



Open Science Grid

Document Name	Compute Element for OSG 0.4.1
Version	0.3
Date last updated	March 17th, 2006
Authors	D.Bradley, T.Martin, S.Timm, T.Wenaus, F. Würthwein

Abstract: The purpose of this document is to present a set of configuration options for the pre-WS GRAM based Compute Element as available in OSG 0.4.1 to improve its robustness, especially for large clusters. Issues raised include configuration of mounts, and application best practices. We do not discuss GIP configuration issues.

Introduction.....	2
1 Problems with the OSG CE as deployed in OSG 0.2.1	2
2 Suggested Configuration to Minimize Problems	3
2.1 Export-free CE Installation	4
2.2 Managed Fork.....	6
2.3 Stdout & stderr.....	6
3 Addressing Remaining Weaknesses.....	6
4 Additional Suggestions and Best Practices.....	7
4.1 Site Administrators.....	7
4.2 Note on Privacy and Security.....	8

4.3	VO Application Developers.....	8
4.3.1	Use of disk areas.....	9
4.3.2	Best practices workflow.....	9

Introduction

With OSG 0.4.1 we are making an attempt at increasing the scalability of the standard pre-WS GRAM CE. This attempt is based largely on changes in configuration, rather than significant changes in the middleware stack. However, it does impact what applications can, and can not do, and thus needs to be augmented with an explanation of “best practices” for running applications on this infrastructure. This document tries to give a brief overview, pointing to additional more detailed instructions elsewhere.

1 Problems with the OSG CE as deployed in OSG 0.2.1

There are crudely speaking the following types of operational problems with the OSG CE:

- The pre-WS GRAM keeps its state in form of processes on the CE. Unless a user uses condor-g and turns on the grid monitor, significant system resources are consumed by pending jobs.
- Prior to OSG 0.4.0, the preferred OSG installation combined the “classic SE” with the CE onto the same hardware. This tight coupling of CE & SE lead to large loads in one degrading performance, and thus increasing the failure rate of the other. In addition, some installations added all the various disk areas used in OSG on the same filesystem such that overflow of one leads to overflow of all. Disk space exhaustion in \$DATA thus can bring down the CE because \$home/ runs out of space, as is discussed in more detail below. Starting with OSG 0.4.0 we strongly encourage a clean separation of “classic SE” and CE onto separate hardware. In addition, separate filesystems for \$APP and \$DATA are suggested.
- Prior to the separation of CE and SE onto different hardware, it was not uncommon for users to use \$home on the CE to deposit files. This was never explicitly supported nor intended as explained in detail below. It was possible at most sites given the way sites were commonly configured, and has let to CE overloads as explained below. We propose that people explicitly make \$home on the CE inaccessible to users as explained in detail below.

- “Guns” handed to users to kill the CE:
 - > Forking of jobs on the CE is neither scheduled nor restricted.
 - > Streaming of stdout and stderr via the CE is neither scheduled nor restricted. Turning on streaming disables the grid monitor in condor-g and thus leads to scalability problems even with zero stdout/stderr.
 - > NFS read/write exports from CE to large clusters opens up the possibility for applications to bring down the CE via massive parallel NFS accesses. This is made worse by the implicit requirement of a “user \$home” area that is shared across the cluster.
 - > There are GRAM failure modes, e.g. disk space exhausted in \$home/... area of a user, that will cause processes to be orphaned on the CE which can cause system overloads. As this disk area is used for spooling of stderr & stdout, storing of user executables, and read/write accessible from the worker nodes, and it’s space is unmanaged, a user can bring down a site’s CE by filling up this space inadvertently. As the total size of this space per user at a site is nowhere advertised, it is impossible for users to behave themselves even if they wanted to.

