LA-UR- 10-03823

Approved for public release; distribution is unlimited.

Title: Xgrid Admin Guide

Author(s): Charlie E. M. Strauss

Intended for: Documentation of LANL xgrid server and for general public education on how to set up a distributed grid server.

Los Alamos
NATIONAL LABORATORY
EST. 1943

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Form 836 (7/06)

Xgrid Admin Guide
Charlie E. M. Strauss

Topics

Contents

What is Xgrid?

Xgrid, with a captial-X is the name for Apple's grid computing system. With a lowercase x, **xgrid** is the name of the command line utility that clients can use, among other ways, to submit jobs to a controller.

An Xgrid divides into three logical components: Agent,Controller and Client. Client computers submit jobs (a set of tasks) they want run to a Controller computer. The Controller queues the Client jobs and distributes tasks to Agent computers. Agent computers run the tasks and report their output and status back to the controller where it is stored until deleted by the Client. The Clients can asynchronously query the controller about the status of a job and the results. Any OSX computer can be any of these. A single mac can be more than one: it's possible to be Agent, Controller and Client at the same time. There is one Controller per Grid. Clients can submit jobs to Controllers of different grids. Agents can work for more than one grid.

Xgrid's setup has a pleasantly small palette of choices. The first two decisions to make are the kind of authentication & authroization to use and if a shared file system is needed. A shared file system that all the agents can access can be very beneficial for many computing problems, but it is not appropriate for every network.

Related XGRID FAQ

- Do I need a powerful computer to run a controller?
- Do I need to install software?
- Do I need a Mac OSX server to be a controller?

Xgrid for Linux

A non apple-created work-alike has been made for Linux.[1]

Planning the Security Envelope

On any Xgrid network the Agents are placing a certain level of trust in the Clients since resources on their computer are in use. There are different degrees of trust granted but there is no purely zero-trust level. A second issue is the authentication required to prevent uninvited bad guys from submitting jobs or reading results. And the final issue is mitigating the damage from accidents, broken trust, and party crashers.

A major factor in choosing a setup depends whether the Xgrid manager will have routine admin access to the Agents. For example, on a dedicated cluster the Xgrid controller Admin is probably the also Admin of the cluster of agents. Whereas on a 'scavenger' grid (A.K.A 'distributed' grid) the xgrid manager will likely have no admin or user level access to the clients at all.

Authentication

Agents must make a choice: they can either advertise their availability to any nearby Controller via Bonjour or they can choose one, and only one, Controller by a hostname or an IP address entered into their preferences. (While exquisitely convenient for self-forming, almost administration-free Xgrids, the Bonjour method when combined with password identification, may be ill-advised on a non-trusted local network.)

Clients (mutually) authenticate to the Controller. The Controller (mutually) authenticate to the Agents. There is a choice between modes of authentication: password mode and Kerberos. One can use a blank password as well but usually this is for debugging since there is no simplification of the setup without a password.

Keberos

On appropriate networks Kerberos can be, by far, the most secure; but it may have limited applicability in many network configurations. Compared to password authentications, Kerberos is more difficult to set up and requires other network services beyond Xgrid. Kerberos is a system by which computers can delegate authentication of allowed users to a trusted kerberos server, typically part of a trusted LDAP or local DNS setup. Setting up security systems is considered a sophisticated sys admin task, but it is made significantly easy by Mac OSX servers which have a GUI process for LDAP and Kerberos. The agents, controller and clients, must also be set up by a qualified sysadmin to bind their authentication to the kerberos server.

Under Kerberos, jobs execute on Agents with the userid (or credentials) of the client that submitted the job. Thus they have the userid of the client and some elevated privileges on the Agent computer that may allow the jobs some enhanced capabilities through a looser resource sandbox (seatbelt) and file access permissions. If the Agents don't trust the Clients it may be unwise to give them elevated privileges.

