Workload management with glideinWMS

Igor Sfiligoi, Fermilab, Batavia, IL, USA

v0.6.1 – Mar 12th

1.Introduction

GlideinWMS is a general purpose Workload Management System (WMS) developed by CMS, based on previous experience in CDF. It relies heavily on Condor software, with additional glideinWMS specific code.

GlideinWMS has been deployed in production by CMS at Fermilab and has been extensively used for single-user analysis by the CMS San Diego group. It is also currently being used in prototype deployments by MINOS and CDF at Fermilab.

This document gives a general overview of glideinWMS, and describes the current and envisioned deployment scenarios for CMS. The security aspects of both software itself and the (anticipated) deployed configuration are emphasized.

2.Structural overview

GlideinWMS is composed of six logical pieces, as shown in Fig 1.:

- a Condor central manager,
- one or more Condor submit machines,
- a glideinWMS collector,
- one or more VO frontends,
- one or more glidein factories, and finally
- the glideins.

The elements composing a glideinWMS installation can be grouped in two classes:

- The Condor pool elements, represented in green, handle the user jobs.
- The glidein handling elements, represented in cyan, regulate the amount of glideins sent to the Grid sites.

The two classses are described in separate sections below.

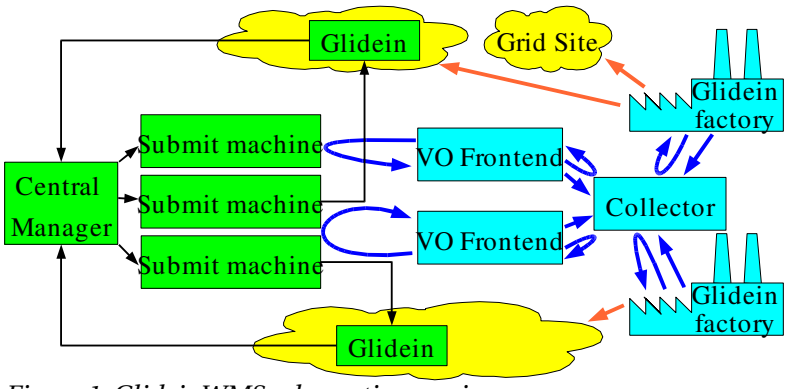

Figure 1: GlideinWMS schematic overview

The Condor pool

The Condor pool is the core of the glideinWMS, as it handles the user jobs and is effectively the only part that the users know about.

A glideinWMS Condor pool is effectively a regular Condor pool, where the execution daemon, called **condor_startd**, has been submitted as a Grid job instead of being pre-installed by the system administrators. This little difference has however a strong security impact; since the **condor_startd** is running as a regular user, it cannot change the UNIX identity of the user job on its own. However, **condor_startd** can use glexec for this task.

A Condor pool is defined by its central manager, in particular the **condor_collector** daemon running there; it collects the information about all the other daemons in the system. See Fig 2.1. All network communications between Condor daemons are provided over a secure channel by CEDAR, a Condor specific mechanism that provides mutual authentication, integrity, and confidentiality. Several authentication mechanisms are supported, but most glideinWMS Condor pools will use GSI authentication and will integrity check all the messages.

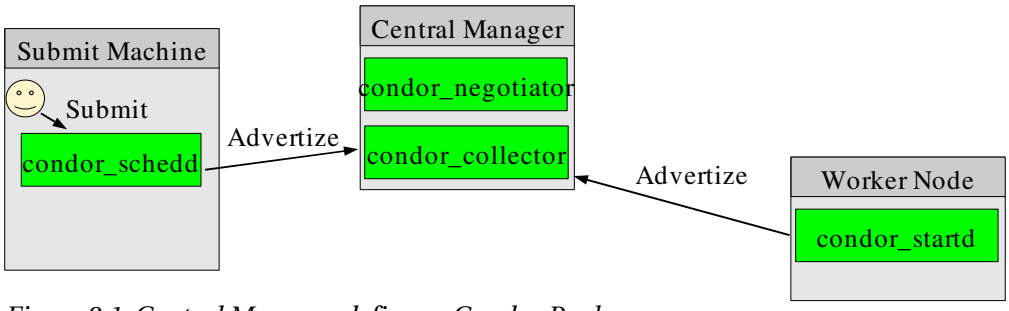

Figure 2.1: Central Manager defines a Condor Pool

The same figure also shows that the user submits his jobs to a local **condor_schedd** daemon. Several authentication methods are available, but most systems are configured to accept either filesystem based authentication or GSI authentication.

Once a job is accepted by **condor_schedd**, the negotiation cycle can begin. The **condor_negotiator** matches the attributes of the user jobs in the **condor_schedd** queue to

the attributes of **condor_startd**'s running on the worker nodes , as shown in Fig 2.2. Once a match is found, the **condor_negotiator** sends a match message to the interested parties.

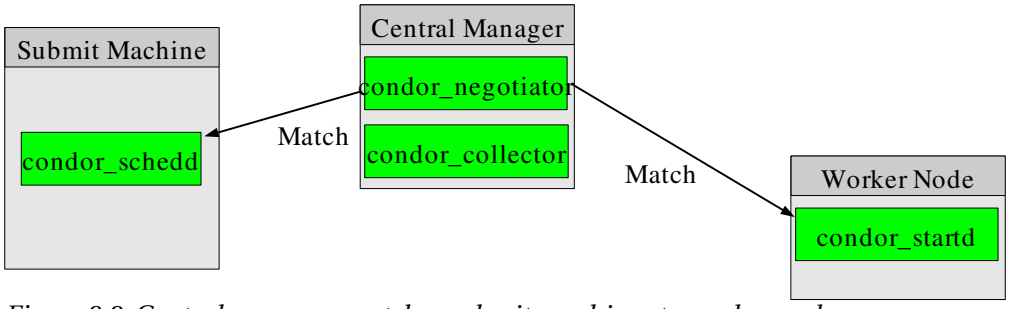

Figure 2.2: Central manager matches submit machines to worker nodes

At this point the **condor_schedd** spawns a new process, called **condor_shadow**, owned by the real user identity. The condor_shadow sends a Claim request to the **condor_startd** that in turn spawns another process, called **condor_starter**. See Fig 2.3b. If using gLExec, the user proxy is delegated during the Claim request and the condor_starter will be running as the real user identity, as shown in Fig 2.3a. Different colors indicate different UNIX identities.

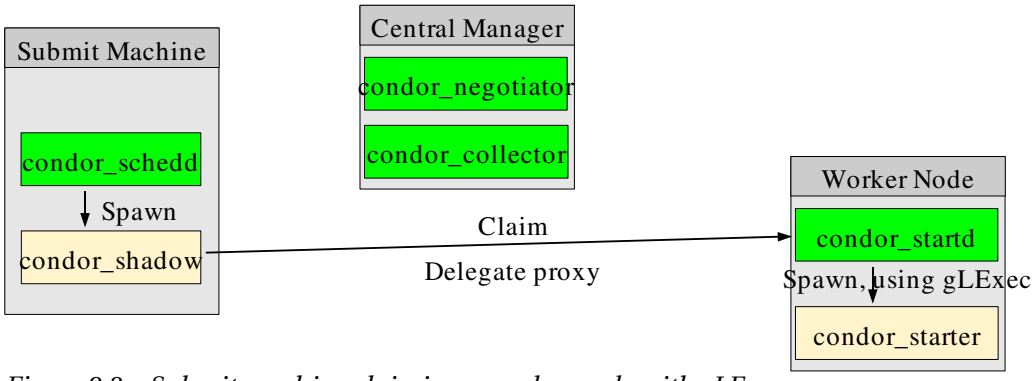

Figure 2.3a: Submit machine claiming a worker node with gLExec

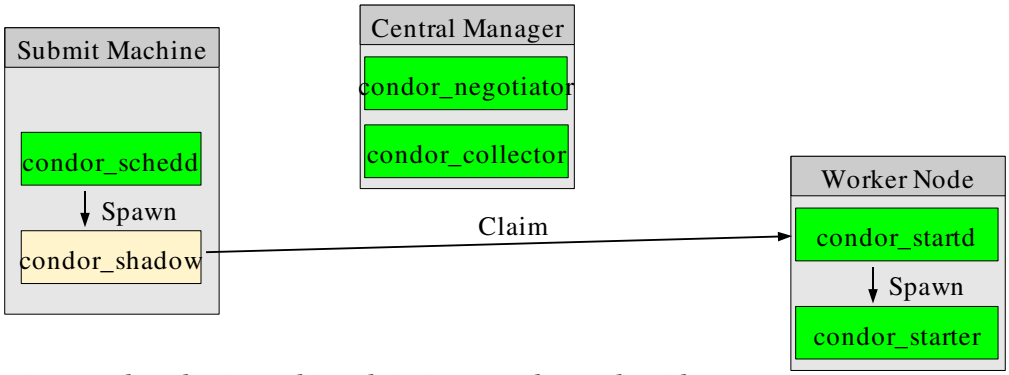

Figure 2.3b: Submit machine claiming a worker node without gLExec

At this point, the condor_shadow can send the user job's input sandbox to the **condor_starter**. Once received, the user job is spawned and executed. After the job is done,

the output sandbox is sent back to the **condor_shadow**, the working directory is cleaned up and the condor_starter terminates. See Fig 2.4a. At this point, the **condor_startd** can either accept a new claim or terminate. Typical configuration will keep the **condor_startd** accepting Claims for a few hours before terminating, as long as new claims arrive within 20 minutes or so.

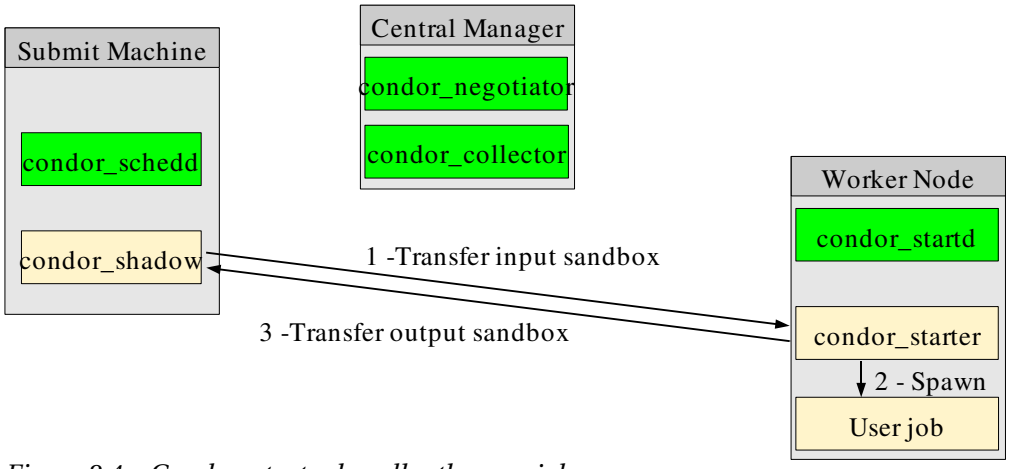

Figure 2.4a: Condor_starter handles the user job

Obviously, running the condor_starter under the same UNIX indentity as the user job is potentially a security risk. However, the risk is minimal, as the only action that the condor_starter can perform is send out the output sandbox; the input sandbox is being pushed to it. Nevertheless, the Condor team is working on improving this by calling gLExec to spawn the job itself, as shown in Figures 2.3b and 2.4b.

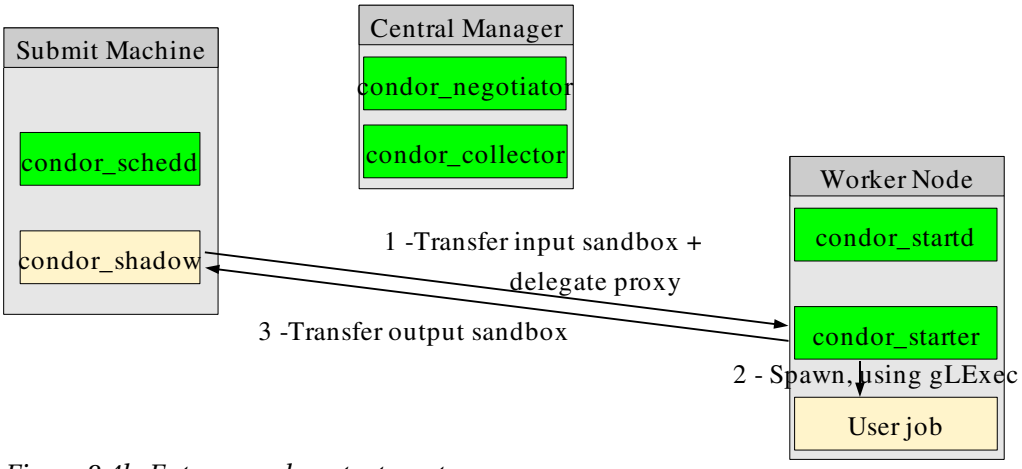

Figure 2.4b: Future condor_starter setup

Please notice that without gLExec in identity switching mode the user job runs under the same identity as the **condor_startd**, as shown in Fig 2.4c. This is a very dangerous setup if more that a single user is using the system; a malicious user could easily compromise **condor_startd** and start farming the proxies of other users. This deployment scenario is strongly discouraged for multi-user setups.

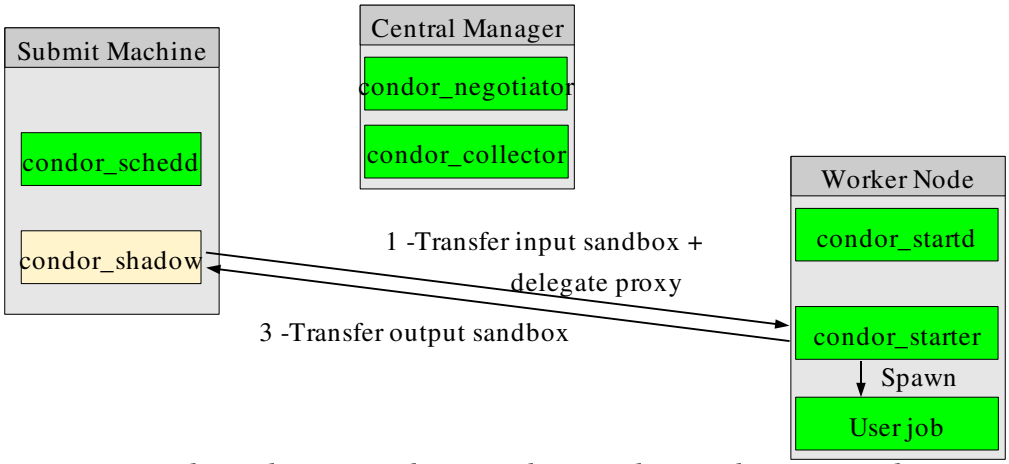

Figure 2.4c: Without identity switching, condor_startd exposed to user attacks

More information about Condor pools can be found in the Condor manual

http://www.cs.wisc.edu/condor/manual/v7.0/

Glidein handling

A glideinWMS Condor pool relies on glideins to start **condor_startd**'s on Grid worker nodes. The glidein submission process is handled by three distinct set of processes, as shown in Fig. 3:

- **Glidein factories** are in charge of submitting the glideins.
- **VO frontends** regulate the number of glideins to be submitted (by the glidein factories), based on the number of jobs waiting in the condor_schedd queues.
- A **condor_collector** is used as a dashboard for message exchange.

As with all other Condor software, CEDAR is used for communication. The typical configuration will use GSI for authentication and will integrity check all the messages.

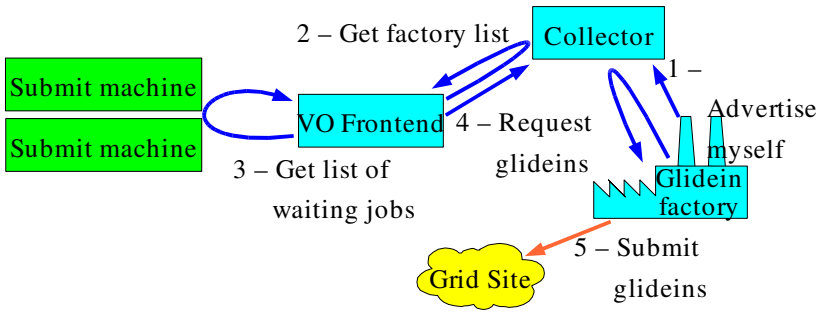

Figure 3: Glidein handling

Glidein factories advertise the known Grid Computing Elements (CEs), along with any attributes that they know, or speculate, about them. VO frontends compare the attributes

published by the glidein factories with the attributes present in user jobs, and decide where to send glideins. Glideins will be sent to all the CEs that match at least one user job.

The glidein submission logic is based on constant pressure; as long as there are suitable jobs waiting in queues on the submission machines, a small number (typically up to 100) of glideins will be kept in the queues of each and every suitable CE. The glidein factories will use Condor-G for actual glidein submission.

A glideinWMS glidein is just a shell script design to download, and possibly execute other files, as shown in Fig 4. These other files are hosted on a Web server, like Apache, usually running on the same machine as the glidein factory. Standard HTTP protocol is used to download the files, but all files are integrity checked using **sha1sum**. A proxy cache, like **Squid**, will be used if available. There is no authentication involved and privacy over the wire is not possible.

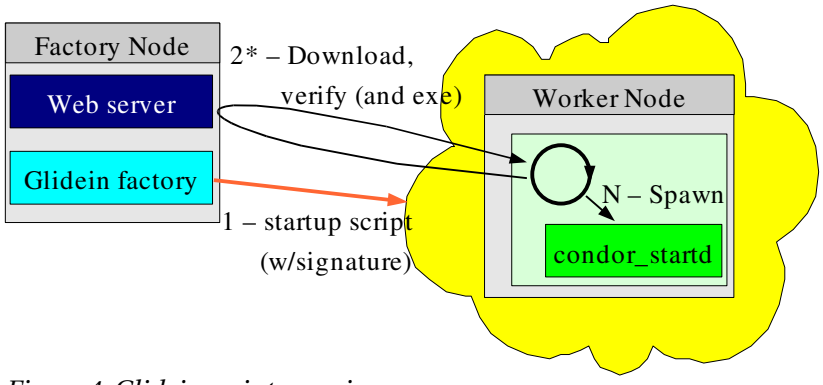

Figure 4: Glidein script overview

The executables launched before **condor_startd** are responsible for gathering machine specific information and create a configuration file for the **condor_startd**. The basic configuration scripts come with glideinWMS, but administrators can add their own executables.

More details can be found in the glideinWMS manual:

http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/doc/manual/

Working in a firewalled world

Condor, and by extension glideinWMS, rely on two way network communications. However, several Grid sites are either behind a NAT or use a restrictive network firewall. To work in environments where only outgoing connectivity is available, Condor provides a product called GCB (Generic Connection Brokering).

With GCB, only a few nodes need to have incoming connectivity: the **Condor central manager** and the **GCB nodes**. All other processes will then use one of the GCB nodes to route their incoming traffic. The process with no incoming connectivity will establish a long lived TCP connection with GCB, obtaining a dedicated port number on the GCB machine. This GCB address is then published as the process' contact point and when a message needs to be send to it, the message will be sent to GCB. GCB will then use the existing TCP channel to relay the message to the waiting process. See Fig. 5 for an example.

At the time of writing, there is no known way to make Condor work in environments where direct outgoing connections are not allowed. glideinWMS is thus unable to use such resources.

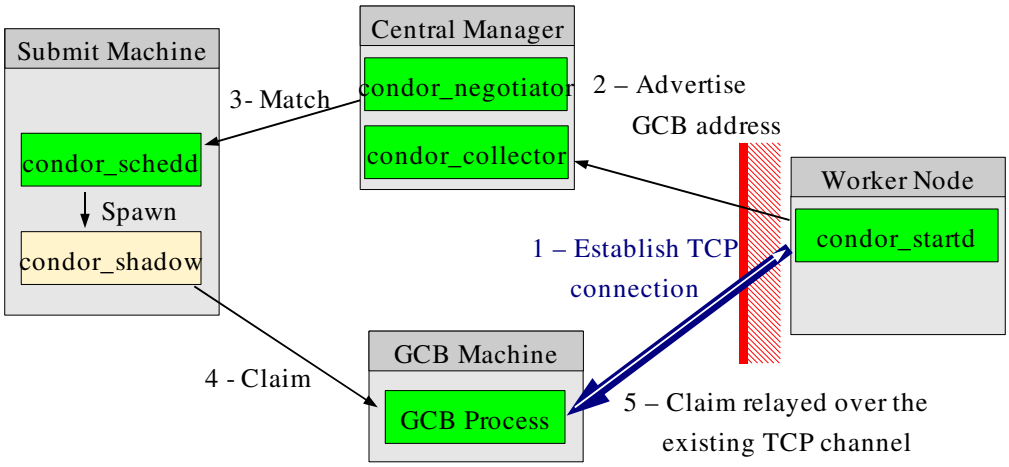

Figure 5: Condor use of GCB to traverse firewalls

Credentials handling

As with all the pilot infrastructures, two kinds of credentials are handled; the **pilot credential** and the **user credentials**.

The **pilot credential** is held by the **glidein factory**. There is no explicit proxy renewal in place. The glidein factory expects a valid proxy to be referenced by the X509_USER_PROXY environment variable; how the proxy is kept valid at all times is currently out of scope for the glideinWMS and may change between deployments. This proxy is used both for authentication with the condor_collector and for submission of the glidein startup script via Condor-G.

When used for submitting glideins, the proxy is transferred to the worker node using standard Grid tools (Condor-G using pre-WS or WS GRAM). Condor-G supports updates of proxies for Grid jobs, but it is currently not used. The pilot proxy is used by **condor_startd** to authenticate with the other Condor daemons and it is needed for the whole lifetime of the glidein. The validity of the proxy at startup is used to determine the maximum lifetime of the glidein.

The **user credentials** are handled by the **condor_schedd**. There are two ways a user can communicate his credentials to **condor_schedd**:

- The user specifies the location of a valid user proxy and condor_schedd will simply take note of that. The user needs to refresh the proxy for the lifetime of the job.
- The user can store a long lived credential in a MyProxy server and instruct condor_schedd to use it. Only Condor-G jobs were supported and the only authentication method available was username&password at the time of writing.

Once the job is ready to start, **condor_shadow** takes the proxy and delegates it to **condor_startd** (or **condor_starter**, depending on configuration). Let me stress that this is

the classic GSI delegation and the credential private key is never transferred over the wire. If the proxy ever gets updated on the submission machine during the lifetime of the job, the proxy is re-delegated to the **condor_starter**.

3.Deployment scenarios by USCMS

USCMS has only a Fermilab-local prototype production installation right now. A world-wide production instance is however expected in the near future, followed by an analysis instance. Detailed installation details for the future installations are not yet finalized, but an educated guess is presented in this document.

Current prototype production installation

The current production installation is located at Fermilab and is sending glideins only to the Fermilab CMS Tier-1 OSG Computing Element. This installation is used by a single production team doing re-reconstruction and skimming.

The Fermilab installation is distributed over 5 machines; one hosts the glidein factory and the glideinWMS collector, one hosts the Condor pool collector, the negotiatior and the VO frontend, and three others host a schedd each. See Fig 6a. for an overview.

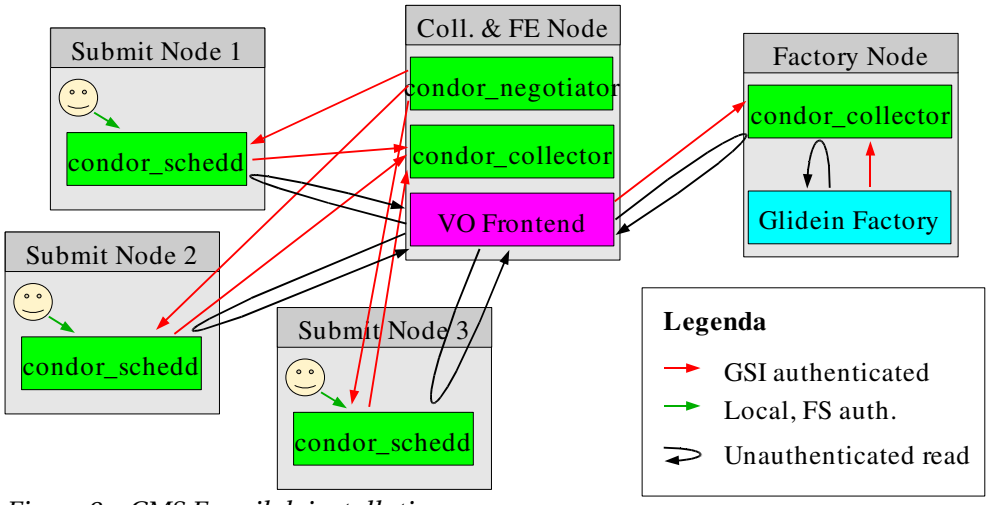

Figure 6a: CMS Fermilab installation

All communications between daemons, except simple information retrieval, are authenticated using GSI authentication and integrity checks are enforced. The glidein factory uses the pilot proxy, while all the other daemons use a host certificate. The Condor daemons are setup to be world readable, so no authentication, nor integrity checking, is used while querying them.

The user job submission is local to the submit nodes, and uses local FS authentication. This means that all authentication is done at a Operating System level, using kerberized ssh. The users do not provide the **condor_schedds** with any x.509 proxy.

The glideins are submitted using the pilot proxy, held by the **glidein factory**. The pilot proxy is really a personal proxy of the production team leader, and it is refreshed by hand by such person on a regular basis.

Once the **condor_startd** starts, it will use this pilot proxy for communication with the **condor_collector**. The submit node authenticate with the glidein using the host certificate. The **condor_shadow** transfers to the **condor_starter** only the user job; no proxy delegation is involved as the user did not provide a proxy. As a result, the user job has access to the pilot proxy. But since this glideinWMS serves only one logical entity, the FNAL CMS production group, this is acceptable.

See Fig 6b. for an overview.

