GFD-I.93 Nathan Stone, Pittsburgh Supercomputing Center (editor)
GridCPR-WG Derek Simmel, Pittsburgh Supercomputing Center
Thilo Kielmann, Vrije Universiteit, Amsterdam
Andre Merzky, Louisiana State University (editor)

Version: 1.0 November 9, 2007

An Architecture for Grid Checkpoint and Recovery Services

Status of This Document

This document provides information to the Grid community regarding an architecture for Grid Checkpoint and Recovery services. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

This document describes an architecture for Grid Checkpoint and Recovery (GridCPR) services. The architecture is based upon findings of the OGF Grid Checkpoint Recovery Working Group (GridCPR-WG) in ongoing discussions and analysis of use cases for Grid Checkpoint and Recovery as described in the GFD-I.92 document, Use Cases for Grid Checkpoint and Recovery [1].

Contents

Preface

The GridCPR working group started to write this document in 2004/2005. After the public comment period in 2005, the document was idling for about two years. Several relevant OGF standards and architectures evolved further, and this document got out of sync with those developments. The authors, nevertheless, think that this informational document still is a useful contribution to the OGF standardization processes, and will in particular serve its main purpose, to drive the development of a CPR API, very well.

Instead of attempting a complete overhaul of the document, the current stewards and the OGF editor decided to finally publish the document "as is". The reader should, however, be aware that (a) the proposed CPR architecture is not a good match with the current WSRF/OGSA based architecture of OGF, and that (b) several of the proposed services by now exist, although often in a different form than proposed here.

1 Introduction

This document describes a GridCPR architecture in terms of the minimum required components for a complete implementation, component functionalities and work flow. The following excerpt from the GridCPR-WG charter is helpful in re-iterating the scope of this effort:

> *A key feature of Grid Checkpoint Recovery service is recoverability of jobs among heterogeneous Grid resources. In other words, resources on which jobs are checkpointed need not be of the same type as those on which the jobs are recovered, as long as the application code operating on the checkpointing resource can be built for and run on the recovery platforms.*

Scope of this document: This necessarily includes the ability to checkpoint and recover application state on a single Grid-connected resource, but also includes the migration of checkpointed jobs to other Grid-connected resources. This does not include the checkpoint and recovery of jobs running simultaneously across multiple Grid-connected compute resources (so called meta-computing). A more extensive list of the scope of this working group is described in the document Use Cases for Grid Checkpoint and Recovery [1].

This document does not detail the Applications Programmer Interface (API) for the GridCPR system, details of which will be covered in a GridCPR API Specification document at a later time. Furthermore, as an architectural prescription, this document does not address the implementation details of services presented herein nor exhaustively detail their interfaces. This information will be covered in a GridCPR Services Interface Specification document at a later time.

2 Components

The GridCPR System is a synthesis of several basic elements, which are all required for a complete GridCPR implementation. These are:

- **Prerequisite Grid Services**
  - standardized grid services on which GridCPR depends
- **GridCPR Library**
  - an interface to the GridCPR functions and services allowing applications to act through or upon the GridCPR system

- **GridCPR Services**
  - persistent services that provide access to GridCPR resources or perform actions on behalf of the user

- **GridCPR Resources**
  - storage, transfer and other resources that the GridCPR system uses for normal function

2.1 Prerequisite Grid Services

Higher-level grid services generally rely upon the features and availability of lower-level grid services; the GridCPR services are typical of higher-level grid services in this respect. Following are the lower-level grid services on which GridCPR relies.

Grid Authentication

A reliable means for authentication is required any time two processes communicate and authorization is required any time a process issues a request to a grid service. For example, in the GridCPR context an application can query the availability of checkpoint data from the State Management service via the GridCPR library. Although authentication/authorization issues per se are outside the scope of the GridCPR charter, other working groups like the Open Grid Service Architecture Authorization (OGSA AUTHZ-WG) group (https://forge.gridforum.org/projects/ogsa-authz) are addressing these and related issues extensively. This effort will utilize whatever methodology and protocols approved by that group to meet its authentication and authorization requirements.