Moreover, on some local networks this might be not useful or local IT policy might oppose kerberos. Specifically, in common unmanaged local networks individual computers authenticate their own user logins and passwords. For example, in scavenger grids composed of cycles donated by underutilized workstations, the workstation owners might not have a single trusted authority they would want to delegate user authentication to. Conversely, in highly managed networks the sysops may have already established a centralized kerberos system and might be averse to having an *ad hoc* kerberos or LDAP server on their network. Thus kerberos authentication is likely to be the natural choice only when the Xgrid manager and the Kerberos manager are in the same IT department.

Under Kerberos when the Controller is authenticating to the Agent it is actually doing so with the Client's credentials.

Open question: how does this works in a hybrid case where clients authenticate by password and Agents authenticate by kerberos. (Some online discussions seem to suggest that there is a mechanism to allow Agents to authenticate to the Controller credentials instead, but this does not seem to be documented.)

Password

Password security on Xgrid should not be confused with that of robust login protocols like SSH. Authentication of the controller to the agent is a fairly weak form of security because the controller uses the same password for all the Agents. This password is stored in a hashed form in a plain text file on all the Agents and so can be read by anyone with admin access on any agent. The file is identical on all agents: It's not salted or varied from agent to agent or signed in any way; the hash is not a secure hash and it can be inverted to recover an equivalent password. If these files are backed up by time machine to an external drive, they may be readable even by people without admin access. Thus many people have potential access to the password file. The situation is thus analogous to automated scripts that do FTP using hard-coded passwords in the scripts. During authentication these passwords are validated in a reciprocal, encrypted manner and are never sent over the network, but unlike SSH all other communication is not encryptend after authentication.

On the clients, there are different kinds of potential exposures of the controller password. Again all clients share a single controller password. To use the xgrid command the client must place the password on the command line or in an environment variable. Thus these passwords are often in clear text on screen and persistently lurking in the shell history, environment and bashrc. Notably, the third party client replacement app, GridStuffer, stores the client password in the keychain and thus is more secure.

The bottom line is it possible that a bad-guy could discover either widely shared password and use it to submit jobs. With the client password jobs could be submitted directly. With the agent password, on a bonjour set up the agent can impersonate the controller to submit jobs. With the agent password, where a specific computer is designated as the controller, then this bad-guy would also have to spoof the hostname or IP address of the controller on the local network. This last threat is less likely: behind router networks using a non-routable IP, it is usually difficult to spoof an IP from outside the router and so is only a threat from insiders on such a network.

Thus password authentication while not requiring additional services like LDAP and keberos to be configured on the Agents, has risks if the network, clients or agents are not trusted.

Setting Limits

As can be seen the authentication choices tend to be either too weak and too invasive/difficult. Apple has not yet offered a middle ground such as ssh-keys where the passwords can be kept securely and differ on every agent and client. Fortunately there is also an authorization layer to the security onion: IP filtering, unix permissions and the sandbox.

Unix permissions and the sandbox

First, in password mode, Xgrid agents always run jobs as user ""nobody"" who has very few system privileges and file access on properly configured computers. Thus one is taking advantage of the natural Unix permissions to limit capabilities by username. This introduces a high degree of safety as long as the unix permissions of the filesystem exist (e.g. FAT formatted thumb drives with lack unix permissions) and have not been misconfigured. Agents, would be wise to run "repair permissions" before activating xgrid.

Second and most important of all, xgrid jobs are run in a sandbox. The sandbox further lowers the system access privileges of the job. It can limit the job to seeing only certain parts of the file system and selectively remove write access regardless of unix permissions, and limit other system calls such as interprocess communications. In particular under password mode, the seatbelt file prevents access drives mounted in /Volumes which is where unprotected FAT32 thumb drives will be automatically mounted. Setuid executables cannot be executed, making privilege escalation security holes difficult. The Xgrid agent also reaps all forked children of a job, thus there is no way for a job to end and leave behind a daemon process that continues on.

The "seatbelt file" that defines the sandbox limits is more restrictive in password mode than in Kerberos mode. Agents may want to consider tightening the sandbox further.

