Site Configuration Guide

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Overview

This guide explains details required to run Swift on various system types, with details for specific installations on which Swift is currently used. For a given system type, most instructions should work on that system type anywhere. However, details such as queue names or file system locations will have to be customized by the user.

2 Prerequisites

This guide assumes that you have already downloaded and installed Swift. It assumes that Swift is in your PATH and that you have a working version of Sun Java 1.5+. For more information on downloading and installing Swift, see the Swift Quickstart Guide.

3 Beagle (Cray XE6)

Beagle is a Cray XE6 supercomputer at UChicago. It employs a batch-oriented computational model where-in a PBS schedular accepts user's jobs and queues them in the queueing system for execution.

The computational model requires a user to prepare the submit files, track job submissions, chackpointing, managing input/output data and handling exceptional conditions manually. Running Swift under Beagle can accomplish the above tasks with least manual user intervention and maximal oppurtunistic computation time on Beagle queues. In the following sections, we discuss more about specifics of running Swift on Beagle. A more detailed information about Swift and its workings can be found on Swift documentation page here:

```
http://swiftlang.org/docs/index.php
```

More information on Beagle can be found on UChicago Beagle website here:

```
http://beagle.ci.uchicago.edu
```

3.1 Requesting Access

If you do not already have a Computation Institute (CI) account, you can request one at https://www.ci.uchicago.edu/accounts/. This page will give you a list of resources you can request access to.

If you already have an existing CI account, but do not have access to Beagle, send an email to support@ci.uchicago.edu to request access.

3.2 Connecting to a login node

Once you have account, you should be able to access a Beagle login node with the following command:

```
ssh yourusername@login.beagle.ci.uchicago.edu
```

3.3 Getting Started with Swift

Follow the steps outlined below to get started with Swift on Beagle:

step 1. Load the Swift and Sun-java module on Beagle as follows: `module load swift sun-java`

step 2. Create and change to a directory where your Swift related work will stay. (say, `mkdir swift-lab`, followed by, `cd swift-lab`)

step 3. To get started with a simple example running `/bin/cat` to read an input file `data.txt` and write to an output file `f.nnn.out`, start with writing a simple swift source script as follows:

```
type file;

/* App definition */
app (file o) cat (file i)
{
  cat @i stdout=@o;
}

file out[]<simple_mapper; location="outdir", prefix="f.",suffix=".out">;
file data<"data.txt">;

/* App invocation: n times */
foreach j in [1:@toint(@arg("n","1"))] {
  out[j] = cat(data);
}
```

step 4. The next step is to create a sites file. An example sites file (sites.xml) is shown as follows:

```
<config>
  <pool handle="pbs">
    <execution provider="coaster" jobmanager="local:pbs"/>
    <!-- replace with your project -->
    <profile namespace="globus" key="project">CI-CCR000013</profile>

    <profile namespace="globus" key="providerAttributes">
                    pbs.aprun;pbs.mpp;depth=24</profile>

    <profile namespace="globus" key="jobsPerNode">24</profile>
    <profile namespace="globus" key="maxTime">1000</profile>
    <profile namespace="globus" key="slots">1</profile>
    <profile namespace="globus" key="nodeGranularity">1</profile>
    <profile namespace="globus" key="maxNodes">1</profile>

    <profile namespace="karajan" key="jobThrottle">.63</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>

    <filesystem provider="local"/>
    <!-- replace this with your home on lustre -->
    <workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
  </pool>
</config>
```

step 5. In this step, we will see the config and tc files. The config file (cf) is as follows:

```
wrapperlog.always.transfer=true
sitedir.keep=true
execution.retries=1
lazy.errors=true
use.provider.staging=true
provider.staging.pin.swiftfiles=false
foreach.max.threads=100
provenance.log=false
```

The tc file (tc) is as follows:

```
pbs cat /bin/cat null null null
```

More about config and tc file options can be found in the Swift documentation.

step 6. Run the example using following commandline:

```
swift -config cf -tc.file tc -sites.file sites.xml catsn.swift -n=1
```

You can further change the value of `-n` to any arbitrary number to run that many number of concurrent `cat`

step 7. Swift will show a status message as "done" after the job has completed its run in the queue. Check the output in the generated `outdir` directory (`ls outdir`)

```
Swift 0.93RC5 swift-r5285 cog-r3322

RunID: 20111218-0246-6ai8g7f0
Progress:  time: Sun, 18 Dec 2011 02:46:33 +0000
Progress:  time: Sun, 18 Dec 2011 02:46:42 +0000  Active:1
Final status:  time: Sun, 18 Dec 2011 02:46:43 +0000  Finished successfully:1
```

3.4 Larger Runs on Beagle

A key factor in scaling up Swift runs on Beagle is to setup the sites.xml parameters. The following sites.xml parameters must be set to scale that is intended for a large run:

- **maxTime** : The expected walltime for completion of your run. This parameter is accepted in seconds.
- **slots** : This parameter specifies the maximum number of pbs jobs/blocks that the coaster scheduler will have running at any given time. On Beagle, this number will determine how many qsubs swift will submit for your run. Typical values range between 40 and 60 for large runs.
- **nodeGranularity** : Determines the number of nodes per job. It restricts the number of nodes in a job to a multiple of this value. The total number of workers will then be a multiple of jobsPerNode * nodeGranularity. For Beagle, jobsPerNode value is 24 corresponding to its 24 cores per node.
- **maxNodes** : Determines the maximum number of nodes a job must pack into its qsub. This parameter determines the largest single job that your run will submit.
- **jobThrottle** : A factor that determines the number of tasks dispatched simultaneously. The intended number of simultaneous tasks must match the number of cores targeted. The number of tasks is calculated from the jobThrottle factor is as follows:

```
Number of Tasks = (JobThrottle x 100) + 1
```

Following is an example sites.xml for a 50 slots run with each slot occupying 4 nodes (thus, a 200 node run):

```
<config>
  <pool handle="pbs">
    <execution provider="coaster" jobmanager="local:pbs"/>
    <profile namespace="globus" key="project">CI-CCR000013</profile>

    <profile namespace="globus" key="ppn">24:cray:pack</profile>

    <!-- For swift 0.93
    <profile namespace="globus" key="ppn">pbs.aprun;pbs.mpp;depth=24</profile>
    -->

    <profile namespace="globus" key="jobsPerNode">24</profile>
    <profile namespace="globus" key="maxTime">50000</profile>
    <profile namespace="globus" key="slots">50</profile>
    <profile namespace="globus" key="nodeGranularity">4</profile>
    <profile namespace="globus" key="maxNodes">4</profile>

    <profile namespace="karajan" key="jobThrottle">48.00</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>

    <filesystem provider="local"/>
    <workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
  </pool>
</config>
```

3.5 Troubleshooting

In this section we will discuss some of the common issues and remedies while using Swift on Beagle. The origin of these issues can be Swift or the Beagle's configuration, state and user configuration among other factors. We try to identify maximum known issues and address them here:

- Command not found: Swift is installed on Beagle as a module. If you see the following error message:

```
If 'swift' is not a typo you can run the following command to lookup the package that ↩
    contains the binary:
    command-not-found swift
-bash: swift: command not found
```

The most likely cause is the module is not loaded. Do the following to load the Swift module:

```
$ module load swift sun-java
Swift version swift-0.93 loaded
sun-java version jdk1.7.0_02 loaded
```

- Failed to transfer wrapperlog for job cat-nmobtbkk and/or Job failed with an exit code of 254. Check the <workdirectory> element on the sites.xml file.

```
<workdirectory >/home/ketan/swift.workdir</workdirectory>
```

It is likely that it is set to a path where the compute nodes can not write, e.g. your /home directory. The remedy for this error is to set your workdirectory to the /lustre path where swift could write from compute nodes.

```
<workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
```

4 Amazon EC2 Cloud

Please refer to the Swift Tutorial for Cloud and Ad hoc Resources.

5 Fusion (x86 cluster)

Fusion is a 320-node computing cluster for the Argonne's Laboratory Computing Resource Center (LCRC). The primary goal of the LCRC is to facilitate mid-range computing in all of the scientific programs of Argonne and the University of Chicago.

This section will walk you through running a simple Swift script on Fusion.

5.1 Requesting Access

If you do not already have a Fusion account, you can request one at https://accounts.lcrc.anl.gov/request.php. Email support@lcrc.anl.gov for additional help.

5.2 Projects

In order to run a job on a Fusion compute node, you must first be associated with a project.

Each project has one or more Primary Investigators, or PIs. These PIs are responsible for adding and removing users to a project. Contact the PI of your project to be added.

More information on this process can be found at http://www.lcrc.anl.gov/info/Projects.

5.3 SSH Keys

Before accessing Fusion, be sure to have your SSH keys configured correctly. SSH keys are required to access fusion. You should see information about this when you request your account. Check ssh FAQ or email support@lcrc.anl.gov for additional help.

5.4 Connecting to a login node

Once your keys are configured, you should be able to access a Fusion login node with the following command:

```
ssh <yourusername>@fusion.lcrc.anl.gov
```

5.5 Creating sites.xml

Swift uses various configuration files to determine how to run an application. This section will provide a working configuration file which you can copy and paste to get running quickly. The sites.xml file tells Swift how to submit jobs, where working directories are located, and various other configuration information. More information on sites.xml can be found in the Swift documentation.

The first step is to paste the text below into a file named sites.xml.

```

```

This file will require one customization. Create a directory called swiftwork. Modify _WORK_ in sites.xml to point to a new directory. For example

```
<workdirectory>/tmp/swiftwork</workdirectory>
```

5.6 Creating tc

The tc configuration file gives information about the applications that will be called by Swift. More information about the format of tc can be found in the Swift user guide.

Paste the following example into a file named tc

```

```

5.7 Copy a Swift Script

Within the Swift directory is an examples directory which contains several introductory Swift scripts. The example we will use in this section is called catsn.swift. The script copies input file's content to another file using the Unix cat utility. Copy this script to the same directory that your sites.xml and tc.data files are located.

```
$ cp ~/swift/examples/misc/catsn.swift .
$ cp ~/swift/examples/misc/data.txt .
```

Tip
The location of your swift directory may vary depending on how you installed it. Change this to the examples/misc directory of your installation as needed.