GridCPR will require the following functionality from a Grid Authentication implementation:

- verification of process identity (note "JMauth" interactions in Figure 1)
- validation of service requests (note "CTauth" interactions in Figure 1)

Job Management

Checkpointing presupposes the execution of an application on a computational resource. Recovery presupposes the subsequent availability of another resource, possibly different nodes or the same nodes at a later time on the same resource.

Automated recovery relies upon the presence of a job management system of some kind, for example a job scheduler, for automated requeueing and resulting recovery of an abnormally terminated execution. Most grid-enabled compute resources have job managers of some kind (generally referred to as "schedulers"), both to facilitate the orderly flow of compute jobs through the resource and to optimize resource utilization. Since GridCPR is itself an optimization that facilitates each of these aspects, it is reasonable to assert the existence of a job management system of some type wherever GridCPR is deployed.

GridCPR will require the following functionality from a Grid Job Management implementation:

- queueing jobs for execution and re-execution (note "USERqueue" and "EHqueue" interactions in Figure 1)
- launching jobs on the compute resource (note "JMlaunch" interactions in Figure 1)
- execution of arbitrary commands (e.g. scripts) both prior to job execution and following job termination (e.g. PBS "prologue" and "epilogue" scripts)

2.2 GridCPR Services

A complete Grid Checkpoint and Recovery system will require the persistent availability of several services, specified below. While the implementations of these services may vary, uniform functionality is central to the success of the GridCPR system. Below we list a minimum set of services required to provide a meaningful system.

Figure 1 shows these components as they relate to one another in a work flow diagram. (See Section 3 for an extended work flow discussion). This drawing utilizes the following conventions:

- Circular objects are Grid services.
- Lines connecting components represent interactions.
- Interactions are labeled with compound names, assembled from:
  - a noun in ALLCAPS, briefly identifying the initiator, and
  - a verb in lower case, briefly identifying the action.
  - Some interactions initiate data flow in multiple directions. In these cases multiple labels are shown.
- Interactions involving significant amounts of data flow have arrows depicting the direction(s) of data flow.

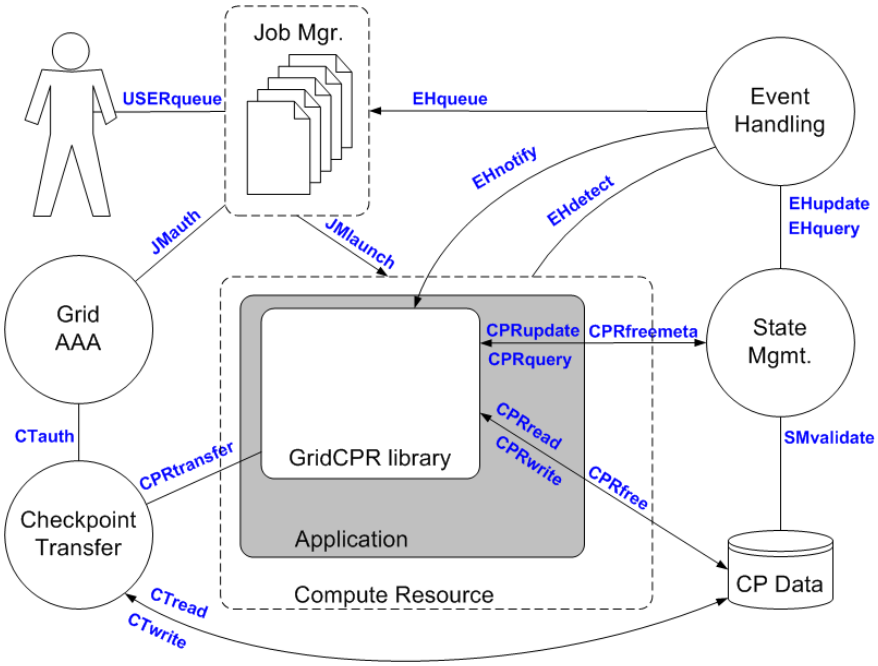

Figure 1: GridCPR Component and Workflow Diagram

2.2.1 Service Descriptions

Each of the required services is discussed below. The interactions between them, the applications and the hardware resources are summariezed summarized in Table 1.

