S&B-SDS-0000

Version：Draft

SWIFT Innovation for BOINC Software Design Spec

Internet Technology R&D Center
Huazhong University of Science & Technology
Wuhan, China
Feb. 2008

Revision History

Date	Name	Version	Description
2008.2.29	Zhou	0.1	
2008.3.16	Zhou		Update architecture

Contents

1. Scope

This document is the Software design specification to develop the Swift Innovation for integrating BOINC with Swift. The major components of software design are all included in this document, which are Software Architecture, Component Description, Operation Flow, Data Structure, Function Definition and Library Description. This document is served as Swift Innovation Software Design accordance; any software activity of it must refer this as guidance whenever possible.

2. References

- ✓ Java CoG Kit Abstraction Guide.
  (http://wiki.cogkit.org/index.php/Java_CoG_Kit_Abstraction_Guide)
- ✓ Java CoG Kit Workflow Guide.
  (http://wiki.cogkit.org/index.php/Java_CoG_Kit_Workflow_Guide)
- ✓ Swift home. (http://www.ci.uchicago.edu/swift/index.php)
- ✓ David P. Anderson: BOINC-A System for Public-Resource Computing and Storage.

3. Requirements

Swift is a professional distributed computing platform performing excellently at workflow control and simplification. But, the computing resource provider is only computer grid. These grid resources are expensive and ability-limited.

Meanwhile, volunteer computing becomes more and more popular, due to the technology development and the enthusiasm for science research of the public. Unlike the grid computing, volunteer computing is unreliable, insecure, uncountable, but with infinite computing resource. Among those volunteer distributed computing systems, BOINC is the most outstanding one.

Swift Innovation is a project which aims at expanding Swift with volunteer computers. Because of BOINC's broad employment, it can serve as the computing resource. This system integrates Swift and BOINC together and composes their advantages. Through this platform, tasks are specified and managed by Swift and executed by BOINC platform. Besides the features Swift already has, Swift Innovation should have following additional ones:

- ✓ Volunteer computing compatible. Swift can use the rich volunteer computing resource now too.
- ✓ Transparent for users. The users who have used Swift before don't need to make any changes. The interfaces to the user will be uniform with Swift.
- ✓ Manageable. Users can choose whether BOINC or traditional grid provider is adopted. Since BOINC is not as reliable as grid computing, when running important project, grid computing may be a better choice.

4. Brief Description about Swift & BOINC

4.1. Swift

Swift is a system for the rapid and reliable specification, execution, and management of large-scale science and engineering workflows. It supports applications that execute many tasks coupled by disk-resident datasets - as is common, for example, when analyzing large quantities of data or performing parameter studies or ensemble simulations.

The technical overview of the Swift architecture is described as below:

- ✓ karajan - the core execution engine
- ✓ Execution layer

The execution layer causes an application program (in the form of a unix executable) to be executed either locally or remotely.

The two main choices are local UNIX execution and execution through GRAM. Other options are available, and user provided code can also be plugged in.

The kickstart utility can be used to capture environmental information at execution time to aid in debugging and provenance capture

- ✓ SwiftScript language compilation layer

Step i: text to XML intermediate form parser/processor. parser written in ANTLR - see resources/VDL.g. The XML Schema Definition (XSD) for the intermediate language is in resources/XDTM.xsd.

Step ii: XML intermediate form to Karajan workflow. Karajan.java - reads the XML intermediate form, compiles to karajan workflow language - for example, expressions are converted from SwiftScript syntax into Karajan syntax, and function invocations become karajan function invocations with various modifications to parameters to accomodate return parameters and dataset handling.

- ✓ Swift/karajan library layer

Some Swift functionality is provided in the form of Karajan libraries that are used at runtime by the Karajan workflows that the Swift compiler generates.

4.2. BOINC

BOINC is a software platform for volunteer computing and desktop Grid computing. BOINC is a centralized system with a project server and an amount of PCs. The server dispatches work to those PCs based on a certain strategy and after PCs having finished the work, results are sent back to the server. BOINC server will decide whether it accepts the results or not, of course, based on a certain strategy, which is defined by project maintainer. In this way, PCs all over the world could help computing a large job and as a result accelerates the job progress.

BOINC is designed to support applications that have large computation requirements, storage requirements, or both. The main requirement of the application is that it be divisible into a large number (thousands or millions) of jobs that can be done independently.

If the project is going to use volunteered resources, there are additional requirements:

Public appeal: An application must be viewed as interesting and worthwhile by the public in order to gain large numbers of participants. A project must have the resources and commitment to maintain this interest, typically by creating a compelling web site and by generating interesting graphics in the application.

Low data/compute ratio: Input and output data are sent through commercial Internet connections, which may be expensive and/or slow. As a rule of thumb, if your application produces or consumes more than a gigabyte of data per day of CPU time, then it may be cheaper to use in-house cluster computing rather than volunteer computing.

5. Software Architecture

5.1. Overview

Since Swift and BOINC can't communicate with each other directly. What we should to do is to implement an interface module "BOINC" below "cog kit" in Swift and add a module "Swift Adaptor" in BOINC. So that BOINC project server can accept remote request of submitting jobs from swift and return back result after complete the jobs. The architecture of the system is illustrated in figure 1.

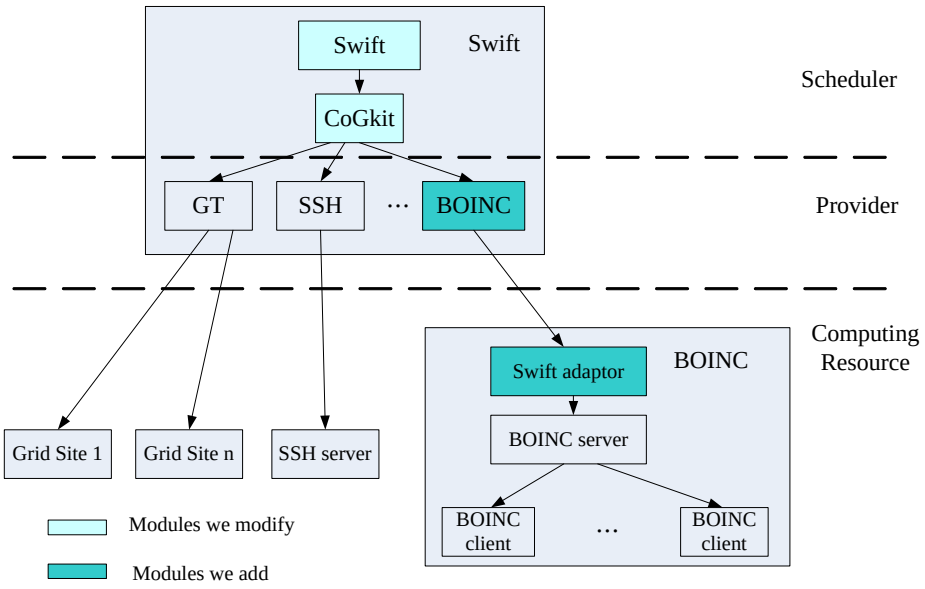

Figure 1: The architecture of the system

The Swift module includes Swiftscript and the swift compiler. It's the interface between system and users. Scientists can write swift programs to define the workflow and get the output here.

The CoGkit layer provides the function of workflow management. It provides interfaces to the Swift layer and dispatches tasks to and receives result from the grid resource. These two modules have been implemented in the Swift system; we needn't to make any changes other than register of the boinc provider so that the scheduler knows what interfaces it provides.

The boinc provider dispatches Swift tasks to boinc system. As the same as any other provider,

it should provide two functions: task management and file transfer. Most of the features of boinc provider are inherited from SSH provider, including file transfer, signaling method and security context. This module interacts with the SSH server in the boinc server side: transfer applications and arguments to the server and execute commands remotely. After application is submitted correctly, boinc provider will wait until BOINC system completes the application and transfers the result back.

The Swift adaptor works at the same machine of the BOINC server. It receives applications from provider and adds them to the BOINC database. Then, it sends a command to BOINC server to update and dispatch the tasks to clients. After the tasks are all completed, it sends the results to boinc provider back.

5.2. Data Flow

BOINC provider first transfers executable and input files to remote “Swift Adaptor” running on the computer that “BOINC project server” locates on. Then “Swift Adaptor” would submit these files to “BOINC project server” and tell it to generate a new job. “BOINC project server” will do everything as it likes based on its own strategy. After having finished the job, “BOINC project server” returns results, that are, output files or error, to “Swift Adaptor” which then would deliver these results to “BOINC provider”. Now, a whole cycle has finished: Cog has successfully executed a job on remote “BOINC project server”.

The workflow of the system is illustrated in figure 2.