5.8 Run Swift

Finally, run the script

```
$ swift -sites.file sites.xml -tc.file tc catsn.swift
```

You should see 10 new text files get created, named catsn*.out. If you see these files, then you have succesfully run Swift on Fusion!

5.9 Queues

Fusion has two queues: shared and batch. The shared queue has a maximum 1 hour walltime and limited to 4 nodes. The batch queue is for all other jobs.

Edit your sites.xml file and edit the queue option to modify Swift's behavior. For example:

```
<profile namespace="globus" key="queue">batch</profile>
```

More information on Fusion queues can be found at http://www.lcrc.anl.gov/info/BatchJobs.

5.10 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at swift-user. When submitting information, send your sites.xml file, your tc.data, and any Swift log files that were created during your attempt.

6 Futuregrid Cloud

Please refer to the Swift Tutorial for Cloud and Ad hoc Resources.

7 Geyser and Caldera (x86 clusters)

The Geyser and Caldera clusters are specialized data analysis and visualization resources at NCAR (National Center for Atmospheric Research).

More information about these clusters can be found at http://www2.cisl.ucar.edu/resources/geyser_caldera.

7.1 Example sites.xml

The following sites.xml has the basic definitions for how to run a job using the LSF scheduler.

```
<config>
  <pool handle="geyser">
    <filesystem provider="local"/>
    <execution provider="coaster" jobmanager="local:lsf"/>
    <profile namespace="globus"   key="jobsPerNode">1</profile>
    <profile namespace="globus"   key="maxTime">3600</profile>
    <profile namespace="globus"   key="maxwalltime">00:05</profile>
    <profile namespace="globus"   key="lowOverallocation">100</profile>
    <profile namespace="globus"   key="highOverallocation">100</profile>
    <profile namespace="globus"   key="nodeGranularity">1</profile>
    <profile namespace="globus"   key="maxNodes">1</profile>
    <profile namespace="globus"   key="project">P93300606</profile>
    <profile namespace="globus"   key="queue">small</profile>
    <profile namespace="karajan"  key="jobThrottle">4</profile>
```

```
    <profile namespace="karajan"  key="initialScore">10000</profile>
    <workdirectory>/glade/scratch/davkelly</workdirectory>
  </pool>
</config>
```

The values for workdirectory, queue, and project will likely need to be adjusted based on the requirements of your project.

8 Grids: Open Science Grid and TeraGrid

Please refer to the Swift Tutorial for OSGConnect.

9 Intrepid (Blue Gene/P)

Intrepid is an IBM Blue Gene/P (BG/p) supercomputer located at the Argonne Leadership Computing Facility. More information on Intrepid can be found at http://www.alcf.anl.gov. Surveyor and Challenger are similar, smaller machines.

9.1 Requesting Access

If you do not already have an account on Intrepid, you can request one here. More information about this process and requesting allocations for your project can be found here.

9.2 SSH Keys

Accessing the Intrepid via SSH can be done with any SSH software package. Before logging in, you will need to generate an SSH public key and send it to support@alcf.anl.gov for verification and installation.

9.3 Cryptocard

This security token uses one-time passwords for controlled access to the BG/P login systems.

9.4 Connecting to a login node

When you gain access to Intrepid, you should receive a cryptocard and a temporary PIN. You must have a working cryptocard, know your PIN, and have your SSH key in place before you may login.

You can connect to Intrepid with the following command:

```
ssh yourusername@intrepid.alcf.anl.gov
```

You will be presented with a password prompt. The first part of your password is your PIN. Enter you PIN, press the Cryptocard button, and then enter the password your crypocard generates. If this is the first time you are logging in, you will be prompted to change your PIN.

9.5 Downloading and building Swift

The most recent versions of Swift can be found at http://swiftlang.org/downloads/index.php. Follow the instructions provided on that site to download and build Swift.

9.6 Adding Swift to your PATH

Once you have installed Swift, add the Swift binary to your PATH so you can easily run it from any directory.

In your home directory, edit the file ".bashrc".

If you have installed Swift via a source repository, add the following line at the bottom of .bashrc.

```
export PATH=$PATH:$HOME/cog/modules/swift/dist/swift-svn/bin
```

If you have installed Swift via a binary package, add this line:

```
export PATH=$PATH:$HOME/swift-<version>/bin
```

Replace <version> with the actual name of the swift directory in the example above.

9.7 What You Need To Know Before Running Swift

Note that on Intrepid, the compute nodes can not create or write to a /home filesystem. Consequently, in order for Swift to interface correctly from login node to the compute nodes, Swift must write all internal and intermediate files to /intrepid-fs0, which is writable by the compute nodes. In order to accomplish this, export the environment variable SWIFT_USERHOME as follows:

```
export SWIFT_USERHOME=/intrepid-fs0/users/`whoami`/scratch
```

Before you can create a Swift configuration file, there are some things you will need to know.

9.7.1 Swift Work Directory

The Swift work directory is a directory which Swift uses for processing work. This directory needs to be writable. Common options for this are:

```
/home/username/swiftwork
/home/username/work
/tmp/swift.work
```

9.7.2 Which project(s) are you a member of?

Intrepid requires that you are a member of a project. You can determine this by running the following command:

```
$ projects
HTCScienceApps
```

If you are not a member of a project, you must first request access to a project. More information on this process can be found at https://wiki.alcf.anl.gov/index.php/Discretionary_Allocations

9.7.3 Determine your Queue

Intrepid has several different queues you can submit jobs to depending on the type of work you will be doing. The command "qstat -q" will print the most up to date list of this information.

