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, please 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://www.ci.uchicago.edu/swift/wwwdev/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 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. 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
```

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

step 1. Load the Swift module on Beagle as follows: `module load swift`

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 definitio */
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 userguide here: http://www.ci.uchicago.edu/swift/wwwdev/-guides/release-0.93/userguide/userguide.html#_swift_configuration_properties.

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
```

Note: Running from sandbox node or requesting 30 minutes walltime for upto 3 nodes will get fast prioritized execution. Suitable for small tests.

3.3 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>
```

3.4 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
Swift version swift-0.93RC5 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 Blues (x86 cluster)

Blues is a 310-node computing cluster for the Argonne National Laboratory community. This section will walk you through running a simple Swift script on Blues.

4.1 Requesting Access

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

4.2 Projects

In order to run a job on a Blues 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.

4.3 SSH Keys

Before accessing Blues, be sure to have your SSH keys configured correctly. SSH keys are required to access Blues. You should see information about this when you request your account. Email support@lcrc.anl.gov for additional help.

4.4 Connecting to a login node

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

```
ssh yourusername@blues.lcrc.anl.gov
```

4.5 Creating sites.xml

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 User's Guide.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://www.ci.uchicago.edu/swift/SwiftSites">
    <pool handle="blues">
        <execution jobmanager="local:pbs" provider="coaster"/>
        <filesystem provider="local" url="none" />
        <profile namespace="globus" key="maxtime">3600</profile>
        <profile namespace="globus" key="jobsPerNode">16</profile>
        <profile namespace="globus" key="ppn">16</profile>
        <profile namespace="globus" key="queue">shared</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">1.00</profile>
        <profile namespace="karajan" key="initialScore">10000</profile>
        <workdirectory>/home/{env.USER}/swiftwork</workdirectory>
    </pool>
</config>
```

4.6 Creating tc.data

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

Paste the following example into a file named tc.data

```
blues hostname /bin/hostname
```

4.7 Create a Swift Script

The following script runs /bin/hostname utility 100 times and writes the outputs. Paste the following script into a file called hostname.swift.

```
type file;

app (file o) hostname ()
{
   hostname stdout=@o;
}

foreach i in [1:100] {
   file hostname_log <single_file_mapper; file=@strcat(i, ".hostname.log")>;
   hostname_log = hostname();
}
```

4.8 Run Swift

Finally, run the script

```
$ swift -sites.file sites.xml -tc.file tc.data hostname.swift
```

When this runs, you should see 100 new text files get created, named *.hostname.log. If you see these files, then you have succesfully run Swift on Blues!

4.9 Queues

Blues has multiple queues available. The command "qstat -q" will print a list.

To edit the queue, modify the following line in sites.xml

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

5 Fusion (x86 cluster)

Fusion is a 320-node computing cluster for the Argonne National Laboratory community. 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. 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 relies on various configuration files to determine how to run. 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 User's Guide.

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

```
<config>
   <pool handle="fusion">
      <execution jobmanager="local:pbs" provider="coaster"/>
      <filesystem provider="local" url="none" />
      <profile namespace="globus" key="maxtime">3600</profile>
      <profile namespace="globus" key="jobsPerNode">8</profile>
      <profile namespace="globus" key="slots">1</profile>
      <profile namespace="globus" key="nodeGranularity">2</profile>
      <profile namespace="globus" key="maxNodes">2</profile>
      <profile namespace="globus" key="queue">shared</profile>
      <profile namespace="karajan" key="jobThrottle">5.99</profile>
      <profile namespace="karajan" key="initialScore">10000</profile>
      <profile namespace="globus" key="HighOverAllocation">1000</profile>
      <profile namespace="globus" key="LowOverAllocation">1000</profile>
      <workdirectory>/homes/{env.USER}/swiftwork</workdirectory>
   </pool>
</config>
```

5.6 Creating tc.data

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

Paste the following example into a file named tc.data

```
fusion hostname /bin/hostname
```

5.7 Create a Swift Script

The following script runs /bin/hostname utility 100 times and writes the outputs. Paste the following script into a file called hostname.swift.

```
type file;

app (file o) hostname ()
{
   hostname stdout=@o;
}

foreach i in [1:100] {
   file hostname_log <single_file_mapper; file=@strcat(i, ".hostname.log")>;
   hostname_log = hostname();
}
```

5.8 Run Swift

Finally, run the script

```
$ swift -sites.file sites.xml -tc.file tc.data hostname.swift
```

When this runs, you should see 100 new text files get created, named *.hostname.log. 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.

