Workload management with glideinWMS

Igor Sfiligoi, Fermilab, Batavia, IL, USA

v0.8.0 – May 28th

1.Introduction

GlideinWMS is a general purpose Workload Management System (WMS) developed by CMS developers in the US, 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.

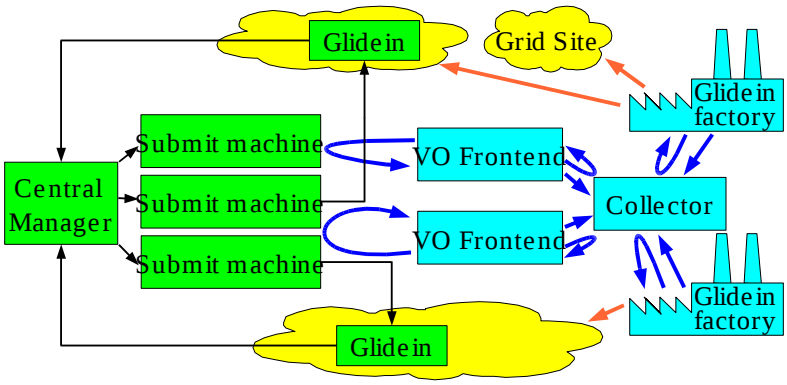

Figure 1: GlideinWMS schematic overview

2.1. 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. When GSI is used, authorization is based on the credential DN. Condor supports both explicit lists as well as regular expressions.

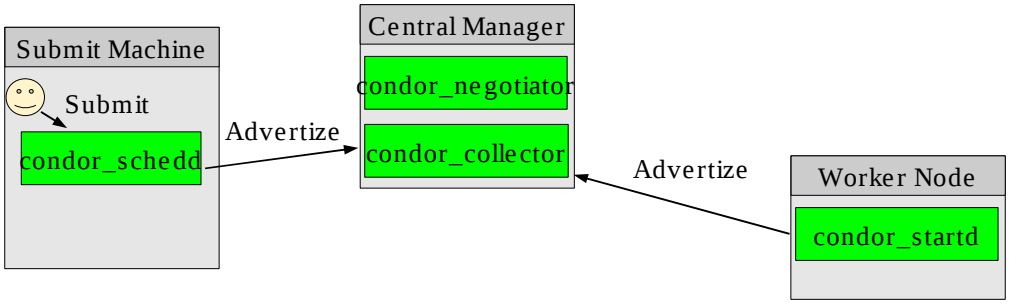

Figure 2.1: Central Manager defines a Condor Pool

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.

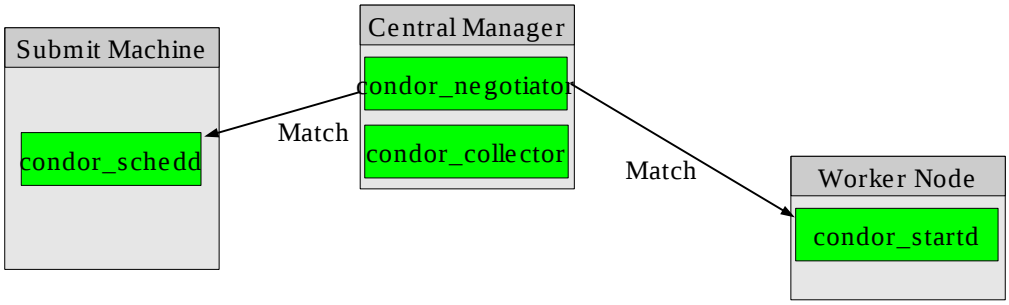

Figure 2.2: Central manager matches submit machines to worker nodes

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.