Table 1: Intrepid Queues

User Queue	Queue	Nodes	Time (hours)	User Maxrun	Project maxrun
prod-devel	prod-devel	64-512	0-1	5	20
prod	prod-short	512-4096	0-6	5	20

Table 1: (continued)

User Queue	Queue	Nodes	Time (hours)	User Maxrun	Project maxrun
prod	prod-long	512-4096	6-12	5	20
prod	prod-capability	4097-32768	0-12	2	20
prod	prod-24k	16385-24576	0-12	2	20
prod	prod-bigrun	32769-40960	0-12	2	20
prod	backfill	512-8192	0-6	5	10

9.8 Generating Configuration Files

Now that you know what queue to use, your project, and your work directory, it is time to set up Swift. Swift uses a configuration file called sites.xml to determine how it should run. There are two methods you can use for creating this file. You can manually edit the configuration file, or generate it with a utility called `gensites`.

9.8.1 Manually Editing sites.xml

Below is the template that is used by Swift's test suite for running on Intrepid.

Copy and paste this template, replace the values, and call it `sites.xml`.

The values to note here are the ones that are listed between underscores. In the example above, they are `_HOST_`, `_PROJECT_`, `_QUEUE_`, and `_WORK_`.

`HOST`
: The IP address on which Swift runs and to which workers must connect. To obtain this, run `ifconfig` and select the IP address that starts with `172`.

`PROJECT`
: The project to use.

`QUEUE`
: The queue to use.

`WORK`
: The Swift work directory.

9.9 Manually Editing tc.data

Below is the tc.data file used by Swift's test suite.

Copy these commands and save it as tc.data.

9.10 Catsn.swift

The swift script we will run is called catsn.swift. It simply cats a file and saves the result. This is a nice simple test to ensure jobs are running correctly. Create a file called data.txt which contains some simple input - a "hello world" will do the trick.

```
type file;

app (file o) cat (file i)
{
  cat @i stdout=@o;
}

string t = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
string char[] = @strsplit(t, "");

file out[]<simple_mapper; location=".", prefix="catsn.",suffix=".out">;
foreach j in [1:@toInt(@arg("n","10"))] {
  file data<"data.txt">;
  out[j] = cat(data);
}
```

9.11 Running Swift

Now that everything is in place, run Swift with the following command:

```
swift -sites.file sites.xml -tc.file tc.data catsn.swift -n=10
```

You should see several new files being created, called catsn.0001.out, catsn.0002.out, etc. Each of these files should contain the contents of what you placed into data.txt. If this happens, your job has run successfully!

9.12 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list. When submitting information, send your sites.xml file, your tc.data, and any Swift log files that were created during your attempt.

10 MCS Compute Servers (x86 workstations)

This sections describes how to use the general use compute servers for the MCS division of Argonne National Laboratory. Swift is available as a soft package on mcs machines. Add +swift to your .soft and run resoft command.

```
$ resoft
```

10.1 Create a coaster-service.conf

To begin, copy the text below and paste it into your Swift distribution's etc directory. Name the file coaster-service.conf.

```

```

10.2 Starting the Coaster Service

Change directories to the location you would like to run a Swift script and start the coaster service with this command:

```
start-coaster-service
```

This will create a configuration file that Swift needs called sites.xml.

Warning
Any existing sites.xml files in this directory will be overwritten. Be sure to make a copy of any custom configuration files you may have.

10.3 Run Swift

Next, run Swift. If you do not have a particular script in mind, you can test Swift by using a Swift script in the examples/ directory.

Run the following command to run the script:

```
swift -sites.file sites.xml -tc.file tc.data yourscript.swift
```

10.4 Stopping the Coaster Service

The coaster service will run indefinitely. The stop-coaster-service script will terminate the coaster service.

```
$ stop-coaster-service
```

This will kill the coaster service and kill the worker scripts on remote systems.

11 Midway (x86 cluster)

Midway is a cluster maintained by the Research Computing Center at the University of Chicago. Midway uses Slurm to handle job scheduling. For more details about Midway, and to request an account, visit http://rcc.uchicago.edu.

11.1 Connecting to a login node

Once you have access to Midway, connect to a Midway login node.

```
$ ssh userid@midway.rcc.uchicago.edu
```

11.2 Loading Swift

Swift is available on Midway as a module. To load the Swift, run:

```
$ module load swift
```