6 Futuregrid Cloud

The NSF-funded FutureGrid cloud is administered by Indiana University. It offers a variety of resources via a multitude of interfaces. Currently, it offers cloud resources via three different interfaces: Eucalyptus, Nimbus (www.nimbusproject.org), and OpenStack (www.openstack.org). The total number of resources at FutureGrid is close to 5000 CPU cores and 220~TB of storage from more than six physical clusters. We use the resources offered by one such cluster via the Nebula middleware.

More information on futuregrid can be found at https://portal.futuregrid.org/.

6.1 Requesting Futuregrid Access

If you do not already have a futuregrid account, you can follow the instructions at https://portal.futuregrid.org/gettingstarted to get started. This page provides information on how to create an account, how to join a project, how to set up your SSH keys, and how to create a new project.

6.2 Downloading Swift VM Tools

A set of scripts based around cloudinitd are used to easily start virtual machines. To download, change to your home directory and run the following command:

```
$ svn co https://svn.ci.uchicago.edu/svn/vdl2/usertools/swift-vm-boot swift-vm-boot
```

6.3 Download your Credentials

Run the following commands to retrieve your credentials:

```
$ cd swift-vm-boot
$ scp yourusername@hotel.futuregrid.org:nimbus_creds.tar.gz .
$ tar xvfz nimbus_creds.tar.gz
```

When you extract your credential file, look at the file called hotel.conf. Near the bottom of this file will be two settings called vws.repository.s3id and vws.repository.s3key. Copy these values for the next step.

6.4 Configuring coaster-service.conf

To run on futuregrid, you will need a file called coaster-service.conf. This file contains many options to control how things run. Here is an example of a working coaster-service.conf on futuregrid.

```
# Where to copy worker.pl on the remote machine for sites.xml
export WORKER_LOCATION=/tmp

# How to launch workers: local, ssh, cobalt, or futuregrid
export WORKER_MODE=futuregrid
```

```
# Do all the worker nodes you're using have a shared filesystem? (yes/no)
export SHARED_FILESYSTEM=no

# Username to use on worker nodes
export WORKER_USERNAME=root

# Enable SSH tunneling? (yes/no)
export SSH_TUNNELING=yes

# Directory to keep log files, relative to working directory when launching start-coaster- ↩
    service
export LOG_DIR=logs

# Location of the swift-vm-boot scripts
export SWIFTVMBOOT_DIR=$HOME/swift-vm-boot

# Futuregrid settings
export FUTUREGRID_IAAS_ACCESS_KEY=XXXXXXXXXXXXXXXXXXXX
export FUTUREGRID_IAAS_SECRET_KEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
export FUTUREGRID_HOTEL_NODES=0
export FUTUREGRID_SIERRA_NODES=2
export FUTUREGRID_CPUS_PER_NODE=1

# Swift information for creating sites.xml
export WORK=/tmp
export JOBS_PER_NODE=$FUTUREGRID_CPUS_PER_NODE
export JOB_THROTTLE=$( echo "scale=5; ($JOBS_PER_NODE * $(($FUTUREGRID_HOTEL_NODES +  ↩
    $FUTUREGRID_SIERRA_NODES)))/100 - 0.00001"|bc )

# Application locations for tc.data
#app convert=/usr/bin/convert
```

Paste your credentials from the hotel.conf file into the FUTUREGRID_IAAS_ACCESS_KEY and FUTUREGRID_IAAS_SECRET_KE fields. Adjust the number of nodes you would like to allocate here by changing the values of FUTUREGRID_HOTEL_NODES and FUTUREGRID_SIERRA_NODES. Add a list of any applications you want to run in the format "#app myapp=/path/to/app".

