SAGA Extension: Checkpoint and Recovery API (CPR)

Status of This Document

This document provides information to the grid community, proposing a standard for an extension to the Simple API for Grid Applications (SAGA). As such it depends upon the SAGA Core API Specification [2], on the GridCPR Use Case document [1] and the GridCPR architecture document [3]. This document is supposed to be used as input to the definition of language specific bindings for this API extension, and as reference for implementors of these language bindings. Distribution of this document is unlimited.

Copyright Notice

Copyright © Open Grid Forum (2007). All Rights Reserved.

Abstract

FIXME: real citations!

This document specifies the an Checkpoint and Recovery (CPR) API extension to the Simple API for Grid Applications (SAGA), a high level, application-oriented API for grid application development. This CPR API is motivated by a number of use cases collected by the GridCPR Working Group in GFD.92 ("Use Cases for Grid Checkpoint and Recovery"). Scope and semantics of the SAGA CPR API extension is motivated by the GridCPR architecture document GFD.93 ("An Architecture for Grid Checkpoint and Recovery (GridCPR) Services and a GridCPR Application Programming Interface").

Contents

1 Introduction 2

1.1 Notational Conventions . . . . . . . . . . . . . . . . . . . . . . 2
1 Introduction

This document specifies an API for the initiation and management of application checkpointing and recovery operations.

1.1 Notational Conventions

In structure, notation and conventions, this document follows those of the SAGA Core API specification \[2\], unless noted otherwise.

1.2 Security Considerations

As the SAGA API is to be implemented on different types of Grid (and non-Grid) middleware, it does not specify a single security model, but rather provides hooks to interface to various security models. In that respect, the SAGA CPR extension covered in this document does not differ from the SAGA Core API specification \[2\], and the Security Considerations from that document apply.
2 SAGA CPR API

2.1 Introduction

This document specifies an API for the initiation and management of application checkpointing and recovery operations. The scope and semantics of this API are motivated by the GridCPR architecture document [3]. Its capabilities fall in the following categories:

A  – checkpoint and recovery operations
   A.1 – specification of application checkpointing capabilities and policies
   A.2 – issuing notification of checkpointing requests
   A.3 – receiving notification of checkpointing requests
   A.4 – issuing notification of recovery requests
   A.5 – receiving notification of recovery requests

B  – management of checkpoints
   B.1 – description of checkpoints and checkpoint meta data
   B.2 – location and movement of checkpoints
   B.3 – security, consistency and lifetime management of checkpoints

The capabilities referenced under A are, at least partly, already included in the SAGA Core Job API, so it seems sensible to define the remaining capabilities in A also as part of the SAGA Core Job API. This document does that by specifying an additional interface (checkpointable) which can optionally be implemented by the saga::job class.

The capabilities listed under B are closely related to the management of files and logical files, which, in the SAGA Core API, share the abstraction of an hierarchical name_space. It seems sensible to define the CPR checkpoint management capabilities in the same framework. This document does that by defining a checkpoint namespace, with the classes cpr_dir and cpr_entry.

2.1.1 Checkpoint URLs

The checkpoint URLs are those URLs which identify cpr_entry and cpr_dir instances (and thus not the URLs pointing to the physical locations of the individual checkpoint files). As this document expects the underlying middleware to adhere to the CPR Architecture described in [?], we recommend the usage
of the scheme gridcpr:// – but that is really up to the implementation, as the required semantics can very likely also be provided by systems which do not follow [?].

