

## **Improving the fault-tolerance level within the GRID computing environment - integration with the low-level checkpointing packages**

*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-0158  
June 16, 2008

Institute on Grid Information, Resource and Workflow  
Monitoring Services

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

# Improving the fault-tolerance level within the GRID computing environment - integration with the low-level checkpointing packages

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-0158*

June 16, 2008

## Abstract

The advantages of utilizing the checkpointing functionality are obvious; however so far the Grid community has not developed a widely accepted standard that would allow the Grid environment to consciously utilize low-level checkpointing packages. Therefore, such a standard named Grid Checkpointing Architecture is being designed within the CoreGRID project. To define, validate, and prove the developed concepts, we utilize small proof-of-concept implementations of checkpointing-aware Grid environments. Implementation, testing and examining these systems allow us to check the feasibility of the devised concepts and provide us with adequate experience. The paper describes the architecture and scenario of one of such proof-of-concept implementations. The presented integration of a low-level checkpointing package with the Grid environment allows the Grid Resource Broker to recover user's jobs in case of failure.

## 1 Introduction

One of the most common benefits of the checkpointing technology is the high level of fault tolerance offered by the applications that can be checkpointed. In case of any failure the checkpointed application can be recovered to the point where the last checkpoint was taken. To date, there have been a few low-level checkpointing packages [1] [2]. Each checkpointing package offers a different functionality and interface. Because of technical issues the checkpointing packages impose some limitations on applications that are to be checkpointed. In case of distributed applications a significant problem is also how to make coherent checkpoint of a few cooperating processes and not lose the just in-transit messages simultaneously. That leads to a conclusion that for now not every application can be checkpointed, if a checkpointing package is able to deal with the given application the another package may not, and even if there is more than one checkpointing package able to deal with the given application, most probably the interfaces to the checkpointing functionality differ. Consequently, due to these features the integration of low-level checkpointing packages with the Grids is a difficult and not yet accomplished task.

---

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

Nevertheless, because of providing the potential, high level of fault-tolerance and jobs migration facilities the checkpointing technology is a very attractive tool from the point of view of the Grid environment. Therefore, just to make the checkpointing packages available to the Grid environment we work on the Grid Checkpointing Architecture [3] [4] which aims to define the novel, Grid-embedded components and associated design patterns that will allow the Grids to utilize a variety of the existing and future low-level checkpointing mechanisms in a conscious way.

To better understand the domain that GCA is dealing with and to check the feasibility of the developed concepts, we are preparing the series of proof-of-concept environments that implement different parts of GCA. In the paper we are going to present one of such proof-of-concept environments. The presented environment was prepared to prove the possibility of integration of the low-level checkpointing package with the Local Resource Manager and with the Grid Resource Broker. The involved individual components and the proposed architecture are described adequately in Section 2 and 3. Relying upon the architecture described in section 3, the two scenarios of experiments have been realized. The user- and workflow-driven scenarios are described in Section 4. As the paper presents only one and most basic proof-of-concept environment, the further ones are briefly mentioned in Section 5. The final conclusions that arise from the performed tests are presented in Section 6.

The described integration was presented at CoreGRID Industrial Conference 2006 in Sophia-Antipolis in France [16]. The presentation was performed in a form of a live Demo Case. Thanks to the GUI provided by the Migrating Desktop [5] [6], the audience had the opportunity to see in a graphic format how the job had been artificially failed and later successfully recovered in the Grid environment. The demo was also presented at our stand at CoreGRID Industrial Showcase, June 4-5, in Barcelona. This event was co-located with the 23rd Open Grid Forum.

## 2 Components

The main functional components involved in the integration are: low-level checkpointer, Execute Manager, Local Resource Manager and Grid Resource Broker. This section provides a short overview of the implementations of the actual components used during the integration. Mutual relations between these components and their role within the considered proof-of-concept environment are described in the next section.

### **Low-level checkpointer - AltixC/R**

The low-level checkpointer utilized during the integration is AltixC/R [7] [8]. It is kernel-level checkpointing package designed by PSNC for Altix systems equipped with IA64 processors and running under the SGI ProPack environment (i.e. Linux-based environment prepared by SGI). The most recent version works with the Linux kernel 2.6. The required kernel-level code is provided in a form of a dynamically loaded kernel module so it is easy to use and install. Contrary to some non kernel-level checkpointers, there is no assumption on the availability of source codes or the programming tools that were used to write the programs that are to be checkpointed. The package is able to checkpoint multi-process programs that communicate through the System V IPC objects. A unique feature of the package is virtualization of some global system keys and identifiers. Thanks to that, when the program is recovered, it is cheated that the identifiers have not changed (even though due to technological reasons, they are very likely to have changed).