Consequences of an attack

To summarize: to gain access to job submission an attacker must learn passwords and in some cases appropriately spoofIP addresses to allowed him to pretend to be a client or pretend to be a controller. To gain higher level access to an agent from a job would require some sort of privilege exploit that defeated both the unix permissions and the sandbox. To accomplish all of this simultaneously is a tall order and no known attack allowing that exists.

However gaining just the ability to submit jobs can be exploited for a Denial of Service attack (using up all CPU, memory, and network resources on the Agent to a point where the computer is effectively paralysed.) Since the Apple seatbelt allows network privileges it could be used as a base to launch network attacks on other computers. The Apple default sandbox seatbelts also allow a job to scan files like hosts and auto_master for attack planning reconnaissance to gather names of users, names other services and computers on the local network, and sometimes passwords.

Security Enhancements

Firewall

An important security feature is that all controller -agent communications are initiated by the agent not the controller. One should not open firewall port 4111 for inbound communication on the agent. Only the controller needs its firewall set for incoming connections. Similarly, there is a significant security improvement in not using bonjour for agents to prevent them from promiscuously offering themselves to any controller. This is also why it's a good idea for controllers not to also be agents.

TCP wrappers

Rather than relying on soley on passwords, you may want to limit client submissions to come from specific subnets, hostnames or IP Addresses. An easy way to do this is the built in TCP wrappers. First check /etc/services to make sure xgrid is a named service for port 4111. Then simply add to the controller's */etc/hosts.deny* :

```
xgridcontrollerd:  ALL:ALL
```

and then add to the controller's */etc/hosts.allow* :

```
xgridcontrollerd:  <list of subnets or IP>
```

However, you must include the agents on this list as well or they too will be cut off. Note: this also means that every agent could pretend to be a client if they know the client password. (It's a shame that apple did not use two ports to segregate clients and agent.) Never-the-less, on many grids the agents are managed locally along with a controller, and it is the possibility of rogue clients one is trying to protect against.

Client Tunneling

If you are willing to give clients a log-in userid on some proxy computer then you can set up real client authentication with per-client passwords or ssh-key files. Would be submitters enable themselves as clients by ssh-ing to that proxy computer and tunneling the connection to the controller computer.

```
ssh -f  -L 4111:controller.host.name:4111 proxy.host.name
```

On the controller, add a TCP wrappers denying to all but the proxy computer and the list of agents. Again note that all agents who know the client password can be clients without the need for an ssh login.

Agent Side

Tunneling for Agents is hard to automate so not generally won't be practical.

Agents currently have the preference setting to either take-all-comers or one specific IP. If you want something in between, set it for bonjour, but then limit the controllers that can connect with TCP wrappers on the agent.

open question: what happens if you put a list of IPS into the specific host configuration in the preferences?

Reported security holes

- It is alleged, but not confirmed, that the Xgrid configuration wizard in the Server Admin gui, silently opens an undocumented NFS share on the local hard drive of the directory /var/xgrid /controller/sfs that is open for read/write to the universe (no IP address restrictions). This security hole was reported to apple who subsequently closed the report as "operates normally", so it is likely to persist in future OS releases. The hole is easily fixed by deleting the line for the share from /etc/exports. Fortunately, a confirmed and unrelated major bug in all 10.6 server admin tools prevents NFS configuration prevents NFS from working when there is a firewall, so this hole is accidentally masked.

Shared File Systems

Commonly an executable needs access to data files and libraries to do it's job. On the agent these may not be available. Likewise results may file-based rather than just the standard out. The Xgrid command offers limited migration facilities for input and output. Specific files and even a whole directory tree can be encoded into the batch file and transiently transported to the agent for the duration of a task or job.

Why shared file systems are desirable

Shared file systems can relax many constraints. For example, they can offer a lightweight approach to inter-task communication and persistence. When used with scoreboard they can permit more sophisticated reporting than just a single art score. But the biggest issue is simply data transport efficiency.

The built-in transport sacrifices efficiency for simplicity. Input files transported to the agent for a task are not cached even when the same agent is running tasks that all use the same input files. The data is retransmitted for every task. The binary data is sent in uncompressed base64 encoded XML, so it's 4/3 larger than the original file size. And for single tasks, the network traffic is doubled since it all goes from client to controller then controller to agent. On the return trip, for output files, the same redundancies occur.