11.3 Example sites.xml

Below is an example that uses two of the queues available on Midway, sandyb and westmere. Be sure to adjust walltime, work directory, and other options as needed.

```
<config>
  <pool handle="midway-sandyb">
    <execution provider="coaster" jobmanager="local:slurm"/>
    <profile namespace="globus" key="jobsPerNode">16</profile>
    <profile namespace="globus" key="maxWalltime">00:05:00</profile>
    <profile namespace="globus" key="highOverAllocation">100</profile>
    <profile namespace="globus" key="lowOverAllocation">100</profile>
```

```
    <profile namespace="globus" key="queue">sandyb</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <filesystem provider="local"/>
    <workdirectory>/scratch/midway/{env.USER}/work</workdirectory>
  </pool>

  <pool handle="midway-westmere">
    <execution provider="coaster" jobmanager="local:slurm"/>
    <profile namespace="globus" key="jobsPerNode">12</profile>
    <profile namespace="globus" key="maxWalltime">00:05:00</profile>
    <profile namespace="globus" key="highOverAllocation">100</profile>
    <profile namespace="globus" key="lowOverAllocation">100</profile>
    <profile namespace="globus" key="queue">westmere</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <filesystem provider="local"/>
    <workdirectory>/scratch/midway/{env.USER}/work</workdirectory>
  </pool>
</config>
```

11.4 Example sites.xml for use with MPI

Below is an example sites.xml that is suitable for running with MPI. Jobtype must be set to single. The value you set for ppn will determine the number of cores/slots your application uses per node. The value of count will set the number of nodes to request. The example below requests 2 nodes with 12 slots per node.

```
<config>
  <pool handle="midway-westmere">
    <execution provider="coaster" jobmanager="local:slurm"/>
    <profile namespace="globus" key="jobsPerNode">1</profile>
    <profile namespace="globus" key="ppn">12</profile>
    <profile namespace="globus" key="maxWalltime">_WALLTIME_</profile>
    <profile namespace="globus" key="highOverAllocation">100</profile>
    <profile namespace="globus" key="lowOverAllocation">100</profile>
    <profile namespace="globus" key="queue">westmere</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <profile namespace="globus" key="jobtype">single</profile>
    <profile namespace="globus" key="count">2</profile>
    <filesystem provider="local"/>
    <workdirectory>/scratch/midway/{env.USER}/work</workdirectory>
  </pool>
</config>
```

11.5 Defining non-standard Slurm options

A Slurm submit script has many settings and options. Swift knows about many of the basic Slurm settings, like how to define a project or a queue, but it does not know about every setting. Swift provides a simple way to pass-thru your own settings into the Slurm submit script.

The general way to do this is:

```
<profile namespace="globus" key="slurm.setting">value</profile>
```

Here is one specific example. Slurm has the ability to notify users via email when a job is done. To make this happen, the Slurm submit script that Swift generates needs a line that contains "--mail-type=END". The following line will make it happen.

```
<profile namespace="globus" key="slurm.mail-type">END</profile>
```

Any valid Slurm setting can be set in a similar way (see the sbatch man page for a list of all settings).

11.6 Various tips for running MPI jobs

- You'll need to load an MPI module. Run "module load openmpi" to add to your path.
- The app that Swift runs should be a wrapper script that invokes your MPI application by running "mpiexec /path/to/yourMPIApp"

12 Persistent Coasters

Coasters is a protocol that Swift uses for scheduling jobs and transferring data. In most configurations, coasters are used automatically when you run Swift. With persistent coasters, the coaster server runs outside of Swift.

This section describes a utility called start-coaster-service that allows you to configure persistent coasters.

12.1 Example 1: Starting workers locally

Below is the simplest example, where the coaster service is started, and workers are launched locally on the same machine.

First, create a file called coaster-service.conf with the configuration below.

coaster-service.conf

```
export WORKER_MODE=local
export IPADDR=127.0.0.1
export JOBSPERNODE=1
export JOBTHROTTLE=0.0099
export WORK=$HOME/swiftwork
```

To start the coaster service and worker, run the command "start-coaster-service". Then run Swift with the newly generated sites.xml file.

```
$ start-coaster-service
Start-coaster-service...
Configuration: coaster-service.conf
Service address: 127.0.0.1
Starting coaster-service
Service port: 51099
Local port: 41764
Generating sites.xml
Starting worker on local machine

$ swift -sites.file sites.xml -tc.file tc.data hostsnsleep.swift
Swift trunk swift-r7153 cog-r3810
RunID: 20131014-1807-q6h89eq3
Progress:  time: Mon, 14 Oct 2013 18:07:13 +0000
Passive queue processor initialized. Callback URI is http://128.135.112.73:41764
Progress:  time: Mon, 14 Oct 2013 18:07:14 +0000  Active:1
Final status: Mon, 14 Oct 2013 18:07:15 +0000  Finished successfully:1
```

You can then run swift multiple times using the same coaster service. When you are finished and would like to shut down the coaster, run stop-coaster-service.

```
$ stop-coaster-service
Stop-coaster-service...
Configuration: coaster-service.conf
Ending coaster processes..
Killing process 8579
Done
```

Note
When you define your apps/tc file, use the site name "persistent-coasters".