State Management (SM) Service

The checkpoint State Management (SM) service is central to the coherent operation of the GridCPR system, analogous to the brain of the system preserving the metadata and knowledge of where its components reside. Normal running conditions will result in the transient creation, migration, and deletion of many checkpoints. For sustainable and scalable operation the SM service must track the status of all active and ill-terminated jobs that are GridCPR-enabled as well as metadata related to the role of each checkpoint. This information must be maintained, e.g., in some database management system. Metadata will be created and updated throughout a running job and can be released upon successful job completion. The checkpoint data can also be released upon job completion and perhaps purged during job execution to efficiently manage storage resources,

which may be limited. Thus the SM service should be supplemented by a service that shepherds checkpoint data storage resources. At recovery time, users or checkpoint transfer services require authoritative information regarding the number and location of required checkpoints. Without this information provided through the SM service, recovery services or applications would have no way of dynamically determining whether sufficient data are available for recovering from a given point in the checkpointed computation.

The SM service must provide the following functionality:

- storing checkpoint and job metadata (note CPRupdate and EHupdate interactions in Figure 1)
- resolving queries for checkpoint and job metadata (note CPRquery interactions in Figure 1)
  - This implies returning arbitrary subsets of information concerning checkpointed jobs or checkpoint data.
- validating incoming metadata and verifying the availability of pre-recorded checkpoint data (note SMvalidate interactions in Figure 1)

The SM service must be supplemented by the following functionality:

- probing for checkpoint data and metadata that is no longer needed in order to release these resources.
  - This can be done via the same application-level GridCPR library, eliminating the need for additional specialized interfaces (see CPRquery and CPRfree interactions in Figure 1)
  - This includes purging earlier checkpoints, keeping only the last N complete checkpoints, to limit resource consumption, which could be an integrated part of the GridCPR library.

Checkpoint Transfer (CT) Services

The Checkpoint Transfer (CT) service provides a reliable means for delivering checkpoint data from one location to another. For example, in the event of a system failure a checkpointed application must undergo a recovery process, possibly migrating all relevant checkpoint data from storage on a failed compute resource to another available grid compute resource. The CT service is essential for the following reasons: (i) checkpoint data generally may not be accessible by restarted processes on different nodes and/or machines; (ii) only the SM

associated with the original execution site will know where the valid checkpoint data resides; (iii) only the local job scheduler will know where the recovered job will execute, and (iv) only GridCPR is expected to be running in both places. While specific possible scenarios exist, in which users could retrieve checkpoints directly like recovering a job on the same compute resource in which checkpoint data reside on a global file system, these are not generally the case. Furthermore, migrating whole jobs from one grid resource to another will require grid-wide transport and authentication services.

The CT service will rely upon the following pre-existing grid services:

- Grid Authentication services to ensure secure and authorized transfer of data within the Grid (note CTauth interactions in Figure 1)
- GridFTP services for high-performance data migration within the grid.

The CT service must provide the following functionality:

- Transferring data from one storage medium to another given the two end-points
  - In this mode, the CT service can interact with GridCPR resources and the SM service via the GridCPR library, thus eliminating the need for additional specialized interfaces. (note the CPRread, CPRwrite, CPRupdate interactions in Figure 1)
  - The actual transfer itself may be built from a number of available services, from a straight UNIX cp to globus-url-copy, or others.

Event Handling (EH) Service

The Event Handling (EH) service is responsible for noticing when certain events occur, both within the compute platform and surrounding the job execution that require a certain response from the GridCPR system. It is analogous to the GridCPR nervous system. Since the goal of the GridCPR system is to facilitate the preservation and recovery of compute applications, the system must be able to determine which jobs have terminated abnormally (see definition above) and whether they merit recovery. Depending upon the outcome, certain steps like marking job-related metadata for recovery may be required.

