

Best Practices for Using the SLAC Batch System

Best Practices for Using the SLAC Batch System

Version of 6/8/2015

Contents:

- [Shared Resources](#)
- [Known problems to avoid](#)
 - [PFILE \(and other\) Simultaneous File Writing Conflicts](#)
 - [Core files](#)
- [Minimizing stress on file servers](#)
 - [General Guidelines for Using Remote File Servers](#)
 - [Local Scratch Space](#)
 - [Monitoring remote file servers](#)
 - [What to be alert for...](#)
- [Summary](#)

([Skip to the bottom](#) of this page for a concise reminder of all Best Practices.)

Shared Resources

When you run batch jobs, you will be sharing various computing resources with your Fermi colleagues and the rest of SLAC.

Shared Resource	Use	Notes
batch machines	LSF general queues	kiso, dole, bullet
interactive login machines	light load, short-running, development/test	rhel6-64, centos7
NFS disks	site-wide, medium performance	Fermi group and user space, in general, NOT backed up
AFS disks	global, medium performance	\$HOME directories for all users, these areas are backed up
xroot disks	site-wide, high performance	Fermi storage for bulk data <ul style="list-style-type: none">• read by everyone• write restricted to pipeline and science groups
network facilities		switches & fabric

It is your responsibility to assess the impact of any significant computing project you wish to run: to ensure it will not unduly stress the system or make it unusable for other users. Such an assessment may start with running successively larger numbers of jobs while carefully monitoring the impact on key servers. In addition, there are some known problems that one must take care to avoid from ever happening. This document attempts to provide some hints on preparing your batch jobs and assessing their impact.

One of the worst scenarios is when someone submits a large number of untested batch jobs and then leaves to go home for the weekend. When the alarms begin to sound, many people will get involved, even awakened in the middle of the night, to investigate. Ultimately, all of the batch jobs may be killed and, in some cases, the user may be prohibited from submitting new jobs until an understanding has been reached. Please try to avoid this sort of situation.

Note that submitting jobs directly to the batch system via 'bsub' and running jobs in the Fermi Pipeline may have slightly different approaches to solving the following potential troubles. The rest of this document will assume individual batch jobs submitted outside the Fermi Pipeline.



Start faster

Please also see [this page](#) to learn how to get your **batch** jobs to start running sooner.

Known problems to avoid

PFILE (and other) Simultaneous File Writing Conflicts

Parameter files, "PFILES", are used by the Fermi ScienceTools, and FTools to store default and last-used options for the commands within these packages. Normally, these small files are stored in \$HOME/pfiles and are written *each time* a command is invoked. If multiple jobs attempt to access these files simultaneously, an unfortunate and painful conflict will result. Not only will your jobs fail to give reliable results, but this sort of activity is very demanding on file servers and can cause severely degraded performance for all users.



This problem may also occur with other files, typically to "dot file" or a "dot directory" in your \$HOME directory. Therefore, it is good practice to redefine \$HOME to a non-shared scratch directory for all projects requiring multiple, simultaneous batch jobs.

Therefore, PFILES should be written to directories which are unique for each job, e.g.,

- Create a unique directory in /scratch for your batch job,

```
mkdir -p /scratch/<userid>/${LSB_JOBID}
```

- Define this directory as your \$HOME and then go there prior to running any ScienceTools/Ftools/etc.,

```
export HOME=/scratch/<userid>/${LSB_JOBID}
cd ${HOME}
```

This will automatically take care of PFILES being unique for your job and avoid overloading the /nfs user disk with large numbers of opens and closes. Create any new files in \$HOME and then copy anything you wish to save at the end of your job.

- Cleanup the scratch directory at end of job (after you have copied out anything you want to save),

```
rm -rf /scratch/<userid>/${LSB_JOBID}
```

Note: Cleaning up the scratch directory is critical! Any scratch file left behind will slowly fill up the /scratch partition and eventually fill it up. (On the batch machines, the local scratch spaces are managed and cleaned up periodically, but on a very long time-scale and much too long to prevent them from filling up!)

Core files

Core files may be produced if your executable crashes. These files can be huge, and are typically written to \$HOME, which if you have taken to heart the preceding section, will reside on a shared NFS server, but equally bad on an AFS server. Imagine the load on a file server if hundreds of jobs suddenly crash and all attempt to simultaneously write multi-GB core dumps. Not only can this bring a file server to its knees, this problem can be very difficult to halt once it gets started. For this reason, and the fact that hundreds of core files are almost never useful, it is strongly urged that core dumps be disabled or severely limited in size for all batch jobs (other than individual **test jobs** from which a core dump may actually be of some use). The appropriate line(s) from the table below should be placed at the beginning of your batch scripts.