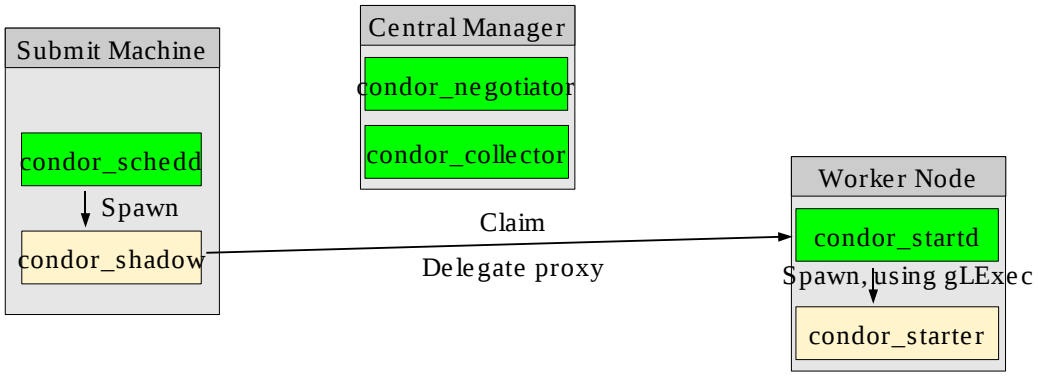

Figure 2.3a: Submit machine claiming a worker node with gLExec

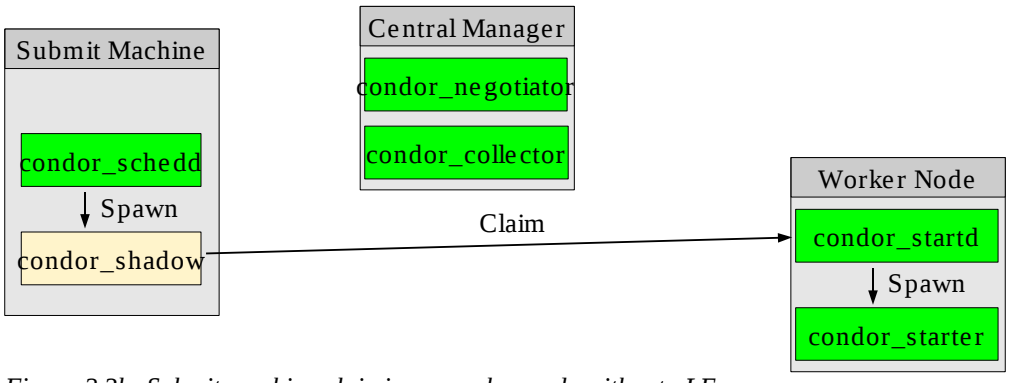

Figure 2.3b: Submit machine claiming a worker node without gLExec

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, **condor_starter** sends the output

sandbox back to the **condor_shadow**, cleans up the working directory, and terminates. See Fig 2.4a. If the **condor_starter** cannot clean up for any reason, the **condor_startd** will do it.

Once the old job is finished, the **condor_startd** can either accept a new claim, force a new match 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. To minimize the latencies re-matching is usually not requested, but can be easily enabled.

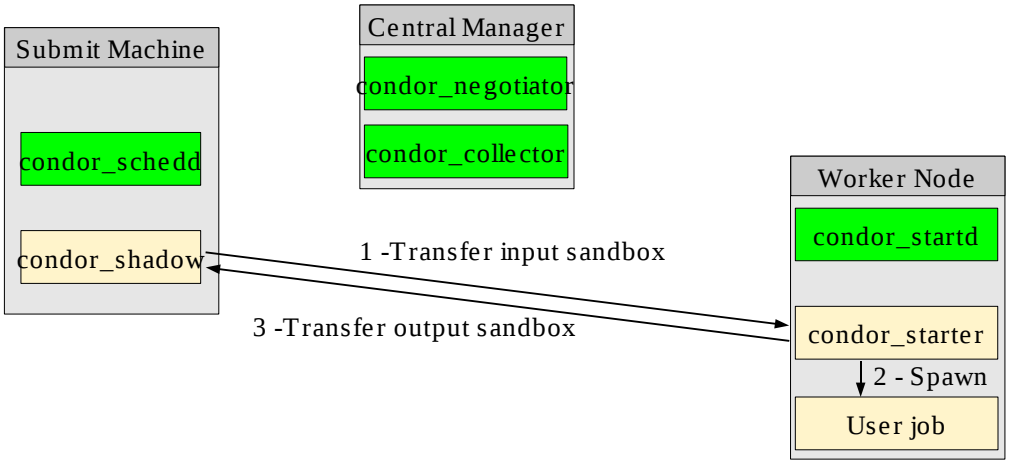

Figure 2.4a: Condor_starter handles the user job

Obviously, running the condor_starter under the same UNIX identity as the user job is potentially a security risk. However, the risk is minimal, as the only external 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.

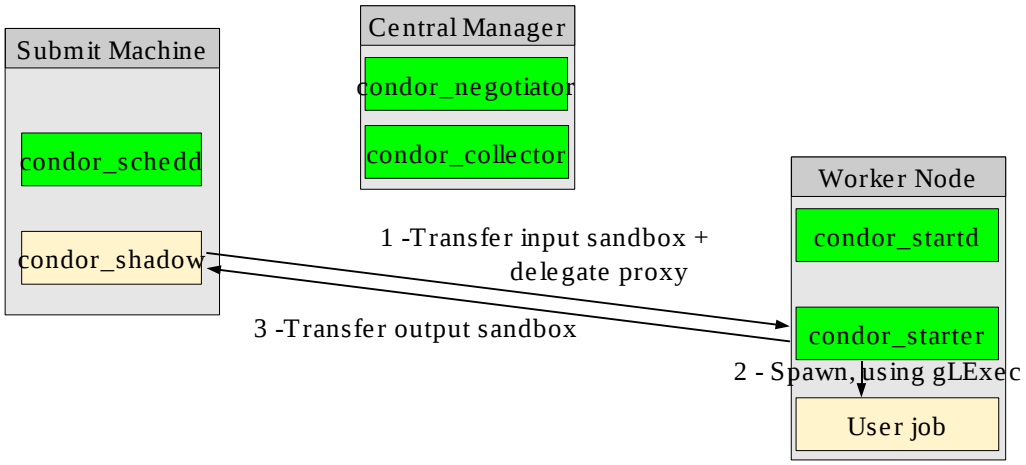

Figure 2.4b: Future condor_starter setup

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.

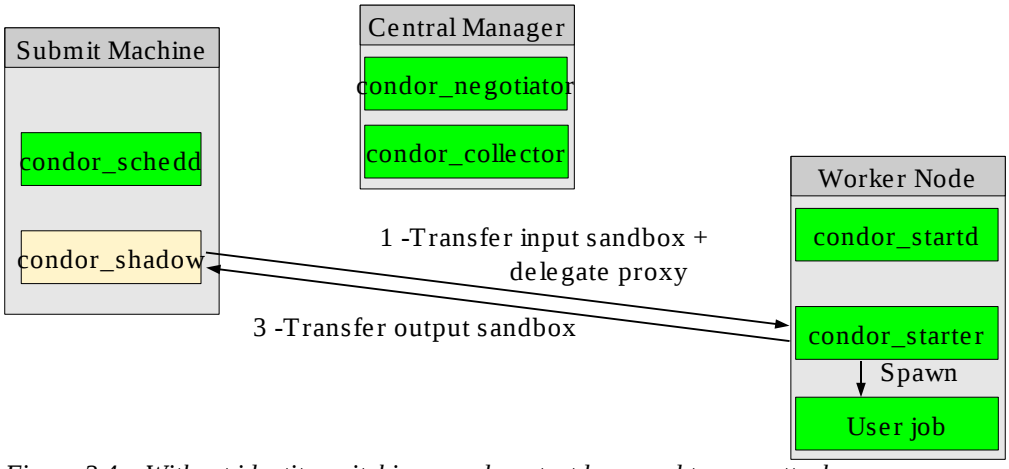

Figure 2.4c: Without identity switching, condor_startd exposed to user attacks

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

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

2.2. 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.

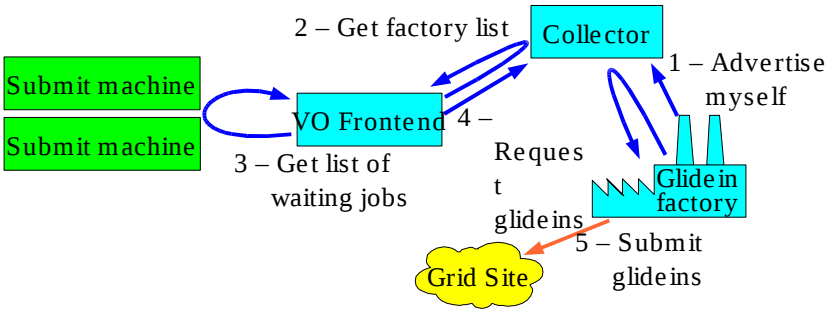

Figure 3: Glidein handling

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 designed 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**. Since there is no authentication involved and privacy over the wire is not possible. HTTP has been chosen over other more secure mechanisms, like HTTPS, because it allows for caching; a proxy cache, like **Squid**, will be used if available.

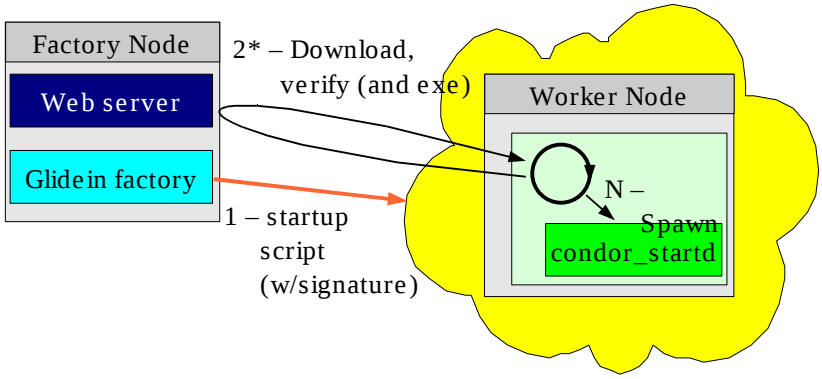

Figure 4: Glidein script overview

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/

2.3. 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 sent 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.

Please notice that **GCB** does not perform any authentication at all; all the requests are honored. The security of the channel is delegated to the end points, **condor_shadow** and **condor_startd** in Fig. 5.

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.

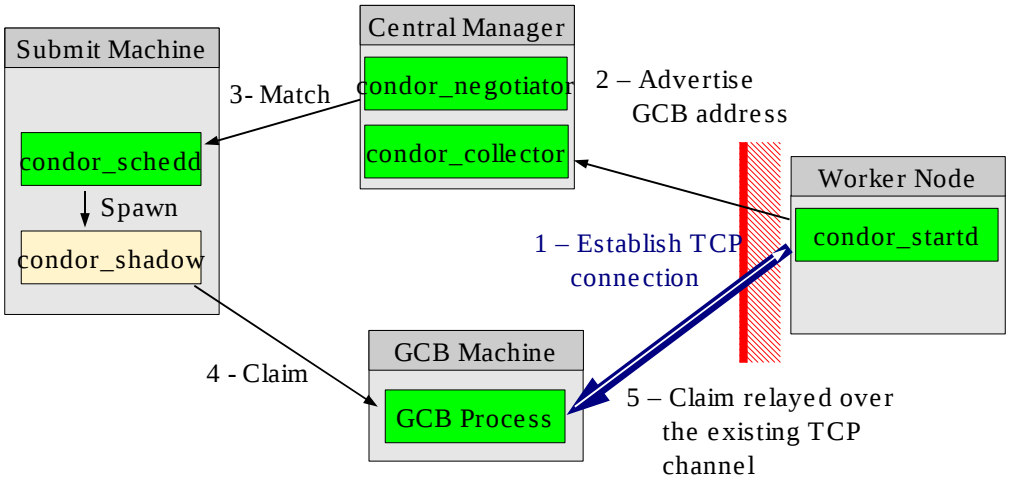

Figure 5: Condor use of GCB to traverse firewalls

2.4. 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. This can easily be changed, if one accepts the risk associated with failed proxy renewals.

The **user credentials** are handled by the **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. There are essentially three ways how this can be done:

- Updating the proxy by hand, typing a password every time.
- Upload the certificate or a long lived proxy into a trusted MyProxy server and use cron to renew the local proxy. The cron script can either be run by the user or by the local system.
- Use cron with a local certificate without a password, or by embedding the password in refresh script. While discouraged by the Grid community, some people do this.

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**.

The delegation does not put any limit on the delegated proxy. For security reasons a limited delegation would be preferable, but this is currently not supported by Condor. The Condor team is aware of this and will provide limited delegation in one of the future Condor releases.

3.Deployment scenarios by USCMS

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

3.1. 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.

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 this person on a regular basis.

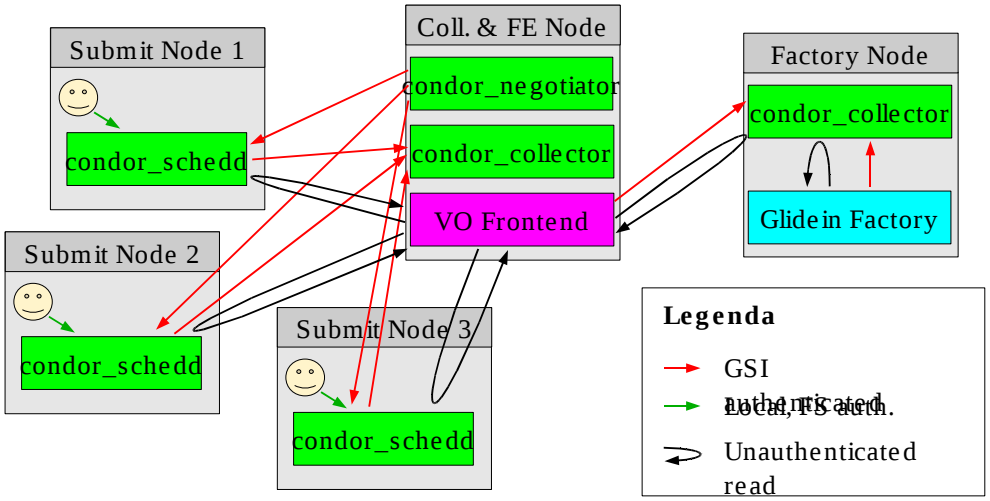

Figure 6a: CMS Fermilab installation

Once the **condor_startd** starts, it will use this pilot proxy for communication with the **condor_collector**. The submit node authenticates 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.

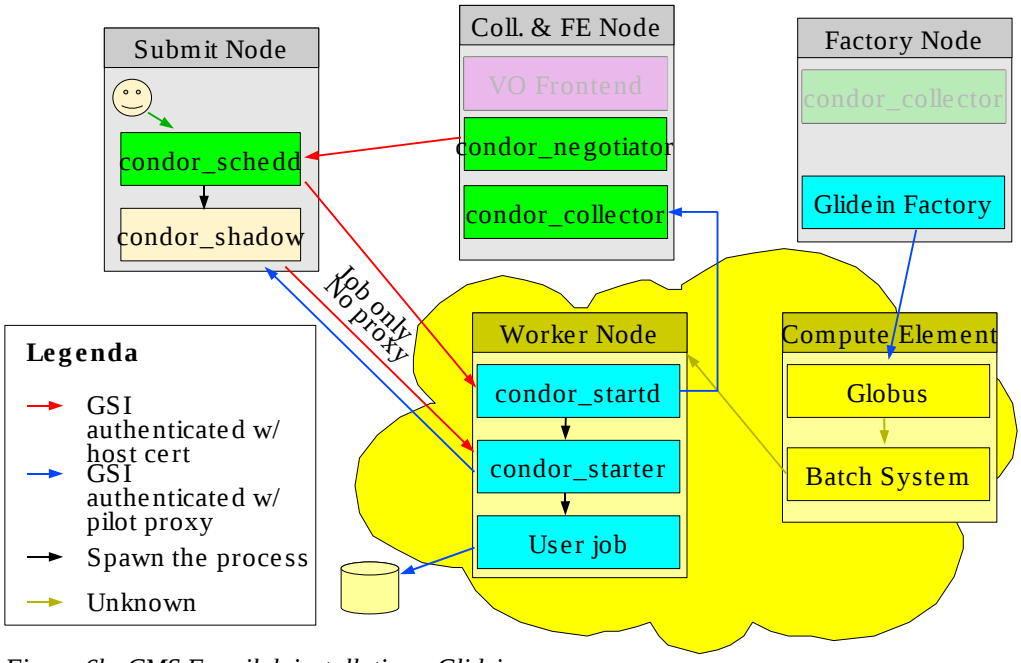

Figure 6b: CMS Fermilab installation - Glideins

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.

3.2. Planned world-wide production installation

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

The CMS production team is currently split between CERN and Fermilab, 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.

The exact number of machines involved is not known yet, and will be determined after the evaluation period. It is however expected that CMS will use the minimum number of machines needed to guarantee the necessary throughput while maintaining enough redundancy to survive the failure of any node. As described above, the nodes involved will be distributed between Fermilab and CERN. Nodes at Fermilab will be maintained by the Fermilab CMS system administrators according to the Fermilab security practices, while CERN nodes will be maintained by the CERN system administrators according the CERN security practices.

User authentication will also be handled similar to the current prototype. All communications between daemons will be GSI authenticated. On the other hand, the CMS production team will have login access to all the submit machines, using kerberized ssh at Fermilab and password authenticated ssh at CERN. Submissions to the local **condor_schedd** will be authenticated by UNIX UID only. The team personal proxies will not be used nor will be they transferred to the worker node.

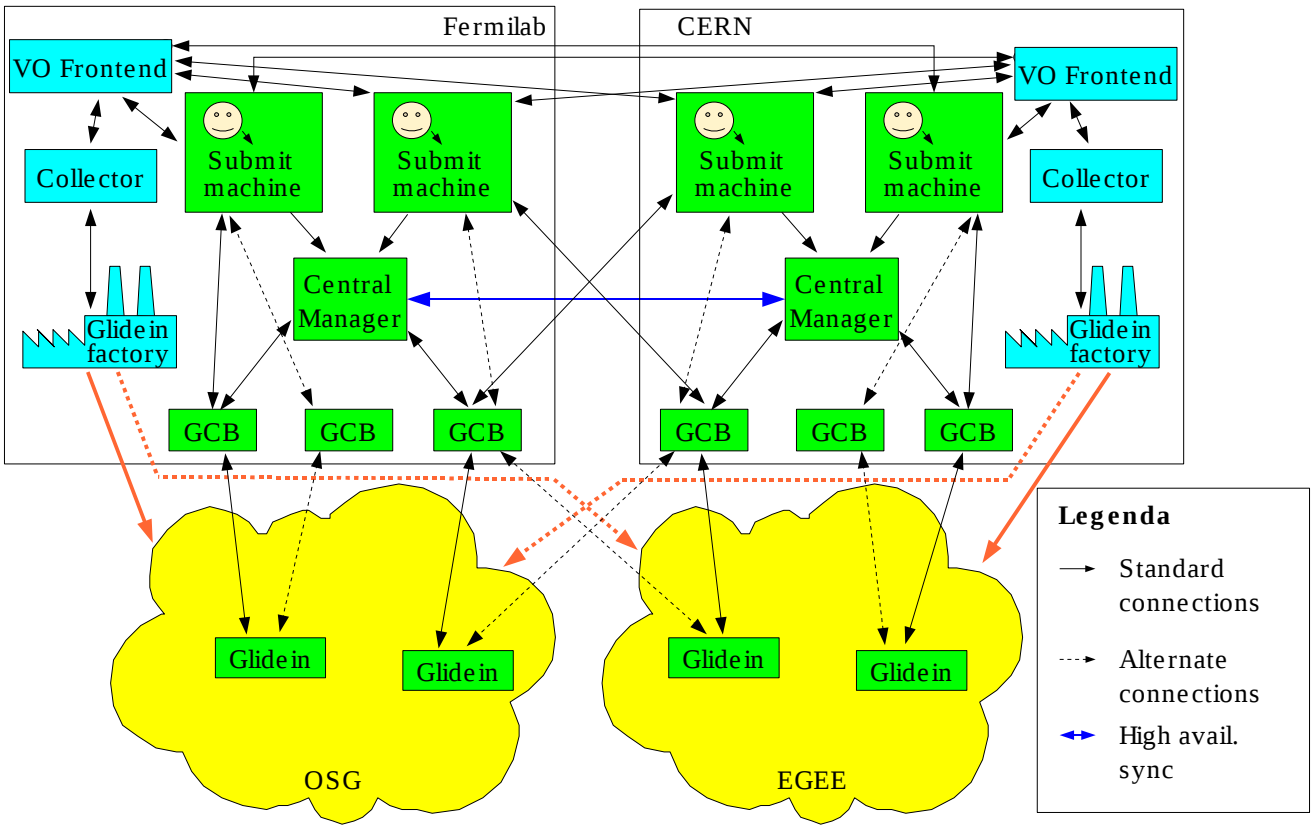

Figure 7: The envisioned CMS world-wide glideinWMS pool

How the glidein factory pilot proxy will be handled is not completely clear yet, but the authors best guess is that CMS will use a service certificate that will be either local to the glidein factory or stored in a MyProxy server. A cron script creating a VOMS-extended short lived proxy out of the certificate will probably be used.

Since CMS is expected to use the glideinWMS both on EGEE and OSG, the glideins will most probably have to be configured to support proxy renewal on the worker nodes. While technically easy, the security and reliability aspects need to be evaluated.

3.3. Planned analysis installation

CMS analysis is handled by the CMS Remote Analysis Builder (CRAB). Users develop their own analysis jobs locally and tests them with a small local data set. Once they are satisfied with the job behavior, they use CRAB to run the jobs on officially published datasets, usually using the Grid resources. CRAB takes care of job submission, job splitting and output retrieval. CRAB hides the usage of GRID middleware from the users, giving them a easy-to-use interface to the computing resources.

CMS is still evaluating the use of the glideinWMS for analysis purposes. CRAB itself is also going through a major redesign phase. The description below is the author's best guess how the combination of the two could be deployed to best serve the CMS analysis needs.

The envisioned CMS analysis infrastructure will have a fully redundant glideinWMS infrastructure, much like the production one described above in Figure 7, but distributed over many more sites, most likely CMS T2 sites. However, users will never login directly directly to any glideinWMS submit machine; each sumbit machine will instead host a CRAB Server, and all user interaction with the Grid will be through the CRAB interfaces. See Figure 8 for a high level overview.

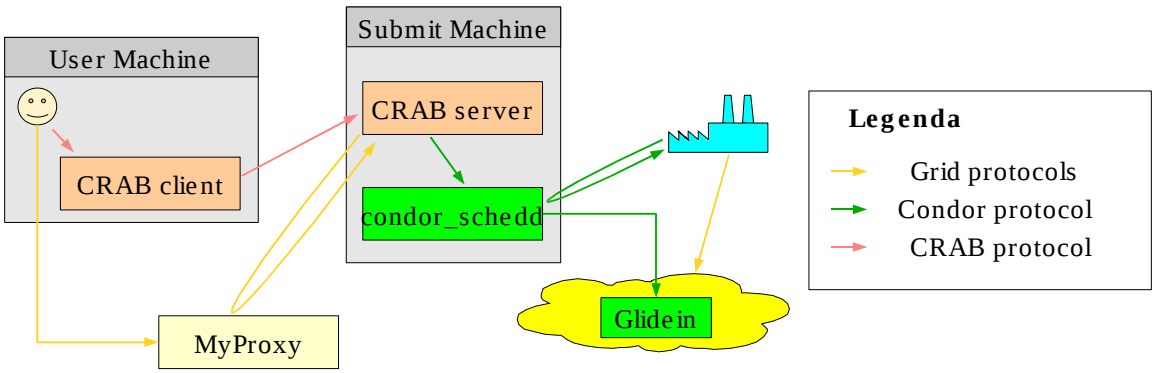

Figure 8: CRAB Server Architecture

The CRAB server uses the EGEE Delegation Web Service Interface paired with a MyProxy instance for proxy handling. The CRAB server Delegation Service needs to be a trusted host of MyProxy for proxy renewal.

The glidein factories will use a service certificate for glidein submissions. Once a glidein starts, all jobs will be paired with the user proxy and gLExec will be used to authenticate and authorize the final user and to start the the user job under the appropriate UNIX identity.

Many technical details about the CRAB Server/glideinWMS interaction are not known at the time of writing, as the integration work has not been finished yet.

4.Conclusions

GlideinWMS is a general purpose Workload Management System (WMS) based on the pilot philosophy. GlideinWMS has been developed with security in mind, but like all pilot-based WMSes it needs some help from the sites to achieve the desired security goals.

The current implementation requires the presence of gLExec in identity switching mode to safely handle multiple users within the same instance. This operation mode has been tested on OSG resources and I have not yet found any security issues with it.

Appendix

Pilot Job Frameworks questionnaire

0. Describe in a schematic way all the components of the system. If a component needs to use IPC to talk to another component for any reason, describe what kind of authentication, authorization, integrity and/or privacy mechanisms are in place. If configurable, specify the typical, minimum and maximum protection you can get.

Described above.

1. Describe how user proxies are handled from the moment a user submits a task to the central task queue to the moment that the user task runs on a WN, through any intermediate storage.

This was described in the **Condor pool** and **Credentials handling** sections.

2. What happens around the identity change on the WN, e.g. how is each task sandboxed and to what extent?

If gLExec is used, it will execute the user job under a different UID. The OS then provides the necessary protection from other users.

See Question 4 for more details about the sandbox handling.

3. How can running processes be accounted to the correct user?

The **condor_starter** keeps track of the user job process tree, and collects the user and sys cpu use. Those, together with the wallclock time are recorded by the collector in the job history log.

If gLExec is used, gLExec can also do the proper accounting of resources. This information can then be used by the local Grid administrators without the need to contact the glideinWMS administrators.

4. How is a task spawned on the WN and how is it destroyed?

The user job is started by the condor_starter in a dedicated directory. This was described in the **Condor pool** section. See also Figures 2.5a, 2.4b and 2.4c for a schematic overview.

When the job ends, the **condor_starter** will clean up the startup directory tree and kill any remaining process. After **condor_starter** terminates, the **condor_startd** will check again if any files or processes are left and do the cleanup if needed. Malicious users can demonize a process and escape the cleanup procedure, but this cannot be avoided without some help from the system.

The coming OSG 1.0 deployment of gLExec is using secondary GIDs to track the process tree (and will destroy the complete process tree when the main process terminate). To the authors best knowledge there is nothing equivalent for EGEE, yet.

5. How can a site be blocked?

IP based controls can be set up either at the system level (like iptables) or in Condor itself.

6. What site security processes are applied to the machine(s) running the WMS?

[Here WMS means the VO WMS, not the gLite WMS.]

The machine is maintained by the hosting site system administrators according to the hosting site policies. Since most of the described use cases are still in the planning stage, not much more details can be given at this time.

Who is allowed access to the machine(s) on which the service(s) run, and how do they obtain access?

Only system administrator and VO service administrators are allowed access to the machines. Hosting site login policies for privileged users are followed.

In the case of production instances, the production team users can login to the machine using a non privileged account. Hosting site login policies are followed.

How are authorized individuals authenticated on the machine(s)?

Following the hosting site login policies.

What is the process for keeping the service(s) and OS patched and up-to-date, especially with respect to security patches?

Hosting site system administrators follow the hosting site patching procedures.

Do you have an identified security contact?

Not yet, as it is not in production use yet.

Describe the incident response plan to deal with security incidents and reports of unauthorized use?

Not defined yet, as it is not yet in production use.

For the prototype, we would simply shut down the glideinWMS instance in case of incident.

What services (in general) run on the machine(s) that offer the WMS service?

Apart from the glideinWMS specific services, only the base services requested by the hosting site policy are run.

What processes exist to maintain audit logs (e.g. for use during an incident)?

Condor maintains extensive daemon and user logs. All those are local and there is no short-term plan to make them remote.

The OS maintain its own standard logs. If the hosting site procedures support remote logging, it is used.

What monitoring exists on the machine(s) to aid detection of security incidents or unauthorized use?

To start with, the same logs described above.

Moreover, all Condor daemons maintain extensive information about jobs and resources in memory and this information can be used for monitoring.

7. Can you limit the users that can submit jobs to the VO WMS? How?

On the production installations, the submission nodes only accept local submissions. A user needs login privileges to submit a job to the VO WMS.

On the analysis installations, <need to find out what CRAB supports>.

Condor pool figures

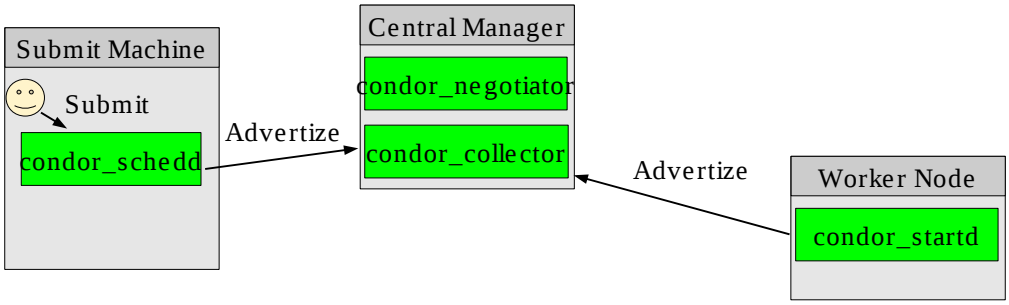

Figure 2.1: Central Manager defines a Condor Pool

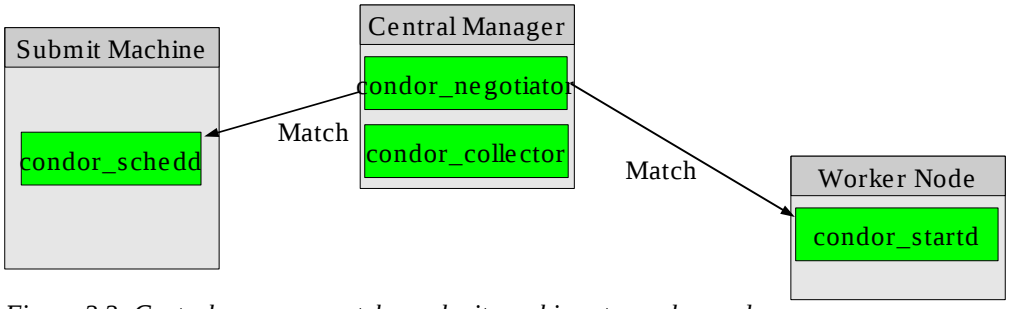

Figure 2.2: Central manager matches submit machines to worker nodes

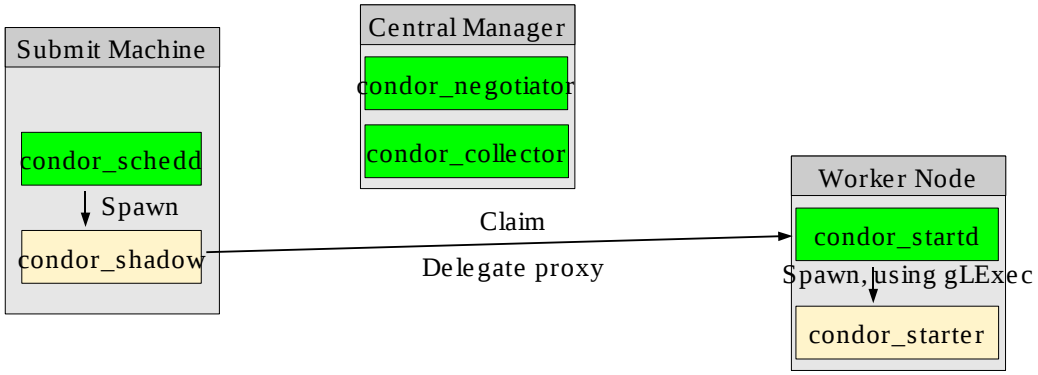

Figure 2.3a: Submit machine claiming a worker node with gLExec

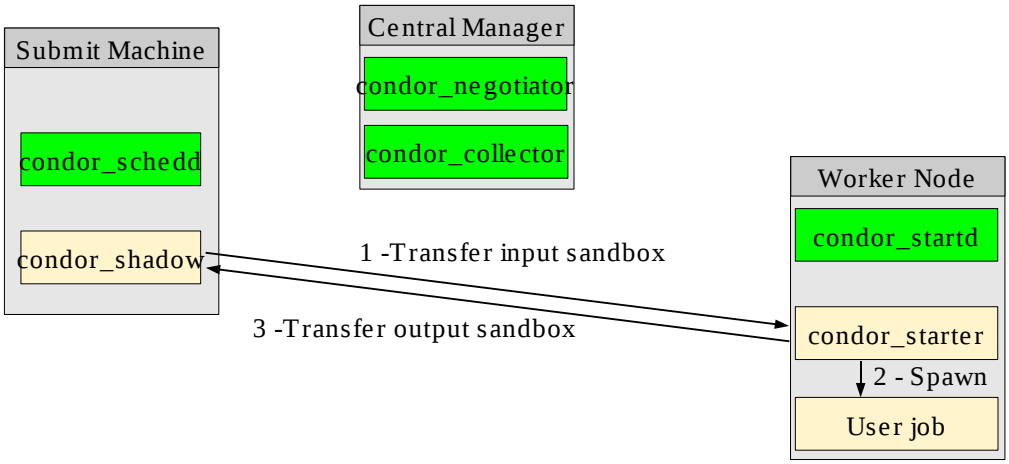

Figure 2.4a: Condor_starter handles the user job

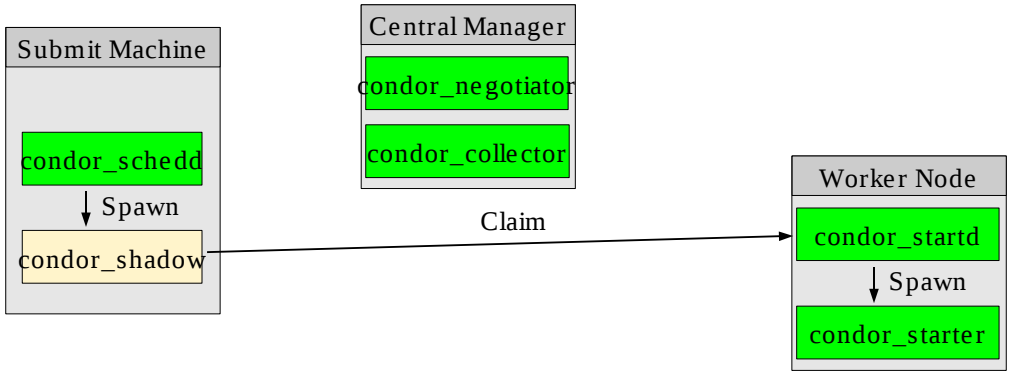

Figure 2.3b: Submit machine claiming a worker node without gLExec

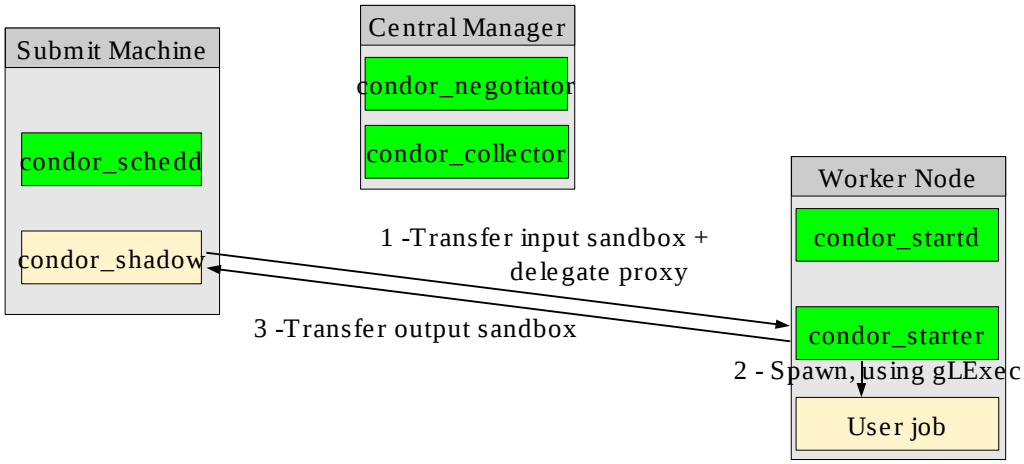

Figure 2.4b: Future condor_starter setup

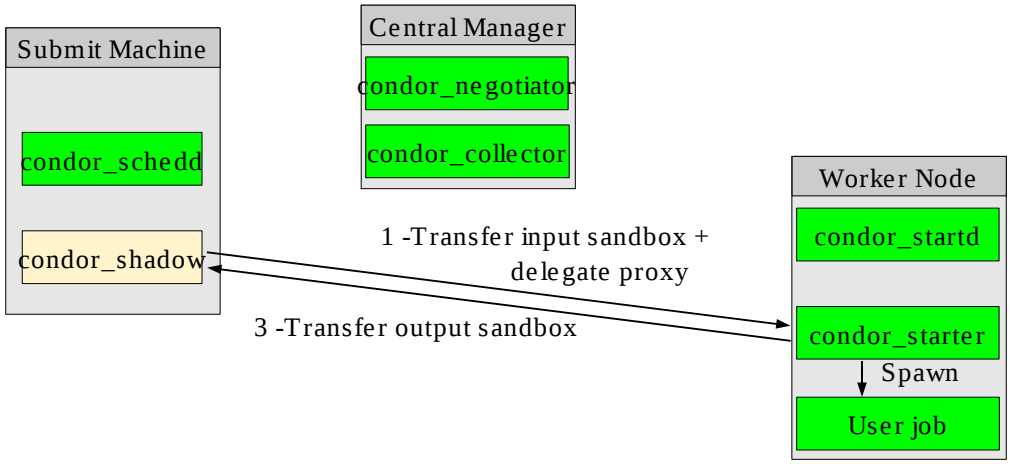

Figure 2.4c: Without identity switching, condor_startd exposed to user attacks