Some of these problems have been resolved already in the OSG 0.4.0 release. E.g., it is recommended (but not required) in OSG 0.4.0 to separate CE and “classic SE” onto different pieces of hardware. This eliminates the gridftp server on the CE, and thus the possibility to gftp files into \$home via the WAN. In addition, application developers are discouraged from using fork to submit gftp clients to pull data into \$DATA, \$APP, or \$home. Instead, a push to the SE is preferred to avoid loading both CE and SE at the same time. The \$home area should never be used to store files!

Additional steps are taken in OSG 0.4.1 to eliminate these problems further. All of these steps are optional. There’s nothing to prevent a site from deploying OSG in such a way as to maximally inflict pain on administrators and users of the site.

2 Suggested Configuration to Minimize Problems

Figure 1. shows a schematics of a suggested site installation. In the present section we discuss this schematic, and point out features towards a resolution of the problems listed in the previous section. Figure 1. identifies three separate pieces of hardware referred to as “UI” for “user interface”, “CE” for “Compute Element”, and “WN” for “worker node”. The UI is the location from which the user submits jobs to the site, using condor-g. It is shown for completeness but outside the scope of this document. In addition to UI,

CE, and WN, the figure shows three disk areas, that should ideally be hosted on file systems separate from each other, and certainly separate from the CE hardware.

