GWD-I.92
GridCPR-WG

Rosa Badia, Barcelona Supercomputing Center
Robert Hood, NASA
Thilo Kielmann, Vrije Universiteit
Andre Merzky, Vrije Universiteit (editor)
Christine Morin, INRIA Rennes
Stephen Pickles, University of Manchester
Massimo Sgaravatto, INFN Padova
Paul Stodghill, Cornell University (editor)
Nathan Stone, Pittsburgh Supercomputing Center
Heon Y. Yeom, Seoul National University

Version: 1.0 RC.2

April 12, 2007

Use-Cases and Requirements for Grid Checkpoint and Recovery

Status of This Document

This document provides information to the Grid community regarding use-case scenarios for Grid Checkpointing and Recovery. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

This document describes use-cases to be addressed by the Grid Checkpoint and Recovery Working Group (GridCPR WG). The scenarios are also used to determine a set of requirements for these standards.

Contents

1 Introduction

One of the goals of the Grid Checkpointing and Recovery Working Group (GridCPR WG)

> *is to define a user-level API and associated layer of services that will permit checkpointed jobs to be recovered and continued on the same or on remote Grid resources [25].*

In order to understand the requirements of these APIs and services, it is necessary to understand the situations in which they will be used. The purpose of this document is to enumerate usage scenarios that the GridCPR WG have decided must be addressed by its specifications. These scenarios will be used to derive a set of requirements for the GridCPR API and services.

There are two fundamental approaches to saving the state of a job, Application-Level CPR (ALC) and SystemLevel CPR (SLC). In Application-Level CPR (ALC), the checkpointing and recovery function is performed by the application. That is, the applications source code contains explicit instructions for performing the CPR. In this case, the critical program variables and data structures are saved. In System-Level CPR (SLC), checkpointing and recovery are done external to the application, i.e. on behalf of the application without modification of the application itself. This may mean saving the processors registers, stack, and memory at the point that the checkpoint is taken.

The current scope of GridCPR WG as described in its charter includes Grid applications that implement ALC. The use-cases in Sections 2 and 3 rely on ALC to various varying degrees. Use-cases based on SLC were judged to be outside the WGs current scope and are discussed in Section 4.

Use-cases that are in scope can be further classified as either describing applications or libraries that could use GridCPR systems or systems that implement CPR and could evolve to be GridCPR implementations. The former type are discussed in Section 2, while Section 3 discuss the latter type. The in scope use-cases can be further classified. Some of the use-cases describe applications or libraries that could use GridCPR systems. We call these consumers of GridCPR function and describe them in Section 2. Other use-cases describe systems that implement CPR and could one day evolve to be GridCPR systems. We call these producers of GridCPR function and describe them in Section 3.

What GridCPR is not

System level checkpointing is a widely researched topic, and also used in Grid environments, in particular in connection with virtualized resources and operating systems. The GridCPR working group in general, and this document in particular, does *not* however, address system level checkpointing, nor the interaction between applications and system level checkpointing. The focus of the Group and this document is solely on application level checkpointing.

2 Consumer Use-Cases Within Scope

2.1 C^3

The C^3 system is a precompiler that can be used to add ALC to existing application source code. The developer adds directives to the application source code to indicate the points at which checkpoints can be taken, and C^3 uses program transformations to augment the application with code to save critical program variables and data structures. C^3 also uses compiler optimizations to reduce the size and overhead of taking checkpoints.

The C^3 system also provides features to ensure that consistent checkpoints of MPI jobs [9, 10]. Work is currently underway to extend this work to handle truly automatic portable checkpointing within a Grid-environment. This will be done using type-safe language (e.g., Java, C#) or type-safe dialects (CCured, Cylone) and process over-decomposition.

The C^3 system has several runtime subsystems that are responsible for providing the checkpoint file and signal management. These subsystems could easily be replaced with a GridCPR system. In this way, C^3 is a user of GridCPR systems.

