GWD-R.96
.
SAGA-WG
Version: 1.0 RC.2

Andre Merzky[1]1
Andre Luckow
Derek Simmel
December 2, 2009

Editor

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.

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

1.1 Notational Conventions 2
1.2 Security Considerations 2

2 SAGA CPR API 3

2.1 Introduction 3
2.2 Specification 4
2.3 Specification Details 13

3 Intellectual Property Issues 14

3.1 Contributors 14
3.2 Intellectual Property Statement 14
3.3 Disclaimer 15
3.4 Full Copyright Notice 15

References 15

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 [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 trhe 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 requstes
- **A.5** – receiving notification of recovery requstes

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 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 can optinally 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 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

```
package saga.cpr
{
  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, // same as in       file::flags
    Append          =  256, // same as in       file::flags
    Read            =  512, // same as in       file::flags
    Write           = 1024, // same as in       file::flags
    ReadWrite       = 2048, // same as in       file::flags
    Binary          = 4096  // same as in       file::flags
  }

  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
//              (diff to last Full 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"   : allways 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  url        url = "",
                     in  int        id  = -1);
  recover           (in  url        url = "",
                     in  int        id  = -1);
                    // implies run() if New


  // manage locality of checkpoints
  cpr_stage_out     (in  url        url = "",
                     in  int        id  = -1);
  cpr_stage_in      (in  url        url = "",
                     in  int        id  = -1);

  cpr_last          (out url        url);
  cpr_list          (out array<url> 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 self : extends       saga::cpr::job
             implements    saga::steerable
          // from cpr::job saga::job::job
          // from job::job saga::async
          // from job::job saga::attributes
          // from job::job saga::task
          // from job::job saga::object
          // from job::job saga::monitorable
          // from job::job saga::permissions
          // from job::job saga::error_handler
{
  // no CONSTRUCTOR
  DESTRUCTOR           (in  job_self        obj);
}

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

    // additional inspection method
    is_checkpoint (in  url        checkpoint,
                   out bool             test);

    // 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  url              checkpoint,
                  in  string           url,
                   in  int             generations = 1);

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

    get_file_num (in  url              checkpoint,
                  out int              nfiles);

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

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

    get_file     (in  url              checkpoint,
                  in  int              id,
                  out url              url);

    open_file    (in  url              checkpoint,
                  in  int              id = 0,
                  in  int              flags = CreateParents | Lock | ReadWrite,
                  out saga::file       file);
```

```
  open_file    (in  url             checkpoint,
                in  url             url,
                in  int             flags = CreateParents | Lock | ReadWrite,
                out saga::file      file);

  remove_file  (in  url             checkpoint,
                in  int             id);

  remove_file   (in  url            checkpoint,
                 in  url            url);

  update_file  (in  url             checkpoint,
                in  int             id,
                in  url             new);

  update_file  (in  url             checkpoint,
                in  url             old,
                in  url             new);

  stage        (in  url             checkpoint,
                in  int             id,
                in  url             target);

  stage        (in  url             checkpoint,
                in  url             file,
                in  url             target);

  stage        (in  url             checkpoint,
                in  url             target);
}


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  url             parent,
                in  int             generations = 1);

  get_parent   (in  int             generations = 1,
                out string          url);
```

```
    get_file_num (out int              nfiles);

    list_files   (out array<url>       files);

    add_file     (in  url              file
                  out int              id);

    get_file     (in  int              id,
                  out url              url);

    open_file    (in  int              id = 0,
                  in  int              flags = CreateParents | Lock | ReadWrite,
                  out saga::file       file);

    open_file    (in  url              url,
                  in  int              flags = CreateParents | Lock | ReadWrite,
                  out saga::file       file);

    remove_file  (in  int              id);

    remove_file  (in  url              url);

    update_file  (in  int              id,
                  in  url              file_new);

    update_file  (in  url              url,
                  in  url              file_new);

    stage        (in  int              id,
                  in  url              target)

    stage        (in  url              url,
                  in  url              target);

    stage        (in  url              target);

    // 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

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

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

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 refering to the last checkpoint available).

3 Intellectual Property Issues

3.1 Contributors

This document is the result of the joint efforts of many contributors, and in particular implementors. 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.

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

The initial version of the presented SAGA API was drafted by members of the SAGA Research Group. Members of this 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:

Shantenu Jha (LSU), Thilo Kielmann (VU), Derek Simmel (PSC), and 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 tech-

nology 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

References

[1] R. Badia, R. Hood, T. Kielmann, A. Merzky, C. Morin, S. Pickles, M. Sgaravatto, P. Stodghill, N. Stone, and H. Yeom. GFD.92 – Use-Cases and Requirements for Grid Checkpoint and Recovery. OGF Informational Document, Open Grid Forum, 2006.

[2] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. GFD.90 – A Simple API for Grid Applications (SAGA). OGF Proposed Recommendation, Open Grid Forum, 2007. Global Grid Forum.

[3] N. Stone, D. Simmel, T. Kielmann, and A. Merzky. GFD.93 – An Architecture for Grid Checkpoint and Recovery Services. OGF Informational Document, Open Grid Forum, 2007.