6.5 Starting the Coaster Service Script

Now that everything is configured, change to the location of the coaster-service.conf file and run this command to start the coaster service:

```
$ start-coaster-service
```

This command will start the VMs, start the required processes on the worker nodes, and generate Swift configuration files for you to use. The configuration files will be generated in your current directory. These files are sites.xml, tc.data, and cf.

6.6 Running Swift

Now that you have all of your configuration files generated, run the following command:

```
$ swift -sites.file sites.xml -tc.file tc.data -config cf <yourscript.swift>
```

If you would like to create a custom tc file for repeated use, rename it to something other than tc.data to prevent it from being overwritten. The sites.xml however will need to be regenerated every time you start the coaster service. If you need to repeatedly modify some sites.xml options, you may edit the template in Swift's etc/sites/persistent-coasters. You may also create your own custom tc files with the hostname of persistent-coasters. More information about this can be found in the Swift userguide at http://www.ci.uchicago.edu/swift/guides/trunk/userguide/userguide.html.

6.7 Stopping the Coaster Service Script

To stop the coaster service, run the following command:

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

This will kill the coaster service, kill the worker scripts on remote systems and terminate the virtual machines that were created during start-coaster-service.

6.8 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at http://mail.ci.uchicago.edu/-mailman/listinfo/swift-user. When submitting information, please send your sites.xml file, your tc.data, and any error messages you run into.

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

8.1 Overview of running on grid sites

- Get a DOEGrids cert. Then register it in the **OSG Engage VO**, and/or map it using `gx-request` on TeraGrid sites.
- Run `GridSetup` to configure Swift to use the grid sites. This tests for correct operation and creates a "green list" of good sites.

- Prepare an installation package for the programs you want to run on grid sites via Swift, and install that package using `foreachsite`.
- Run `swift-workers` to start and maintain a pool of Swift workers on each site.
- Run Swift scripts that use the grid site resources.

Note
This revision only supports a single-entry sites file which uses provider staging and assumes that the necessary apps are locatable through the same tc entries (ie either absolute or PATH-relative paths) on all sites.

Note
This revision has been testing using the bin/grid code from trunk (which gets installed into trunk's bin/ directory, and the base swift code from the 0.93 branch. No other configurations have been tested at the moment. I intend to put this code in bin/grid in 0.93, as it should have no ill affects on other Swift usage.

8.2 Requesting Access

For OSG: Obtain a DOEGrids certificate and register the certificate in the OSG "Engage" VO following the procedure at:

https://twiki.grid.iu.edu/bin/view/Engagement/EngageNewUserGuide

FIXME: access to OSG wiki pages may request the user to present a certificate. Is this a problem from users without one? If so, make a copy of the page on the Swift web.

For TeraGrid: Obtain a DOEGrids certifcate using the OSG Engage instructions above. Ask a TeraGrid PI to add you to a TeraGrid project. Once you obtain a login and project access (via US Mail), use gx-request to add your certificate.

A detailed step-by-step instructions for requesting and installing your certificates in the browser and client machine are as follows:

Step1. Apply for a certificate: https://pki1.doegrids.org/ca/; use ANL as affiliation (registration authority) in the form.

Step2. When you receive your certificate via a link by mail, download and install it in your browser; we have tested it for firefox on linux and mac., and for Chrome on mac.

On firefox, as you click the link that you received in the mail, you will be prompted to install it by firefox: passphrase it and click install. Next take a backup of this certificate in the form of .p12. This is in Preferences > Advanced > Encryption > View Certificate > Your Certificate

Step3. Install DOE CA and ESnet root CA into your browser by clicking the top left links on this page: http://www.doegrids.org/

Step4. Go to the Engage VO registration point here: https://osg-engage.renci.org:8443/vomrs/Engage/vomrs from the same browser that has the above certs installed. Also see https://twiki.grid.iu.edu/bin/view/Engagement/EngageNewUserGuide for more details.