Functional requirements

- API for application state writing and reading.
- Services for failure notification.
- Services for checkpoint data transport.
- Services for checkpoint data management.

2.2 Cactus

[3, 7, 14] provides application-level checkpointing for any codes written within the framework. The functionality is transparent to the application developer and user, and checkpointing and recovery is usually requested in the parameter file at run time. Checkpointing may also be requested dynamically during a run through a steering interface such as the HTTPD web interface, or the application code can dynamically react to simulation data and request checkpointing itself through the Cactus API. Checkpoints can be written and read in several different architecture independent binary data formats, including HDF5 and FlexIO. Parallel Cactus applications can be checkpointed on one machine and then recovered on a different architecture machine using a different number of processors.

Functional requirements

- API for application state writing and reading.
- Services for failure notification.
- Services for checkpoint data transport.
- Services for checkpoint data management.

2.3 RealityGrid

The RealityGrid [4, 21, 20] project provides limited support for jobs that contain ALC functionality. RealityGrid does not provide a complete set of CPR services. Rather, application developers are expected to instrument applications to read and write checkpoint files in whatever format they want and RealityGrid provides functions for managing and transporting these files. RealityGrid is able to provide transport within a heterogeneous computing environment, but it is the developers responsibility to generate heterogeneous checkpoint files.

RealityGrids CPR support enables jobs to implement fault-tolerance and to implement strategies for long running computations to save state at the end of a fixed length batch run. In either case, jobs can be restarted from checkpoints in subsequent batch allocations. This support also enables job migration in heterogeneous computing environments.

RealityGrid also enables jobs to provide rewind capabilities that are based upon the CPR system. Rewind can be used not only for debugging, but for computational steering as well [11, 12]. A common use involves the user running a computationally intensive simulation in a mode that produces low-resolution results. The user can then rewind the simulation to a point of interest and rerun it with options to produce high-resolution results only in the region of interest.

The RealityGrid project enables this sort of parametric exploration by supporting checkpoint trees [22]. That is, the RealityGrid system enables checkpoint files to be linked together in a manner that encodes the causal relationship between them. This enables a user to construct and manage exploration trees. Combined with the visualization tools rewind features discussed in the previous section, these capabilities provide the computational scientist with very powerful tools for scientific discovery.

RealityGrid applications and the steering library have the following functional requirements:

Functional requirements

- API for application state writing and reading.
- Services for checkpoint data transport.
- Services for checkpoint data management.
- Services for job management.

Technically, RealityGrid is also a provider of GridCPR functions. It provides the following key functions:

- Services for checkpoint data transport.

2.4 XCAT3

XCAT3 [19] is a Common Component Architecture application framework based on Grid standards. One of the functions that XCAT3 provides is checkpointing for CCA-based applications. Because these applications can be executed on a number of distributed computing resources, consistency is a consideration when checkpointing. XCAT3 handles this by providing Application Coordinators. When a checkpoint is required, the user or some other an agent notifies the Application Coordinator, which then executes a blocking coordination protocol between the distributed components.

In order to provide checkpointing within a heterogeneous computing environment, XCAT3 uses application-level checkpointing. Also, checkpoint data is stored in XML to ensure maximum portability.

In order to ensure the availability of checkpoint data in the event of processor failure, XCAT3 assumes a Storage Service Federation, which can provide stable storage for checkpoint data.

Functional requirements

- API for application state writing and reading.
- Services for checkpoint data management.

2.5 GridWay Metascheduler

The GridWay Metascheduler [15, 16], now a Globus project, adapts job execution to changing grid conditions by providing fault recovery mechanisms,

dynamic scheduling, migration on-request and opportunistic migration [17]. Migration is implemented by restarting the job on the new candidate host, therefore the job should generate restart files at regular intervals in order to continue execution from a given point. If checkpointing files are not provided, the job is restarted from the beginning. GridWay periodically retrieves to the client machine or a checkpoint server (GridFTP URL) the restart architecture-independent files.