2.2 Specification

```java
package saga.cpr
{
    class cpr_job_description : implements saga::job_description
    // from job_description saga::attributes
    // from job_description saga::object
    // from object saga::error_handler
    {
        // Attributes:
        //
        // name: CPRPolicy
        // desc: checkpoint policy
        // type: Enum
        // mode: ReadWrite
        // value: ''
        // notes: - the attribute can have the values:
        //          - External: checkpoints are triggered by an
        //            external application
        //          - Internal: checkpoints are triggered by the
        //            job internally.
        //          - an application with 'Timed' CPR policy can
        //            still create internally and externally
        //            triggered checkpoints.
        //
        // name: CPRFrequency
        // desc: checkpoint frequency for 'Timed' CPR policy
        // type: Int
        // mode: ReadWrite
        // value: '86400'
        // notes: - specifies the number of seconds between two
        //         consecutive timed checkpoints.
        //          - Defaults to one checkpoint per day.
        //          - The value is ignored if CPR policy is not
        //            set to 'Timed'
        //
        // name: CPRSequence
        // desc: sequence of checkpoint types
        // type: String
```
// mode: ReadWrite
// value: '"
// notes: - the attribute is a sequence of the letters
// - 'F': Full checkpoint
// - 'I': Incremental checkpoint
// - 'i': Incremental checkpoint
// (diff to last checkpoint)
// - the sequence is repeated infinitely
// - Incremental checkpoints are always relative
to some preceding checkpoint. That implies
that the first checkpoint is *always* a
full checkpoint.
// - Examples:
// - "F" : always do full checkpoints
// - "FIFI": alternate full and incremental
// Checkpoints
// - "i" : always do incremental checkpoint,
// using the previous (incremental)
// CP as base. First CP will be
// full.
// - This attribute is informational, to optimize
the checkpoint management. The application
and backend need to ensure that this
sequence is actually applied. To
simplify that, the SAGA CPR implementation
SHOULD put the attributes value into the
application's environment, as
'SAGA_CPR_SEQUENCE'.
// - If application and backend do not actually
apply this sequence, it MUST NOT imply
invalid checkpoints.
// - SAGA CPR implementation MAY be able to
enforce this sequence.

// name: CPRTimeToLive
// desc: lifetime for checkpoint files
// type: Int
// mode: ReadWrite
// value: '2500000'
// notes: - specifies the number of seconds
// checkpoints are guaranteed to be valid
// - Defaults 2,500,000 seconds (ca 29 days)
// - the value can be changed for each individual
// checkpoint - see the respective cpr_entry
// attribute with the same name.
- the SAGA CPR implementation SHOULD make sure that no Full checkpoints are deleted for which derived Incremental checkpoints still exist.
- for application internal checkpoints, the application itself is responsible to enforce that checkpoint location. To simplify that, the SAGA CPR implementation SHOULD put the attributes value into the application’s environment, as ‘SAGA_CPR_TIME_TO_LIVE’.

name: CPRHistoryLength
desc: number of checkpoints to keep
type: Int
mode: ReadWrite
value: '-1'
notes: - specifies the number of previous generations of checkpoints to be kept in the system. If that number is exceeded, the backend MAY delete older checkpoints.
- Negative values specify an unlimited number of generations to be kept.
- the SAGA CPR implementation MUST make sure that no Full checkpoints are deleted for which derived Incremental checkpoints still exist.
- Defaults to -1.

name: CPRBaseLocation
desc: cpr_directory to be used for storing checkpoints
type: URL
mode: ReadWrite
value: 'any:///#UserID#/JobID#/'
notes: - specifies the cpr_directory to be used when registering the checkpoint files.
- if the directory does not exist, it is created, as are its parents.
- the '#UserID#' wildcard can be used to specify the value of the UserID attribute
- the '#JobID#' wildcard can be used to specify the value of the job’s jobid.
- for application internal checkpoints, the application itself is responsible to enforce that checkpoint location. To
// simplify that, the SAGA CPR implementation
// SHOULD put the attributes value into the
// application's environment, as
// 'SAGA_CPR_BASE_LOCATION'.