Step5. For installation of certificate on client machine, you need to have the certificate that is in the browser put in your client's ~/.globus directory from where you want to access OSG resources. The certificate has to be in the form of .pem files with a seperate .pem file for key and cert. For this conversion use the above backed up .p12 file as follows:

```
$ openssl pkcs12 -in your.p12 -out usercert.pem -nodes -clcerts -nokeys
$ openssl pkcs12 -in your.p12 -out userkey.pem -nodes -nocerts
```

Above commands are taken from: http://security.ncsa.illinois.edu/research/grid-howtos/usefulopenssl.html For more on openssl: http://www.openssl.org/docs/apps/openssl.html

Step6. Test it:

```
$ voms-proxy-init --voms Engage -hours 48
```

To run jobs using the procedure described here, you need to be logged in to a "submit host" on which you will run Swift and other grid-related utilities. A submit host is any host where the OSG client stack or equivalent tools are installed. Such hosts include the OSG Engage submit host (engage-submit3.renci.org), and the two Swift lab servers {bridled,communicado}.ci.uchicago.edu.

Obtain a login on engage-submit3.renci.org following instructions on the OSG URL above.

Obtain a CI login with access to the Swift lab servers by requesting "OSG Gridlab" access at:

http://accounts.ci.uchicago.edu

8.3 Connecting to a submit host

```
ssh yourusername@bridled.ci.uchicago.edu
ssh yourusername@communicado.ci.uchicago.edu
ssh yourusername@engage-submit.renci.org
```

8.4 Downloading and install Swift

The current version of Swift can be downloaded from http://www.ci.uchicago.edu/swift/downloads/index.php.

Fetch and untar the latest release. Then add the Swift bin/ directory to your PATH. For example:

```
cd $HOME
wget http://www.ci.uchicago.edu/swift/packages/swift-0.92.1.tar.gz
tar txf swift-0.92.1.tar.gz
export PATH=$PATH:$HOME/swift-0.92.1/bin
```

8.5 Set up OSG environment

Depending on your shell type, run:

```
source /opt/osg-<version>/setup.sh
or
source /opt/osg-<version>/setup.csh
```

Note
This above step is not required on engage-submit3 host.

8.6 Create a VOMS Grid proxy

```
$ voms-proxy-init -voms Engage -valid 72:00
```

8.7 Generating Configuration Files

```
cd $HOME
mkdir swiftgrid
cd swiftgrid
gen_gridsites
# Wait a few minutes to a few hours for Swift to validate grid sites
get_greensites >greensites
```

You can repeatedly try the get_greensites command, which simply concatenates all the site names that sucessfully resturned an output file from site tests.

8.8 Installing software on OSG sites

The command "foreachsite" will execute a local shell script passed to it as an argument, on each OSG site in the Engage VO ReSS site catalog. The user's installscript will execute on either the head node (as a GRAM "fork" job) or on a worker node, as controlled by the -resource option. Its syntax is:

```
$ ./foreachsite -help
./foreachsite [-resource fork|worker ] [-sites alt-sites-file] scriptname
$
```

To install your software, create a script similar to "myapp.sh", below, which (in this example) reads a tar file of a pre-compiled application "myapp-2.12" and executes a test script for that application. The test script should print some reconizable indication of its success or failure:

```
$ cat myinstall.sh
renice 2 -p $$
IDIR=OSG_APP/extenci/myname/myapp
mkdir -p $IDIR
cd $IDIR
wget http://my.url.edu/~mydir/myapp-2.12.tar.tgz
tar zxf myapp-2.12.tar.tgz
myapp-2.12/bin/test_myapp.sh
if [ $? = 0 ]; then
  echo INSTALL SUCCEEDED
else
  echo INSTALL FAILED
fi
$
$ foreachsite -resource fork myinstall.sh
$
$ # Wait a while here, then poll for successfully installed apps...
$
$ grep SUCCEEDED run.89/*/*.stdout
$
```