Jobs submitted with GridWay could benefit from GridCPR systems providing standard and uniform APIs and services for portable checkpoint generation and storage.

Functional requirements

- API for application state writing and reading.
- Services for failure notification.
- Services for checkpoint data management.

3 Producer Use-Cases Within Scope

3.1 SRS

The Stop Restart System (SRS) [27] provides a user-level checkpointing library and a Runtime Support System (RSS) that manages the checkpointed data. A unique feature of SRS is that it allows for reconfiguration of the executing MPI application both in terms of the number of machines used for application execution and the data distributions used in the application between checkpoints and continuations. SRS is primarily intended for Grid scheduling and resource management systems to migrate executing parallel application across distributed heterogeneous sites that do not share common file systems. It also provides fault-tolerance by enabling the application to withstand and recover from non-deterministic errors caused during application execution.

SRS provides for the transport of checkpoint data between Grid computing resources using IBP [23]. Applications register with an external agent, the RSS, in order to transfer information about checkpoint locations and to coordinate job management.

- API for application state writing and reading.

- Services for limited job management.

An external agent, the RSS, is used for maintaining configuration information across job instances and for coordinating job stopping and resuming.

- Services for checkpoint data management and transport. The

RSS maintains information about a jobs checkpoint data and checkpoint data is moved between computing resources using IBP.

3.2 TCS

The checkpoint system for the Pittsburgh Supercomputer Centers Terascale Computing System (TCS) [26] allows for the automated recovery of jobs following both machine failures and scheduled maintenance periods. When a node failure is detected, the system determines whether that node was currently running a users job. If so, the user account is credited for the lost of time and the job is rescheduled for further execution.

The system also provides for user termination and migration of jobs. The user is provided with interfaces for checkpoint and halting a running job, for migrating the applications checkpoint and data files to a different computational resource, and for resuming the job on the new resource. There is also a means of querying the CPR system about the state of jobs and checkpoint data.

This checkpointing system is not transparent to the application; the user must modify their application to use the appropriate APIs. Also, if the user wished to migrate a running job to a different cluster, then the user is responsible for ensuring that the checkpoint data is written in a portable manner.

One of the novel features of TCS is that it allows the user to set the policy for where checkpoint data should be stores. Currently supported policies include on node-local disks, using a parity scheme over several nodes, and entirely off-processor.

Key functions

- API for application state writing and reading.
- Services for failure notification.
- Services for job management.

- Services for checkpoint data transport.
- Services for checkpoint data management.
- Collaboration with accounting services.

3.3 European DataGrid

CPR in the European DataGrid (EDG) [13] and EGEE [2], its successor project, is used to provide some form of fault-tolerance to jobs, which is particularly important for long-running jobs, such as those in High Energy Physics. In this system, the developer is responsible for determining the job state that must be saved and restored. The system is responsible for noticing failures and automatically resubmitting jobs for further execution.

The primary purpose of Grid checkpointing within the EDG project [13] is for fault-tolerance. In the event of a failure, it attempts to avoid having to re-run jobs from the beginning. This provides better resource utilization, since computations are only performed. This is in particular important for long running jobs, as is the case for many of the target HEP (High Energy Physics) applications that can run for many hours or days.

In the EDG, the user is responsible for determining what part of the job state must be saved in order to correctly restart. It is also up to the user to determine the points in jobs execution at which the state must be saved. Furthermore, the application Also, the application must be instrumented to be able to restart from a previously saved state. This is all done by instrumenting the code with the proper EDG Grid Checkpointing APIs.

Given this framework, EDG now supports two main use-cases:

- An instrumented job runs on a computing resource and periodically saves its state. Lets suppose that a Grid failure, i.e. a failure external to the job (e.g. a failure in the computing resource where the job was running) occurs. If the Grid middleware is able to detect the failure, the EDG Workload Management System automatically (assuming that the user has enabled this option) reschedules the job and resubmits the job to a (possibly different) compatible resource. When the job restart its execution, the last saved state is retrieved, and the application restarts the computation from that point.
- If some other undetected failure occurs while an instrumented job is running, EDG allows the user to manually restart the job from one of the previously saved checkpoints. Although it is not possible to use this approach to recover from arbitrary failures (e.g., incorrect input data), it

is possible to correct certain failures before resuming (e.g., missing input data file).

