ExTENCI: SAGA Condor Integration

The aim of this document is to describe the use-cases and requirements for an integration of SAGA and Condor. Based on these requirements, an implementation concept for a Condor plug-in for SAGA (a so-called *Adatpor*) is derived and described here in detail. With this proposed *Condor Adaptor*, it will be possible to use Condor, Condor-G as well as Condor glide-in programmatically through the standardized SAGA C++ and Python interfaces. This will allow distributed applications - including those based on the Cactus Computational Framework - to utilize distributed infrastructure based on the Globus Toolkit, Condor and others through a unified interface.

This project is part of the the Extending Science Through Enhanced National Cyberinfrastructure (ExTENCI) project, which is a joint Open Science Grid (OSG) and TeraGrid project, funded by the National Science Foundation Office of Cyberinfrastructure (NSF OCI).

Status of This Document This document is still work in progress.

Contents

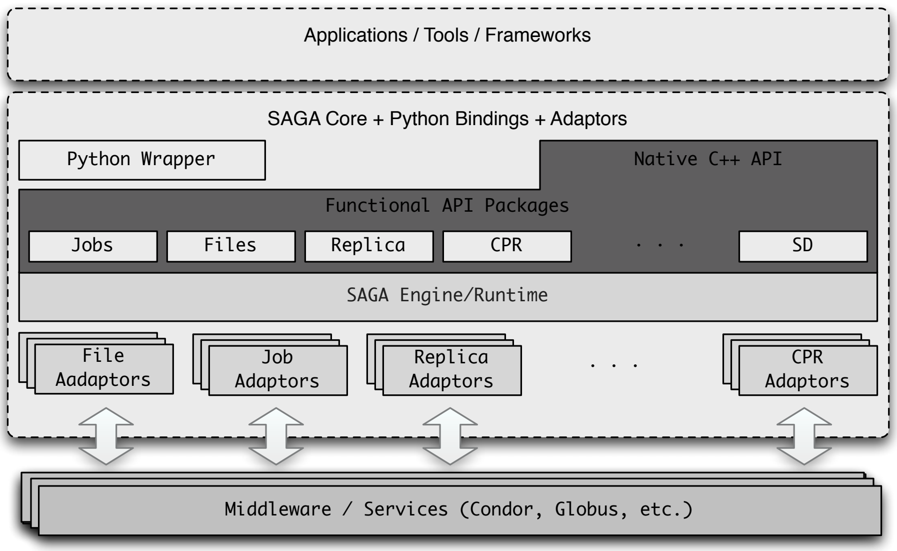

Figure 1: Layered schematic of the different components of SAGA. Middleware specific adaptors make applications developed using SAGA portable. Schematic showing the different ways in which SAGA can be used to develop distributed applications. (i) Using native SAGA calls to implement distributed functionality; (ii) Through the use of frameworks which provide either application-level usage modes, patterns and thus shielding the application from directly interfacing with the infrastructure.

1 SAGA

1.1 API Specification

A set of application groups expressed the desire for a simple programmatic interface that would be widely-adopted and widely-available, which led to the need and role for the SAGA API. The goal of such an interface is to provide a "distributed computing counterpart to MPI" (in impact if not in details), and to supply developers with a simple, uniform, and standard programmatic interface with which to develop applications. Thanks to the efforts of many contributors, an initial specification of such an interface was released in 2008: the Simple API for Grid Applications (OGF-GFD-90). The scope and requirements of the SAGA API have been formally defined by OGF's SAGA Research Group (SAGA-RG). The SAGA-RG collected use cases from a broad user community and published them as OGF-GFD.70. The requirements and design for the SAGA API were directly derived from these use cases; this process has been documented and published in OGF-GFD.71. The end result of this process is the 1.0 version of the SAGA core specification (OGF-GFD.90), which defines language-independent syntax and semantics for the SAGA core API (error handling, session and context management, permissions, monitoring, attribute and task model) and functional packages (jobs, name-spaces, files, replicas, streams, RPC). Several API extension, such as CPR (OGF-GWD-R.96), Adverts (GFD-R-P.XX), and Messaging (GWD-R.94) are currently under development and follow a similar, well-defined community-driven standardization and approval process. SAGA is now an Open Grid Forum (OGF) proposed recommendation on the path to becoming a standard. The standardization is important because it makes it more likely that infrastructures will support SAGA, which may make it widely available for users of most national CI projects.

