Structural overview of the glideinWMS

Igor Sfiligoi
Fermilab
Batavia, IL, USA

Last updated: July 28th, 2008

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.

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.

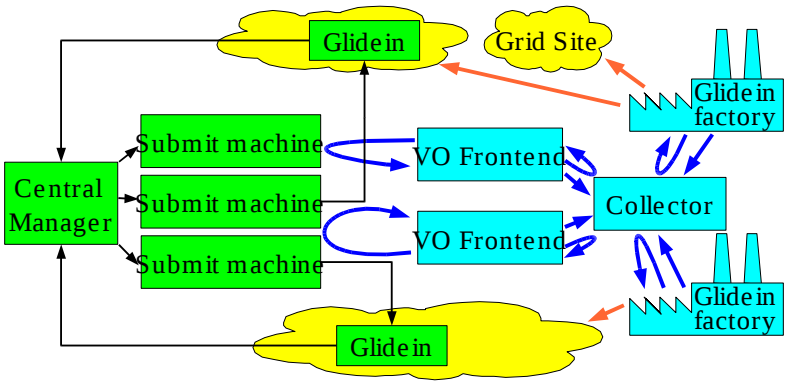

Figure 1: GlideinWMS schematic overview

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; if other similar tools become in the future, Condor could evolve to support those, too.

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.

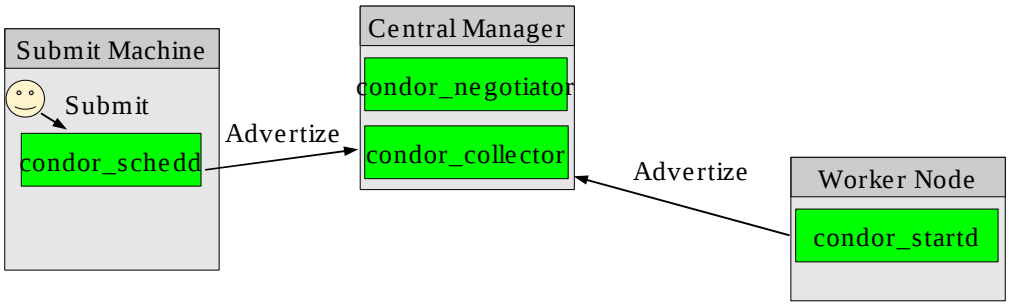

Figure 2.1: Central Manager defines a Condor Pool

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.

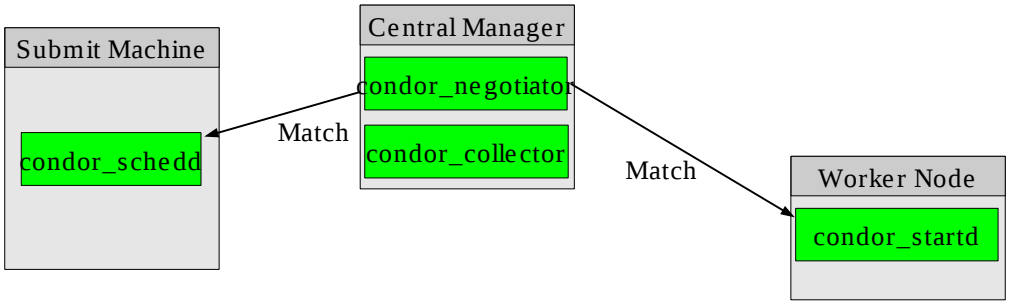

Figure 2.2: Central manager matches submit machines to worker nodes

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.

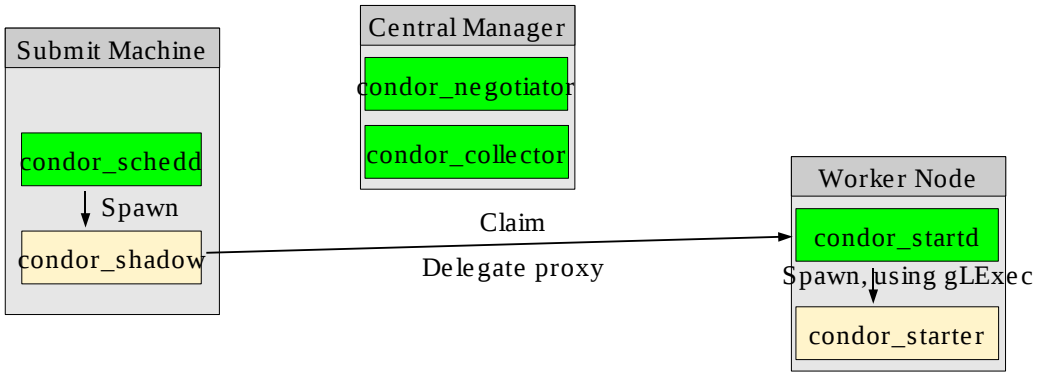

Figure 2.3a: Submit machine claiming a worker node with gLExec

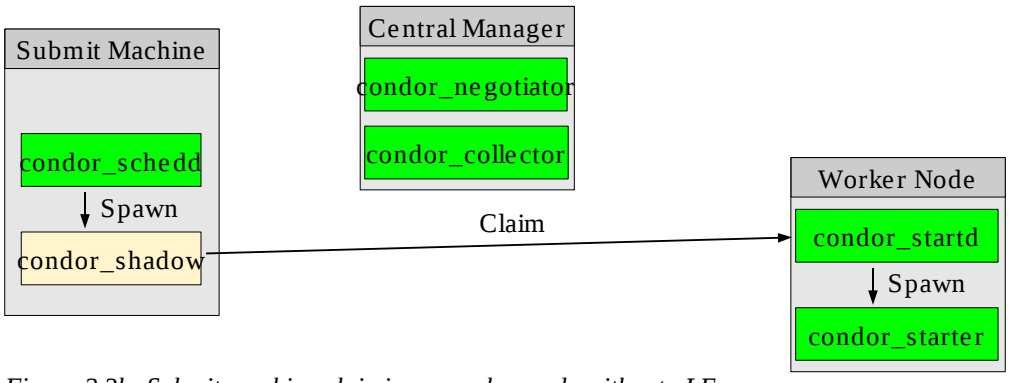

Figure 2.3b: Submit machine claiming a worker node without gLExec

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 job from the same **condor_schedd**, force a new matchmaking cycle 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.

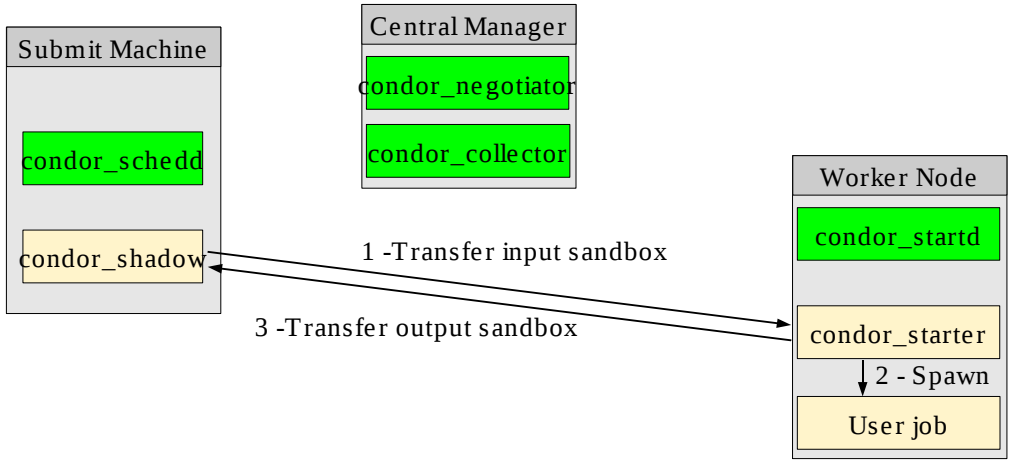

Figure 2.4a: Condor_starter handles the user job

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.

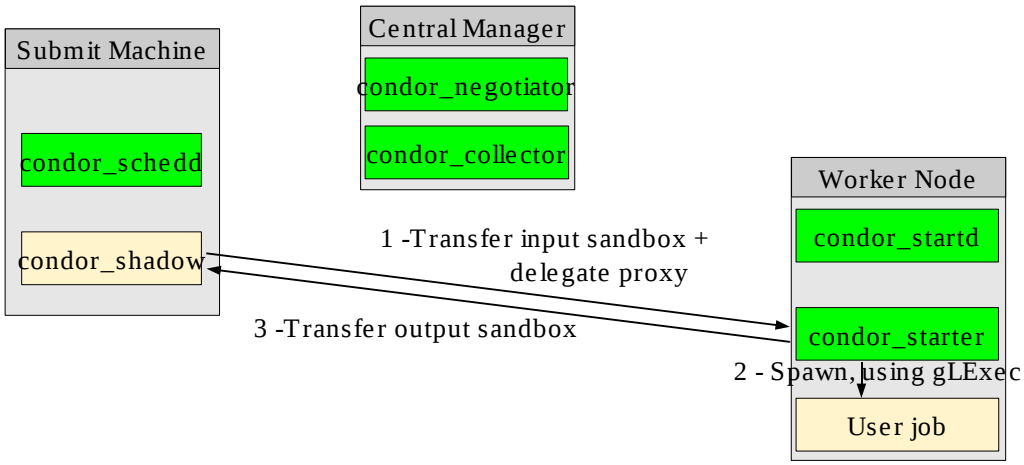

Figure 2.4b: Future condor_starter setup

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.

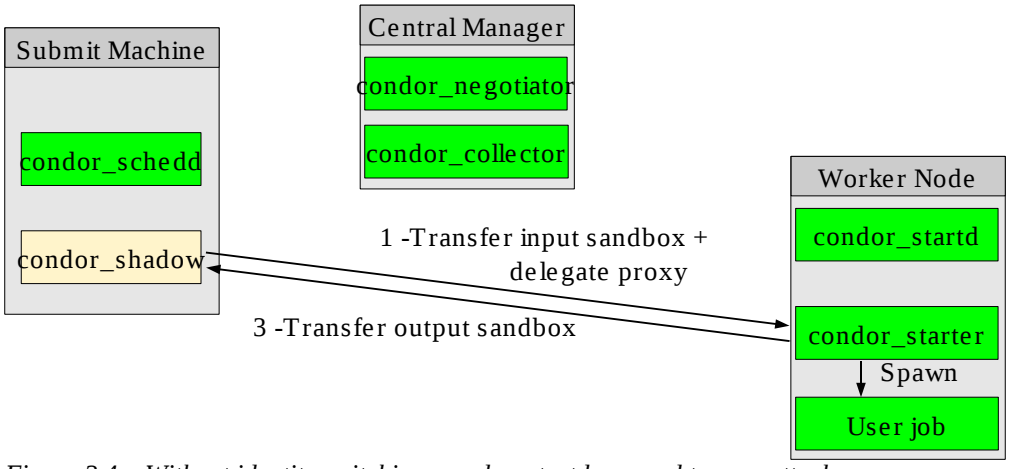

Figure 2.4c: Without identity switching, condor_startd exposed to user attacks

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 **glideinWMS 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.

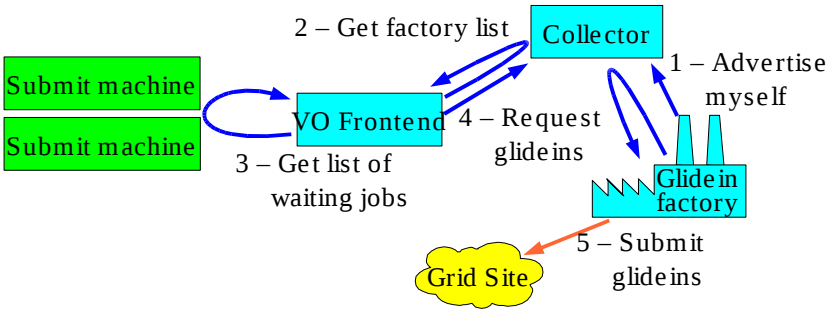

Figure 3: Glidein handling

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; the standard Grid mechanisms are thus used for the transport of the glidein payload and glidein proxy to a worker node. Other mechanisms, like the gLite WMS, could be used but are not supported in the current implementation.

A glideinWMS glidein is 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.

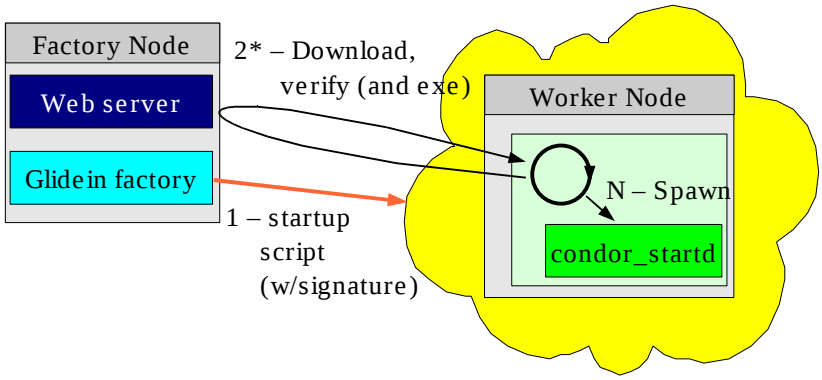

Figure 4: Glidein script overview

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, relies 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.

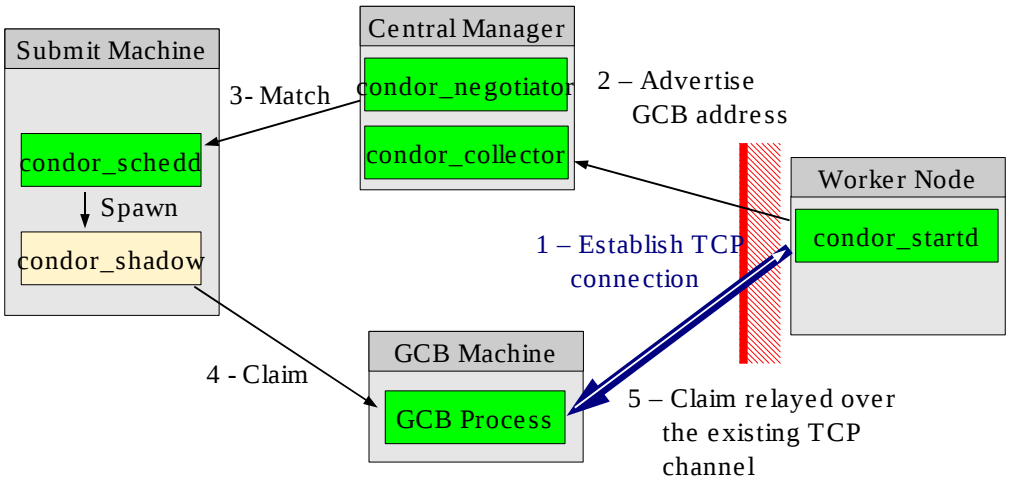

Figure 5: Condor use of GCB to traverse firewalls

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.
- Uploading 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.
- Using 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.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

Condor pool figures

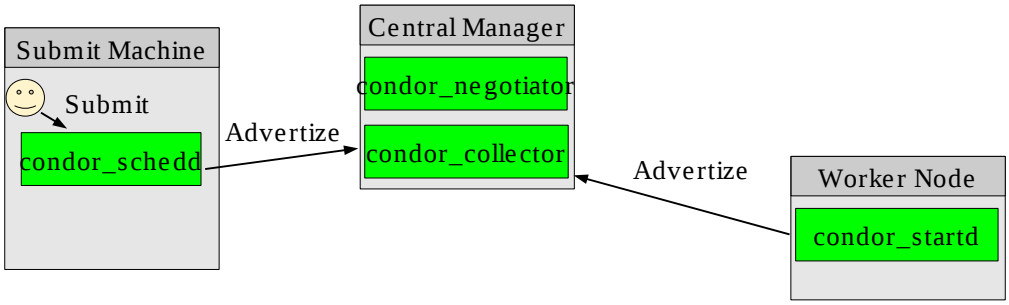

Figure 2.1: Central Manager defines a Condor Pool

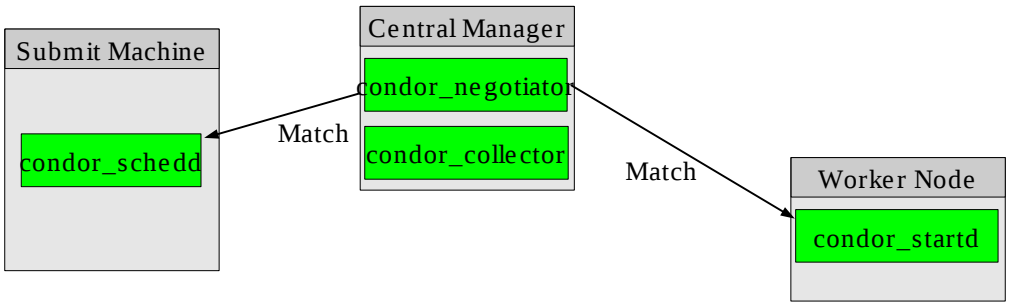

Figure 2.2: Central manager matches submit machines to worker nodes

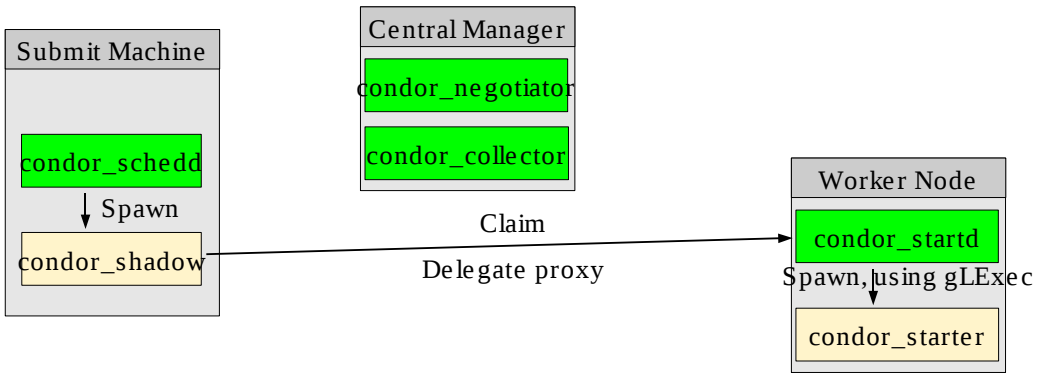

Figure 2.3a: Submit machine claiming a worker node with gLExec

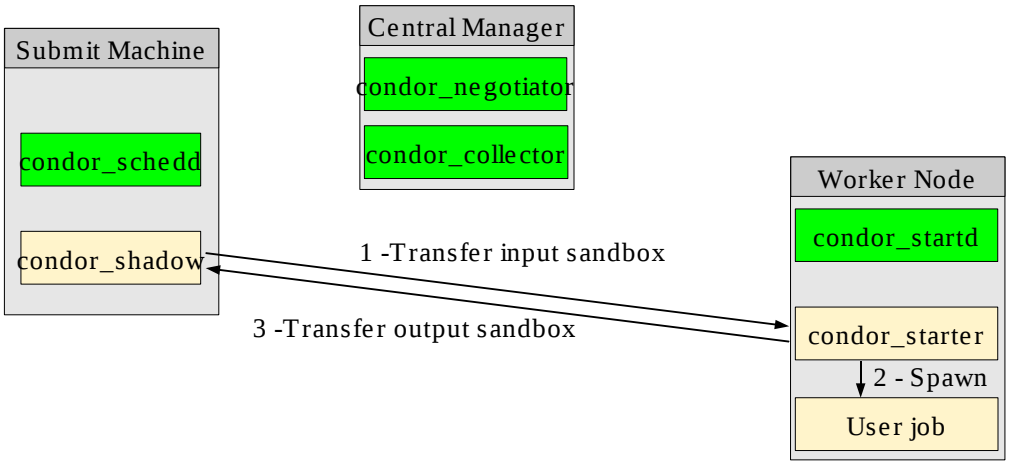

Figure 2.4a: Condor_starter handles the user job

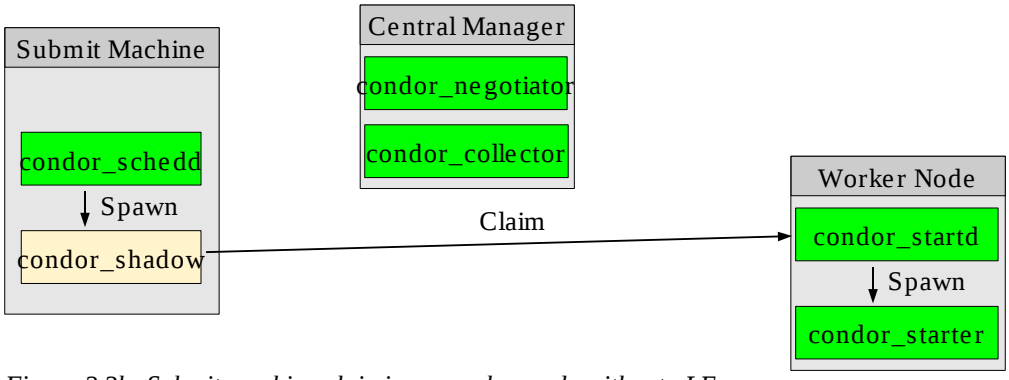

Figure 2.3b: Submit machine claiming a worker node without gLExec

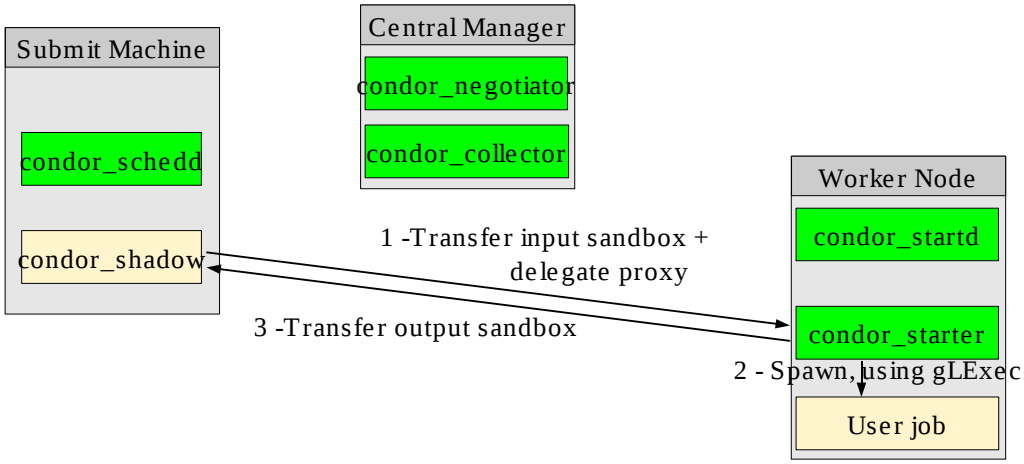

Figure 2.4b: Future condor_starter setup

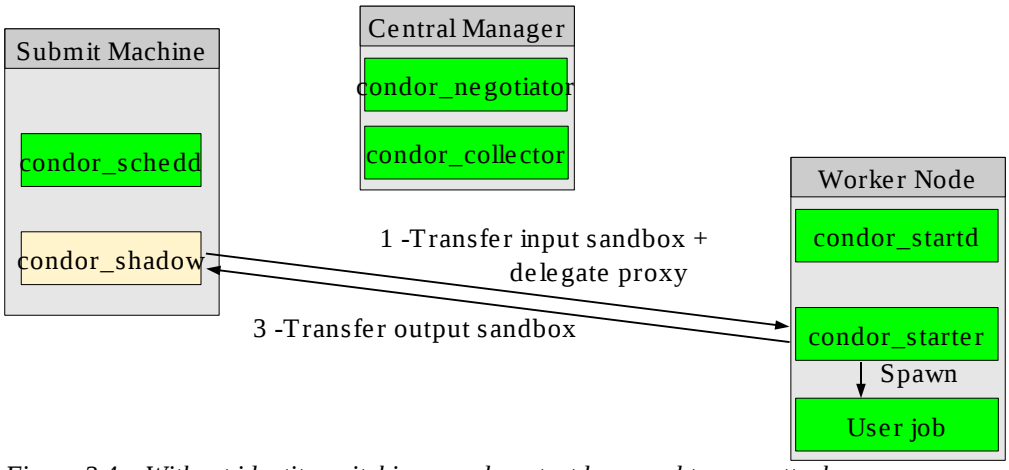

Figure 2.4c: Without identity switching, condor_startd exposed to user attacks