Following is an example of the installation script for DSSAT app on OSG:

```
#!/bin/bash

cd ${OSG_APP}/extenci/swift/

#pull
wget http://www.ci.uchicago.edu/~ketan/DSSAT.tgz

#extract
tar zxf DSSAT.tgz

# test
cd DSSAT/data

../DSSAT040.EXE A H1234567.MZX > std.OUT

if [ $? = 0 ]; then
  echo INSTALL SUCCEEDED
else
  echo INSTALL FAILED
fi
```

8.9 Starting a single coaster service

This single coaster service will service all grid sites:

```
start-grid-service --loglevel INFO --throttle 3.99 --jobspernode 1 \
                   >& start-grid-service.out
```

8.10 Starting workers on OSG sites through GRAM

Make sure that your "greensites" file is in the current working directory.

The swiftDemand file should be set to contain the number of workers you want to start across all OSG sites. Eventually this will be set dynamically by watching your Swift script execute. (Note that this number only includes jobs started by the swift-workers factory command, not by any workers added manually from the TeraGrid - see below.

The condor directory must be pre-created and will be used by Condor to return stdout and stderr files from the Condor jobs, which will execute the wrapper script "run-workers.sh".

Note
this script is current built manually, and wraps around and transports the worker.pl script. This needs to be automated.

```
echo 250 >swiftDemand mkdir -p condor

swift-workers greensites extenci \
              http://communicado.ci.uchicago.edu:$(cat service-0.wport) \
              >& swift-workers.out &
```

8.11 Starting workers on OSG sites through GlideinWMS

As an alternative to the above gram based *direct* worker submission, a GlideinWMS based worker submission can be made. The service start step would be same as above.

GlideinWMS is a Glidein Based WMS (Workload Management System) that works on top of condor.

As with the case of Gram based workers, the condor directory must be pre-created and will be used by Condor to return stdout and stderr files from the Condor jobs, which will execute the wrapper script "run-workers.sh".

```
run-gwms-workers http://communicado.ci.uchicago.edu:$(cat service-0.wport) \
100 >& gwms-workers.out &
```

Note
The run-gwms-workers is available from the bin/grid directory of swift trunk code. You will need to include it in your PATH.

In the above commandline, one can change the number of workers by changing the second commandline argument, which is 100 in this example.

8.12 Adding workers from TeraGrid sites

The job below can be used to submit jobs to TareGrid (Ranger only at the moment) to add more workers to the execution pool. The same requirements hold there as for OSG sites, namely, that the app tools listed in tc for the single execution site need to be locatable on the TeraGrid site(s).

```
start-ranger-service --nodes 1 --walltime 00:10:00 --project TG-DBS123456N \
                     --queue development --user tg12345 --startservice no \
                     >& start-ranger-service.out
```

Note
Change the project and user names to match your TeraGrid parameters.

8.13 Running Swift

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

```
export SWIFT_HEAP_MAX=6000m # Add this for very large scripts

swift -config cf.ps -tc.file tc -sites.file sites.grid-ps.xml \
      catsn.swift -n=10000 >& swift.out &
```

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 on the grid sites.

8.14 More Help

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

9 Intrepid (Blue Gene/P)

Intrepid is an IBM Blue Gene/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 at https://accounts.alcf.anl.gov/accounts/request.php. More information about this process and requesting allocations for your project can be found at http://www.alcf.anl.gov/support/-gettingstarted/index.php.

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://www.ci.uchicago.edu/swift/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

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
```

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
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 for running on PADS.

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 on PADS!

9.12 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at https://lists.ci.uchicago.edu/cgi-bin/mailman/listinfo/swift-user. When submitting information, please 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.