1.2 C++/Python Implementation

The SAGA implementation referred to in this document is a reference implementation of the SAGA API specification written in C++. It is under active development by our group at the Center for Computation and Technology at Louisiana State University with many contributions from external groups and collaborators and strives to implement the complete set of functional API packages as defined in OGF-GFD-90 while focusing on maximum practical relevance by providing middleware-bindings (Adaptors) for as many distributed middleware services as possible.

SAGA is open source software, released under the Boost Software License. As shown in Figure 1, SAGA can be divided into three major components : (1) the *Core Components* which provide the API functionality, (2) the *Adaptors* which translate API calls into native middleware calls, and the (3) the Python API *Language Bindings* - a thin Python layer on top of the native C++ API. The SAGA *Core Components* are a collection of dynamic libraries and header files that represent the functional API packages (see section 2.1) along with a lightweight, highly-configurable *Runtime* that manages call dispatching and the dynamic runtime loading of the *Middleware Adaptors*.

More information about SAGA can be found on the website:
http://saga.cct.lsu.edu

1.3 SAGA Middleware Adaptors

The following list gives an overview of the currently available SAGA middleware adaptors:

Adaptor Suite	API Package	Middleware
Default	Job	Fork
	File	Local FS
	Streams	TCP
SQL	Replica	PostgreSQL, SQLite
	Advert	PostgreSQL, SQLite
Globus	Job	GRAM2/GRAM5
	File	GridFTP
	Replica	RLS
EC2	Job	EC2-compatible (Amazon EC2, Eucalyptus, Nimbus
GridSAM	Job	OMII GridSAM
LSF	Job	Platform LSF
gLite	Job	gLite CREAM WS
HDFS	File	Hadoop FS
PBS	Job	PBS Pro, Torque
DRMAA	Job	DRMAA-compatible (SGE, PBS/Torque, GridWay, FedStage
OGSA BES	Job	BES-compatible (ARC, Unicore, Genesis-II

1.4 SAGA-based Applications, Tools and Frameworks

In the absence of a formal theoretical taxonomy of distributed applications, there are three types of distributed applications: (i) Applications where local functionality is swapped for distributed functionality, or where distributed execution modes are provided. A simple but illustrative example is an application that uses distributed resources for bulk submission. Here, the application remains unchanged and even unaware of its distributed execution, and the staging, coordination, and management are done by external tools or agents. Most applications in this category are classified as implicitly distributed. (ii) Applications that are naturally decomposable or have multiple components are then aggregated or coordinated by some unifying or explicit mechanism. DAG-based workflows are probably the most common example of applications in this category. Also, in this category, are applications that remain unchanged but are enabled to utilize tools and services that provide them the ability to utilize distributed resources. Applications that use Pilot Jobs to submit to multiple distributed resources also fall in this category. Finally, (iii) applications that are developed using frameworks, where a framework is a generic name for a development tool that supports specific application characteristics (e.g. hierarchical job submission), and recurring patterns (e.g. MapReduce, data parallelism) and system functionality.

It is important to note that SAGA provides the basic API to implement distributed functionality required by applications (typically used directly by the first category of applications), and is also used to implement higher-level APIs, abstractions, and frameworks that, in turn, support the development, deployment and execution of distributed applications [1]. Merzky et al. [2] discusses how SAGA was used to implement a higher-level API to support workflows. In this paper, we will discuss how SAGA can be used to implement runtime frameworks to support the efficient execution of the distributed applications.

