# SAGA Big Job

**ABSTRACT**

The document contains the BigJob architecture, Control structure and its deployment and execution on LONI.

## Table of Contents

## Introduction

The Pilot-Job abstraction has been shown to be an effective abstraction to address many requirements of scientific applications. Specifically, Pilot-Jobs support the decoupling of workload submission from resource assignment; this results in a flexible execution model, which in turn enables the distributed scale-out of applications on multiple and possibly heterogeneous resources. SAGA-based Pilot-Job, which supports a wide range of application types, and is usable over a broad range of infrastructures, i.e., it is general-purpose, extensible and interoperable. SAGA-based Pilot-Job is used for different application types and supports the concurrent usage across multiple heterogeneous distributed infrastructure, including concurrent usage across Clouds and traditional Grids/Clusters.

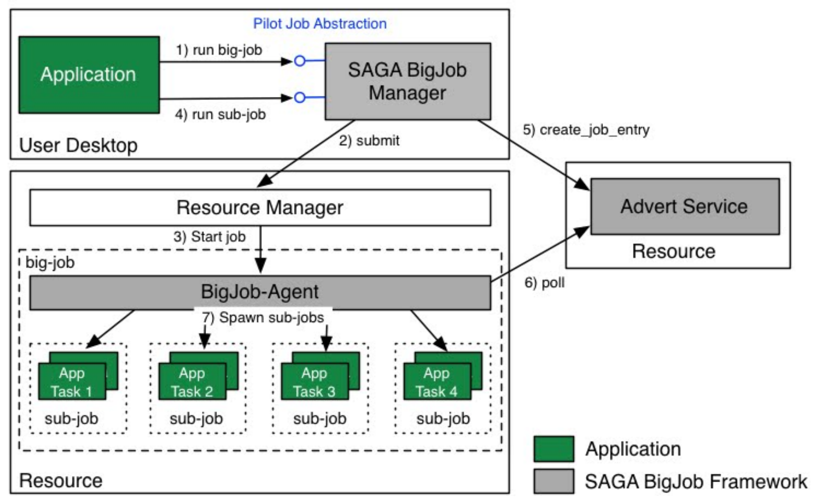
## BigJob - A SAGA-based Pilot Job Implementation

Condor GlideIn provides a very useful way to efficiently utilize distributed resources. It allows the execution of jobs without the necessity to queue each individual job. This concept is also referred to as pilot job. A pilot job is a job, which is started through the regular Grid resource manager, usually the Globus GRAM (Grid Resource Allocation Manager). The pilot job provides a container for many sub-jobs, i.e. applications submit these sub-jobs through the pilot job and not the resource manager. A major advantage of this approach is that the waiting time at the local resource manager, which usually significantly contributes the overall time-to-completion, is avoided.

The Simple API for Grid Applications (SAGA) is a high-level, easy-to-use API for accessing distributed resources. Unlike other common pilot job systems SAGA BigJob

(i) natively supports MPI job and
(ii) works on a variety of back-end systems, generally reflecting the advantage of using a SAGA-based approach. The following figure gives an overview of the SAGA BigJob architecture.

Before running regular jobs, an application must initialize a bigjob object. The BigJob Manager then queues a pilot job, which actually runs a BigJob Agent on the respective resource. For this agent a specified number of resources is requested. Subsequently, sub-jobs can be submitted through the BigJob Manager using the jobID of the BigJob as reference. The BigJob Manager ensures that the subjobs are launched onto the correct resource based upon the specified jobID using the right number of processes. Communication between the BigJob Agent and BigJob Manager is carried out using the SAGA advert service, a central key/value store. For each new job, an advert entry is created by the BigJob Manager. The agent periodically polls for new jobs. If a new job is found and resources are available, the job is dispatched, otherwise it is queued.



## Push vs. Pull

- In general, the way SAGA – BigJob has been used is by pushing all the sub-jobs in advance to different machines, before the BigJob is active.
- But if there is need, it can be used to "pull" the jobs after the BigJob is active.
- The "pull" option is useful if many machines/BigJobs are being used and when the sub-jobs need to be run in a particular order.