However, that modest inefficiency is can be dwarfed by another consideration. Consider an calculation needing just a few items from a very large database. If the items needing to be read cannot be known beforehand then the whole database must be transported across rather than just the records of interest. When using canned software, one may not have the luxury of optimizing the data storage and retrieval format for the idiosyncrasies of Xgrid. An analogous issue arrises in dynamic loading of libraries in code. A given piece of code may want just a few modules in a very large library. Consider, for example, Python code that uses extensions at runtime not found natively on the Agent. You may not know which parts of a large distribution library are required and so have to migrate it all.

To accommodate these ad hoc run-time read/writes needs it highly desirable to be able to mount a shared filesystem on all the agents. This also allows the agents to share intermediate results and otherwise be more tightly coupled.

Strategies for Shared file systems

Here we will ignore the two extreme cases: on a remotely distributed system where sharing a disk mount is prohibitively difficult and on dedicated clusters with complete control of Agents and so there are no constraints on what can be done. Instead we examine the case of password authentication where jobs are run as user nobody and are severely limited by the sandbox. Even stronger constraints arise in grids of inhomogeneous workstations with limited admin access.

Can a job mount an external file system on demand?

Not easily. Xgrid has no provision for this on the agents and user-nobody cannot execute a mount command, so conventional approaches are out. The sandbox allows custom socket communication with a server, but this would not be a universal solution since many pre-existing programs expect a normal file system. You can't even fake this with a UNIX fifo-file because the sandbox prohibits a pipe between different process IDs under password authentication.

Thus admin intervention is needed on the Agents to set up a file mount. The easiest way to make this persistent across both reboots and temporary network failures is to include it as an automount. Such a one-time configuration can even be done by the workstation owners themselves using the directory utility GUI (in leopard). However for simplicity and for greater NFS option selections it may be better to run a script that directly edits /etc/auto_master.

Where in the agent filesystem should a shared file system be mounted?

Since the mount is not for the benefit of the workstation owners, the mount point is best kept out-of-plain-sight for various reasons

1. The mount point should be unseen in file dialogs to avoid sluggish response times from querying remote disks.
2. Workstation users might mistakenly save their files on your remote disk.
3. Put up a speed bump against unsophisticated Agents users from messing with your file system.
4. No intrusive icon on the desktop or finder sidebar for the remote disk mount.
5. Avoid time machine backup or spotlight indexing

First off we can only consider places that user-nobody and the sandbox allow access. The most obscure place is perhaps in $TEMPDIR which is the nobody-specific folder Darwin unix creates as a per-user-name temp directory. A problem with this is assuring that path will be the same path on all computers. Uniformity of the mount point is desired since our executables often need to know the absolute path to access this. Another possible place is just one node above where xgrid creates the working folders for the tasks. This seems logical and consistent but since it's not a documented path it might change or get over-written on system updates. (We note there is a cryptic empty directory already there suggestively called 'sfs' which one suspects might be a vestige of Apple experimenting with this.)

Charlie Strauss suggests /tmp/.xsfs as a worthy candidate. The finder has /tmp on it's list of hidden directories, and unix will hide the .xfsf name, making it hard to accidentally appear in a file dialog or directory listing. Because apple NFS dynamically creates the mount directory on demand (unlike other some other *NIX) the folder ".xsfs" does not have to pre-exist, so there's no problem with the /tmp folder getting wiped every reboot. The absolute path is also nicely short making it easy to type in arguments to commands. It has the added virtue that if the workstation owner did manage to accidentally save a precious document there and subsequently lost it, the owner can hardly blame the xgrid manager since /tmp is always a risky place to save something. And there's no need to uninstall it later since /tmp get's wiped every reboot.

Should the shared file system be password protected?