1.4.1 BigJob: A SAGA-based Pilot-Job Implementation:

BigJob is a SAGA-based Pilot-Job implementation. In contrast to other Pilot-Job implementations, e.g., Falkon, BigJob natively supports parallel applications (e.g. based on MPI) and works independent of the underlying Grid infrastructure across different heterogeneous backend, e. g. Grids and Cloud, reflecting the advantage of using a SAGA-based approach. Further, the framework is extensible and provides several hooks that can be used to support other resource types and Pilot-Job frameworks.

currently and allocate resources via the big-job interface and bind sub-jobs to these resources. We describe how SAGA interfaces to different backends, while exposing the same user-level BigJob interface and semantics. A tutorial that describes the BigJob API can be found at [3].

Figure 2 shows an overview of the SAGA BigJob implementation for computational Grids. The Grid BigJob comprises of three components: (i) the BigJob Manager that provides the Pilot-Job abstraction and manages the orchestration and scheduling of BigJobs (which in turn allows the management of both big-job objects and sub-jobs), (ii) the BigJob Agent that represents the pilot job and thus, the application-level resource manager on the respective resource, and (iii) advert service which is used for communication between the BigJob Manager and Agent.

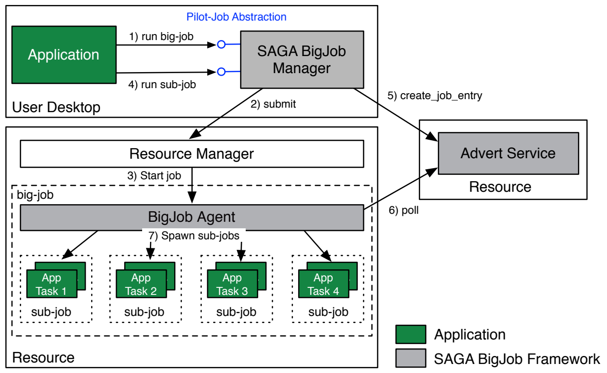

Figure 2: BigJob Architecture: The core of the framework, the BigJob Manager, orchestrates a set of sub-jobs via a BigJob Agent using the SAGA job and file APIs. The BigJob Agent is responsible for managing and monitoring sub-jobs.

2 Condor

Condor is an open source high-throughput computing software framework for coarse-grained distributed parallelization of computationally intensive tasks. Condor can be used to manage workload on a dedicated cluster of computers, and/or to farm out work to idle desktop computers. Condor is developed by the Condor team at the University of WisconsinMadison and is licensed under the Apache License 2.0.

Condor provides a SOAP web-service API called *BirdBath* [1], which can be used to submit, control and monitor jobs. BirdBath is part of the standard Condor distribution, but it has to be enabled explicitly on the server side. The far more common usage mode of Condor is through the condor command line tools. Since they are available on all Condor systems, they seem to be the better candidate for a generic interface to SAGA.

More information about Condor can be found on the website: ***http://www.cs.wisc.edu/condor/***

[1] `http://www.cs.wisc.edu/condor/birdbath/`

2.1 Condor-G

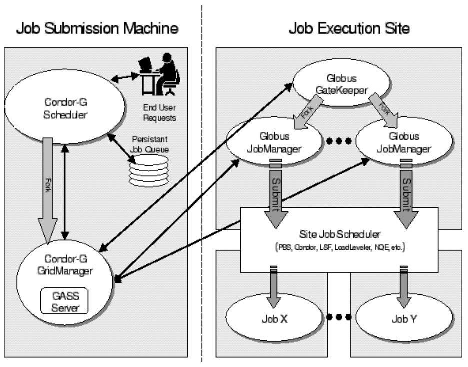

Figure 3: Remote Execution by Condor-G on Globus managed resources. (Image taken from the Condor manual)

Condor-G allows Condor jobs to use resources not under its direct control. It is mostly used to talk to Grid and Cloud resources, like pre-WS and WS Globus, Nordugrid ARC, UNICORE and Amazon EC2. But it can also be used to talk to other batch systems, like Torque/PBS and LSF. Condor-G is a ”plugin” for Condor that allows the user to use the familiar Condor command line utilities (e.g. condor_submit, condor_q) to submit jobs to a Globus resource.

More information about Condor-G can be found on the website: ***http://www.cs.wisc.edu/condor/condorg/***

2.2 Condor Glide-in

Glide-in allows a user to temporarily extend a condor pool with additional resources that are not managed by condor but by globus. This is being achieved by submitting and executing a condor_startd executable (which is also running on every ’regular’ condor resource) along with the right parameters through the Globus resource manager (GRAM). The submission is done via Condor-G, but could in principle also be done via another method.