```
# Keep all interesting settings in one place
# User should modify this to fit environment

# Location of SWIFT. If empty, PATH is referenced
export SWIFT=

# Where to place/launch worker.pl on the remote machine
export WORKER_LOCATION=/home/$USER

# How to launch workers: local, ssh, futuregrid, or cobalt
export WORKER_MODE=ssh
export WORKER_USERNAME=$USER

# Worker host names for ssh
export WORKER_HOSTS="crush.mcs.anl.gov thwomp.mcs.anl.gov stomp.mcs.anl.gov crank.mcs.anl. ↩
    gov
steamroller.mcs.anl.gov grind.mcs.anl.gov churn.mcs.anl.gov trounce.mcs.anl.gov
thrash.mcs.anl.gov vanquish.mcs.anl.gov"

# Directory to keep log files, relative to working directory when launching start-coaster- ↩
    service
export LOG_DIR=logs

# Manually define ports. If not specified, ports will be automatically generated
export LOCAL_PORT=
export SERVICE_PORT=

# start-coaster-service tries to automatically detect IP address.
# Specify here if auto detection is not working correctly
export IPADDR=
```

```
# Gensites values
export WORK=/sandbox/$USER

# If SHARED_FILESYSTEM is set to no, provider staging will be turned on
export SHARED_FILESYSTEM=no

# If running outside of mcs network, set WORKER_RELAY_HOST below
# export WORKER_RELAY_HOST="login.mcs.anl.gov"

export WORKER_LOGGING_LEVEL=DEBUG
export WORKER_LOG_DIR="/home/$USER/logs"
export JOBSPERNODE="4"
export JOBTHROTTLE=100

# Set applications here
#app cat=/bin/cat
```

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>
```

12 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).

12.1 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"

13 Open Science Data Grid

To run Swift on Open Science Data Grid, connect to http://www.opensciencedatacloud.org, and select OSDC Console Login. Select Instances→Launch Instance, select Ubuntu, and click the down arrow next to the launch button. Select Launch Cluster, pick your options, and launch.

SSH to the sullivan login node (sullivan.opensciencedatacloud.org) and run the command "nova list". This will print information about your instances. You will see one instance with a name containing headnode. SSH to the headnode.

The headnode should have Swift and Java available. The easiest way to do this is to have a copy of those packages available in /glusterfs/users/$USER, and have those directories in your $PATH.

Set the environment variable $SWIFT_USERHOME to /glusterfs/users/<yourusername>.

Use the following sites.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://www.ci.uchicago.edu/swift/SwiftSites">
  <pool handle="osdc">
    <execution provider="coaster" jobmanager="local:pbs"/>
    <profile namespace="globus" key="jobsPerNode">1</profile>
    <profile namespace="globus" key="ppn">1</profile>
    <profile namespace="globus" key="queue">batch</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
```

```
    <profile namespace="globus" key="slots">10</profile>
    <profile namespace="globus" key="nodeGranularity">1</profile>
    <profile namespace="globus" key="maxNodes">1</profile>
    <filesystem provider="local"/>
    <workdirectory>/glusterfs/users/dkelly/swiftwork</workdirectory>
  </pool>
</config>
```

You will need to change workdirectory to reflect your username. Slots should be set to the maximum number of nodes you have available. jobsPerNode should be set to the number of cores available per node.

Run Swift as normal with a command like this:

```
$ swift -sites.file sites.xml -tc.file tc.data myscript.swift
```

14 SSH

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

14.1 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.

14.2 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.

14.3 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.

14.4 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.

14.5 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.

14.6 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
```

15 SSH-CL

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

15.1 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
```

15.2 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.

15.3 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
```

16 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.

16.1 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.

16.2 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.

16.3 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.

16.4 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
```

16.5 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.

16.6 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
```

16.7 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.

17 UC3 (x86 cluster)

17.1 Requesting Access

To request access to UC3, you must have a University of Chicago CNetID and be a meimber of the UC3 group. More information about UC3 can be found at https://wiki.uchicago.edu/display/uc3/UC3+Home or uc3-support@lists.uchicago.edu.