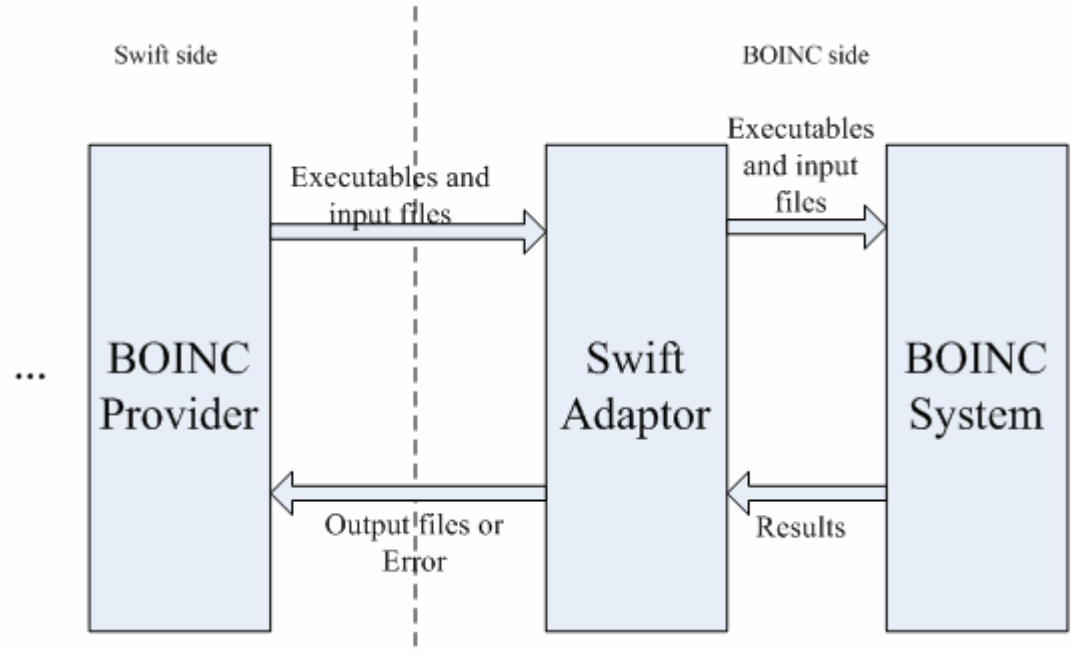

Figure 2: Work Flow of System

6. Component Description

The Swift Innovation project is based on Swift and BOINC and it acts as a bridge between them then Swift can use BOINC as a computing grid as any other computing provision.

Since Swift and BOINC are different systems and the data format and behaviors are different

too. Besides, the Swift and BOINC will be working in different hosts. So, we have to build a translator between them.

As for Swift, different grid nodes are unified by “providers”. Each kind of computing node is abstracted to a provider. In the same way, BOINC provider is added in the Swift system below CoGkit module.

In the BOINC side, since BOINC doesn’t support remote job submission, additional Swift adaptor is added to receive applications and arguments from Swift and submit the tasks to BOINC.

6.1. BOINC Provider

The provider module communicates with the boinc system side. The tasks it needs to complete include: authentication, task management, and result disposal. According to this, boinc provider can be divided to 3 submodules, and the work flow is as follows:

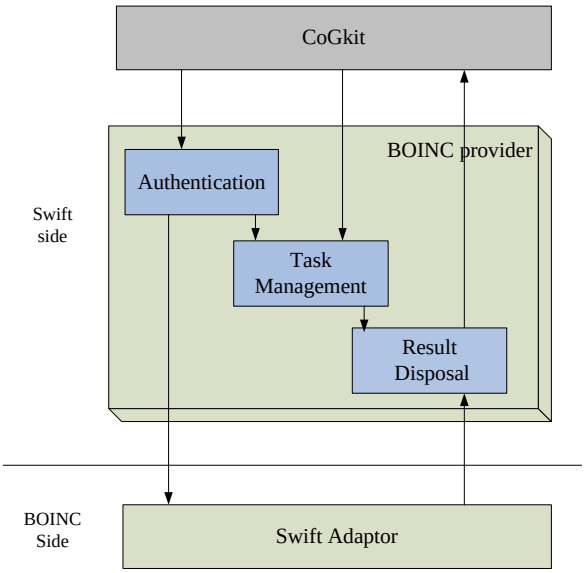

Figure 3: BOINC provider modules

A simple description of the work flow goes: when tasks are dispatched to BOINC provider, it will authenticate whether the user is legal; if passed, provider begins to transfer application data to the Swift adaptor and tell it to start work; if no error happens, result disposal module will add the task to its monitor list; when the task is over, it gets the result back.

6.1.1. Authentication

Since boinc provider is inherited from ssh provider, it uses the same authenticate as ssh. Detailed, it supports two ways to identify. One is username and password pair, and another is public and private key pair. The way user chooses should be specified in the configure file: auth.defaults in folder: ~/.ssh.

6.1.2. Task Management

Applications include several components: executables, arguments, parameters, input and output files. All these should be specified in the swift script files. After been compiled, these components will be stored in an attribute structure. At first, the provider transfers all the input files to the swift adaptor, then it will generate a shell command according to the structure and transferred to the SSH server in the adaptor host to execute. This command will execute the swift adaptor and pass the structure data as the arguments.