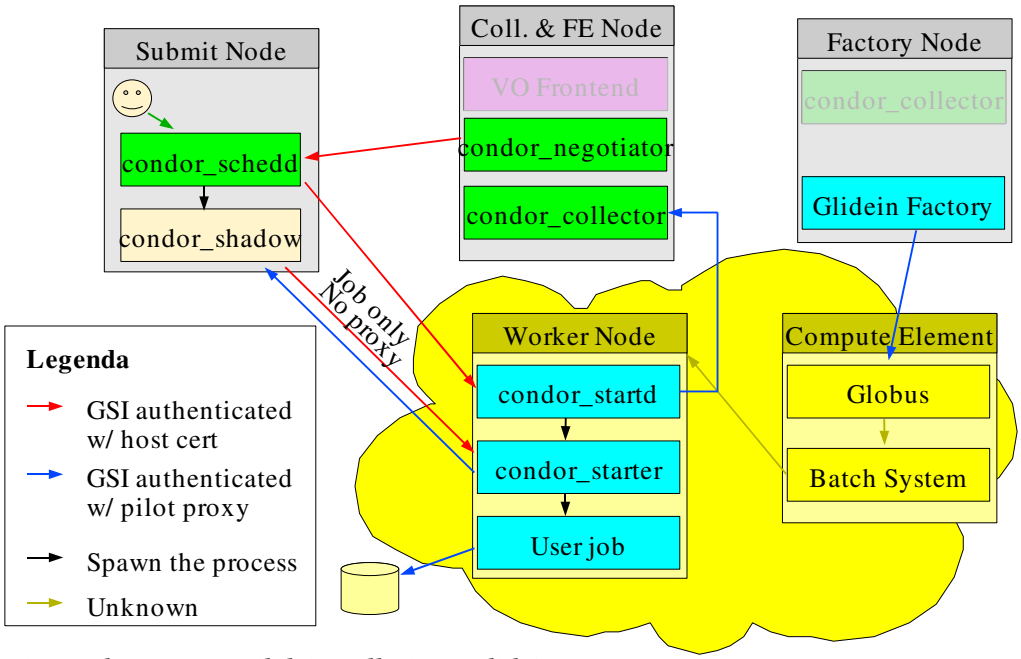

Figure 6b: CMS Fermilab installation - Glideins

The access to all of the above machines is restricted to the people in the FNAL CMS production team. Kerberized ssh is used for login. The production team does not have root access on these machines.

All the submitted jobs contain only CMS production binaries. This restriction is implemented as policy only, and it is not enforced by any technical means.

Planned world-wide production installation

CMS is still evaluating the use of the glideinWMS for production purposes. The description below is author's best guess how it would be deployed if no problems are found.

The CMS production team is currently split between CERN and Fermlab, in order to cover the 24 hours using only day shifts. Similarly, the submission machines are distributed between the two sites to maximize the uptime.

The envisioned glideinWMS-based production pool would also be deployed both at CERN and at Fermilab to minimize the downtimes due to problems at one of the sites, while still maintaining a single logical view of the system. Each and every service has at least two instances running in normal conditions, and any one of them can go offline without drastically affecting the resource pool. Most services are simply not critical, while the central manager would use the Condor high availability features to switch the negotiation handling from one site to the other. See Fig. 7 for a visual overview.