### **Execution Manager WS GRAM**

The Execution Manager is meant to provide the Grid Resource Broker with a uniform interface to a variety of underlying computing nodes. Thanks to that the interface to different types of clusters (which are accessed through the Local Resources Managers) and even to single computing machines is similar and well abstracted in a form of the Execution Manager-imposed protocol. In the considered integration we have used the WS GRAM [9] as the Execution Manager. The WS-GRAM stands for Web Services Grid Resource Allocation and Management and is part of the Globus Toolkit. According to the Globus Toolkit website [10] the WS GRAM component of the toolkit comprises a set of WSRF-compliant Web Services [11] to locate, submit, monitor, and cancel jobs on Grid computing resources.

### **Local Resource Manager Torque**

The component that provides access to local computing resources is named Local Resource Manager (LRM). The actual LRM used in the presented integration is Torque [12] which is an open source implementation of the manager that provides control over jobs distributed among computing nodes of the cluster. In the simplest scenario the cluster together with management infrastructure can be scaled to one node.

### Grid Resource Broker GRMS

The Grid Resource Broker is a component that is able to coordinate resources allocation and especially job submissions in a Grid environment. This is also the component that the end users interact with in order to submit, monitor and control their jobs. The user interface to the Grid Resource Broker can vary from the specialized GUI or CLI tools to WWW or WAP-based pages. The Grid Resource Broker used in the presented integration is the Grid Resource Management Service (GRMS) [13] which is part of the GRIDGE Grid Toolkit [14] [15] being a set of integrated, ready-to-use Grid services. The GRMS supports building and deploying resource management systems for large-scale distributed computing infrastructures. Comparing to other similar products, a unique feature of GRMS is the ability to deal with jobs defined as a set of tasks with precedence relationships where the execution of a child task can be triggered by any status of a parent task. It is noteworthy that during the integration activity we have experienced active support from the GRMS development team.

### User Interface Migrating Desktop

As the described integration was presented in a form of a Demo Case at the CoreGRID Industrial Conference, specific requirements had appeared. The interface should provide a neat graphical interface that would allow presenting the benefits of the checkpointing technology. Apart from the common features for controlling and monitoring the job, the interface should provide the possibility of presenting the intermediate computing results. It turned out that the product that offers such unique functionality is the Migrating Desktop [5] [6] (MD). Even though the most noticeable part of MD is an intuitive GUI, this product in fact means much more. It is actually a whole framework that hides the details and complexity of the underlying Grid services and which is meant to deploy interactively controlled, Grid technology-based complex systems. Internally the GUI of the MD is just a front end to the Roaming Access Server (RAS) which further interacts with different Grid services, including the Grid Resource Broker. Thanks to the Java Web Start technology, the GUI part of MD can be accessed from any Java-equipped terminal. Consequently, the user has the impression that the GUI together with all settings and jobs is migrating between terminals. As MD is natively not ready to cooperate with the GRMS (the Grid Resource Broker we choose), the cooperation with the MD development team was required. However, both the RAS and GRMS are well designed and based on Web Services so the integration was quite smooth.

## 3 Architecture details

The outline of the architecture of the described proof-of-concept environment is depicted in Figure 1. The presented components correspond to the ones introduced in Section 2, but in some cases they are presented together with internal subcomponents not mentioned in Section 2. The figure has been divided into four parts that represent the division of the architecture from the deployment point of view. The next part of the section describes the architecture basing on the bottom-up approach. First, the lowest level components are presented and then the higher level ones.