12.2 Example 2: Starting workers remotely via SSH

The start-coaster-service script can start workers on multiple remote machines. To do this, there are two main settings you need to define in your coaster-service.conf. The first is to set WORKER_MODE=ssh, and the second is set WORKER_HOSTS to the list of machines where workers should be started.

coaster-service.conf

```
export WORKER_MODE=ssh
export WORKER_USERNAME=yourusername
export WORKER_HOSTS="host1.example.edu host2.example.edu"
export WORKER_LOCATION="/homes/davidk/logs"
export IPADDR=swift.rcc.uchicago.edu
export JOBSPERNODE=1
export JOBTHROTTLE=0.0099
export WORK=/homes/davidk/swiftwork
```

If there is no shared filesystem available between the remote machines and the local machine, you will need to enable coaster provider staging to transport files for you. Below is an example Swift configuration file to enable it:

cf

```
wrapperlog.always.transfer=false
sitedir.keep=false
execution.retries=0
lazy.errors=false
status.mode=provider
use.provider.staging=true
provider.staging.pin.swiftfiles=false
use.wrapper.staging=false
```

Run start-coaster service to start coaster and workers. When you run Swift, reference the cf file to enable provider staging.

```
$ start-coaster-service
Start-coaster-service...
Configuration: coaster-service.conf
Service address: swift.rcc.uchicago.edu
Starting coaster-service
Service port: 41714
Local port: 41685
Generating sites.xml
Starting worker on host1.example.edu
Starting worker on host2.example.edu

$ swift -sites.file sites.xml -tc.file tc.data -config cf hostsnsleep.swift
Swift trunk swift-r7153 cog-r3810
RunID: 20131014-1844-7flhik67
Progress:  time: Mon, 14 Oct 2013 18:44:43 +0000
Passive queue processor initialized. Callback URI is http://128.135.112.73:41685
Progress:  time: Mon, 14 Oct 2013 18:44:44 +0000  Selecting site:4 Finished successfully:4
Final status: Mon, 14 Oct 2013 18:44:45 +0000  Finished successfully:10
```

Note
This requires that you are able to connect to the remote systems without being prompted for a password/passphrase. This is usually done with SSH keys. Please refer to SSH documentation for more info.

12.3 Example 3: Starting workers remotely via SSH, with multihop

This example is for a situation where you want to start a worker on nodes that you can't connect to directly. If you have to first connect to a login/gateway machine before you can ssh to your worker machine, this configuration is for you.

The coaster-service.conf and cf files are the same as in Example 2.

Assume that node.host.edu is the machine where you want to start your worker, and that gateway.host.edu is the machine where you must log into first. Add the following to your $HOME/.ssh/config file:

```
Host node.host.edu
   Hostname node.host.edu
   ProxyCommand ssh -A username@gateway.host.edu nc %h %p 2> /dev/null
   ForwardAgent yes
   User username
```

This will allow you to SSH directly to node.host.edu. You can now add node.host.edu to WORKER_HOSTS.

12.4 Example 4: Starting workers remotely via SSH, with tunneling

The coaster workers need to be able to make a connection back to the coaster service. If you are running Swift on a machine behind a firewall where the workers cannot connect, you can use SSH reverse tunneling to allow this connection to happen.

To enable this, add the following line to your coaster-service.conf:

```
export SSH_TUNNELING=yes
```

12.5 Example 5: Starting workers remotely via SSH, hostnames in a file

The variable WORKER_HOSTS defines the list of hostnames where workers will be started. To set this to be the contents of a file, you can set WORKER_HOSTS as follows:

coaster-service.conf

```
export WORKER_HOSTS=$( cat /path/to/hostlist.txt )
```

12.6 Example 6: Starting workers via a scheduler

To start workers via some other script, such as a scheduler submit script, export WORKER_MODE=scheduler. Once the coaster service has been initialized, start-coaster-service will run whatever user defined command is defined in $SCHEDULER_COMMAND.

The contents of SCHEDULER_COMMAND will vary greatly based on your needs and the system you are running on. However, all SCHEDULER_COMMANDs will need to run the same command exactly once on each worker node:

```
$WORKER $WORKERURL logname $WORKER_LOG_DIR
```

Here is an example that runs on a campus cluster using the Slurm scheduler: .coaster-service.conf

```
export WORKER_MODE=scheduler
export WORKER_LOG_DIR=/scratch/midway/$USER
export IPADDR=10.50.181.1
export JOBSPERNODE=1
export JOBTHROTTLE=0.0099
export WORK=$HOME/swiftwork
export SCHEDULER_COMMAND="sbatch start-workers.submit"
```

The SCHEDULER_COMMAND in this case submits a Slurm job script and starts the workers via the following commands:

```
#!/bin/bash

#SBATCH --job-name=start-workers
#SBATCH --output=start-workers.stdout
#SBATCH --error=start-workers.stderr
#SBATCH --nodes=1
#SBATCH --partition=westmere
#SBATCH --time=00:10:00
#SBATCH --ntasks-per-node=12
#SBATCH --exclusive

$WORKER $WORKERURL logname $WORKER_LOG_DIR
```

12.7 List of all coaster-service.conf settings

Below is a list of all settings that start-coaster-service knows about, along with a brief description of what it does.

The settings are defined in terms of bash variables. Below is an example of the format used

```
export WORKER_LOGGING_LEVEL=DEBUG
```

Below is a list of the options that coaster-service.conf recognizes and what they do.

