SAGA
A Simple API for Grid Applications

SAGA ON FUTUREGRID

oweidner@cct.lsu.edu | sjha@cct.lsu.edu

Introduction

Overview

SAGA (Simple API for Grid Applications) is an abstract API that describes the basic functionality required to build distributed applications, tools and frameworks so as to be independent of the details of the underlying infrastructure. Any implementation of SAGA can be used to provide simple access layers for distributed systems and abstractions for applications and thereby address the fundamental application design objectives of Interoperability across different infrastructure, Distributed Scale-Out, Extensibility, Adaptivity whilst preserving simplicity.

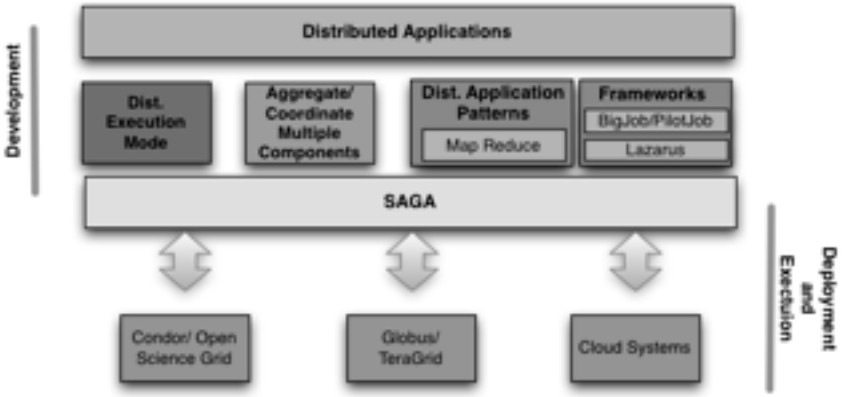

Figure 1: SAGA's role in the distributed applications landscape

SAGA is an OGF (Open Grid Forum) interface standard (GFD-R-P.90[1]) which is supported by a strong and active user community. There are several partial and complete implementations of the SAGA API standard, like JSAGA[2], JavaSAGA and DESHL[3]. In this document, we request the transparent deployment of the **SAGA C++/Python Reference Implementation**[4] on all FutureGrid testbed systems to enable computational scientists to develop a new generation of intelligent distributed scientific applications, frameworks and toolkits that can exploit an emerging global Cyberinfrastructure landscape.

---

[1] http://www.ogf.org/documents/GFD.90.pdf

[2] http://grid.in2p3.fr/jsaga/

[3] http://deisa-jra7.forge.nesc.ac.uk/

[4] http://saga.cct.lsu.edu

C++/Python Implementation

The SAGA C++/Python (reference) implementation (CPP/P-SAGA) is being developed at the Center for Computation and Technology at Louisiana State University and is currently the most feature-complete implementation of the OGF SAGA standard. It supports a wide range of distributed grid/cloud middleware systems (see Table 1) and is being actively used by different research groups on TeraGrid, EGI, NAREGI and other grid infrastructures.

> **CPP/P-SAGA is a software library in the classical sense. It consists of a set of dynamic and static libraries, C++ header files and Python modules than can be used to incorporate SAGA functionality in any application code. It does not provides services nor does it require any persistently running background processes.**

Design

The CPP/P-SAGA library components are separated in two completely orthogonal dimensions. The user of the library may use and combine these freely and develop additional suitable components usable in tight integration with the provided modules.

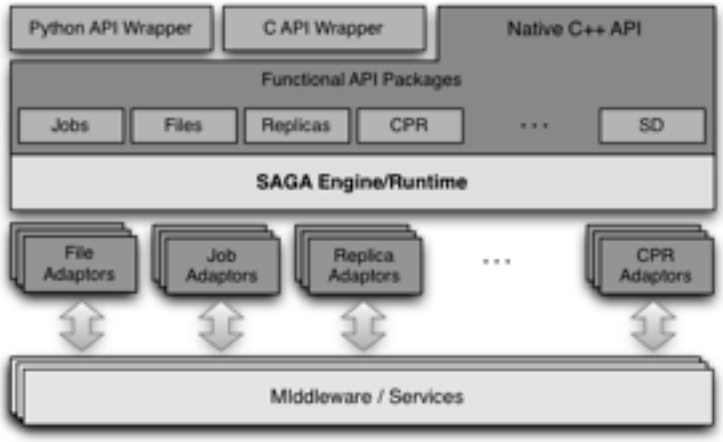

Figure 2: CPP/P-SAGA's modular architecture

Horizontal Extensibility – API Packages

CPP/P-SAGA uses the grouping of sets of API functions as defined by the SAGA specification to define API packages. These packages are: file management, job management, remote procedure calls, replica management, advert management and data streaming. These modules depend only on the SAGA engine. The user is free to use and link only those actually needed by the application, minimizing the memory footprint of his application. It is straightforward to add new packages (as the SAGA specification is expected to evolve) since all common operations needed inside these packages (such as adaptor loading and selection, or method call routing) are imported from the SAGA engine. The creation of new packages is essentially reduced to: (1) add the API package files, and declare the classes, (2) reflect the SAGA object hierarchy (see section , and (3) add class methods.