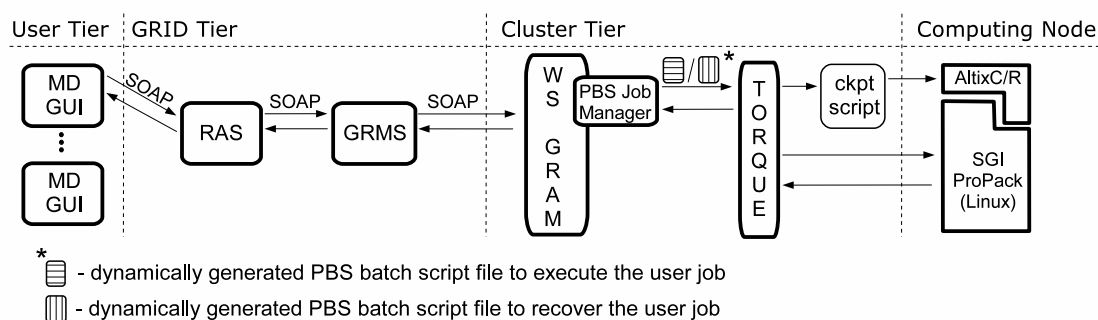


Figure 1: Architecture

Apart from the operating system, the lowest level component in Figure 1 is the AltixC/R checkpointing package. The checkpointing package provides dynamically loaded kernel modules and command line tools that allow for taking checkpoints and recovering users' jobs. The AltixC/R package has to be deployed on each Computing Node on which

the checkpointing functionality should be available. The AltixC/R assumes that the checkpointing functionality is exposed only to locally logged users, so it does not expose any interface to the Grid or Cluster environment.

Figure 1 shows that the jobs are submitted to the Computing Node through the TORQUE LRM. The TORQUE possesses support for low-level checkpointers but by default the support is disabled. To enable the support the TORQUE has to be recompiled and the following line has to be added to the `src/include/pbs_config.h` file.:

```
#define MOM_CHECKPOINT 1
```

The TORQUE allows for employing the low-level checkpointing packages by means of special checkpointing script and additional arguments passed to the `qsub`<sup>1</sup> command. It is important to understand that this script is not shipped together with the TORQUE. The script has to be written by the system administrator or another actor who has adequate knowledge about the integrated checkpointer, the TORQUE and the way TORQUE executes the script. The `qsub` command of syntax `qsub -c c=<time> <PBS batch script>` submits the job defined within the `<PBS batch script>` file to the cluster. The arguments `-c c=<time>` determine that every `<time>` minutes the checkpointing script will be executed. The assumption is that the checkpointing script is available on the Computing Node to which the job will be submitted. The path to the script is defined within the configuration files of the Computing Nodes of the TORQUE environment. For example, in our testing environment the file `/var/spool/torque/mom_priv/config` that resides on the Computing Node contains the following line.:

```
$checkpoint_script /home/fujisan/bin/mom-checkpoint.sh
```

The most important arguments "automatically" passed to the checkpointing script are: PID of the PBS batch script that represents the job that is to be checkpointed, JOB ID assigned by the TORQUE to the job and unix like USER ID of the owner of that job. These arguments are available in the checkpointing script through the \$1, \$2 and \$3 variables. These variables and knowledge about locally available low-level checkpointing package should be enough to write the checkpointing script that successfully takes checkpoints of the user's job.

To resume a job in this environment the user has to prepare a dedicated PBS batch script and resubmit the job. The PBS batch script is finally executed on the Computing Node. The assumption is that this time, instead of executing the user job from the beginning, the PBS batch script uses a locally available checkpointing package to recover the job. Obviously, the image of the recovered job has to be available at this point but it is a matter of agreement between the authors of the checkpointing script and PBS batch script.

At this point the environment is able to checkpoint and recover jobs that are submitted by users directly to the TORQUE. However, as it is shown in Figure 1, the cluster managed by TORQUE is further exposed to the Grid environment. The element that links the cluster with the Grid is WS GRAM which is able to transform SOAP messages to the dynamically generated PBS batch scripts and submit them to TORQUE. The part of the WS GRAM that is responsible for preparing and submitting the PBS batch script is the PBS Job Manager. The problem is that by default the PBS Job Manager is not able to submit the job with the additional `"-c c=<time>"` parameter and cannot prepare the PBS batch script that recovers the job using the locally available checkpointing package. Therefore, to achieve this functionality the PBS Job Manager had to be adjusted. The PBS Job Manager is written in the Perl language and the source file resides in the `pbs.pm` file that is part of the Globus Toolkit. The WS GRAM accepts the request of job executing in a form of a job description defined as a special xml file. Even though the format of this file does not contain any checkpointing related elements explicitly, it allows for customizing the job description through the `<extensions>` xml element. So, we defined the following xml elements as the `<extension>` children: `<ckpt_id>`, `<checkpointable>`, `<period>`, `<recovery>` and `<grms_id>`. A detailed description of these elements is omitted. However, as the adjusted PBS Job Manager is passed the whole job descriptor it extracts from it the newly defined checkpointing related elements and basing on them prepares the adequate PBS batch script.