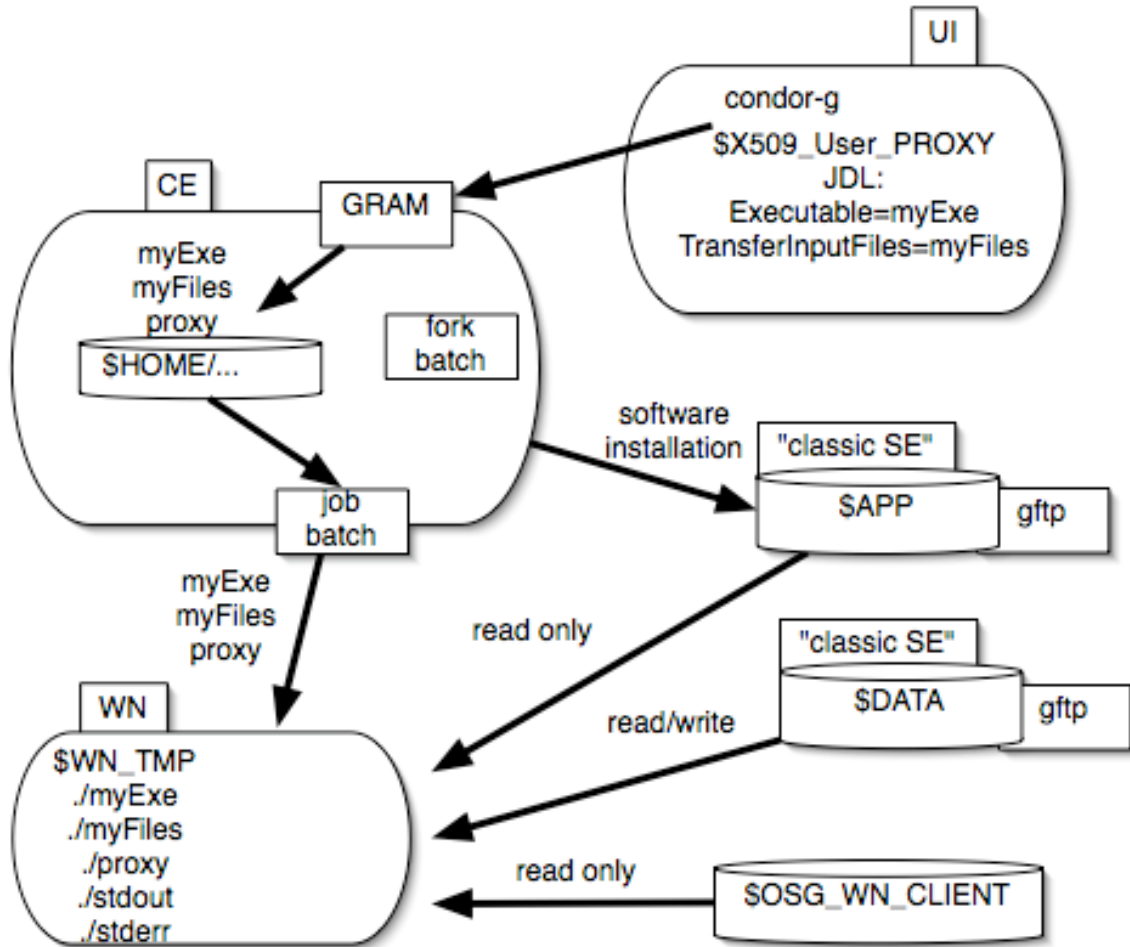


Figure 1. Schematics of a suggested site installation.

2.1 Export-free CE Installation

We suggest a CE installation without any NFS exports. The three NFS exported areas `$APP`, `$DATA`, and `$OSG_WN_CLIENT` are all hosted on a separate fileserver, and depending on size of the cluster, possibly more than one file server. The

\$OSG_WN_CLIENT area could in fact be installed on the worker nodes themselves, as its content is static, and under the control of the site.

In OSG 0.4.0, there are the following reasons for NFS exporting CE disk areas to worker nodes:

- The executable, proxy, and a list of auxiliary files transferred from the UI as part of job submission are made available to the WN at runtime via NFS mounting of \$home/.
- Stdout & stderr are available for streaming, as well as end-of-job export to submission client via NFS mounting of \$home/.

For OSG 0.4.1, we propose that \$home is not longer exported to the worker nodes. Streaming is thus fundamentally no longer supported. Instead, the required files should be transferred to the worker nodes via processes inherent to the site's batch system. Explicit instructions for how to do this for sites using condor as batch system are provided elsewhere [noNFS]. As this is not a requirements on sites, we do not think it is necessary to have this capability for all possible batch systems. As sites with different batch systems figure out how to do this, they are of course encouraged to provide explicit instructions in analogy to [noNFS].

We propose to adopt the following conventions for sites that implement export-free CEs:

- \$WNTMP is a dynamically created directory, created only for the duration of the batch lease, and deleted in its entirety by the batch system when the user application vacates the batch slot. This implies that \$WNTMP is resolvable as absolute path only on the worker node. We propose that sites indicate this in gridcat by setting "\$WNTMP" to "\$OSG_WN_TMP".
- The user application should always be launched in \$WNTMP.
- The user application should expect all files transferred to the worker node as part of the job submission process to be copied into \$WNTMP.
- Users need to be able to operate transparently across sites that use NFS exports and sites that don't. This implies that the following environment variables are set:

```
X509_USER_PROXY=$OSG_WN_TMP/x509_up
SCRATCH_DIRECTORY=$OSG_WN_TMP
```

The first of these is needed to point to the proxy file. The second is normally pointing to the gram_scratch directory in \$home on the CE. The GRAM copies the files transferred by condor-g into this location on the CE. By setting these two variables to point to the standard environment variable for \$WNTMP, we thus follow the same environment variable conventions whether or not \$home is mounted on the worker nodes.

2.2 Managed Fork

We suggest to replace the fork jobmanager with a new version referred to as “managed fork”[mfork]. Managed fork prevents overloads by limiting the maximum number of simultaneously executing fork jobs. We suggest to allow about 50 simultaneous fork jobs. The condor grid monitor itself that’s needed for successful operations of the pre-ws GRAM is presently running a fork job, and the existing implementation of the managed fork [mfork] does not distinguish between grid monitor and other fork jobs. It is thus suggested to dial the number of allowed fork jobs as high as possible without compromising CE functionality.

Large sites may in addition run fork jobs on hardware separate from the CE. If they choose to do so then this has to be transparent to users. This implies that the complete directory structure visible by a fork job that executes on the CE, also has to be visible from this separate hardware via the appropriate mount points. We do not know of any site that implements this at present. However, there’s nothing in the OSG deployment rules that would prevent sites from doing this. In fact, this might be considered a reasonable alternative to the managed fork available with OSG 0.4.1 [mfork].

