Whitepaper: A Case for SAGA as Access Layer for Production Distributed Computing Infrastructures

Shantenu Jha[*1,2], Andre Merzky[1], Ole Weidner[1]

[1] *Center for Computation & Technology, Louisiana State University, USA*

[2] *Department of Computer Science, Louisiana State University, USA*

[*] *Contact Author* sjha@cct.lsu.edu

December 19, 2010

Aim and Audience of this White Paper

The aim of this document is to inform providers, design architects and users of production Distributed Computing infrastructures (DCI) of the role and relevance of the SAGA API for their DCI. In particular, we focus on how SAGA can increase the number of users and application usage modes by providing SAGA as an access layer to DCI.

1 Introduction

SAGA is an acronym for "Simple API for Grid Applications". As the name suggests, a simple API which facilitates the development and execution of distributed applications on most types of distributed infrastructure. Modern distributed computing environments are very complex infrastructures, and allowing applications to make use of these complex systems is not trivial. By defining a simple API, one requires those complexities to be dealt with at levels other than application code and development. Simplicity of the interface is the primary design principle and objective of SAGA. Functional goals of SAGA are:

1. Provide a stable programming interface to distributed application programmers and tool developers
2. Shield developer from heterogeneous and evolving infrastructures and middlewares
3. By providing the building blocks to distributed and remote operations enable the expression of high-level abstractions and support of distributed application requirements

The fact that SAGA is an OGF standard ensures the community-wide adoption and stability of the API.

2 The SAGA Landscape

2.1 SAGA API Specification

The SAGA API specification is object oriented, and language independent (the API is defined in IDL). The API is structured into various packages (e.g. jobs, replicas, streams, etc.). Those packages have limited dependencies amongst each other - not all SAGA implementations implement all packages. All API packages share certain properties: how are synchronous methods expressed, how are notifications realized, how are security tokens expressed, what types of exceptions are defined, etc. Those properties are specified in the SAGA-Core, the API's look and feel.

That design of the SAGA API allows to specify additional API packages, which adhere to the same look-and-feel. In fact, several such API packages have already been defined (e.g., Service Discovery, Remote Procedure Calls etc.), and are standardized as well, or are in the process of being standardized.

2.2 SAGA: A Community Specification

The SAGA API specification has been developed and guided by the broader distributed computing community at the OGF. (`http://www.ogf.org/`). An analysis of the requirements led to abstractions that were mapped into different SAGA API packages, while ensuring that (a) the overall usability (e.g. the API look-and-feel) was consistent over the whole scope of the API, (b) the API functionality maps relatively well onto existing middleware features, and (c) the API is simple to use. The API is simple, even if the semantic translation and across layers and maintaining implementation fidelity for middleware specific features is not trivial.

***SJ: Andre to put in a few sentences about SAGA and other OGF Efforts, e.g., GIN/PGI – how they cross fertilize and enable ...

***SJ: SJ to shorten a bit..

2.3 SAGA Components

As discussed, the SAGA API specification is language independent. The SAGA distributions contain various language *bindings* for that API. The SAGA-Core distributions deliver those bindings as class files for Java, as modules for Python, and as shared and/or static libraries for C++.

SAGA as an API would be rather useless if it would not also offer bindings to the various middlewares. Those bindings are, for all major SAGA implementations, provided as *adaptors*. Some simple adaptors are usually packaged with the SAGA-Core distributions, but otherwise are packaged and distriubuted separately (details see in section **??**).

While SAGA is foremost an API, the SAGA distributions support end users in a variety of ways. In particular, the SAGA distributions also include command line tools implemented via the SAGA API, and higher level libraries for common distributed programming patterns,

also basing on the SAGA API. Further, the SAGA distributions provides comprehensive support to compile, link and run SAGA applications (configure scripts, make support, runtime wrappers, developer tutorials , etc).

Command line tools are, in our experience, amongst the first components of any distributed middleware to be exploited by end users. SAGA-C++ provides a set of command line tools which basically cover the complete semantic set of SAGA API calls, such as job submission and management, file management, replica management, etc.

Several SAGA based projects are actively developing and using higher level programming abstractions, such as pilotjobs, bigjobs, mapreduce, or workflows. Such components are routinely installed and used by a number of user communities, and represent significant added avlue, although they are not part of the SAGA core code base. It should be noted that the SAGA Python bindings, which are usually installed by default, are very frequently used to provide tooling and higher level programming abstractions.

2.4 SAGA Core: Implementations and Deployment

2.4.1 SAGA Implementations

The language independent SAGA API specification has been mapped to multiple programming languages, in particular to C++, Java and Python. Multiple implementations exists, the most notable ones are SAGA-C++, JSAGA and JavaSAGA.

SAGA-C++ is...

JSAGA and JavaSAGA are

All three implementations provide python bindings - the Java implementations realize those via Jython, the C++-implementation via boost-python. The two python bindings are at the moment being unified, and have already been shown to be interoperable.

Interestingly, all three discussed SAGA implementations are adaptor based: they implement a relatively small library which provides the SAGA API, and a set of adaptors which translate the SAGA API calls into the respective middleware operations. It is those adaptors which encapsulate most of the complexity which was formerly present in the applications layer. While SAGA adaptors are relatively easy to implement, at least as a prototype, they require significant maintainance effort to keep up with the middleare intricacies and evolution.

This white paper focuses, from here on, on the SAGA-C++ implementation.