The next crucial element in Figure 1 is GRMS. The GRMS is a kind of a Grid Resource Broker which, basing on the request received from the user, finds the adequate Computing Resource and submits to it the user's job. The GRMS submits the job to the Computing Resource with the use of the WS GRAM service. In order to submit the job the GRMS prepares the aforementioned job descriptor and sends it to the selected WS GRAM. In the described proof-of-concept environment the GRMS has been extended with the ability of preparing job descriptors that contain the previously defined checkpointing-related elements. As the GRMS itself accepts its own xml job descriptors provided by the users, this descriptor has also been extended with the additional checkpointing-related elements.

---

<sup>1</sup>`qsub` command submits the user job to the cluster to be executed, which job will be executed is defined within the user provided PBS batch script.

As the GRMS provides the command line interface, the environment described up to now is already sufficient to allow end users to submit jobs to the Grid and to indicate that their jobs have to be executed on the Computing Resource equipped with the checkpointing functionality. However, as the command line interface is not suitable for live demo purposes, the last element in a form of a user-friendly graphical interface has been introduced.

The mentioned GUI has been borrowed from the Migration Desktop project. As it is shown in Figure 1, the environment provided by this project consists of two main components: the GUI itself and the RAS server. Thanks to the Java Web Start Technology, the end user can fetch the GUI from the RAS server on whatever machine. The already started GUI provides robustness and a very intuitive way to control user jobs in a complex Grid environment. According to the user's commands, the MD GUI sends a request to the RAS server. The communication is based on the standardized SOAP protocol. The RAS server is the component that on behalf of the end user deals with all the complexity of the underlying Grid environment. In our case, the RAS server is in charge of forwarding the user requests to the GRMS. In fact, out-of-box RAS does not cooperate with the GRMS. However, both RAS and GRMS depend on SOAP messages, so the two components have been integrated thanks to the MD and GRMS development teams. Additionally, the GUI part of MD had to be extended with checkpointing-related widgets just to allow the end user to indicate that a given job has to be executed on the Computing Node that supports the checkpointing functionality.

At this point it is already the whole proof-of-concept environment. The two simple usage scenarios of the presented architecture are described in the next section.

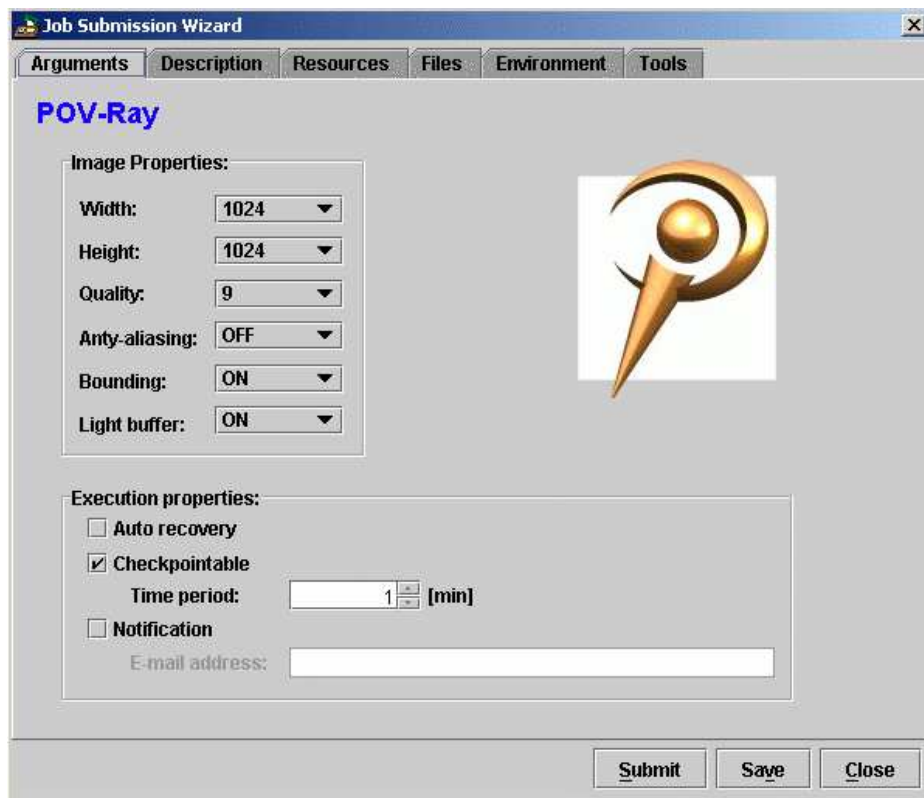


Figure 2: Migrating Desktop GUI

## 4 Demo scenario

Basing on the proof-of-concept environment described in the previous section we have performed a set of tests and experiments. Two of them have been prepared in an attractive form dedicated to be presented as a live demo at CoreGRID Industrial Conference [16] and are described in this section.