Automated job recovery – which obviates any actions on the part of system operators – is an important step, of value both to the scientific user and the compute resource administrators. Automated recovery, however, is critically dependent upon the automated detection of system (e.g. hardware) failures and

subsequent re-allocation of compute resources for the recovered job. There exist a diverse number of ways for detecting hardware failure (e.g. heartbeat mechanisms), which will remain implicitly implementation-dependent, and therefore outside the scope of this document. But integration between the GridCPR system, an event handler and the job manager will provide the fullest experience of the GridCPR system to all parties involved.

Without the EH service, applications may be unable to determine the cause of job failures and may be forced to repeat calculations. This service is critical to distinguish between active jobs, successfully completed jobs and jobs that have failed abnormally, due to some specific failure of the system hardware or software. Each of these cases requires different handling by the SM service, as informed by the EH service.

The EH service must provide the following functionality:

- identification of hardware or system software failures within the compute resource (note EHdetect interactions in Figure 1)
  - This may be implemented in a variety of ways, but the implementation itself is outside the scope of this document.
- determination of whether detected events affect running or recently terminated jobs, based on available checkpoint metadata
- notification to the SM service marking the job as abnormally terminated (see definitions above note EHupdate interactions in Figure 1)
- notification to the application via the GridCPR library that a component has failed, if the application is still running (see EHnotify interactions in Figure 1)
- requeue failed jobs for recovery via integration with the job management (see EHqueue interactions in Figure 1)

2.3 GridCPR Library

The GridCPR Library is, by definition, the interface between the GridCPR system and all other agents and processes. It exclusively defines this entire interface and encapsulates the back-end implementations. It must present appropriate bindings to external processes that are to interact with GridCPR, for example: header files and libraries for C, C++ and Fortran applications; client binaries for interactive shell use; Java Interface definitions and class files for Java applications; etc. A clear and thorough definition of this interface is required to ensure the ability to impelement GridCPR uniformly on a diverse set of platforms.

The Use Cases document identified a common set of requirements for a GridCPR system that fulfills the charter of the GridCPR working group. A list of the API categories from that document and a corresponding list of interactions is identified in Table 2 .

Interaction	Description
USERqueue	Submit a job to the JM for eventual execution.
EHqueue	(Re)submit a job to the JM for recovery and restart.
EHnotify	Notify a running application of hardware failures or other conditions that would merit job recovery. This is advisory in nature, and does not force the application to abort.
EHdetect	Inform the GridCPR system of hardware or other failures that may merit job recovery. If any active jobs are affected by such a notification, the EH service would subsequently initiate an EHnotify interaction.
EHupdate	Update the recorded status of an active or recovering job, revising its metadata to reflect current conditions.
EHquery	Provide information to the EH subsystem, for example, to aid in the determination of what jobs may be affected by current events.
JMauth	Verify the requestor's identity (authentication) and permission (authorization) to perform the pending job management functions.
JMlaunch	Launch a queued application on the compute resources.
CPRupdate	Update the recorded metadata, e.g., concerning that status of stored data or of the running application.
CPRquery	Request information concerning the status of stored checkpoint data.
CPRfreemeta	Release expired metadata.
CPRread / CPRwrite	Read or write checkpoint data.
CPRfree	Release expired checkpoint data.
CPRtransfer	Request transfer of checkpoint data from one location to another.
CTauth	Validate the identity or permission of a requestor to migrate checkpoint data.
CTread / CTwrite	Read or write checkpoint data from within the CT service.
SMvalidate	Validate the status and/or availability of previously stored checkpoint data.

Table 1: GridCPR interaction definitions.

These APIs will be fully specified in a GridCPR API Specification follow-on document.