Another scenario where job checkpointing is used in the EDG environment is called job partitioning. The idea is that a job can be partitioned in sub-jobs, which can be executed in parallel. Then a job aggregator is responsible to collect the results of these sub-jobs (represented by their final states) and provides the overall results.

The EDG project also plans to exploit Grid Checkpointing for job preemption. In this scenario, it might be necessary to migrate jobs from a computational resources for a certain reason (e.g. because that machine must be used to run an other job with higher priority), but this functionality is not yet supported.

Key functions

- API for application state writing and reading.
- Services for failure notification.
- Services for job management.
- Services for checkpoint data transport.
- Services for checkpoint data management.
- Services for sub-job result collection and aggregation.
- Services for priority-based scheduling and preemption (planned).

3.4 GridLab

In the GridLab project [24, 5], a job, consisting of one or more processes, is running on a Grid machine. In the middle of the run, the job may be forced to migrate to a different machine, possibly with a different architecture and/or number of CPUs. The application program may either decide by itself to migrate (e.g. poor performance on the current machine) or may be forced to do so, either by the user (via an application manager) or by the local resource management software that wishes to evict the job. The main purpose of GridCPR in GridLab thus is the ability to interrupt and migrate a job until it finally terminates. Fault-tolerance is only a secondary aspect.

An extension of the above use-case is dealing with jobs that run concurrently at multiple Grid sites.

Applications save their state to regular files. Checkpoint meta data can be stored in GridLabs "advert service", allowing the checkpoint file(s) to be found and retrieved after restart. File transport is done via GridLabs data movement service (or via GridFTP) [6].

Key functions

- Services for checkpoint data transport, via GridLabs data movement service or GridGTP.
- Services for checkpoint data management, via Advert Service.
- Services to enable checkpoint of jobs at multiple Grid sites.

4 Use-Cases Outside of Scope

A number of SLC-based use-cases were submitted to this working group for consideration. Since they do not allow application developers to write portable, resource- independent code to handle checkpointing and recovery operations in a consistent manner across different Grid resources, SLC systems are outside of the current scope of the GridCPR working group. We mention them here for completeness [1].

4.1 Kerrighed.

The Kerrighed system [8] provides a single system image OS than can be run within and across clusters. Its goal is to provide SLC of sequential and parallel jobs, and to enable transparent failover and migration of such jobs within a cluster federation. Loosely-coupled distributed jobs are handled by forcing processes to checkpoint when certain communication occurs and by affixing certain causality information to messages.

4.2 MPICH-GF.

MPICH-GF [18] supports user-transparent SLC for fault tolerance of MPI jobs running within a homogeneous computing environment. MPICH-GF is provided as a library that is linked with the unmodified application source code.

[1] One can envision SLC checkpointing mechanisms leveraging functions of GridCPR systems (e.g., checkpoint data reading/writing, transport and management). However, since they are fundamentally system-level, homogeneous or not Grid related, as applications ??, they are outside of scope.

The system provides checkpointing and message logging and a job management system that monitors the job, periodically sends checkpoint signals to the job, and restart the job if a failure occurs.

4.3 Dj Vu.

Dj Vu [1] provides transparent SLC for stock native jobs. In addition to state-saving, it also uses dynamic linking to provide alternative versions of certain system libraries. This enables Dj Vuto support the execution of certain system calls across checkpoints and to support reliable transport protocols for network communication.

5 Requirements

There are a minimum set of APIs and services that are required in order to implement the use-cases described above outlined in this document. In this section, we discuss these requirements, vis-a-vis the use-cases. In the The Architecture document, that will also be also being developed by this working group, will be described these APIs and services in much greater detail.