## Deployment & Implementation

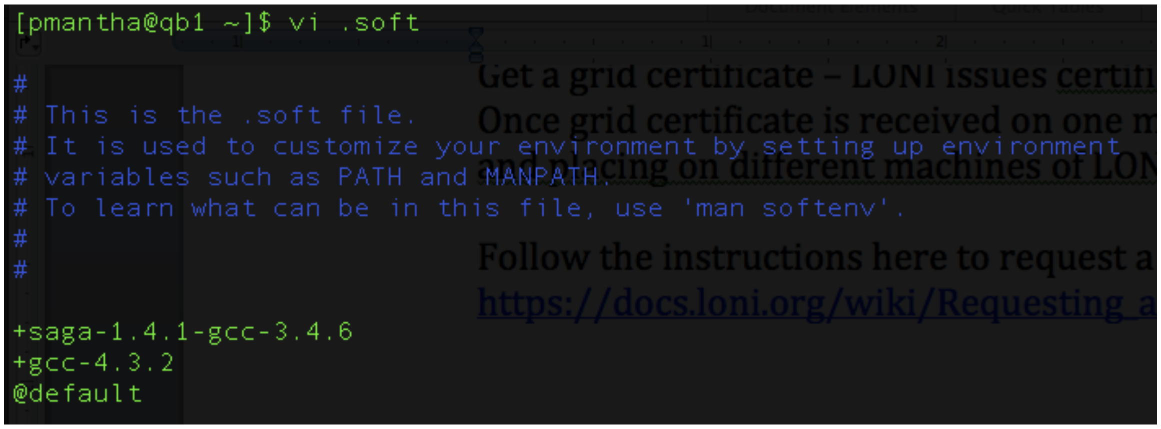
1. *BigJob uses GRAM to submit jobs which requires proxy.*

Get a grid certificate – LONI issues certificates which work on all LONI machines. Once grid certificate is received on one machine, the same can be used by copying and placing on different machines of LONI.

Follow the instructions here to request a certificate on QB/LONI:
https://docs.loni.org/wiki/Requesting_a_LONI_Grid_Certificate

2. *SAGA is already available on LONI.*

Please add saga-1.4.1-gcc-3.4.6 and gcc-4.3.2 as shown below to .soft file in the home directory. SAGA Python bindings will also be set automatically.



```
[pmantha@qb1 ~]$ vi .soft

#
# This is the .soft file.
# It is used to customize your environment by setting up environment
# variables such as PATH and MANPATH.
# To learn what can be in this file, use 'man softenv'.
#
#

+saga-1.4.1-gcc-3.4.6
+gcc-4.3.2
@default
```

do resoft after modifying the .soft file

```
$ resoft
```

3. *Make sure you are on the Head node to submit the job.*

4. *Execute* "`ssh localhost` ".

If it doesn't login and asks for password make it password less login using the following commands.

```
[pmantha@eric2 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pmantha/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pmantha/.ssh/id_rsa.
Your public key has been saved in /home/pmantha/.ssh/id_rsa.pub.
The key fingerprint is:
f6:aa:d6:df:c7:38:a9:e0:6b:f0:9a:02:73:a9:29:9b pmantha@eric2
[pmantha@eric2 ~]$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
[pmantha@eric2 ~]$ ssh localhost
Last login: Tue Jan 25 16:16:13 2011 from localhost.localdomain
```

5. *SAGA Advert Adaptor configuration Need to be changed.*

It is necessary to configure the SAGA adaptors. The adaptor that need to be configured is the advert adaptor.

$SAGA_LOCATION provides the path in which saga is installed.

Create a directory in $HOME & copy all the files from SAGA location to home directory(preferably). Make sure SAGA environment is set.

```
mkdir $HOME/saga_adaptors
cp -fr $SAGA_LOCATION/share/saga/* $HOME/saga_adaptors
```

The advert adaptor configuration has to be changed.

```
cd $HOME/saga_adaptors
```

Modify the content of saga_adaptor_default_advert.ini file as below.

```
[saga.adaptors.default_advert]
  name      = default_advert
# path      = $[saga.location]/lib

[saga.adaptors.default_advert.preferences]
  # adaptor configuration

  # specify what backend to use
  dbtype     = postgresql


[saga.adaptors.default_advert.preferences.sqlite3]
  # specify a file name where sqlite3 can store its data.
  # relative and absolute paths are both supported.  Relative paths are
  # relative to the applications cwd.

  dbconnect = ${HOME}/.saga_advert.db


[saga.adaptors.default_advert.preferences.postgresql]
  # The set of parameters used in the dbconnect string
  # for PostgreSQL is the same as accepted by the PQconnectdb
  # function from the libpq library (see here:
  # http://www.postgresql.org/docs/8.1/interactive/libpq.html#LIBPQ-CONNECT)

  dbconnect =
dbname=advertdb;host=advert.cct.lsu.edu;port=5432;user=SAGA;password=SAGA_client
```

create ".saga_ini" hidden file in $HOME directory and provide the path of saga_adaptors file location in that and the content of file should look like below code

```
[saga]
  # saga install root,  points to what is set in the environment, as
  # SAGA_LOCATION, or use the configure time prefix as fallback.
  location = ${SAGA_LOCATION:/var/tmp/saga-1.4.1_python2.6.4}

  # where to find adaptor ini files
  ini_path = ${HOME}/saga_adaptors/
```