More information about Condor-G can be found on the website: ***http://www.cs.wisc.edu/condor/condorg/***

3 Integration

3.1 Use-Cases

These are the use-cases that have been identified within the ExTENCI project. The final outcome of this integration project will be tested against these use-cases to make sure that the implementation is feature-complete.

- **U.01:** A user submits, monitors and controls Condor jobs through the SAGA Job API.
- **U.02:** A user can select Condor-G ("Globus via Condor") resources as well as regular Condor pools for job execution through the SAGA Job API.
- **U.03:** A user can utilize SAGA Pilot Job to submit jobs to both the TeraGrid and the OSG
- **U.04:** A user can utilize the above capabilities either via the "command line" or via the TeraGrid Cactus Gateway.

3.2 Requirements

In addition to the use-cases, we define a set of requirements that the implementation has to fulfill:

- **R.01:** E.g. stuff like types of system where this has to work, etc.

3.3 Adaptor Architecture

3.3.1 Interface

We will implement the Condor adaptor on top of the Condor command line tools to communicate with the Condor scheduler, monitor and control jobs. Although Condor provides additional APIs for job submission, these are usually not enabled by default (see *Sec. 2*), so using the command line tools of Condor presents the safest and most portable option.

We have already gained some experience interfacing SAGA with other command line tools (e.g. lsf, pbs, torque), including error handling and encapsulation using the `boost::process` library. We consider it as one of the most effective ways of interfacing with distributed middleware. Another benefit from using Condor's command-line tools is, that the tools itself are easily replaceable with an alias, so we can easily add remote submission capabilities without changing the adaptor code or even implement it. It also allows us to run the Condor adaptor on a machine without condor installed. Consider the following simple example: we're using SAGA on `Host A` which doesn't have Condor installed. We do have SSH access to `Host B` which is a condor submit host. The only thing we have to to, is set up aliases on `Host A`, similar to this:

```
alias condor_q="/bin/ssh HostA
   /usr/bin/condor_q" alias condor_glidein="/bin/ssh HostA
   /usr/bin/condor_glidein"
```

This gives us (limited) remote submission and interaction capabilities at no additional charge and without having to rely on potentially disabled remote API functionality.

3.3.2 Job Description Mapping

SAGA defines its own job description interface which is similar to JSDL. Since Condor jobs will be created programmatically through the SAGA job package API, it is required to find an appropriate mapping from a *saga::job::description* object to a Condor job description. We propose the following mapping:

```
[validation.job.description]

Queue               = Optional, "Vanilla", "Standard", "Scheduler",
                      "Local", "Grid", "MPI", "Java", "VM"

Interactive         = Optional, False,

Cleanup             = Optional, True

SPMDVariation       = Unsupported // Set through universe / queue
ProcessesPerHost    = Unsupported
ThreadsPerProcess   = Unsupported

[condor.job.description]

Queue               = job.description.number_of_processes
Universe            = job.description.queue, "Vanilla"

Executable          = job.description.executable,
Arguments           = job.description.arguments
Environment         = job.description.environment
```

```
Remote_InitialDir    = job.description.working_directory

Input                = job.description.input
Output               = job.description.output
Error                = job.description.error

Deferral_Time        = job.description.job_start_time

Notify_User          = job.description.job_contact

TGProject            = job.description.job_project

[condor.job.requirements]

Machine              = job.description.candidate_hosts

CPUs                 = job.description.total_cpu_count
Memory               = job.description.total_physical_memory

Arch                 = job.description.cpu_architecture
OPSys                = job.description.operating_system_type
```

References

[1] Yaakoub El-Khamra and Shantenu Jha. Developing autonomic distributed scientific applications: a case study from history matching using ensemblekalman-filters. In *GMAC '09: Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, pages 19–28, New York, NY, USA, 2009. ACM.