17.2 Connecting to a login node

To access the UC3 login node, you will use your CNetID and password.

```
ssh -l <cnetid> uc3-sub.uchicago.edu
```

17.3 Installing Swift

Swift should be available by default on the UC3 login nodes. You can verify this by running the following command:

```
swift -version
```

If for some reason Swift is not available, you can following the instructions at http://www.ci.uchicago.edu/swift/guides/release-0.93/quickstart/quickstart.html. Swift 0.94 or later is required to work with the condor provider on UC3.

17.4 Creating sites.xml

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 User's Guide.

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

```
<config>
  <pool handle="uc3">
    <execution provider="coaster" url="uc3-sub.uchicago.edu" jobmanager="local:condor"/>
    <profile namespace="karajan" key="jobThrottle">999.99</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <profile namespace="globus"  key="jobsPerNode">1</profile>
    <profile namespace="globus"  key="maxWalltime">3600</profile>
    <profile namespace="globus"  key="nodeGranularity">1</profile>
    <profile namespace="globus"  key="highOverAllocation">100</profile>
```

```
    <profile namespace="globus"  key="lowOverAllocation">100</profile>
    <profile namespace="globus"  key="slots">1000</profile>
    <profile namespace="globus"  key="maxNodes">1</profile>
    <profile namespace="globus"  key="condor.+AccountingGroup">"group_friends.{env.USER}"</ ↩
        profile>
    <profile namespace="globus"  key="jobType">nonshared</profile>
    <filesystem provider="local" url="none" />
    <workdirectory>.</workdirectory>
  </pool>
</config>
```

17.5 Creating tc.data

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

Paste the following example into a file named tc.data:

```
uc3 echo /bin/echo null null null
```

17.6 Create a configuration file

A swift configuration file enables and disables some settings in Swift. More information on what these settings do can be found in the Swift User's guide.

Paste the following lines into a file called cf:

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

17.7 Creating echo.swift

Now we need to create a swift script to test with. Let's use a simple application that calls /bin/echo.

```
type file;

app (file o) echo (string s) {
   echo s stdout=@o;
}

foreach i in [1:5] {
  file output_file <single_file_mapper; file=@strcat("output/output.", i, ".txt")>;
  output_file = echo( @strcat("This is test number ", i) );
}
```

17.8 Running Swift

Putting everything together now, run your Swift script with the following command:

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

If everything runs successfully, you will see 5 files get created in the output directory.

17.9 Controlling where jobs run

Swift will automatically generate condor scripts for you with the basic information about how to run. However, condor has hundreds of commands that let you customize how things work. If you need one of these advanced commands, you can add it to your sites.xml. The basic template for this is:

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

For example, let's assume that you want to control where your jobs run by adding a requirement. The condor command that will control the run is:

```
Requirements = UidDomain == "osg-gk.mwt2.org"
```

To have this generated by Swift, you will add a line to your sites.xml in the key/value style shown above.

```
<profile namespace="globus" key="condor.Requirements">UidDomain == "osg-gk.mwt2.org"</ ↩
    profile>
```

17.10 Installing Application Scripts on HDFS

Note
This section will only work if the application you want to use is an interpreted script (bash, python, perl, etc). HDFS does not have the ability to give programs an execution bit and run them. If the application you want to run on UC3 is a compiled executable, skip this section and read ahead.