API category	**interactions in that category**
Application state writing and reading	CPRread, CPRwrite, CPRupdate
Checkpoint data management	CPRupdate, CPRfree, CPRfreemeta, SMvalidate
Failure/event notification	EHdetect, EHnotify, EHupdate, EHquery
Job management	USERqueue, EHqueue, JMauth, JMlaunch
Checkpoint data transport	CPRtransfer, CPRread, CPRwrite, CPRupdate
Authentication, authorization, and accounting	JMauth, CTauth

Table 2: GridCPR API categories (from the "Use Cases" document) and associated interactions, as shown in Figure 1

This library will be utilized not only by user applications but by administrative tools directed at the effective operation and administration of the GridCPR system. For example, the Checkpoint Transfer (CT) service will necessarily read and write checkpoint data and update the State Management (SM) service noting the locations of redundant copies of checkpoint data. But it will use the same read/write/update interface used by the scientific applications an effective code-reuse strategy.

2.4 GridCPR Resources

The GridCPR system will require a significant amount of resources for successful operation, primarily storage resources. GridCPR must be able to store multiple complete snapshots of large amounts of memory and other transient information for all active jobs on a grid computing resource. These checkpoints and metadata, while traditionally viewed as files, may be stored in any medium or format. The GridCPR system must have access to such storage through the GridCPR API from each compute node.

Checkpoint storage may be in the form of large parallel or distributed file systems, collection management systems, databases or other storage venues. And, as described above, checkpoints must be managed, i.e. automatically released after successful job completion. The SM service itself will internally require some amount of storage in the form of some type of database management system.

Given the nature of GridCPR, requisite resources will extend beyond basic storage to include adequate network resources to exchange complete checkpoints and job-related information between grid-connected compute resources.

3 GridCPR Workflow Scenarios

For thoroughness we present below a sample list of common workflow operations where each transaction (as defined above) is followed by the reasoning behind this operation. By so doing we demonstrate how higher-level operations are constructed from a series of well-defined transactions.

1. Job Launch
    - **USERqueue** - All original job requests originate with a user and/or his proxy.
    - **JMauth** - All jobs submitted to Grid resources are authenticated by Grid Authentication.
    - **JMlaunch** - The scheduler is the arbiter of job execution.
2. Writing a checkpoint
    - **CPRwrite** - The application will write a checkpoint when appropriate.
    - **CPRupdate** - After the checkpoint is complete the GridCPR library must notify the SM that a checkpoint exists. (optional: The following steps represent a redundant storage option.)
    - **CPRtransfer** - The GridCPR library may use this means to request a duplicate copy of checkpoint data, if redundant storage is desired.
    - **CPRread** - The CT will fetch the original checkpoint from the location of its initial storage.
    - **CPRwrite** - The CT will write a duplicate copy of the checkpoint on other reliable media.
    - **CPRupdate** - The CT record the availability of replicated data with the SM service.
3. Failure Detection
    - **EHdetect** - The EH service must monitor the system to detect mission-critical failures that will affect running jobs.
    - **EHupdate** - The EH service must notify the SM of an abnormal system condition so that a) the job can be flagged for later recovery and b) the SM will not free checkpoint data potentially necessary for future recoveries.
    - **EHnotify** - The EH service must notify the running application of a failure condition, so that it may terminate, avoiding deadlocks, and release valuable resources.
    - **EHqueue** - The EH may automatically resubmit failed jobs to the Scheduler for automated recovery and restart.

4. Checkpoint Recovery
    - **JMlaunch** - The scheduler re-launches a failed job.
    - **CPRquery** - The application must consult the SM service to ascertain the availability of checkpoint data for job recovery, if any exists.
    - **CPRtransfer** - The application, via the GridCPR library, requests retrieval of a coherent checkpoint.
        - **CPRread** - The CT service retrieves the checkpoint data from its current location and transports it to the requested destination.
        - **CPRwrite** - The CT service writes the checkpoint data to a location accessible to the recovered application.
        - If the CT service fails to retrieve, transfer or write any checkpoint data then the recovery agent will have to repeat this cycle, beginning again with a CPRquery to determine whether there is any other available checkpoint data. This process can iterate until all recorded/available checkpoints are exhausted.
    - **CPRread** - If checkpoint retrieval by the CT was successful, the application will read the checkpoint data via the GridCPR library from the checkpoint storage resource.
5. Normal Termination / Cleanup
    - **CPRupdate** - A running application notifies the SM that it has completed successfully, implying that the checkpoint recovery data is no longer needed.
    - **CPRfree** - The SM service releases all resources held by checkpoint data
    - **CPRfreemeta** - The SM service releases resources held by checkpoint metadata.
        - Some checkpoint- or job- related metadata may be kept for tracking purposes.