// name: CPRBaseName
// desc: cpr_directory to be used for storing
// checkpoints
// type: URL
// mode: ReadWrite
// value: '#JobID#.Generation#.cpr
// notes: - specifies the cpr_entry name to be used
// when registering the checkpoint files.
// - if the entry exists when the checkpoint is
to be created, its content is overwritten!
// - The following wildcards are available:
// - '#JobID#': as for CPRBaseLocation
// - '#UserID#': as for CPRBaseLocation
// - '#Generation#': number of snapshot.
// - Generation numbering starts at 0, and MAY be
// padded with zeros to a fixed length.


class cpr_job_service : implements saga::job_service
    // from job_service saga::object
    // from job_service saga::async
    // from object saga::error_handler
{
    create_job
        (in job_description jd_start,
         in job_description jd_rec,
         out job job);
}

class cpr_job : extends saga::job,
    implements saga::steerable
    // from job saga::task
    // from job saga::async
    // from job saga::attribute
    // from task saga::object
    // from task saga::monitorable
    // from object saga::error_handler
{
    list_checkpoints (out array<string> urls);

    // cpr actions
checkpoint (in string url = "");
recover (in string url = "");

// implies run() if New

// manage locality of checkpoints

cpr_stage_out (in string url = ",
    in int id = -1);
cpr_stage_in (in string url = ",
    in int id = -1);

get_last_cpr (out string url);

// Metrics:

// name: Checkpoint
// desc: to be fired when an application level checkpoint is requested
// mode: ReadWrite
// unit: 1
// type: String
// value: ''
// notes: - the metric acts as trigger
// - the value can optionally be set to an cpr_entry URL to be used for the resulting checkpoint

// name: Checkpointed
// desc: to be fired when application level checkpoint is finished
// mode: ReadWrite
// unit: 1
// type: Trigger
// value: ''

// name: Recover
// desc: to be fired when application level recovery is requested
// mode: ReadWrite
// unit: 1
// type: String
// value: ''
// notes: - the metric acts as trigger
// - the value can optionally be set to an cpr_entry URL to be used for the recovery

// name: Recovered
// desc: to be fired when application level recovery is finished
// mode: ReadWrite
// unit: 1
// type: Trigger
// value: 
}

class directory : extents saga::ns_directory
  implements saga::attribute
  // from ns::directory saga::ns_entry
  // from ns_entry saga::object
  // from ns_entry saga::async
  // from object saga::error_handler
{
  enum flags
  {
    None = 0, // same as in name_space::flags
    Overwrite = 1, // same as in name_space::flags
    Recursive = 2, // same as in name_space::flags
    Dereference = 4, // same as in name_space::flags
    Create = 8, // same as in name_space::flags
    Excl = 16, // same as in name_space::flags
    Lock = 32, // same as in name_space::flags
    CreateParents = 64, // same as in name_space::flags
    Truncate = 128,
    Append = 256,
    Read = 512,
    Write = 1024,
    ReadWrite = 2048,
    Binary = 4096
  }

  // open flags default to CreateParents and Lock
  // for open on checkpoint files.

  // find checkpoints based on name and meta data
  find (in string name_pattern,
        in array<string> meta_pattern = (),
        in int flags = None,
        in string spec = "",
        out array<string> urls);

  set_parent (in saga_url checkpoint,
              in string url,
get_parent (in saga_url checkpoint, in int generations = 1, out string url);

create_child (in saga_url checkpoint, in int flags = None, out cpr_entry child);

add_file (in saga_url checkpoint, in saga::url file, out int id);

remove_file (in saga_url checkpoint, in int id);

update_file (in saga_url checkpoint, in int id, in saga::url file);

get_file (in int id, out saga::url url);

list_files (in saga_url checkpoint, out array<saga::url> files);

get_file_num (out int nfiles);

stage (in saga_url checkpoint, in saga::url target, in int id = -1);

class checkpoint : extends saga::ns_entry implements saga::attribute
    // from ns_entry saga::object
    // from ns_entry saga::async
    // from object saga::error_handler
{
    // get parent checkpoint url
    set_parent (in string url, in int generations = 1);

    get_parent (in int generations = 1, saga-rg@ogf.org
10
out string url);
create_child (in int flags = None,
out cpr_entry child);
add_file (in saga::url file
out int id);
remove_file (in int id,
out saga::url file);
update_file (in int id,
in saga::url file_new);
get_file (in int id,
out saga::url url);
list_files (out array<saga::url> files);
get_file_num (out int nfiles);
stage (in saga::url target,
in int id = -1);