6.1.3. Result Disposal

When applications are delivered to the swift adaptor successfully, the main thread exits, the left works will be done by the result disposal.

This module works as an independent thread. It maintains a task list including all the tasks dispatched to the boinc system. This thread will inquire the swift adaptor about the status. If the tasks have been completed successfully, it let the adaptor transfer the result back. If there is any error, it will terminate the task or resume the task, according to the parameter in the configure file or the command line.

6.2. Swift adapter

Here is the work flow:

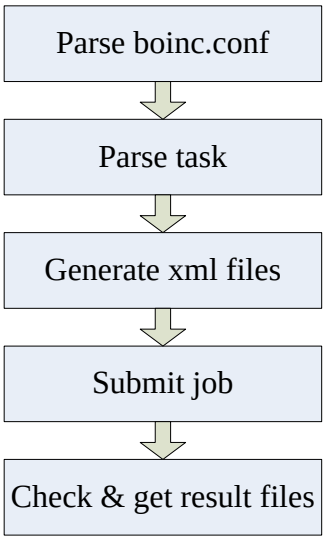

Figure 4: The workflow of Swift adaptor

“Swift adapter” implements these functions:

- Parse configuration file named boinc.conf, which provides enough info about BOINC project for us to submit a job.
- Parse task received from “BOINC provider” to get job parameters, such as app name, input files, output files, platform, etc.
- Generate xml files based on job parameters. They are job.xml (needed by wrapper

program), workunit template and workunit result template.

- Execute some programs and python scripts to submit a job to BOINC project.
- Check job status periodically and get result files.

If any error happens, “Swift adapter” will exit with non-zero code.

File transfer and task-description transfer issues are handled by SSH server. “swift adapter” is called by SSH server only if BOINC provider asks so. Once “swift adapter” starts, the command line argument gives the task description.

6.2.1. Parse configuration file

There should be a file named boinc.conf in home directory of the user who sets up the BOINC project. The file is in the format item=”value”, here is an example file:

```
db_name="cplan"
db_user="liufan"
db_passwd=""
db_host=""
boinc_proj_dir="/home/liufan/projects/cplan/"
download_dir="/home/liufan/projects/cplan/download"
apps_dir="/home/liufan/projects/cplan/apps"
templates_dir="/home/liufan/projects/cplan/templates"
wrapper_win32="/home/liufan/wrapper_5.10_windows_intelx86.exe"
wrapper_linux32="/home/liufan/wrapper_5.10_i686-pc-linux-gnu"
results_dir="/home/liufan/projects/cplan/sample_results"
validator="sample_bitwise_validator"
assimilator="sample_assimilator"
```

This file provides enough information for us to submit a job to BOINC project and get result files. So it’s definitely necessary..

6.2.2. Parse task

In order to submit a job to BOINC project, not only info about BOINC project is needed, but also info about job itself. These include app name, input files, output files, platform, deadline, command arguments, etc.

After knowing info about BOINC project and job, “swift adapter” is ready to generate needed files and then submit the job to BOINC project.

6.2.3. Generate xml files

Before generating xml files, “Swift adapter” will first check if necessary files exist, based on the result parsing task. If no, quits.

Several xml files are generated, including job.xml, workunit template, workunit result template. Job.xml is needed by the wrapper program. Workunit template describes info except output files of a job. Workunit template describes info about result files of a job.

Hint: wrapper program is used to run applications written without BOINC APIs in BOINC system.

6.2.4. Submit job

This includes three procedures: add application to database, add application version to database, and create workunit.

- Add application
  "swift adapter" use mysql C interface to add record to BOINC project database.
- Add application version
  An application may has 1.0, 1.1 or other versions, also, it may run in Linux or Windows or Mac platform, and these make "application version" necessary.
  Using swift, we don't think versions like 1.0, 1.1 is necessary. Instead, if someone wants to run a new version, for example 2.0, he just submits the job with a totally new application name.
  Platform is considered in "swift adapter".
  "Swift adapter" executes a python script, which is provided by BOINC, to add application version to BOINC database.
- Create workunit
  A workunit corresponds to running an application once. An application can run again with different input files and parameters, and this gives the concept workunit.
  Here in "swift adapter", a workunit corresponds to a task received from "BOINC provider". "swift adapter" will generate a workunit for each task..
  "swift adapter" calls the program "create_work" provided by BOINC to create workunit.

After these procedures, a job has been submitted to BOINC project.

6.2.5. Check & get result files

"swift adapter" uses the default validator and assimilator to validate and assimilate result files.

"swift adapter" will check BOINC project database periodically and after the job completes, it would move result files to specific destination, or, in case an error occurs, exit with non-zero code.