The outline of these two scenarios is similar. The user submits a POV-Ray based-job to the Grid using MD GUI. After the Grid Resource Broker assigns and submits the job to the Computing Resource, the failure of the computing infrastructure is simulated. Next, depending on the scenario, a user- or workflow-driven recovery action is performed. The scenarios utilize the POV-Ray application as the example user's job due to the following reasons: it can be checkpointed and recovered by means of an AltixC/R checkpointing package<sup>2</sup>, and in conjunction with MD GUI features it is possible to represent intermediate computing results (even the crash and recovery point can be noticed).

#### 4.1 User driven recovery scenario

By the term "user-driven recovery scenario" we mean that when the failure occurs, the recovery action has to be triggered by the user and not by the system itself. The scenario begins with the user dealing with the MD GUI. Using a dialog window (see figure 2), the user chooses the input parameters and files for the POV-Ray application and where the result files are to be stored. Additionally, the dialog window lets specify that the job is to be checkpointed<sup>3</sup> and the period of time in which the next checkpoints should be taken. When all parameters are already set, in order to submit the job to the Grid the user has to simply click the "Submit" button. From now on the job is submitted to the GRMS which is in charge of finding the Computing Resource which fulfills all the job's requirements (including the one related to the checkpointing functionality). The MD provides an easy-to-use Jobs Monitor on which the user can monitor the state of all submitted jobs. When GRMS ultimately executes the job on any Computing Resource, the state of the Jobs Monitor indicates the state of the job as ACTIVE. If a job is already in the ACTIVE state, the Jobs Monitor allows displaying the window with partial results generated by the job. As in this case the job is the just POV-Ray application, the partial result is simply the fragment of a graphical file. While the job is being executed, partial results presented to the user are refreshed on line at the same time. Basing on extended formats of underlying job descriptors, the checkpointing-related parameters are obviously forwarded up to the WS GRAM. Therefore the WS GRAM executes the job with the periodical checkpointing mechanism activated with the use of a customized PBS Job Manager.

The next step is failure simulation. It can be done in a few ways; however, we just find out the Computing Node where the job is executed and then kill the process that constitutes the job. The user can notice the failure just by observing the window with partial results (the image will stop refreshing) or by observing the state of job displayed by the Jobs Manager. It is also possible that the user is notified about the failure via email. No matter how the user gets informed about the failure, he or she would surely like to recover the job from the point where the last checkpoint was taken. To do so, the failed job has to select on the list of jobs in the Jobs Monitor and then the "Resubmit" button has to be clicked. After that action a simple wizard allowing to adjust the checkpointing-related parameters appears. Finally, the "Submit" button submits the job to the GRMS which further submits the job to the same Computing Resource on which it was originally executed. As the checkpointing-related parameters are forwarded from MD GUI up to the WS GRAM, the PBS Job Manager recovers it instead of executing the job from the beginning.

#### 4.2 Workflow driven recovery scenario

The workflow driven scenario is even more attractive because the job failures are automatically detected and recovered by the GRMS. This functionality has been achieved thanks to the GRMS feature that lets manage complex workflows. The workflow consists of a number of tasks and an individual task can be triggered by status changes of other tasks. Therefore, when the task under which the POV-Ray is running changes the state to the FAILED, the workflow manager automatically triggers the task that recovers the failed job.

To submit the job to the Grid in this scenario, the end user uses the same dialog window as in the first scenario but additionally the check box labeled "Auto recovery" has to be selected. After the job is submitted to the Grid, the window with partial results can be displayed. However, the failure simulation does not stop refreshing the window. It means that when the job fails, the GRMS itself will prepare and submit an appropriate job descriptor to the related WS GRAM.

---

<sup>2</sup>Each checkpointing package is able to checkpoint only a finished set of applications.

<sup>3</sup>The dialog window has a special check box labeled "Checkpointable". It allows to indicate that a job is to be checkpointed.