Shell	command
bash	ulimit -c 0
csh	limit coredumpsize 0
python	import resource resource.setrlimit(resource.RLIMIT_CORE, (0,-1))

One can sometimes get a traceback from a truncated core dump file, but not a whole lot more, so the favored approach is to disable the core file completely.

Minimizing stress on file servers

The single largest point of stress is I/O overload on the file servers. There are many ways to stress a server. Different servers will "run out of steam" at different thresholds depending on # of CPUs, CPU speed, amount of memory, I/O system, number of network connections, disk speed/buffering/controller details, RAID striping, etc.

The number of simultaneous jobs that may be run without causing severe stress will vary depending upon exactly what the jobs are doing. For example, some jobs perform heavy I/O at the beginning and end, while others perform I/O continuously. Every job is a bit different and so requires its own assessment.

 Any large task of more than a few 10's of batch jobs must be ramped up slowly in order to allow for monitoring the relevant servers for adverse impact. Additionally, some tasks may require you to trickle in jobs rather than submitting them as a large batch to prevent overloading of, for example, the Gleam and/or ScienceTools code server.

General Guidelines for Using Remote File Servers

The most basic rule is to avoid prolonged I/O to a remote file server. (The one exception is xroot which seems able to handle very large loads of this type.) This includes both file **reading & writing** as well as directory operations, such as **creating, opening, closing, deleting** files. A good way to design your job is to copy needed input files to local scratch space for reading, and to write output data products to local scratch, then copy to a remote file system at job completion.

Exceptions to this rule include: reading files from xroot (no need to copy them to local scratch space first); the main executable itself (usually one or more compiled programs); and reading of a small number of small files a small number of times during the job execution (e.g., your .cshrc, and other one-time configuration files stored remotely).

These are **very general** guidelines and should be taken with a large grain of salt. While they will not guarantee good performance, think of this as a checklist of possible problem areas. In this context, 'reading' and 'writing' means a file is open for an extended period of time, i.e., the duration of the job. In contrast, **copying** a file from local scratch to /nfs or /afs (or vice versa) is typically a much less stressful approach for the file servers.

operation	local scratch	xroot	NFS	AFS	Notes
writing large files (>100MB)	✓	✓	✗	✗	
reading large files (>100MB)	✓	✓	✓	✗	
writing small files (<100MB)	✓	✓	✓	✓	okay only in small numbers (NFS/AFS)
reading small files (<100MB)	✓	✓	✓	✗	
copying whole files (local<->remote)	✓	✓	✓	✓	typically at job start and/or end
frequently creating/opening/closing/deleting files	✓	✗	✗	✗	best to avoid this completely
frequently stat'ing files	✓	✗	✗	✗	
multiple jobs writing to the same file	✗	✗	✗	✗	don't do this!

Notes.

1. xroot is the repository for Fermi on-orbit data and Monte Carlo data. It is readable by anyone, but typically not writable except by pipeline accounts.
2. NFS refers to the collection of servers dedicated for Fermi use. Typically one server (machine) has multiple disks attached, so that stressing a server can cause a problem for multiple groups of users.
3. AFS is the filesystem used for user \$HOME directories and a relatively small collection of Fermi group areas. This is a world accessible file system (if one has proper access credentials) and caches file on local machines.

Finally, keep track of how much space is available on the directories/partitions you write to. Writing to a 100% full partition is known to cause a lot of stress on the file server. It is easy to check the available space: cd to the directory of interest and then issue the "df -h ." command, which will tell you the size and remaining space on that partition. (Less frequently, one may encounter a different limit: inodes. Check your inode quota with the "df -hi ." command. This quota runs against the sum: number of files + directories + symlinks)

Local Scratch Space

Local scratch directories are available on all SLAC linux machines. They vary in size from several GB to several 100 GB. This space is shared by all users logged into a given host. On batch machines, it is vitally important to **clean up** this space at the end of your job or it will, over time, fill up (and this has happened). Common practice for using scratch space is to create a directory with your username and put all files in there. Note that if using the Fermi pipeline to manage your job flow, you will need to devise a 'stream-dependent' method of naming your scratch sub-directories to prevent jobs running on the same host from overwriting each other's files.

Machine type	Local scratch space location
batch	/scratch
interactive (e.g., rhel6-64)	/usr/work
desktop	/scratch (usually)

The newest class of batch machine, bullets, have a high-performance *global* file system called Lustre. Fermi pipeline jobs have access to additional scratch space only on these machines via the path `/lustre/ki/pfs/fermi_scratch`. It is hoped that at some future date a similar global scratch space will be opened up to all users.

