Rosa Badia, IBM Barcelona
Robert Hood, NASA
Thilo Kielmann, Vrije Universiteit
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
November 5, 2004

GridCPR
Request for Comments: GWD-XXX-00x-7
Obsoletes: number
Category: informational

Use-Cases for Grid Checkpoint and Recovery

Status of This Memo

This memo 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. This is a DRAFT document and continues to be revised.

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.

Full Copyright Notice

Intellectual Property Statement

Contents

1 Introduction

One of the goals of the Grid Checkpointing and RecoveryWorking 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. [21]

In order to understand the requirements of these API's 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 System-Level CPR (SLC). In *Application-Level CPR (ALC)*, the checkpointing and recovery function is performed by the application. That is, the application's 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 done external to the application, i.e. on the application's behalf without modification of the application itself. This may mean saving the processor's 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 degrees. Use-cases based on SLC were judged to be outside the WG's current scope and are discussed in Section 4.

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.

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 [5, 6]. 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

[8, 1, 12] 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 [20, 17, 16] 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 developer's responsibility to generate heterogeneous checkpoint files.

RealityGrid's 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 [7, 9]. 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" [18]. 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,

- 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 [15] 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 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.

3 Producer Use-Cases Within Scope

3.1 SRS

The **S**top **R**estart **S**ystem (SRS) [23] 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 [19]. 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 job's checkpoint data and checkpoint data is moved between computing resources using IBP.

3.2 TCS

The checkpoint system for the Pittsburgh Supercomputer Center's Terascale Computing System (TCS) [22] 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 user's job. If so, the user's 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 API's. 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) [11] 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 [11] is for fault-tolerance. In the event of a failure, it attempts to avoid having to rerun 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 job's 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 API's.

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

- An instrumented job runs on a computing resource and periodically saves its state. Let's 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 [13, 2], 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 GridLab's "advert service", allowing the checkpoint file(s) to be found and retrieved after restart. File transport is done via GridLab's data movement service (or via GridFTP) [3].

Key functions:

- Services for checkpoint data transport, via GridLab's 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].

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

Kerrighed. The Kerrighed system [4] 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.

MPICH-GF. MPICH-GF [14] 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. 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.

Déjà Vu. Déjà Vu[10] 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 Déjà Vuto support the execution of certain system calls across checkpoints and to support reliable transport protocols for network communication.

5 Summary and Requirements

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

Figure 1 shows the relationship between an application and the various API's and services. The first thing to notice is that the box labelled "Application" contains a box labelled "Computation". Some of the use-cases 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.

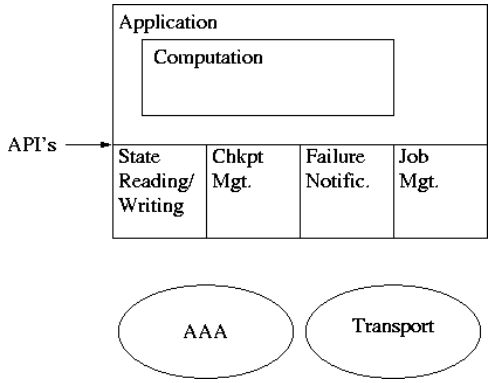

Figure 1: Architecture from Application point-of-view

Whatever the configuration, the Application interacts with the GridCPR system via a set of four API's[2]. In addition to the API's, 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 API's shown in Figure 1 are as follows,

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.

[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 API's.

Checkpoint data management. Once the checkpoint data has been created, it needs to be 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 API's 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 user's account for time lost when a computing resource fails.

6 Editor Information

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

Acknowledgments

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 reflect the views of the National Science Foundation.

References

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

[2] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf, and Ian Taylor. Enabling applications on the grid: A gridlab overview. *International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications*, August 2003.