## 5 Further work

From the technical point of view, the demo described in the paper is the simplest proof-of-concept implementation related to the work on GCA. There are more advanced implementations extended with the cluster-level and Grid-level jobs migration features. However, these implementations lack convenient GUI interface, so they do not suit the live-demo requirements very well. The most advanced proof-of-concept implementation is the Grid Checkpointing Service (GCS) [17] which provides a flexible framework allowing for integration of any low-level checkpointer with the Grid environment. However, to become a production-quality service the GCS should be extended with features related to: garbage collection of outdated images, access control to the images and images encryption and versioning.

## 6 Conclusions

The proof-of-concept environment described in the paper has been developed as part of work on the GCA [3] [4]. Nevertheless, the presented environment does not constitute the implementation of GCA as such. The environment has been utilized to check the feasibility of ideas the GCA is based on. So far the results of the tests and experiments that were performed have been promising and proved sanity of the most recent GCA assumptions. In particular, the possibility of conscious interacting of the Grid Resource Broker with Computing Resource equipped with low-level checkpointing package has been confirmed. It has also been proved that knowledge of the given low-level checkpointing package allows to provide scripts or programs that generalize the low-level checkpointing interface to a more abstract form exposed to higher layers of the system. Simultaneously, the described proof-of-concept environment was able to improve the fault-tolerance level of the Grid computing environment. Even though failures were simulated, the workflow- and user-driven recovery procedures were able to recover the computing process to the point of the most recent checkpoint. Thanks to that, no significant computing cycles wasting took place and potential time or CPU cycles limitations imposed on the computing process can be fulfilled.

The described integration was presented to the industry community at CoreGRID Industrial Conference 2006 in Sophia-Antipolis in France [16]. The scientific community was also presented the demo at CoreGRID Industrial Showcase, June 4-5, in Barcelona. This event was co-located with the 23rd Open Grid Forum.

## Acknowledgment

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

## References

- [1] A survey of Checkpoint/Restart Implementations, Eric Roman, Lawrence Berkley National Laboratory, CA.
- [2] <http://www.checkpointing.org>
- [3] Gracjan Jankowski, Jozsef Kovacs, Norbert Meyer, Radoslaw Januszewski and Rafal Mikolajczak, Towards Checkpointing Grid Architecture, PPAM2005 proceedings.
- [4] Gracjan Jankowski, Rafal Mikolajczak, Radoslaw Januszewski, Jozsef Kovacs, Attila Kertesz, Maciej Stroinski, Grid Checkpointing Architecture - Integration of low-level checkpointing capabilities with GRID, CoreGRID Technical Report TR-0075.
- [5] <http://desktop.psnk.pl/>
- [6] Mirosław Kupczyk, Rafal Lichwala, Norbert Meyer, Bartek Palak, Marcin Plociennik, Maciej Stroinski, Pawel Wolniewicz, The Migrating Desktop - the General Entry Point to the Grid, 6th CARNET Users Conference, 27-29 September 2004, Zagreb, Croatia
- [7] Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak, Checkpoint/Restart mechanism for multiprocess applications implemented under SGIGrid Project, CGW2004.

- [8] <http://checkpointing.psnc.pl/>
- [9] <http://www.globus.org/toolkit/docs/4.0/execution/wsgam/>
- [10] <http://www.globus.org/toolkit/>
- [11] [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsrf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf)
- [12] <http://www.clusterresources.com/pages/products/torque-resource-manager.php>
- [13] <http://www.gridlab.org/WorkPackages/wp-9/>
- [14] <http://www.gridge.org/>
- [15] Juliusz Pukacki, Michal Kosiedowski, Rafal Mikolajczak, Marcin Adamski, Piotr Grabowski, Michal Jankowski, Mirosław Kupczyk, Cezary Mazurek, Norbert Meyer, Jarek Nabrzyski, Tomasz Piontek, Michael Russell, Maciej Stroinski, Marcin Wolski, Programming Grid Applications with Gridge, COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY 12(1), X-XX (2006)
- [16] <http://www.coregrid.net/mambo/content/view/262/288/>
- [17] Gracjan Jankowski, Radosław Januszewski, József Kovacs Grid Checkpointing Service - Integration of low-level checkpointing packages with the Grid environment, 3rd CoreGRID Workshop on Grid Middleware, June 5-6, 2008, Barcelona, Spain.