2.3 Stdout & stderr

Stdout & stderr streaming is fundamentally not supported any longer, and jobs that require it may be denied access to the site. We propose that the site may impose a limit on the size of stdout/stderr that it supports for transfer, and that that limit may be zero.

3 Addressing Remaining Weaknesses

Even after the site is configured as suggested in the previous section, there are two remaining weaknesses, both of which are related to submission file transfer. The user executable presently is transferred for each and every job submitted, despite the fact that many applications basically submit the same application, sometimes with slightly modified arguments, 100’s or 1000’s of times to a site. As a result, the total amount of disk space consumed by these applications can add up. In addition, the stdout & stderr files which are transferred back to \$home/... after the application vacates the batch slot can add up. If the space in \$home fills up then GRAM processes will hang, possibly leading to an overload of the CE.

We propose that sites provision sizeable amounts of disk space for \$home/... on the CE to minimize the risk of disk overflows. We propose that sites clean up \$home/... , and not rely on anybody else for this cleanup. In principle, the GRAM itself cleans up this area. However, it is not uncommon for this cleanup to fail. As a result, we consider it mandatory for site administrators to regularly clean up \$home/... .

We suggest that applications stage out their own stdout/stderr together with anything else they want to keep. This may be done either via \$DATA, srmcp to the site's SRM, or directly from the worker node to some VO controlled location outside the site.

It is worth pointing out in this context that there are two little known fields in the GLUE Schema that specify the network topology of a site. Their default values as specified in osg-info-static-cluster.conf are:

```
GlueHostNetworkAdapterOutboundIP    TRUE
GlueHostNetworkAdapterInboundIP     FALSE
```

This implies that a default site is assumed to have an open network outgoing, and a closed one incoming, as is the case for a typical public/private network boundary with a NAT. Site administrators are expected to set these attributes appropriately such that users can rely on them.

We suggest that the GOC will generate tickets if these attributes are not set correctly. This implies that testing for network connectivity from the worker node becomes part of the site functional test.

We note that some CE's on OSG, e.g. FNAL_FERMIGRID, provide an entry point to multiple sites. Entering via such a CE does not guarantee that the same \$DATA and \$APP is visible from all worker nodes.

4 Additional Suggestions and Best Practices

In this section we present additional suggestions for site admins and users, some of which go beyond the ones presented in previous sections.

4.1 Site Administrators

We strongly suggest to provision at least two separate pieces of hardware. One to implement the CE, and one as "classic SE" and fileserver to export \$DATA, \$APP, and \$OSG_WN_CLIENT to the compute cluster. There should be no exporting of disks required from the CE.

The classic SE hosts the gftp server.

The CE mounts \$APP, \$DATA, and \$OSG_WN_CLIENT in the same way as the worker nodes. It is thus possible to verify proper installation of software in \$APP via a fork job. Sites are not required to allow write access to \$APP via fork jobs for all users.

No space management or quotas of any sort are expected on \$DATA or \$APP by the applications. \$DATA may be subjected to cleanup of old files. For \$APP this must be avoided as application software installation may include files that are rarely accessed. It

would be disastrous for applications to loose some rarely accessed files within a VOs software release, as this might alter application behavior in ways that are not easily detectable! \$APP may thus NOT be subject to automatic cleanup scripts by the site administrator!

Installation of software on \$APP are best handled by providing role based access to a batch slot that is dedicated for this purpose on the fileserver that exports \$APP. A simple example implementation of this is to have a special batch queue that keys in on the uid a user is mapped to. This is in production at UCSanDiegoPG since OSG 0.2.1. Sites may provision one such batch queue for each of their primary client VOs plus another one shared among all other VOs allowed on the site. The queues then send their jobs to dedicated batch slots on appropriate pieces of hardware. The intention here is two-fold. First, write access to \$APP should be allowed to a privileged subset of the VO users, the VO admin, rather than all VO users. Second, installation can be time and resource intensive. It is desirable to not have this activity happen as fork jobs on the CE.