Vertical Extensibility – Middleware Bindings

A layered architecture (see figure 2) allows us to vertically decouple the SAGA API from the used middleware. Separate adaptors, either loaded at runtime, or pre-bound at link time, dispatch the various API function calls to the appropriate middleware. These adaptors implement a well defined Capability Provider Interface (CPI) and expose that to the top layer of the library, making it possible to switch adaptors at runtime, and hence to switch between different (even concurrent) middleware services providing the requested functionality.

The top library layer dispatches the API function calls to the corresponding CPI function. It additionally contains the SAGA Engine component, which implements: (1) the core SAGA objects such as session, context, task or task container. These objects are responsible for the SAGA look & feel, and are needed by all API packages, and (2) the common functions to load and select matching adaptors, to perform generic call routing from API functions to the selected adaptor, to provide necessary fall back implementations for the synchronous and asynchronous variants of the API functions.

Supported Middleware Systems

The following table shows all available middleware adaptors for SAGA and the corresponding SAGA packages (files, jobs, replica, etc.) they interface with. The table also show the prerequisites the individual adaptor required.

Adaptor Name	SAGA Package	Requirements
AWS/EC2	saga::job	Amazon EC2 API tools
Condor	saga::job	Condor cmd. line tools
Local (Fork) Exec	saga::job	-
Local Files	saga::file	-
PostgreSQL	saga::advert	postgresql client libraries
PostgreSQL	saga::replica	postgresql client libraries
SQLite3	saga::advert	sqlite3 client libraries
SQLite3	saga::replica	sqlite3 client libraries
gLite Cream	saga::job	gLite Cream client-side libraries
Globus GRAM2 / GRAM5	saga::job	Globus Toolkit >= 4.x
Globus GridFTP	saga::file	Globus Toolkit >= 4.x
Globus RLS	saga::replica	Globus Toolkit >= 4.x
GridSAM	saga::job	OpenSSL
Platform LSF	saga::job	LSF cmd. line tools
NAREGI	saga::job	NAREGI cmd. line tools
PBS Pro	saga::job	PBS cmd. line tools
SSH	saga::job	ssh libs
SCP	saga::file	fuse libs
Torque	saga::job	torque cmd. line tools

Table 1: List of Available SAGA Adaptors

A Simple Code Example

The following Python code snippet is a complete SAGA application that copies a set of input files to a remote machine, runs a remote job and copies the results back to the local machine. The remarkable property of this code is, that it can be taken from one environment (e.g. condor-based) to another (e.g. gLite-based) and the only things that need to be changed are the URLs for the job submission and file copy calls:

```
import saga

input = saga.filesystem.file("file://localhost//tmp/data/input01.xml")
input.copy("gridftp://remotehost.remotedomain.org/home/alone/")

jd = saga.job.description()
jd.executable = "/home/alone/mysim"
jd.arguments = ["-i", "input01.xml", "-o", "output01.dat"]

js = saga.job.service("gram://remotehost.org")
mysim_job = js.create_job(jd)
mysim_job.run()

while mysim_job.state == saga.job.Running:
  time.sleep(1)

output = saga.filesystem.file("gridftp://remotehost.org/home/alone/output01.dat")
output.copy("file://localhost//tmp/data/")
```

Deployment

Context

Given the fact that SAGA is a software library that provides a uniform abstraction for existing middleware implementations, **SAGA should be installed in an environment where as many as possible of the adaptor prerequisites (Table 1) are accessible**. We propose to make SAGA part of any HPC image/VM on FutureGrid that provides access to Grid functionality, queueing systems and other middleware supported by SAGA.

This topic needs to be discussed in detail including the question which of SAGA's middleware adaptors should be supported on FutureGrid.

Updates

Although SAGA has reached a substantially stable state, it is still considered work in progress. Especially in the early usage stage in the context of an experimental environment like FutureGrid, we expect problems and bugs to show up. The SAGA Core Components follow an (adjustable) three-monthly release cycle for bugfix releases and on-demand feature releases. It would be desirable to have a simplified deployment / approval procedure that can cope with updates roughly every three months or maybe even more frequently in the early stage of this effort.

Prerequisites

Besides the requisites that are required by individual adaptors and that are listed in Table 1, SAGA requires the following tools and libraries:

- The Boost C++ Libraries (>= 1.34) -- Available at http://www.boost.org/
- Python (>= 3.4) -- compiled with shared library support
- A C++ compiler & runtime -- tested compilers are currently gcc (>=3.4)

Installation

For automated deployment, we provide an installation tool written in Perl called *Mephisto (http://faust.cct.lsu.edu/trac/mephisto).* It allows the installation of SAGA, SAGA's middleware adaptors and any required prerequisite following a predefined installation protocol that can be hosted in a central location. We propose to use Mephisto as the principal deployment tool for SAGA on FutureGrid. It provides the required flexibility and simplicity and can easily be changed to suit FutureGrid-specific needs.