While it might seem like a password is desirable, however, *as commonly implemented*, it has only modest protection and added inconvenience. First, discriminating soley by password rather than IP could be more vulnerable since it relies on keeping the password secret. That can be a hazard since auto_mount passwords are by default kept in plain text configuration files readable by all users on the agent machine *including third-party xgrid jobs*. (See sandbox section for a way to hide this from xgrid jobs.) Second, one must be careful how the incoming user-ids are enabled on the file server since simply creating standard user-IDs would in turn grant login access to the server that would be undesired. Third, since the mounted file system has to be readable by user nobody it may also be accessible for other agent workstation users as well.

Issues to consider when choosing between NFS,AFS or SMB

From the server-side, NFS export is attractive because it does not require a userID and password from the client but easily restricts by IP addresses. It can enforce user-id squashing so all written files are accessibly by user nobody.

AFP and SMB authenticate by user-id and password which will need to be exposed in /etc/auto_master on every agent mounting the disk. They don't natively discriminate by IP address of the agent, so by default anyone who learns the userid and password can access the server disk from any computer. You won't be able to remotely edit these if you need to change them without getting admin access on the Agent again. To limit the scope of a password breach, It could be a good idea to also implement additional IP address filtering such as using TCP wrappers. You also need to make sure that user nobody on the agent is able to access the mount by the User ID chosen Note: If you simply choose nobody as the AFP mount user-id then all agents will be sharing the same password (for user nobody) reducing its security value. This is also a trick that can only be played once: one could not export a different filesystem to a different grid this way since the Nobody user ID is unique. Finally you need to make sure that files written by agents by default bear the nobody user-id not the user-id of the diskmount or else the agents won't be able to access each other's shared data. (recipe for how to do this on AFP?)

One minor virtue of AFP/SMB over NFS is the ease of opening the firewall ports (1 for AFP and 6 for NFS of which 4 are varying port numbers). That is, if you are having to beg an overworked IT staff to open ports in company firewalls, you might find that AFP is open already but because of the variable RPC ports NFS usually can't be open by default.

open questions: Is it possible to store an AFP or SMB passwords in a keychain to keep it out of the eyes of casual users on the agent, but since it needs to mount at boot, how can these key chains be unlocked? Don't know. Also there are other issues (http://support.apple.com/kb/TS1234) with passwords stored in keychains. Is there some way to authenticate AFP by LDAP that avoids these issues? Don't know.

Xgrid Client Choices

While the agent and controller are specific software daemons, there is no one thing that fills the roll of client. Apple has designed a set of APIs that any program can use to talk to the controller for job submission and information retrieval. Thus many different forms of clients can exist for various special purposes. However for use more as traditional batch job system one has the client tools xgrid and MPI built into all OSX versions. There are also third party queue's like GridStuffer[2], and demo apps in apple's development package.

xgrid

All OSX machines have the xgrid command for submitting jobs from the command line. The man page documents most of the command so we wont repeat that here. But it leaves out a few important details.

1. xgrid -job log -id 1234 will print out important status info of job 1234 and in particular has the -art scores.
2. paths for -art scripts on the command line must be relative from the current working directory. Absolute paths fail on the command line for -art scripts.
3. when using -art, one must also specify -artid and also a condition like -artequal or -art will be silently ignored.
4. Arguments to art scripts are possible only in batch mode submissions.
5. scheduler hints are only merely suggestions not requirments to the controller.

MPI

As of 10.5, openMPI is built into every mac and jobs can be launched form the command line using "mpirun". Conveniently, openMPI will autodetect an xgrid and use it if the shell contains environment variables specifying the controller and password. What could be easier! Yet there's a snag....

Can jobs submitted to openMPI use Xgrid?

As of 10.5, openMPI is built into every mac and jobs can be launched form the command line using "mpirun". Conveniently, openMPI will autodetect an xgrid and use it if the shell contains environment variables specifying the controller and password. What could be easier! Unfortunately, As of April 2010, the support for Xgrid built into leopard openMPI has been broken for 15 months due to a specific known bug that remains unfixed in v1.3 and will likely remain unfixed in the next release (v1.4) of openMPI accroding to this bug ticket (https://svn.open-mpi.org/trac/ompi/ticket/1777) . It is reported that the older release, openMPI 1.2 can be installed and will work. Some guidance on using openMPI 1.2 can be found at open-mpi.org here (http://www.open-mpi.org/faq/?category=osx#xgrid-howto) with greater detail here (http://www.open-mpi.org/community/lists/users/2006/01/0539.php) .