Figure 1 shows the relationship between an application and the various APIs and services, from the application point of view. The first thing to notice is that the The outer box labelled Application contains a box labelled Computation, which accounts for some of the those use-cases which envisions an existing computation being directly modified to interact with a GridCPR system. This could be done via an automatic program transformation, tool like C^3, or by changes to an application framework, like Cactus or XCAT3. In other use-cases, there is an agent that is external to the core computation that is responsible for ensuring the continuation of the computation. EDG and RealityGrid are examples of this.

Whatever the configuration, the Application interacts with the GridCPR system via a set of four APIs[2]. In addition to the APIs, several of the use-cases also presupposed the existence of certain services. These are shown as disconnected components of the archicture in Figure 1. That is, the services are necessary for the use-cases to be implemented, but the Applications do not necessarily need to directly interact with these services.

The APIs shown in Figure 1 are as follows:

[2]None of the use-cases clearly required accessing the GridCPR system by, for instance, command-line or GUI tools, but these tools are clearly desirable and implementable using these APIs.

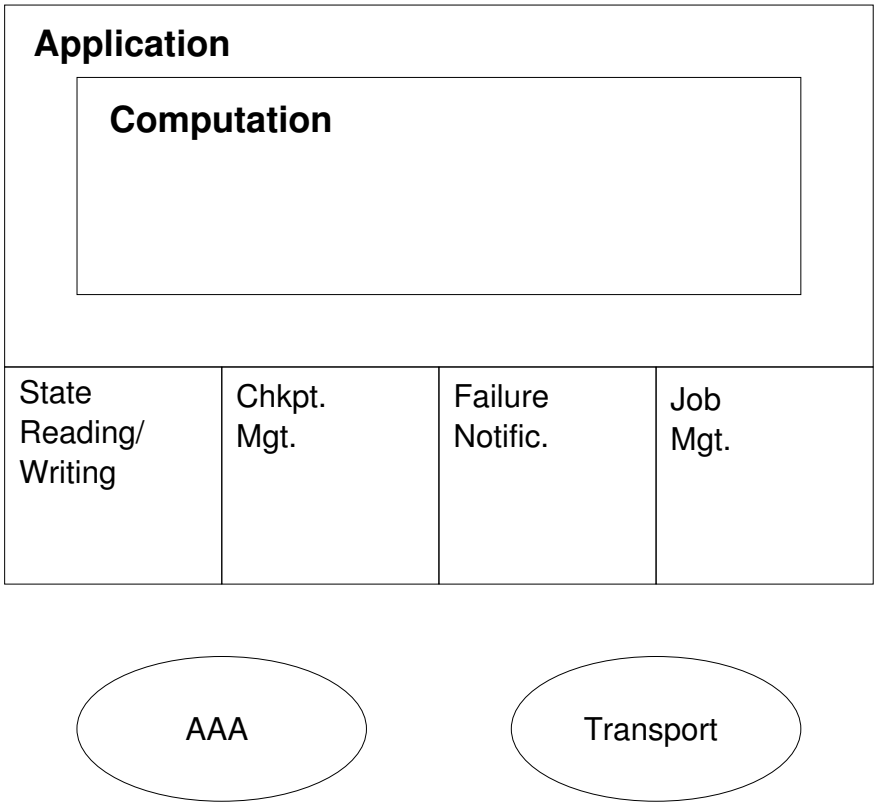

Figure 1: Architecture from Application point-of-view

Application state writing and reading. Functions must be provided for writing and reading the application variables to the checkpoint data that is managed by the GridCPR system.

Checkpoint data management. Once the checkpoint data has been created, it needs to be stored and managed. For example, there needs to functions for deleting checkpoint data that is no longer needed. Also, this API should provide a mechanism for querying the meta-data that is associated with checkpoint data.