6. *Download Bigjob*

```
svn co https://svn.cct.lsu.edu/repos/saga-projects/applications/bigjob/
```

7. *After you downloaded BigJob you should be able to use it right away with the default adaptors that come with SAGA. First, you must configure the framework. In particular, the agent requires some setup:*

- Configure agent launcher script which is used to set the necessary environment entries and to call the main agent program:
  ```
  cp bigjob_agent_launcher.sh.template bigjob_agent_launcher.sh
  ```
- review bigjob_agent_launcher.sh

- Configure basic agent settings:
  ```
  cp bigjob_agent.conf.template bigjob_agent.conf
  ```
- review bigjob_agent_launcher.conf

8. Having setup the agent, the included example ( `example-local.py` ) can be reviewed:

- Creation of a bigjob:

```
    bj = bigjob.bigjob("advert.cct.lsu.edu")  // Advert database location.

    bj.start_pilot_job("fork://localhost", # remote resource SAGA URL. To run job on
queenbee the URL look like gram://qb1.loni.org/jobmanager-pbs
                      "/home/<user>/.../bigjob_agent_launcher.sh", #Path to BigJob agent
script
                       "1",    # number of nodes requested
                       None,   # optional: queue ( workq, single, checkpt etc., )
                       None,   # optional: project/allocation
                       "/tmp", # optional: working directory
                       None,   # optional: path to Globus proxy
                       None    # optional: walltime in minutes
                      )
```

- Submit subjob:

```
# create job description
jd = saga.job.description()
jd.executable = "/bin/date"
jd.number_of_processes = "1"
jd.spmd_variation = "single" # mpi, openmp can be other options.
jd.arguments = [""]
jd.working_directory = os.getcwd()
jd.output = "stdout.txt"
jd.error = "stderr.txt"
```

- Submit subjob through bigjob

```
sj = bigjob.subjob(advert_host)
sj.submit_job(bj.pilot_url, jd)
```

The below table shows the different Job description attributes available in SAGA.

| Attribute Name | Data Type | Purpose |
|---|---|---|
| Executable | string | what executable to run |
| Arguments | list of strings | the job arguments |
| CandidateHosts | list of strings | where to run |
| SPMDVariation | string | MPI type (if any) |
| TotalCPUCount | Int | # CPUs, total |
| NumberOfProcesses | Int | # processes, total |
| ProcessesPerHost | Int | # processes per host |
| ThreadsPerProcess | Int | # threads per process |
| Environment | list of strings | environment vars |
| WorkingDirectory | string | working directory |
| Interactive | bool | usage type |

| | | |
|---|---|---|
| Input | string | stdin source file |
| Output | string | stdout log file |
| Error | string | stderr log file |
| FileTransfer | list of strings | file staging |
| Cleanup | bool | cleanup-after-run |
| JobStartTime | datetime | when to start job |
| TotalCPUTime | Int | how long will job run |
| TotalPhysicalMemory | Int | memory the job needs |
| CPUArchitecture | string | architecture to run on |
| OperatingSystemType | string | OS to run on |
| Queue | string | queue to submit to |
| JobContact | list of strings | notification contact |

9. *Initialize the grid proxy and then run the example. The same example can be modified to submit different jobs giving the job description as mentioned above.*

   ```
   $grid-proxy-init
   
   $python example-local.py.
   ```

10. *The agent log files is written to the. ./*`agent` *directory. You can find the output of the sub-jobs in the locations specified in the job description* `(./agent/stdout.txt and ./agent/stderr.txt).`

11. *Multiple bigjobs can be submitted. Please find the example example-manyjob.py. The bigjob parameter details should be mentioned in resource_list list variable. The list is iterated and each bigjob is submitted with the corresponding details.*

## References

*http://randomlydistributed.blogspot.com/2009/10/bigjob-saga-based-pilot-job.html*
*http://www.cct.lsu.edu/~sjha/select_publications/interop_pilot_job.pdf*
*http://www.cct.lsu.edu/~sjha/select_publications/coupled_simulations_framework.pdf*