2.4.2 Important SAGA Deployments

The different SAGA implementations, and in particular SAGA-C++, have by now been in use in different user communities for a number of years, and thus have matured enough to enter the field of production cyber infrastructures. At the same time, the number of supported backends has grown to a level that basically all current production infrastructures are supported (e.g. for jobs we support ARC, gLite, Globus, Condor, PBS, Torque, DRMAA, EC2, Eucalyptus, BES, Naregi, Unicore, SMOA, Genesis-II, fork, and ssh).

SAGA has so far been successfully and routinely used on the TeraGrid, LONI, FutureGrid, DEISA, NAREGI/RENEKI and a number of smaller, more localized DCI. It has also been used by a range of e-Science projects such as EGEE, DGrid, VPH etc.

SAGA-C++ has one major external dependency, which is boost[1], a set of C++ headers and lbraries widely used in the C++ community. While boost itself is a fairly complex code base, it is routinely packaged by the various Unix distributions, and thus usually available out of the box. Further, SAGA-C++ is able to compile against a wide range of boost releases, including those which are the default versions for the currently used Linux distributions. Postgresql client libraries are not strictly required, but recommended for some of the core SAGA functionality.

While SAGA is relatively easy to deploy in applications space (i.e. user space), its overall goal of improving the end user experience of distributed systems benefits greatly from system level installations. It is straight forward to install SAGA in user space (`configure; make; make install`), but in our experience, the correct environment setup to use a SAGA installation in user space is still a stumbling stone for many end users (`LD_LIBRARY_PATH` etc). We thus prefer SAGA to be available in system space, which lowers the entry barrier significantly.

We are currently working to provide binary releases, in the form of Debian and Ubuntu packages, and RPMs. Also, we are currently integrating SAGA into NMI and ETICS build and testing environments, which we hope will lower the effort for system level installations significantly.

2.5 SAGA Adaptors: Implementation and Deployment

For a list of adaptors that are currently supported, refer to: `http://saga.cct.lsu.edu/software/cpp`

2.6 SAGA usage modes

Although SAGA is foremost an API, the SAGA distributions support end users in a variety of ways. In particular, the SAGA distributions also include command line tools implemented via the SAGA API, and higher level libraries for common distributed programming patterns, also basing on the SAGA API.

[1] `http://www.boost.org/`

3 SAGA Usage and Active Projects

3.1 SAGA Active Projects

3.1.1 Standards promote Interoperability

ExTENCI

ExTENCI: TeraGrid-OSG (2010-12)
Cactus Application Scenarios

- Problem size varies determinant of Infrastructure used
  - TG, OSG or either...
- MPI-based applications have a very complex SW environment that they need to worry about
- Application Scenarios/Usage Modes
  - Ensemble of Cactus Simulations (NumRel, EnKF (Petroleum Eng))
  - Multiphysics Code (GR-MHD, CFD-MD)
  - Spawning Simulations (Realtime outsourcing from BlueWaters/Ranger to specialised architectures or less powerful resources)

DEISA/TG/interop VPH (under virt phys human), pilotjob (mult ensembles of 16/32 way par jobs)

gLite / Ganga

glite+globus/CERN/HEP /

RENKEI/NAREGI

3.1.2 Applications Scenarios

Cybertools

Mapper: multiphysics,

NeuGrid / UWE

medical imaging, workflow

3.1.3 Tooling

JSAGA, Service Discovery

SAGA-based Pilot Jobs (BigJobs)

Computational Biology Gateways

4 Analysis of Use Cases

SAGA is used for multiple reasons. Three primary usage modes of SAGA are the following: (i) Simplifying access layer, (ii) building block for tools and distributed execution execution, and (iii) a distributed scripting and programming capability.

Distributed Computing is more than just submitting isolated jobs.

It is about federating resources dynamically; about coordinated execution of heteregeneous and dynamic workloads; it is about distributed data management etc..

A primary objective of SAGA is to support and simplify the implementation of distributed applications. As described earlier, that is achieved by providing a library which implements the SAGA API.

Application Prototyping and Tooling

The SAGA python bindings have been proven to be immensely helpful for application prototyping. But also, they are very helpful when interactively testing remote operations (in the interactive python interpreter / python shell). Finally, it is very easy to implement small command line tools in python, which are able to mimic and test smaller portions of the overall application. For example, it is straight forward to implement a specific job control component of an application in a stand alone python script, and to later include the same functionality in the application proper, with the confidence that the semantics of the remote operations will be well preserved.

Application Development

The SAGA API provides very concise and high level method calls which cover the vaste majority of distributed operations, as required by the target user community – scientific application and tool developers. Further, as the API specification and implementation is *standardized*, and thus stable, it allows for a 'write once, run anywhere' approach, which is in general not available otherwise (or at least not without *significantly* increase of application complexity).

Application Deployment

Runtime Configuration

5 Relevance to EGI/UMD

Access Layer to EMI - ARC, gLite and Unicore, while working well with Globus (IGE). In fact first few success stories of IGE are built around applications that use SAGA.

Distributed Scripting/programmatic access

Interoperability with OSG (and TeraGrid-XD).

6 SAGA as a Standardized Programmatic and Access Layer: Advantage to DCI Providers

As part of the ExTENCI project, SAGA will make major advances towards becoming a broadly usable programmatic access layer to Condor/OSG. For example the Structural Biology Grid that currently (`http://SBGrid.org`) currently implements sophisticated analysis and user-defined pipelines. However, these are inherently localized and confined to specific infrastructure. Replacing "local python" calls with "distributed (SAGA) python" calls enables the seamless utilization of DCI. This provides a simple mode of extensibililty of infrastructure, without any major refactoring of code. The advantages of this to the end-user is obvious; the lowered barrier-to-entry for novel users and communities will increase the ease and uptake of DCI thus benefitting DCI providers/organizations.

7 SAGA Future/Roadmap

References