Failure/event notification. Some of the use-cases provide agents that reschedule jobs that fail during their execution. In order to do this, there must be a mechanism for the agents to discover that failures have occurred.

Job management. In order for an agent to reschedule a failed job, these must be functions for interacting with a scheduling service.

It is assumed that there are services associated with the APIs described above.

In addition to these, the following services are shown in Figure 1:

Checkpoint data transport. In order for a job to be started on a different Grid computing resource than the one on which its checkpoint data was created, there must be a mechanism for transferring checkpoint data between Grid computing resources. Several of the use-cases, such as SRS and TCS, currently provide such transport mechanism for checkpoint data.

Authentication, authorization, and accounting. None of the use-cases explicitly discussed security, but data integrity is clearly a necessary requirement for Grid computing. There must be associated services to support these functions. One of the use-cases mentioned crediting a users account for time lost when a computing resource fails.

6 Summary

This document derives a set of requirements for Grid Checkpoint and Recovery Systems, by analyzing a number of CPR use-cases and several existing CPR systems for distributed environments. With the focus on application level checkpointing, this document identifies, (i) a set of APIs required to allow applciations to write, manage and use checkpoints, and (ii) several services required to implement such a CPR system in Grid environments.

These requirements are to be consumed by the GridCPR working group, and are to be taken into regard by designing the GridCPR API, and by defining a GridCPR architecture. These requirements will be consumed by the GridCPR working group, and will influence the design of a GridCPR architecture and the definition of a GridCPR API.

7 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0085969. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reect the views of the National Science Foundation.

8 Security Considerations

As stated at the end of Section 5, none of the use-cases explicitly discussed security, but data integrity is an obvious requirement for Grid computing.

Apart from that specific requirement, any implementation of a GridCPR system should strive to ensure secure communication, data (i.e. checkpoint) management etc, as usual. These requirements are, however, not specific for CPR systems, but apply to Grid service infrastructures in general, and hence, are out of scope for this document.

9 Intellectual Property Issues

9.1 Editor information

This document is the result of the joint efforts of several authors, listed on the title page. The editors listed here are those committed to taking permanent stewardship for this document. They can be contacted in the future for inquiries about this document.

Paul Stodghill
stodghil@cs.cornell.edu
Upson Hall, Cornell University
Department of Computer Science
Ithaca, NY, 14853
United States
USA Phone: 607-254-8838

Andre Merzky
andre@merzky.net
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

9.2 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

9.3 Disclaimer

9.4 Full Copyright Notice

References

[1] FIXME: Missing DejaVu Reference.

[2] Enabling Grids for E-sciencE (EGEE). `http://public.eu-egee.org/`.

[3] The Cactus Code Server. `http://www.cactuscode.org/`.

[4] The RealityGrid Project. `http://www.realitygrid.org/`.

[5] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[6] G. Allen, T. Goodale, H. Kaiser, T. Kielmann, A. Kulshrestha, and R. v. N. Andre Merzky. A Day in the Life of a Grid-Enabled Application: Counting

on the Grid. In *Proceedings of the Workshop on Grid Application Programming Interfaces September 20, 2004, Brussels, Belgium In conjunction with GGF12*, 2004.

[7] G. Allen, T. Goodale, G. Lanfermann, T. Radke, D. Rideout, and J. Thornburg. Cactus Users Guide, 2004.

[8] R. Badrinath, C. Morin, and G. Valle. Checkpointing and recovery of shared memory parallel applications in a cluster. In *Proc. Intl. Workshop on Distributed Shared Memory on Clusters (DSM 2003, Held in conjunction with CCGrid 2003.)*, page 471477, May 2003. `http://www.inria.fr/rrrt/rr4806.html`.

[9] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill. Automated application-level checkpointing of MPI programs. In *ACM Symposium on Principles and Practice of Parallel Programming (PPoPP 2003)*, 2002.

[10] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill. Collective Operations in an Applicationlevel Fault Tolerant MPI System. In *International Conference on Supercomputing (ICS) 2003, San Francisco, CA, June 2326*, 2003.

