Evaluation of the components of a distributed workload managers on new platforms

Rami Aboushadi – Jackson State University – MS – OMNI Internship

FERMILAB-POSTER-22-185-STUDENT

Introductions: GlideinWMS

The purpose of the GlideinWMS is to provide a simple way to access the Grid resources. GlideinWMS is a Glidein Based WMS (Workload Management System) that works on top of HTCondor. Glideins are like placeholders, a mechanism by which one or more remote resources temporarily join a local HTCondor pool. The HTCondor system is used for scheduling and job control.

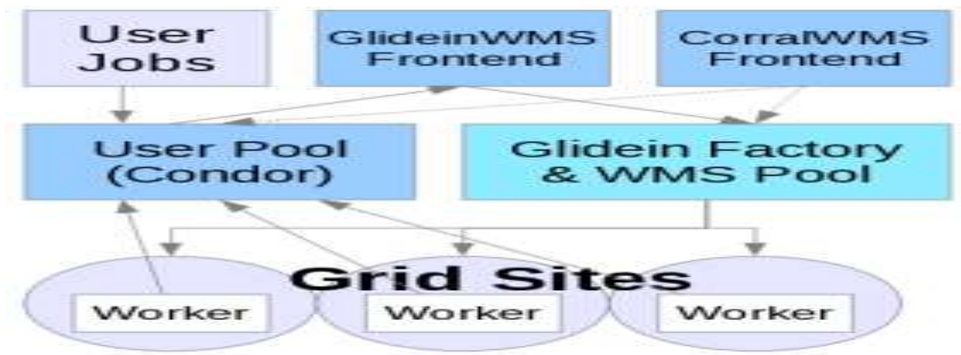

Figure 1: GlideinWMS Structures

Source: https://glideinwms.fnal.gov/doc.prd/frontend/index.html

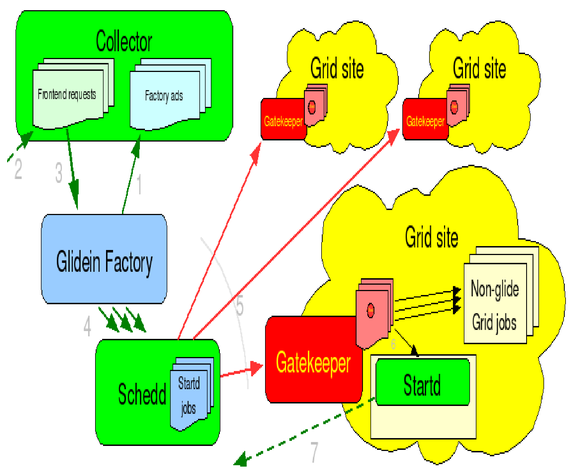

Figure 2: The main task of the VO Frontend is to look for user jobs and ask the Glidein Factories to provide glideins, if needed.

Source: https://glideinwms.fnal.gov/doc.prd/frontend/index.html

The GlideinWMS Factory and Frontend and HEPCloud's Decision Engine are coded in Python.

To run on well supported platforms and to take advantage of new features, over time it is necessary to support new platforms, new Operating Systems and new Python versions.

Python 3.6 is the supported Python on EL7 and was EOL in December 2021.

GlideinWMS and HEPCloud decided to move to Python 3.9 and to EL8, which provides better support for Python versions newer than 3.6.

Also Worker Nodes, the computers where the scientific software is run, evolve over time and it is useful to know the effectiveness of new architectures in running scientific applications.

The goal of this project is to test different part of the systems, adapt GlideinWMS and write recommendations for the use of new platforms.

```
PYUNITTEST_FILES_CHECKED_COUNT=42
PYUNITTEST_ERROR_FILES="./test_lib_fork.py"
PYUNITTEST_ERROR_FILES_COUNT=1
PYUNITTEST_ERROR_COUNT=0
BRANCH=LOCAL
PYUNITTEST_TIME=723
PYUNITTEST=success
```

Figure3 : Example of some files that failed unit test. Debugging and troubleshooting

Methods:

To evaluate the GlideinWMS and HEPCloud components on REHL8 I followed a series of procedures:

- Setup of AlmaLinux 8 VMs with Python 3.9
- Running the continuous integration tests on RHEL8 with Python 3.9. And troubleshooting and fixing some failing unit tests
- Installation, testing and adaptation of GlideinWMS Frontend and Factory on RHEL8 with Python 3.9
- Installation, testing and adaptation of Decision Engine on RHEL8 with Python 3.9
- Benchmarking of new Hyper-threaded worker nodes with different loads from a real DUNE workflow

Figure4 : Results of pyunittest after troubleshooting and debugging with zero file error.

```
PYUNITTEST_FILES_CHECKED_COUNT=42
PYUNITTEST_ERROR_FILES=""
PYUNITTEST_ERROR_FILES_COUNT=0
PYUNITTEST_ERROR_COUNT=0
BRANCH=LOCAL
PYUNITTEST_TIME=175
PYUNITTEST=success
----------------
runtest.sh INFO: Tested branch LOCAL (0): success
RESULT_pyunittest_LOCAL=0:success
runtest.sh INFO: Complete with local files (ec:0)
RESULT_pyunittest=0
runtest.sh INFO: Logs are in /home/rami_l/work39/output
runtest.sh INFO: Tests Complete - Success
```

Conclusions:

I learned ...

I've got more experienced ...

I was able to troubleshoot the unit tests and fix some problems with deprecated Python constructs and with a different handling of multi processing. We decided to skip the test of epoll and remove it from the possible alternatives to handle multi processing.

I was able to install all the systems on Alma Linux 8 (RHEL8 compatible) with Python 3.9: GlideinWMs Frontend, VO Pool, and Factory, and HEPCloud's Decision Engine

The DUNE workflow was able to scale on the Worker node using all cores

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy