GWD-R.73
SAGA-RG

Version: 0.1

Andre Merzky
Vrije Universiteit, Amsterdam

January 28, 2007

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 [1], on the GridCPR Use Case document [?]and the GridCPR architecture document [?]. 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

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.xx ("Use Cases for Grid Checkpoint and Recovery"). Scope and semantics of the SAGA CPR API extension is motivated by the GridCPR architecture document GFD.yy ("An Architecture for Grid Checkpoint and Recovery (GridCPR) Services and a GridCPR Application Programming Interface").

Contents

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 documents follows those of the SAGA Core API specification [1], 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 – see the documentation of the `saga::context` class in Section **??** for details.

A SAGA implementation is considered secure if and only if it fully supports (i.e., implements) the security models of the middleware layers it builds upon, and neither provides any (intentional or unintentional) means to by-pass these security models, nor weakens these security models' policies in any way.

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 [?]. Its capabilities fall in the following categories:

A – checkpoint and recovery operations

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

B – management of checkpoints

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

The capabilites referenced under **A** are, at least partly, already included in the SAGA Core Job API, so it seems sensible to define the remaining capabilies in **A** also as part of the SAGA Core Job API. This document does that by specifying an additional interface (checkpointable) which is to 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 hierachical `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 The `checkpointable` Interface

The SAGA CPR API defines a checkpoint (`cpr_entry`) to be a represent a complate snapshot of a state of an application. An application (`saga::job`) can consist of multiple proceses, and each process may write any number ($0...n$) of checkpoint files; checkpoints thus represent a number of individual checkpoint files. The files the checkpoint is comprised of are not managed by the application, but by the middleware. The files are refered to by a integer number **FIXME: string?**, and the application can open the individual files 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 stablished 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 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 only zero or one parent. This allows effectively for a tree of checkpoints, which allow applications to rewind to older checkpoints, or to checkpoints with a different

A checkpointable job (`saga::cpr_job`) offers, compared to a normal `saga::job`, some additional methods (`checkpoint()` and `recover()`) and metrics (`Checkpoint`, `Checkpointed`, `Recover` and `Recovered`) for checkpoint and recovery operations.

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 taken), It is also possible to enforce the opposite, and to stage out a checkpoint file to ensure its availability on node shutdown etc. (`cpr_stage_out()`, also by default refering to the last checkpoint taken).

2.1.2 The Checkpoint Name Space – `cpr_dir` and `cpr_entry`

FIXME: high level description

2.1.3 Checkpoint URLs

FIXME: same conventions as in core, but recommend `gridrpc` as scheme.

2.2 Specification

```
package saga.cpr
{
  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 = "");
    cpr_stage_in      (in  string        url = "");

    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 cpr_dir : 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 Binary, Truncate,
      // CreateParents and Lock for open on cpr_entry.

      // 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 );
    }


    class cpr_entry : extends        saga::ns_entry
                      implements     saga::attribute
                   // from ns_entry  saga::object
                   // from ns_entry  saga::async
                   // from object    saga::error_handler
    {
      get_parent (in  int    generations = 1,
                  out string url);  // cpr-entry url
      get_file   (in  int    num = 0,
                  out string url);  // file/lfile url
      open_file  (in  int    num = 0,
                  out file   file); // saga::file

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

2.3 Specification Details

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 thst 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:

Andrei Hutanu (LSU), Hartmut Kaiser (LSU), Pascal Kleijer (NEC), Thilo Kielmann (VU), Shantenu Jha (LSU).

3.2 Intellectual Property Statement

3.3 Disclaimer

3.4 Full Copyright Notice

References

[1] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.xx, 2007. Global Grid Forum.