[3] Gabrielle Allen, Tom Goodale, Hartmut Kaiser, Thilo Kielmann, Archit Kulshrestha, Andre Merzky, and Rob van Nieuwpoort. A day in the life of a grid-enabled application: Counting on the grid. In *Workshop on Grid Application Programming Interfaces*, Brussels, Belgium, September 20 2004. Held In Conjunction With GGF12. Available at http://www.cs.vu.nl/ kielmann/papers/ggf12ws-gat.pdf.

[4] Ramamurthy Badrinath, Christine Morin, and Geoffroy Vallée. Checkpointing and recovery of shared memory parallel applications in a cluster. In *Proc. Intl. Workshop on Distributed Shared Memory on Clusters (DSM 2003)*, pages 471–477, May 2003. Held in conjunction with CCGrid 2003. Available at http://www.inria.fr/rrrt/rr-4806.html.

[5] Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. Automated application-level checkpointing of MPI programs. In *ACM Symposium on Principles and Practice of Parallel Programming (PPoPP 2003)*, 2002.

[6] Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. Collective operations in an application-level fault tolerant MPI system. In *International Conference on Supercomputing (ICS) 2003*, San Francisco, CA, June 23–26 2003.

[7] 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. Available at http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/179.pdf.

[8] The cactus code server. http://www.cactuscode.org, May 14 2004.

[9] 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. Available at http://taylorandfrancis.metapress.com/openurl.asp?genre=article&eissn=1366-5812&volume=44&issue=5&spage=417.

[10] *fixme: Missing dejavu reference*.

[11] Alessio Gianelle, Rosario Peluso, and Massimo Sgaravatto. Datagrid: Job partitioning and checkpointing. https://edms.cern.ch/document/347730, June 3 2002.

[12] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing - VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[13] Gridlab: A grid application toolkit and testbed. Available at http://www.gridlab.org/, July 21 2004.

[14] Sangbum Kim, Namyoon Woo, Heon Y. Yeom, Taesoon Park, and Hyoungwoo 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.

[15] Sriram Krishnan and Dennis Gannon. Checkpoint and restart for distributed components in xcat3. In *Grid 2004, 5th IEEE/ACM International Workshop on Grid Computing*, November 2004.

[16] Stephen Pickles. On the use of checkpoint/recovery in realitygrid. Available at http://gridcpr.psc.edu/GGF/docs/ReG-GridCPR-use-cases.pdf, January 2004.

[17] Stephen Pickles, Robin Pinning, Andrew Porter, Graham Riley, Rupert Ford, Ken Mayes, David Snelling, Jim Stanton, Steven Kenny, and Shantenu Jha. The realitygrid computational steering api - version 1.0. unpublished, July 9 2003.

[18] Stephen M. Pickles, Peter V Coveney, and Bruce M Boghosian. Transcontinental realitygrids for interactive collaborative exploration of parameter space (triceps). Available at http://www.scconference.org/sc2003/inter_cal/inter_cal_detail.php?eventid=10701#5. Winner of SC'03 HPC Challenge competition in the category "Most Innovative Data-Intensive Application".

[19] James S. Plank, Micah Beck, Wael R. Elwasif, Terry Moore, Martin Swany, and Rich Wolski. The internet backplane protocol: Storage in the network. In *NetStore99: The Network Storage Symposium*, Seattle, WA, USA, 1999.

[20] The RealityGrid Project. Realitygrid. http://www.realitygrid.org.

[21] Derek Simmel, Thilo Kielmann, and Nathan Stone. Draft charter v.1.2, Grid Checkpoint Recovery Working Group (GridCPR). Technical report, Global Grid Forum, January 1 2004. Available at http://gridcpr.psc.edu/GGF/charter/GridCPR-WG-charter.1.2.txt.

[22] Nathan Stone, John Kochmar, Raghurama Reddy, J. Ray Scott, Jason Sommerfield, and Chad Vizino. A checkpoint and recovery system for the pittsburgh supercomputing center terascale computing system. http://www.psc.edu/publications/tech_reports/chkpt_rcvry/checkpoint-recovery-1.0.html, December 3 2003.

[23] 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.