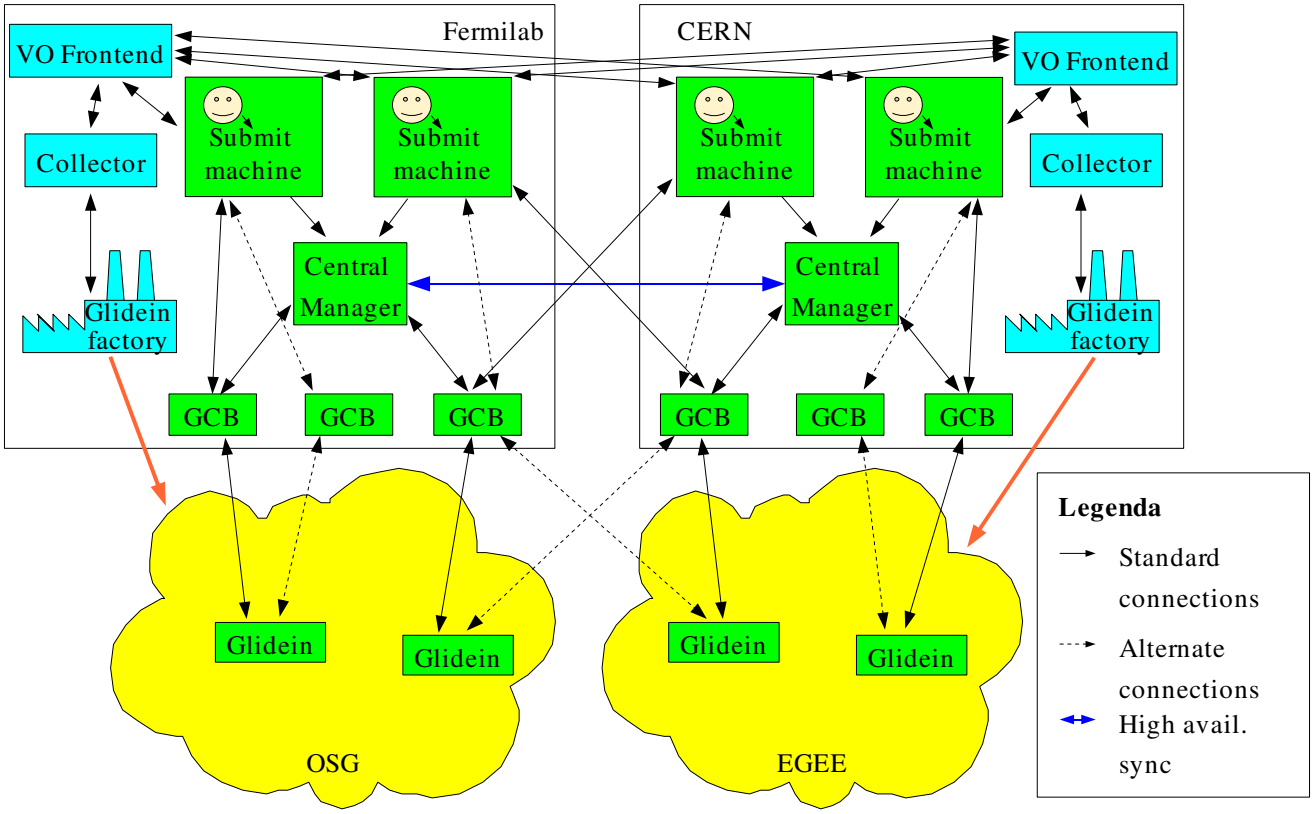

Figure 7: The envisioned CMS world-wide glideinWMS pool

Planned analysis installation

To be written

4.Conclusions

To be written

Appendix

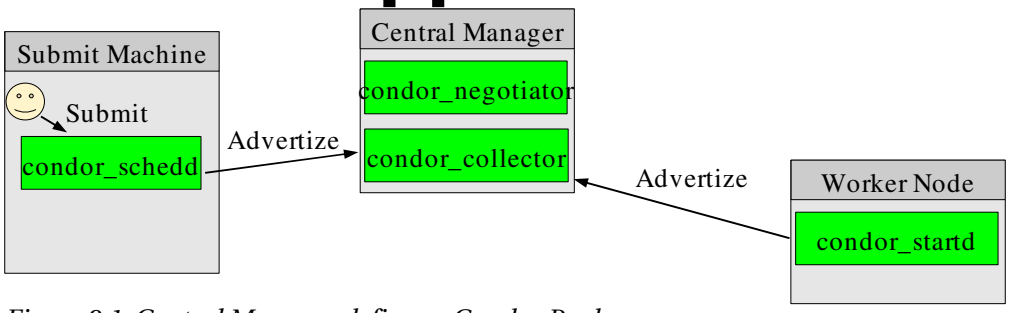

Figure 2.1: Central Manager defines a Condor Pool

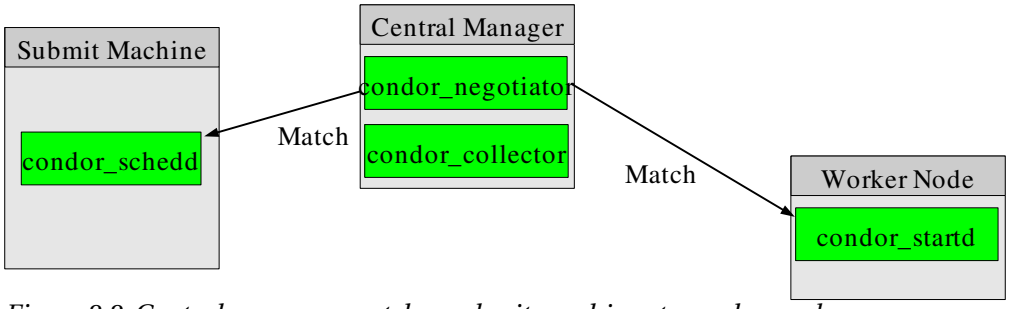

Figure 2.2: Central manager matches submit machines to worker nodes

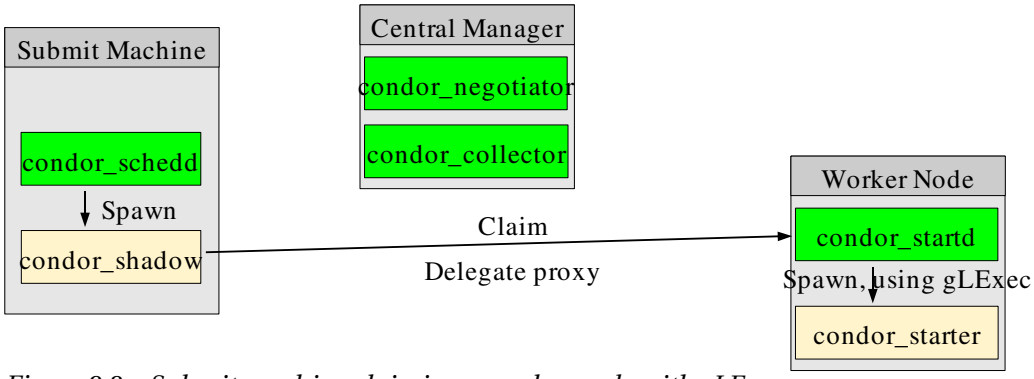