Can jobs submitted to xgrid launch openMPI themselves?

Apple provides a second, completely undocumented and unsupported, way to launch MPI jobs. Included in Xcode is a demo project called GridSampler intended to be a GUI for job submission. Building this produces not one but three apps, one of which will boot MPI from jobs on the grid. When run you will see a job entered into the xgrid controller that executes multiple tasks on one or more machines. Inspecting the job specification for this (recoverable from the controller) reveals that it simply submit ordinary concurrent tasks with a wrapper that launches the desired executable. The wrapper binary bootstraps the MPI connection from each agent's task to establish the network links for MPI. Unfortunately there is no documentation so it's not known how to invoke any of the many options mpirun has to offer. Moreover, in use it seems to be unstable, sometimes failing to set up communications. But at least it does work somewhat. The GUI app is not required: just include the stand-alone wrapper binary in your own custom job specifications batch files submitted from the command line. This is easily done by copying the specification pattern found in any of the jobs submitted from the GUI app.

Objective C API

Apple has a documented API and some examples of using it in the developer tool kit. There is also a third party API toolkit called EZGrid which purports to greatly simplify setting up an Objective C client.

Other clients, APIs, and batch file generators

GridStuffer[2] is a terrific replacement client with a Gui interface. XgridFuse[3] is a replacement for the xgrid Admin Tool that simplifies access to the job status and results.

There are projects in Ruby and Python that will import client functions. As of mid 2010 the RxGrid[4] appears to be in active development. PyXG [5] project has no online-documentation but there is documentation in the tarball download[6] . Both of these use a syntax and capability that mimics the xgrid client tool.

Additionally, there are dedicated batch file generators. These are not client replacements but rather simplify generating sophisticated batch files that can then be submitted with the standard xgrid client too. The Xgrid Batch Editor[7] is a stand alone gui interface. Xgrid.rb[8] is a ruby-based batch file generator.

xgridstatus[9] is a must-have single purpose tool that accesses undocumented features of the API and offers some unique functions. It returns grid IDs, availability and processor details of the agents. Currently no other tool, including apple's own offer those features. It is not used for submitting jobs.

Resource Management and Etiquette

The XGRID FAQ covers topics on:

- how to measure load
- how to know what agents are on a grid
- how to know what logical grids are present

Forks, Threads and Interprocess Signals

The XGRID_FAQ has a detailed section on this as well.

Tasks can fork or use multiple threads. However, with great power comes great responsibility. The Xgid controller queue has no provision for a job to hint at that it will fork, or for a single task to request multiple processor slots on the agent. When the queue is full the controller will assign additional tasks to the Agent till it reaches the maximum allowed level of tasks. If these contain forks the number of running processes may exceed what is desirable for good workflow on the agent. It may even impair a workstation's responsivity to it's desktop user: even though the xgrid jobs work at nice level -20, empirically for ~4xCores *purely computational* processes working full time, the workstation will be crippled. Process that use up memory, disk, or network resources can impact even more quickly.

Other than micromanging the queue there are several blunt hacks that somewhat ameliorate this. First, the xgrid agent preferences plist allows the setting of the maximum number of jobs to accept. In 10.5 it was 2xCores and in 10.6 it is 0.5x cores[10]. If one's process flow has a predictable number of forks per task, then one can adjust this number so the resultant number of processes is acceptable. However, for process flows that use variable numbers of forks this solution would lead to too conservative a processor restriction. To help somewhat with inhomogenous task forking needs, one can divide the agent pool into several logical grids and set the agents on each to have a different cores/max_tasks ratio. Altering the max_tasks in the xgridagent.plist can not be done from xgrid itself, as it requires admin access to the Agent. Thus it cannot be done dynamically as the processing needs change.