[2] A. Merzky, K. Stamou, S. Jha and D. S. Katz, A Fresh Perspective on Developing and Executing DAG-Based Distributed Applications: A Case-Study of SAGA-based Montage, accepted for IEEE Conference on eScience 2009, Oxford. http://www.cct.lsu.edu/~sjha/dpa_publications/saga_montage_G09.pdf.

[3] BigJob Tutorials, http://saga.cct.lsu.edu/projects/abstractions/bigjob-a-saga-based-pilot-job-implementation.

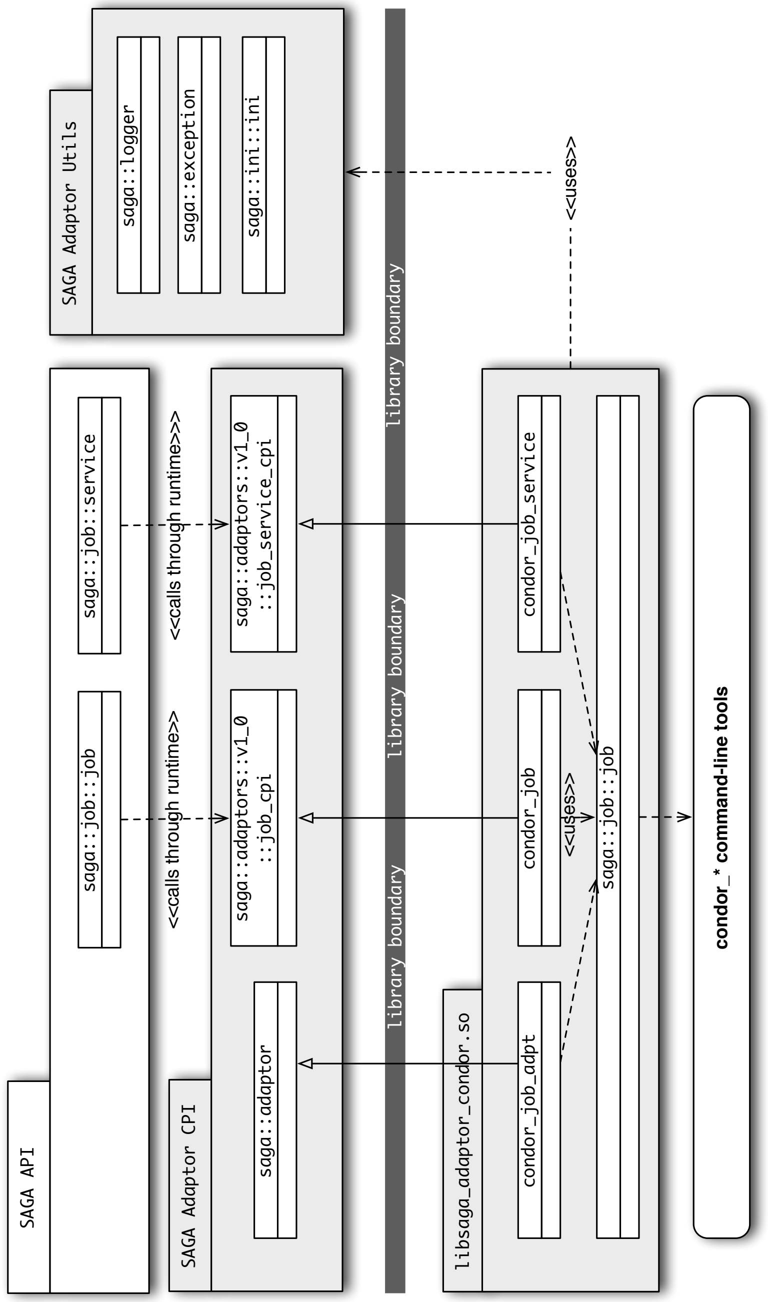

Figure 4: The SAGA Condor adaptor architecture. The adaptor interfaces with SAGA's well defined Capability Provider Interface (CPI), or *Adaptor API*. This ensures that the adaptor fully integrates with SAGA's intelligent adaptor selection mechanism, logging and error handling facilities. To ensure maximum versatility across platforms, the Condor adaptors uses the boost::process API to call the Condor command line tools.