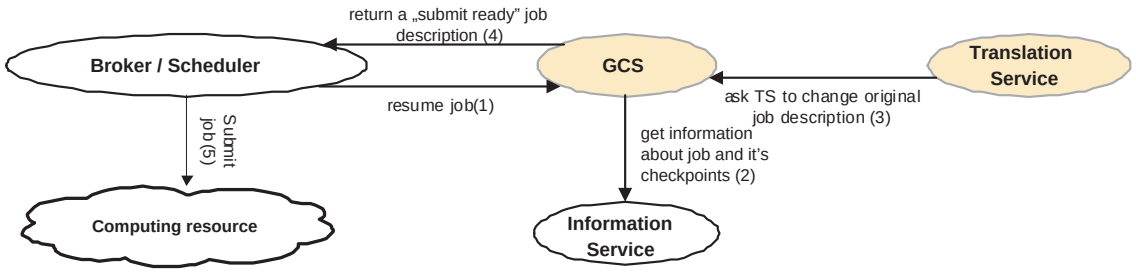

Figure 6: Message exchange during job restart

The stages of job restart:

1. The Broker sends a **resume_job** [interface 1] message to the GCA. This message is sent according to some policy (perhaps after a job failure detection or part of a migration procedure).

2. The GCS checks if there are any images for the given job. If such image is prepared, the GCS has to select the most appropriate image (probably the most recently created). At the next stage the GCS has to find a TS for the *checkpointer* that created the image and ask it to prepare a set of changes in the original job description [**prepare_job_resume** interface 2].

3. The TS has to parse the job description along and specify a set of changes in the original job description that will cause the job to be resumed. The set of changes/modified job description is sent back to the GCS.

4. The GCS may make some further changes (e.g. adding the files of the image to the job description in order to cause them to be copied to the destination node) and sends a modified job description.

5. The Broker finds an appropriate node according to the requirements in the job description and submits the job.

5.4 Job migration

The job migration is an important functionality; however, no singled out command is designed in GCA exclusively for migration. Essentially the migration is the following sequence of commands issued by the Broker: **Checkpoint** the job, **Terminate** the job, and **Restart** the job. The Checkpoint and Restart are described in sections 5.2 and 5.3 of this paper. The **Terminate** is a command that forces the application to finish its work. We assume that this is a standard functionality of any broker.

6 Relations to other projects

Apart from the CoreGrid checkpointing work group we are aware of the existence of one more team working on a similar subject. The Global Grid Forum [8] Grid Checkpoint Recovery Working Group strives to define an architecture for the Grid checkpointing service. The differences between the CoreGrid working group and the GGF WG are extremely significant the source of those differences are the underlying assumptions. The GGF team focused on creating a tool that would enable an application that runs in the Grid environment to checkpoint itself. In their approach the tool should be a part of an application and is tightly connected to various Grid services (such as authentication, storage etc.). There is a draft of their checkpointing architecture described in a draft "An Architecture for Grid Checkpoint Recovery Services and a GridCPR API" [9]. In the architecture proposed by CoreGrid WG, the GGF checkpointer architecture can be treated as a core/low level/native *checkpointer*.

7 Conclusions

The GCA in the current form is a very flexible architecture that is able to use much of the existing and future checkpoint and restart toolkits. Introduction of this service should encourage users to use the Grid computing because of providing higher fault tolerance/safety level of application which is the Achilles' heel of a system consisting of thousands of distributed nodes.

There are other projects focusing on similar topics such as GGF WG [8]; however, due to a lack of space their comparison with our project was omitted.

References

[1] http://checkpointing.psnc.pl/Progress/psncLibCkpt/

[2] Checkpoint/Restart mechanism for multiprocess applications implemented under SGIGrid Project, Gracjan Jankowski, Rafal Mikolajczak, Radoslaw Januszewski, CGW2004.

[3] Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, Michael Litzkow, Todd Tannenbaun, Jim Basney, and Miron Livny; Computer Sciences Department University of Wisconsin-Madison.

[4] Libckpt: Transparent Checkpointing under Unix, Conference Proceedings, Usenix Winter 1995 Technical Conference, New Orleans, LA, January, 1995.

[5] Next Generation Grid(s), European Grid Research 2005-2010, Expert Group Report, 16th June 2003.

[6] Next Generation Grids 2, Requirements and Options for European Grids Research 2005-2010 and Beyond, Expert Group Report, July 2004.

[7] Towards the Grid Checkpointing Architecture, G. Jankowski, J. Kovacs, R. Mikoajczak, R. Januszewski, N. Meyer, Poznan Supercomputing and Networking Center.

[8] http://www.ggf.org/

[9] An Architecture for Grid Checkpoint Recovery Services and a GridCPR API, Derek SImmel, Thilo Kielmann, Nathan Stone.