Local scratch space must be manually cleaned up at the end of the job, `'rm -rf $MYSCRATCHSPACE'`

Finally, note that all linux machines have a `/tmp` disk partition. It is strongly recommended that `/tmp` **NOT** be used because of the danger of its becoming full which will cause the machine to crash.

Monitoring remote file servers

First, one must identify the **server** holding all of the job's needed input and future output files.

1. 'cd' to the location holding the existing or future file
2. 'df .' this give tell you one of three answers:
 - a. AFS

```
Filesystem      1K-blocks    Used Available Use% Mounted on
AFS              9000000         0   9000000    0% /afs
```

This tells you it is an AFS server. Then, follow up with the command `'fs whereis .'`, e.g.,

```
File . is on host afs03.slac.stanford.edu
```

The server is afs03

- b. old NFS (single wain-class server)

```
Filesystem      1K-blocks    Used Available Use% Mounted on
wain025:/g.glast.u55/dragon
10485760 3731456   6754304   36% /nfs/farm/g/glast/u55/dragon
```

This is an NFS location and the server is wain025

- c. new NFS (GPFS as underlying file system with clusteredNFS 'heads', as of May 2015)

```
GPFS

Filesystem      Size    Used Avail Use% Mounted on
fermi-cnfs1bl1:/gpfs/slac/fermi/fs2/u
10G   3.6G   6.5G   36% /nfs/farm/g/glast/u
```

This is also an NFS location, but one must monitor four different servers: fermi-gpfs03, fermi-gpfs04, fermi-cnfs01, fermi-cnfs02

- d. local

```
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda1       18145092 8530464   8692904   50% /
```

This is a local disk – *and will not be visible to any batch jobs*

Next, armed with a list of needed servers, you may monitor their performance with a web-based tool called Ganglia. Select the server(s) of interest and set the time interval to the minimum (1 hour). You may have several of these web pages open at once. Here are short-cuts to three storage domains used by Fermi. Once on this page, click on the specific server to be monitored.

- Fermi NFS, http://ganglia.slac.stanford.edu:8080/ganglia/glast/?c=nfs-glast&m=load_one&r=12_hours&s=descending&hc=4&mc=2
- SLAC AFS, http://ganglia.slac.stanford.edu:8080/ganglia/fileservers/?c=afs&m=load_one&r=hour&s=by%20hostname&hc=3&mc=2

- Fermi xrootd, http://ganglia.slac.stanford.edu:8080/ganglia/glast/?m=load_one&r=3_days&s=by%2520name&c=glast-xrootd&h=&sh=1&hc=4&z=small

The Ganglia pages display strip charts for CPU usage and I/O rates as a function of time. Here is an historical image of the Fermi user/group space disk, [wain025](#), Ganglia page's first few plots. The third plot shows CPU time. User jobs should avoid a situation where the "System CPU" exceeds about 50%. Greater loads will begin to cause file-access delays, then timeouts and, eventually, the machine will hang or crash. Now take a look at [wain025's performance](#) at this very moment. In addition to the top five plots (as shown in the historical image), you can scroll down to see activity on each user and group partition on the server. Near the very bottom, keep an eye open for the "nfs_server_badcalls" plot; experience has shown that any entries at all in this plot is a **very bad thing**.

What to be alert for...

- CPU utilization > 50% (especially "System CPU")
- NFS disk I/O > 30 MB/s
- AFS disk I/O > 5-10 MB/s
- xroot disk I/O >> 200 MB/s (wains only)

Summary

- **Store analysis code and scripts** in your AFS home directories (which are backed up)
- **Assessment.** For every new task, assess its impact on key servers to ensure they will not be overloaded
- **File staging.** Files that remain open for the duration of the job (either reading or writing) should be located in local scratch space. Copy needed input files to local scratch at the beginning of your job; write output data products to their final destinations at the end of the job.
- **Submitting jobs.**
 - **Never** submit a large number (~>50) jobs without first assessing their impact on key shared resources.
 - If your jobs are known to produce a large I/O load only during the start-up phase, then submit jobs in small batches, wait for those to run and pass the start-up phase and only then submit another small batch, etc.
 - If you are planning a large batch operation of, say, more than 50 simultaneous jobs, please inform and coordinate with SAS management ([Richard Dubois](#)).
- **PFILES.** Arrange that the parameter files for ScienceTools, FTools, etc. be stored in a directory *unique* to the batch job.
- **Avoid Disk Thrashing**
 - Completely disable core dumps
 - Avoid unnecessary file open() and close() operations, as well as file creates/deletes.
 - Avoid writing to a full disk partition.
- **Cleanup.** Be sure to **perform a cleanup** on the local scratch space after your jobs have completed!