4.2 Note on Privacy and Security

We note that the use of group accounts is still widespread on OSG, despite the fact that there are technical solutions available to avoid them. We point out that members of a VO that uses group accounts have access to each other's proxies. Combined with the practice of long lived proxies, we consider "identity theft" an unnecessary risk we currently live with.

There are several ways to eliminate this risk. The simplest way might be to eliminate exports of \$home/... to the worker nodes, and move fork jobs off to a separate piece of hardware that does not have access to \$home/... . In such a configuration opportunities for "identity theft" are eliminated significantly even for group accounts because a running job has access only to proxies for people running under the same group account on the same piece of hardware at the same time.

Of course, ideally sites should move to pool accounts, and VOs should use roles to avoid this problem altogether. These two are coupled because the site admins need to know which role within a VO should have write access to \$APP. Site admins tend to use group accounts for all members of a VO if they have to provide write access to \$APP for all members of the VO. VOs who don't use roles, or communicate their operational intentions otherwise, thus take unnecessary security risks.

4.3 VO Application Developers

In the present section we describe what we consider "intended use" of the various features of OSG. This is only an initial set, and by no means the only way to use the OSG infrastructure. Comments are more than welcome!

4.3.1 *Use of disk areas*

- **\$APP:** This area is meant for application software installations that have utility beyond the lifetime of a single job, and for many users. It is mounted on all worker nodes as a read only filesystem. Installations are performed ideally by a specially privileged VO-administrator via the normal jobmanager using a special role. However, only very few sites support this type of installation at this point, and only few VOs use roles. More commonly, \$APP is mounted read/write on the CE, and installations are done via fork jobs. The files needed for software installation may be placed in \$APP via the WAN using gftp, or pulled by installation jobs submitted via the CE. Most sites provide write-access to \$APP to all DN's allowed at the site, and only via fork jobs and gftp. The LocalStorageConfiguration instructions for OSG 0.4.0 specify "At least 10GB of space should be allocated per VO" in \$APP. At present, there is no way to know which sites support what forms of \$APP installation, short of asking the site admin, or reading the site policy statement, and hoping that this is described there. This disk area is expected to exist long term in OSG.
- **\$DATA:** This area is used for staging data in and out. It is therefore read/write accessible from the worker nodes, and accessible via the WAN by gftp. There is generally no space management in \$DATA, and some form of automatic clean-up is generally in place. This disk area is going to be replaced long term with SRM controlled storage elements. Documentation for OSG 0.4.0 indicates that a VO should expect 10GByte times the number of worker nodes.
- **\$SRM:** More and more sites support SRM controlled storage elements. The vast majority of disk space on the OSG is only accessible via these SEs.
- **\$home:** At present, many sites still allow gftp access to \$home on the CE, and export this area to all worker nodes. Nevertheless, we strongly discourage the use of this area by the application!
- **\$WNTMP:** This is the primary work space for the application at run time.

Typically, the space available is best described as \$APP << \$DATA << SRM-SE. SRM-SE's typically hold many tens of TB's, most of which is only accessible to some VOs.

4.3.2 *Best practices workflow*

A simple but typical workflow will include the following parts:

- Install re-useable applications into \$APP via a combination of gftp and installation job submission.

- Stage data into \$DATA using gftp.
- Stage user & job specific application libraries etc. into \$DATA.
- Submit a very small script, O(100kB), for each job via the GRAM.
- The script establishes a work space for the job inside \$WNTMP by unpacking applications and data from \$DATA or SRM-SE and softlinking libraries from \$APP. The script then starts the executable. After completion of the executable, the script cleans up, storing all output data, including stdout/stderr if desired, to \$DATA or the SRM-SE. The script deletes everything in \$WNTMP before vacating the batch slot.