12.7.1 IPADDR

Defines IP address where the coaster-service is running. Workers need to know the IP address where to connect back to. Example: export IPADDR=192.168.2.12

12.7.2 LOCAL_PORT

Define a static local port number. If undefined, this is generated randomly. Example: export LOCAL_PORT=50100

12.7.3 LOG

LOG set the name of the local log file to be generated. This log file is the standard output and standard error output of the coaster-service and other commands that start-coaster-service runs. This file can get large at times. To disable, set "export LOG=/dev/null". Default value: start-coaster-service.log

12.7.4 SCHEDULER_COMMAND

In schedule mode, this defines the command to run via start-coaster-service that will start workers via the scheduler. Example: export SCHEDULER_COMMAND="qsub start-workers.submit".

12.7.5 SERVICE_PORT

Sets the coaster service port number. If undefined, this is generated randomly. Example: Export SERVICE_PORT=50200

12.7.6 SSH_TUNNELING

When the machine you are running Swift on is behind a firewall that is blocking workers from connecting back to it, add "export SSH_TUNNELING=yes". This will set up a reverse tunnel to allow incoming connections. Default value: no.

12.7.7 WORKER_HOSTS

WORKER_HOSTS should contain the list of hostnames that start-coaster-service will connect to start workers. This is only used when WORKER_MODE is ssh. Example: export WORKER_HOST="host1 host2 host3".

12.7.8 WORKER_LOCATION

In ssh mode, defines the directory on remote systems where the worker script will be copied to. Example: export WORKER_LOCATION

12.7.9 WORKER_LOG_DIR

In ssh mode, defines the directory on the remote systems where worker logs will go. Example: export WORKER_LOG_DIR=/home/john

12.7.10 WORKER_LOGGING_LEVEL

Defines the logging level of the worker script. Values can be "TRACE", "DEBUG", "INFO ", "WARN ", or "ERROR". Example: export WORKER_LOGGING_LEVEL=NONE.

12.7.11 WORKER_USERNAME

In ssh mode, defines the username to use when connecting to each host defined in WORKER_HOSTS.

13 SSH

This section describes how to use the SSH provider to connect to remote sites and to handle data transfer.

13.1 Generate a unique SSH key

It is recommended that you create a new SSH key exclusively for this purpose. In order to avoid being prompted for passwords/passphrases, your SSH passphrase will be stored in a read protected file. Run this command on the machine where you will be running Swift:

```
ssh-keygen -t dsa -f $HOME/.ssh/id_dsa-swift
```

You will be prompted to create a passphrase. This will create two files: $HOME/.ssh/id_dsa-swift and $HOME/.ssh/id_dsa-swift.pub.

13.2 Add your public key to the remote host

On the remote host where you will be running, edit or create the file $HOME/.ssh/authorized_keys. Paste in the contents of the newly created $HOME/.ssh/id_dsa-swift.pub from the previous step to the end of the file.

13.3 Verify your new key works

From the host where you will be running Swift, run the following command to verify your keys are working:

```
$ ssh -o IdentitiesOnly=true -i $HOME/.ssh/id_dsa-swift user@login.remotehost.edu
```

You should be prompted for your new passphrase and be able to connect.

13.4 Create auth.defaults

Create a file called $HOME/.ssh/auth.defaults on the host where you are running Swift. Use the following commands to create this file:

```
$ touch $HOME/.ssh/auth.defaults
$ chmod 600 $HOME/.ssh/auth.defaults
```

Next, edit $HOME/.ssh/auth.defaults and add the following lines:

```
login.remotehost.edu.type=key
login.remotehost.edu.username=your_remote_username
login.remotehost.edu.key=/your/home/.ssh/id_dsa-swift
login.remotehost.edu.passphrase=your_passphrase
```

Replace login.remotehost.edu with the hostname you want to use, replace the values for "your_remote_username", "your_passphrase", and set the correct path of private key you generated.

13.5 Create a sites.xml file

Here is an example sites.xml file that will allow you to connect and transfer data to a remote host:

```
<config>
   <pool handle="remotehost">
      <execution provider="coaster" jobmanager="ssh:local" url="login.remotehost.edu"/>
      <filesystem provider="ssh" url="login.remotehost.edu"/>
      <profile namespace="karajan" key="jobThrottle">0</profile>
      <profile namespace="karajan" key="initialScore">10000</profile>
      <workdirectory>/path/to/remote/workdirectory</workdirectory>
   </pool>
</config>
```

Note
This example will run work directly on login.remotehost.edu. In many cases you will not want to do this. You'll like want to modify your sites.xml to use a remote scheduler, by setting jobmanager to ssh:pbs or ssh:slurm, for example. This usually requires also setting things like queues and walltimes. This example is provided for simplicity and testing.

13.6 Setting your properties

Since you want to data transfer via ssh, you'll want to verify that you're not using any other file transfer mechanisms. Make sure you have the following swift properties defined in your configuration file:

```
use.provider.staging=false
use.wrapper.staging=false
```

14 SSH-CL

This section describes how to use the SSH command line provider (ssh-cl) to connect to remote sites.

14.1 Verify you can connect to the remote site

The first step of this process is to verify that you can connect to a remote site without being prompted for a password or passphrase.

```
$ ssh my.site.com
```

Typically to make this work you will need to add your SSH public key to the $HOME/.ssh/authorized_keys file on the remote system.