// Attributes:
// time
// nfiles
// ttl
// mode (full, inc 1, inc 2)
// parent (url for cpr-entry)
// childs (array of cpr-entry urls)

2.3 Specification Details

2.3.1 The checkpointable Interface

As described above, the CPR job extends the SAGA Core Job API. In particular, SAGA jobs will implement the checkpointable interface defined here. Otherwise, the job class actually stays the same as defined in the original job package in the SAGA Core API. The changes to the SAGA job service are similar small: only the create_job method gets overloaded with a version which
accepts an additional job description to be used on job restart (i.e. on recovery). The job description, however, has a number of additional attributes to define the job's behaviour in the scope of CPR: the default checkpoint policy, the default checkpoint life time, the application specific checkpoint trigger mechanism, etc. FIXME: add all those.

The checkpointable interface (implemented by saga::cpr_job) offers, compared to the normal saga::job, several additional methods, such as (checkpoint() and recover()), and also offers a number of new metrics (Checkpoint, Checkpointed, Recover and Recovered). The cpr_job's state model does not change if compared to the saga::job state model. Various backends MAY, however, report 'Checkpointing' or 'Recovering' or similar as state detail to the 'Running' state, whenever the job is performing either of the two actions. We always assume that the cpr job continues to utilize compute, network and storage resources while performing CPR operations.

2.3.2 The Checkpoint Name Space – cpr_dir and cpr_entry

The SAGA CPR API defines a checkpoint (cpr_entry) to be a representation of complete application state at a specific point in time. An application (saga::job) can consist of multiple processes, and each process may write any number (0..n) of checkpoint files; a single cpr_entry can thus represent any number of checkpoint files. The individual files are referred to by an integer number (index), and applications can open them separately for reading and/or writing.

Checkpoints are organized in a SAGA namespace (i.e. saga::cpr_entry and saga::cpr_dir inherit saga::ns_entry and saga::ns_dir). An additional relationship between cpr_entries is established by their order in time: a checkpoint taken directly before another checkpoint is named parent, a checkpoint taken directly after another checkpoint is named child. The CPR middleware SHOULD be able to identify parent/child relationships automatically – this can, however, be enforced and also changed on API level, by using the set_parent() / remove_parent() and set_child() / remove_child() methods. Also, a parent may have more than one child, but a child may have at most one parent. This allows effectively for a tree of checkpoints, which allow applications to rewind to older checkpoints, or to restart with a checkpoint from a different application configuration.

The exact physical location of checkpoint files is, in general, not under application control - it is, however, possible to ensure co-location of the job execution host and checkpoint files (cpr_stage_in(), by default fetching the last checkpoint available), It is also possible to enforce the opposite, and to stage out a checkpoint file to ensure its continued availability on node shutdown etc. (cpr_stage_out(), also by default referring to the last checkpoint available).
3 Intellectual Property Issues

3.1 Contributors

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

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

The initial version of the presented SAGA API was drafted by members of the SAGA Research Group. Members of that group did not necessarily contribute text to the document, but did contribute to its current state. Additional to the authors listed above, we acknowledge the contribution of the following people, in alphabetical order:

Thilo Kielmann (VU), Shantenu Jha (LSU), Derek Simmel (PSC), Nathan Stone (PSC).

3.2 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.
3.3 Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

3.4 Full Copyright Notice

Copyright (C) Open Grid Forum (2006). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.
References