Once your simple echo test is running, you'll want to start using your own applications. One way to go about doing this is by using the Hadoop filesystem. This filesystem is only available on the UC3 Seeder Cluster. In order to limit yourself to machines that can access this filesystem, add the following line to your sites.xml file:

```
<profile namespace="globus" key="condor.Requirements">UidDomain == "osg-gk.mwt2.org" &amp;& ↩
    amp; regexp("uc3-c*", Machine)</profile>
```

Now you can install your script somewhere under the directory /mnt/hadoop/users/<yourusername>. Here is an example with putting everything together.

sites.xml

```
<config>
  <pool handle="uc3">
    <execution provider="coaster" url="uc3-sub.uchicago.edu" jobmanager="local:condor"/>
    <profile namespace="karajan" key="jobThrottle">999.99</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <profile namespace="globus"  key="jobsPerNode">1</profile>
    <profile namespace="globus"  key="maxWalltime">3600</profile>
    <profile namespace="globus"  key="nodeGranularity">1</profile>
    <profile namespace="globus"  key="highOverAllocation">100</profile>
    <profile namespace="globus"  key="lowOverAllocation">100</profile>
    <profile namespace="globus"  key="slots">1000</profile>
    <profile namespace="globus"  key="maxNodes">1</profile>
    <profile namespace="globus"  key="condor.+AccountingGroup">"group_friends.{env.USER}"</ ↩
        profile>
    <profile namespace="globus"  key="jobType">nonshared</profile>
    <profile namespace="globus" key="condor.Requirements">UidDomain == "osg-gk.mwt2.org" & ↩
        amp;&amp; regexp("uc3-c*", Machine)</profile>
    <filesystem provider="local" url="none" />
    <workdirectory>.</workdirectory>
  </pool>
</config>
```

tc.data

```
uc3 bash /bin/bash null null null
```

myscript.swift

```
type file;

app (file o) myscript ()
{
   bash "/mnt/hadoop/users/<yourusername>/myscript.sh" stdout=@o;
}

file out[]<simple_mapper; location="outdir", prefix="myscript.",suffix=".out">;
int ntasks = @toInt(@arg("n","1"));

foreach n in [1:ntasks] {
   out[n] = myscript();
}
```

/mnt/hadoop/users/<yourusername>/myscript.sh

```
#!/bin/bash

echo This is my script
```

cf

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

Example run

```
$ swift -sites.file sites.xml -tc.file tc.data -config cf myscript.swift -n=10
Swift trunk swift-r6146 cog-r3544

RunID: 20130109-1657-tf01jpaa
Progress:  time: Wed, 09 Jan 2013 16:58:00 -0600
Progress:  time: Wed, 09 Jan 2013 16:58:30 -0600  Submitted:10
Progress:  time: Wed, 09 Jan 2013 16:59:00 -0600  Submitted:10
Progress:  time: Wed, 09 Jan 2013 16:59:12 -0600  Stage in:1  Submitted:9
Final status: Wed, 09 Jan 2013 16:59:12 -0600  Finished successfully:10
$ ls outdir/*
outdir/myscript.0001.out  outdir/myscript.0003.out  outdir/myscript.0005.out  outdir/ ↩
    myscript.0007.out  outdir/myscript.0009.out
outdir/myscript.0002.out  outdir/myscript.0004.out  outdir/myscript.0006.out  outdir/ ↩
    myscript.0008.out  outdir/myscript.0010.out
```

17.11 Staging in Applications with Coaster Provider Staging

If you want your application to be as portable as possible, you can use coaster provider staging to send your application(s) to a remote node. By removing the condor requirements in the previous section, you will have more cores available. Here is a simple script that stages in and executes a shell script.

```
type file;

app (file o) sleep (file script, int delay)
{
   # chmod +x script.sh ; ./script.sh delay
   bash "-c" @strcat("chmod +x ./", @script, " ; ./", @script, " ", delay) stdout=@o;
}

file sleep_script <"sleep.sh">;

foreach i in [1:5] {
  file o <single_file_mapper; file=@strcat("output/output.", i, ".txt")>;
  o = sleep(sleep_script, 10);
}
```

Mapping our script to a file and passing it as an argument to an app function causes the application to be staged in. The only thing we need on the worker node is /bin/bash.

Tip
If the program you are staging in is an executable, statically compiling it will increase the chances of a successful run.

Tip
If the application you are staging is more complex, with multiple files, add everything you need into just one compressed tar file, then extract on the worker node.