This SSH connection must work without specifying a username on the command line. If your username differs on your local machine and the remote machine, you will need to add an entry like this to your local $HOME/.ssh/config:

```
Host my.site.com
  Hostname my.site.com
  User myusername
```

14.2 Create a sites.xml file

Once you have verified that you can connect without prompt to the remote machine, you will need to create a sites.xml file that contains the host information. The example below will assume there is no scheduler on the remote system - it simply connects to the remote machine and runs work there.

```
<config>
  <pool handle="mysite">
    <execution provider="coaster" jobmanager="ssh-cl:local" url="my.site.com"/>
    <profile namespace="globus" key="jobsPerNode">1</profile>
    <profile namespace="globus" key="lowOverAllocation">100</profile>
    <profile namespace="globus" key="highOverAllocation">100</profile>
    <profile namespace="karajan" key="jobThrottle">1</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <workdirectory>/home/username/work</workdirectory>
  </pool>
</config>
```

Note
This requires that the remote site can connect back to the machine where you are running Swift. If a firewall is configured to block incoming connections, this will not work correctly.

14.3 Enabling coaster provider staging

If there is a shared filesystem between the two machines, you can set this as your work directory and skip this step. Otherwise, you will need to enable coaster provider staging.

To do this, add the following line to your "cf" file:

```
use.provider.staging=true
```

To run swift, then:

```
swift -sites.file sites.xml -tc.file tc.data -config cf script.swift
```

15 Stampede (x86 cluster)

Stampede is a cluster managed by the Texas Advanced Computing Center (TACC). It is a part of the XSEDE project. For more information about how to request an account, a project, how to log in and manage SSH keys, please see the More information about the system can be found in the Stampede User Guide.

15.1 Downloading and building Swift

The most recent versions of Swift can be found at the Swift downloads page. Follow the instructions provided on that site to download and build Swift.

15.2 Overview of How to Run

You will need to do the following steps in order to run.

1. Connect to a system that has the Globus myproxy-init command. This will be the system where Swift runs and from where Swift submits jobs to Stampede.
2. Obtain a grid certificate.
3. Run Swift with configuration files that define how to start remote jobs to Stampede via gram.

15.3 Verify System Requirements and Environment

The system where you run Swift needs to have the myproxy-init tool installed. Ideally it should also have globus-job-run for testing purposes.

Swift uses two environment variables in order for remote job execution to work. They are $GLOBUS_HOSTNAME and $GLOBUS_TCP_PORT_RANGE.

GLOBUS_HOSTNAME should contain the full hostname of the system where you are running. It may also contain the IP address of the machine.

GLOBUS_TCP_PORT_RANGE defines a range of ports to which a remote system may connect. You will likely need this defined if you are behind a firewall with somewhat restricted access.

15.4 Obtain a Grid Certificate

Once you have verified you have everything you need on the submit host where you are running, you can obtain an XSEDE grid certificate with following command:

```
$ myproxy-logon -l username -s myproxy.teragrid.org
```

15.5 Create sites.xml file

You may use the following example as a guide to run on Stampede. You will likely need to make a few modifications, as described below.

```
<config>
  <pool handle="stampede">
    <execution provider="coaster" jobmanager="gt2:gt2:slurm" url="login5.stampede.tacc. ↩
        utexas.edu:2119/jobmanager-slurm"/>
    <filesystem provider="gsiftp" url="gsiftp://gridftp.stampede.tacc.utexas.edu:2811"/>
    <profile namespace="globus"  key="jobsPerNode">16</profile>
    <profile namespace="globus"  key="ppn">16</profile>
    <profile namespace="globus"  key="maxTime">3600</profile>
    <profile namespace="globus"  key="maxwalltime">00:05:00</profile>
    <profile namespace="globus"  key="lowOverallocation">100</profile>
    <profile namespace="globus"  key="highOverallocation">100</profile>
    <profile namespace="globus"  key="queue">normal</profile>
    <profile namespace="globus"  key="nodeGranularity">1</profile>
    <profile namespace="globus"  key="maxNodes">1</profile>
    <profile namespace="globus"  key="project">yourproject</profile>
```

```
    <profile namespace="karajan" key="jobThrottle">.3199</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <workdirectory>/scratch/01503/yourusername</workdirectory>
  </pool>
</config>
```

You will need to modify the XSEDE project name to the match the name that has been allocated to you. In most cases you'll want to set the work directory to your Stampede scratch directory. This is defined, on Stampede, in the environment variable $SCRATCH.

15.6 Running Swift

You may now run your Swift script exactly as you have before.

```
$ swift -sites.file sites.xml -tc.file tc -config cf myscript.swift
```

15.7 Debugging

If you are having problems getting this working correctly, there are a few places where you may look to help you debug. Since this configuration is slightly more complicated, there are several log files produced.

1. The standard swift log, created in your current working directory on the machine where you are running from. This will be named something along the lines of myscript-<datestring>.log.
2. The bootstrap log. These are located on Stampede in your home directory and have the name coaster-bootstrap-<datestring>.log.
3. The coaster log. This is located on Stampede in your $HOME/.globus/coasters/coasters.log.
4. The gram log. This is located on Stamped in $HOME/gram-<datestring>.log.

For help in getting this configuration working, feel free to contact the Swift support team at swift-support@ci.uchicago.edu.

16 UC3 / OSGConnect (x86 cluster)

Please refer to the Swift Tutorial for OSGConnect.