Figure 2.3a: Submit machine claiming a worker node with gLExec

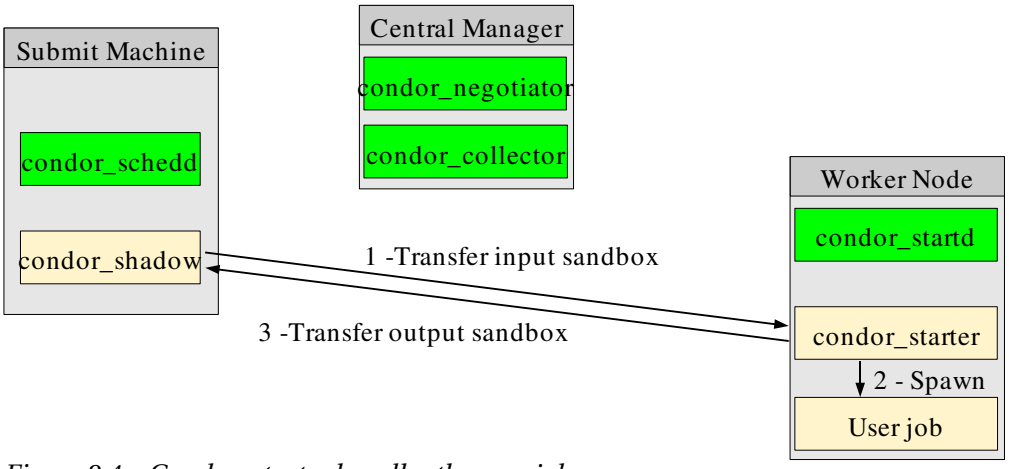

Figure 2.4a: Condor_starter handles the user job

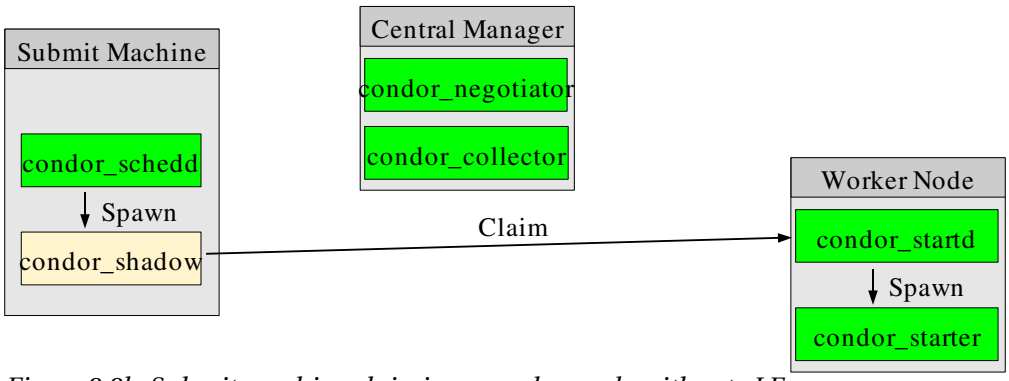

Figure 2.3b: Submit machine claiming a worker node without gLExec

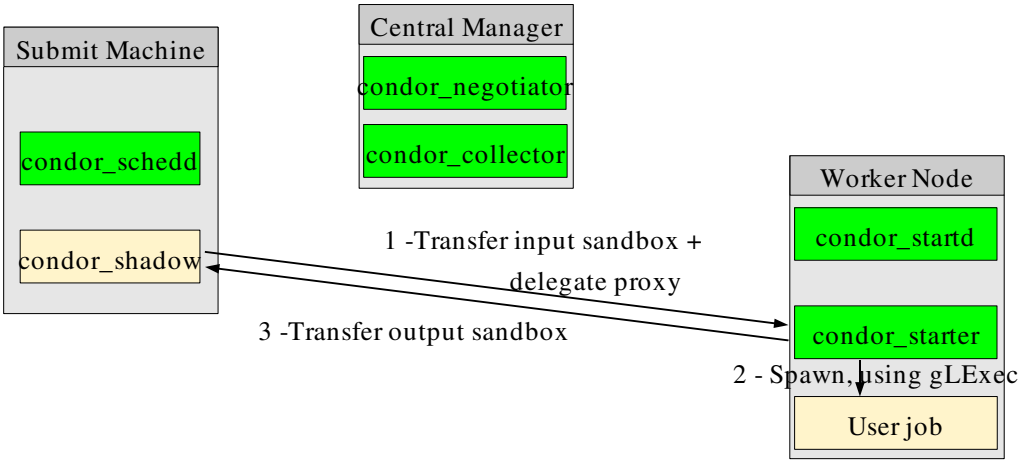

Figure 2.4b: Future condor_starter setup

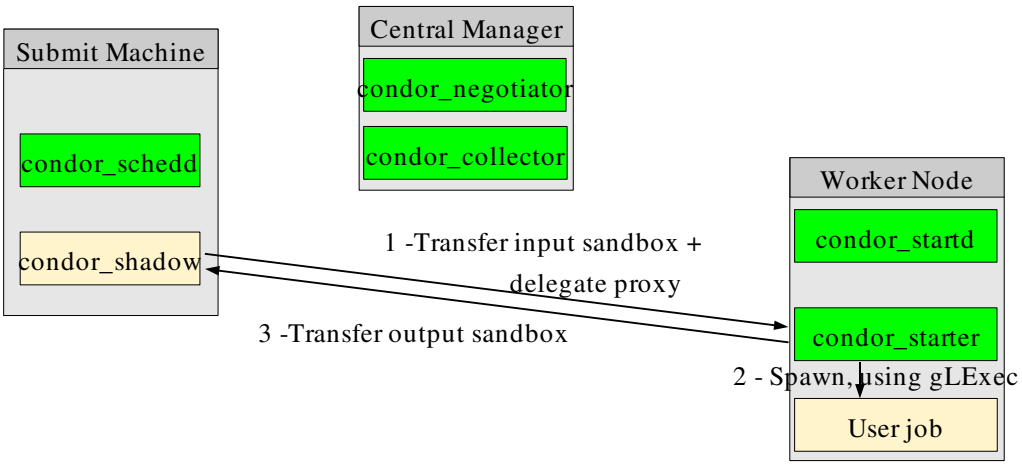

Figure 2.4b: Future condor_starter setup

5.Questions

To be written.

6.Condor pool figures