4 Security Considerations

The ability to resume computational jobs from checkpoint files has obvious security implications. Care must be taken to protect checkpoint data on behalf of the user. Secure storage (e.g. POSIX file system security) and data transport (e.g. GridFTP) are required to keep GridCPR users from losing data. By relying on independent services like GridFTP this system relies upon existing infrastructural tools that are already written in such a way as to prevent users from inadvertently getting elevated user identity/permissions.

5 Intellectual Property Issues

5.1 Editor information

This document is the result of the joint efforts of many contributors. The authors listed here and on the title page are those taking responsibility for the content of the document, and all errors. The editors (underlined) are committed to taking permanent stewardship for this document and can be contacted in the future for inquiries.

Thilo Kielmann
kielmann@cs.vu.nl
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

Andre Merzky
andre@merzky.net
Center for Computation and Technology
Louisiana State University
216 Johnston Hall
70803 Baton Rouge
Louisiana, USA

Derek Simmel
dsimmel@psc.edu
Pittsburgh Supercomputing Center
300 South Craig St.
Pittsburgh, PA 15213
United States
Phone: +1 (412) 268-1035

Nathan Stone
stone@psc.edu
Pittsburgh Supercomputing Center
300 South Craig St.
Pittsburgh, PA 15213
United States
Phone: +1 (412) 268-4960

5.2 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.

5.3 Disclaimer

5.4 Full Copyright Notice

References

[1] R. Badia, R. Hood, T. Kielmann, A. Merzky, C. Morin, S. Pickles, M. Sgaravatto, P. Stodghill, N. Stone, and H. Y. Yeom. Use Cases for Grid Checkpoint and Recovery. Grid Forum Document GFD.92, May 2007. Open Grid Forum.

[2] Grid Checkpoint and Recovery Working Group. Open Grid Forum. `http://forge.ogf.org/projects/gridcpr-wg/`.

A Glossary

We present here definitions for a number of terms to ensure clarity and standardized usage for the purposes of all documents presented by the OGF GridCPR Working Group. These definitions are as follows:

- **Abnormal Termination**

  All computational applications terminate eventually. However termination is judged to be "abnormal" if it is not a result of the implementation of the application itself or its input data. Typical scenarios involve failed file systems, networks or computational hardware.

- **Checkpoint**

  A checkpoint is a persistent snapshot of the running state of an application. It exists in some storage medium, whether that is in one or more files, database records or other storage media. It has associated metadata without which it is useless.

- **Checkpoint Metadata**

  Metadata are additional data that provide a richer context for the primary data. Checkpoint metadata identify, for example, the origin, sequence, ownership or history of the primary checkpoint data. For effective interoperability these metadata must follow a prescribed structure or schema. For job portability the metadata must be passed with the checkpoint data, to ensure proper location, distribution, utilization, etc.

- **Service**

  Services are persistently available agents (e.g. web services or daemons) that perform pre-defined actions according to their specification on a transaction/request basis. The GridCPR system relies upon a number of services identified within the Open Grid Forum (OGF) for its effective operation.

- **Interoperability**

  For purposes of this document, interoperability is defined as the ability to take an application checkpointed on one grid-connected resource and restart it on another. This includes both checkpoint data portability as well as checkpoint job portability. The latter explicitly requires the portability of the checkpoint metadata in order to effectively utilize the checkpoint data. This definition does not, however, imply the ability to interchange individual GridCPR services from independent implementations. This interpretation is consistent with the GridCPR charter[2].