[11] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning, and A. R. Porter. Computational Steering in RealityGrid. In *Proceedings of the UK e-Science All Hands Meeting*, September 2-4 2003. `http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf`.

[12] J. Chin, J. Harting, S. Jha, P. V. Coveney, A. R. Porter, and S. M. Pickles. Steering in Computational Science: Mesoscale Modelling and Simulation. *Contemporary Physics*, 44:417 – 434, 2003. `http://taylorandfrancis.metapress.com/openurl.asp?genre=article&eissn=13665812&volume=44&issue=5&spage=417`.

[13] A. Gianelle, R. Peluso, and M. Sgaravatto. Datagrid: Job Partitioning and Checkpointing, June 3 2002. `https://edms.cern.ch/document/347730`.

[14] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *5th International Conference on Vector and Parallel Processing - VECPAR2002*, Lecture Notes in Computer Science, Berlin, 2003. Springer.

[15] GridWay Metascheduler. `http://www.gridway.org/`.

[16] E. Huedo, R. Montero, and I. Llorente. A framework for adaptive execution on grids. *Software – Practice and Experience*, 34(7):631–651, 2004.

[17] E. Huedo, R. S. Montero, and I. M. Llorente. Evaluating the reliability of computational grids from the end user's point of view. *Journal of Systems Architecture*, 52(12):727–736, 2006.

[18] S. Kim, N. Woo, H. Y. Yeom, T. Park, and H. Park. Design and Implementation of Dynamic Process Management for Grid-enabled MPICH. In *Proceedings of the 10th European PVM/MPI Users Group Conference, Venice, Italy*, September 2003.

[19] S. Krishnan and D. Gannon. Checkpoint and Restart for Distributed Components in XCAT3. *Grid 2004, 5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[20] S. Pickles. On the Use of Checkpoint/Recovery in RealityGrid, January 2004.
`http://gridcpr.psc.edu/GGF/docs/ReG-GridCPR-use-cases.pdf`.

[21] S. Pickles, R. Pinning, A. Porter, G. Riley, R. Ford, K. Mayes, D. Snelling, J. Stanton, S. Kenny, and S. Jha. The RealityGrid Computational Steering API – Version 1.0. Technical report, July 2003. unpublished.

[22] S. M. Pickles, P. V. Coveney, and B. M. Boghosian. Transcontinental RealityGrids for Interactive Collaborative Exploration of Parameter Space (TRICEPS). Winner of SC03 HPC Challenge competition in the category Most Innovative Data-Intensive Application, `http://www.scconference.org/sc2003/inter_cal/inter_cal_detail.php?eventid=10701#5`.

[23] J. S. Plank, M. Beck, W. R. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the Network. In *NetStore99: The Network Storage Symposium, Seattle, WA, USA*, 1999.

[24] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski. GridLab - a Grid Application Toolkit and TestBed. *Future Generation Computer Systems – Grid Computing: Towards a new Computing Infrastructure*, 18(8):1143–1153, October 2002.

[25] D. Simmel, T. Kielmann, and N. Stone. Charter, Grid Checkpoint Recovery Working Group (GridCPR). Technical report, Global Grid Forum, January 2004.
`http://gridcpr.psc.edu/GGF/charter/GridCPR-WG-charter.1.2.txt`.

[26] N. Stone, J. Kochmar, R. Reddy, J. R. Scott, J. Sommerfield, and C. Vizino. A Checkpoint and Recovery System for the Pittsburgh Supercomputing Center Terascale Computing System. Technical report, Pittsburgh Supercomputing Center, December 3 2003. `http://www.psc.edu/publications/tech_reports/chkpt_rcvry/checkpoint-recovery-1.0.html`.

[27] S. Vadhiyar and J. Dongarra. SRS: A Framework for Developing Malleable and Migratable Parallel Applications for Distributed Systems. *Parallel Processing Letters*, 13(2):291–312, 2003.