Another approach would be turn this down to a single process per node and have a pre-processing step before job submission to combine jobs into tasks sets that must start simultaneously. This approach will breakdown since the marriage of convenience of unrelated tasks may have ones that finish sooner than others.

Another approach for a task known to fork N times is to submit the task along with N-1 bogus tasks that simply wait idle for the true task to complete. If we specify that all N tasks must start at the same time this forces the accounting of processor slots to respect the fork count. The problem with this approach is that it only works if xgrid places all the taks on a single agent rather than distributing them. Unfortunately there no way to instruct xgrid to place all the tasks on a single Agent without also specifying the exact agent the jobs must run on using a narrow -art command. That micromangement subverts the desirable feature of the queues handler to place a job on the next available Agent. see drone_thread for an example of code for creating a job with bogus tasks.

intriguing observation: Normally if an agent rapidly forks normally all the children will have consecutive process IDs. On 10.5 xgrid agent with user Nobody, the processes IDs will be two apart instead of consecutive. Speculatively, perhaps there is some intervening shadow process created on forks that is also not propagating all of the POSIX/BSD signals.

Inter process signaling

Two easy ways processes on the same machine can communicate efficiently are BSD/POSIX signals (i.e. sending kill events) and through named pipes (fifo). Unfortunately, Xgrid on 10.5 (user Nobody) prevents forked processes from reading and writing to a named pipe. (oddly they can read from a named pipe and they can write to a named pipe, but not when both ends of the pipe are xgrid processes.) Second, BSD/POSIX signals only work to a limited extent. A process cannot send a signal to another process group. A parent process can send a signal to a child process provided the child has not created a new process group. In 10.5 a process cannot send a signal to itself.

One signal that does work between process groups is file-locking. One xgrid process can flock a file and another process will respect the file system flock (warning this can fail on NFS mounted file systems). Thus one can create semiphores in any mutually accessible directory and poll them as areplacement for signalling.

See the XGRID FAQ for further details

Sandbox

First you would have to be extremely foolish to monkey with the security sandbox the xgrid runs the tasks in. However fools may wish to forge ahead. First a couple caveats, the security sandbox is not documented, and it is machine generated. So next system update you might find your sandbox modifications either overwritten or, worse, causing a complete malfunction of some new sandbox system such that your jobs are running unsandboxed. Yikes. So why try? Well the default one allows network connections. Do you really want your agents to have that privilege? It also allows agents to write to anywhere in /var that user nobody can write. It lets the agents read all sorts of things in your /System, /Library, /var, and /etc folders that you might not want outsiders to know the details of. Here is Charlie

E.M. Strauss's suggestion for a more tightly restricted while still practically useful replacement for */usr/share/sandbox/xgridagentd_task_nobody.sb* sandbox.

style, useful commands and scripts

Not filled in

Xrid Pitfalls

- crashing jobs can fill up crash reporter dump
- Using too many threads or forks.

Mastering Xgrid Agent

Processes

The following are from a 10.6 agent with no jobs

```
UID   PID  PPID CPU PRI NI      VSZ    RSS WCHAN  STAT   TT       TIME COMMAND
  0     1     0   0  31  0  2456700    848 -      Ss     ??   43:43.17 /sbin/launchd
 86  3165     1   0  33  0  2465544   4840 -      Ss     ??    0:00.24 /usr/libexec/xgrid/xgridagentd
  0  3168  3165   0  33  0  2460892   2108 -      Ss     ??    0:00.02 /usr/libexec/xgrid/xgridagenthelp
```

All jobs appear to be subprocesses of xgridagenthelperd. Each job appears to have a separate processgroup ID equal to the process ID of the parent fork of the job. All jobs appear to share the session ID of the xgridagenthelper.

Agent Xgrid Paths

Some directories are not populated until xgridagentd or xgridcontrollerd are turned on for the first time. THe following if from a computer with an agent turned on for password mode in 10.5

- /etc/xgrid

```
drwxr-xr-x  4 root  wheel  136 May  8 13:51 agent
drwxr-xr-x  3 root  wheel  102 Dec 10 22:30 controller
```

- /etc/xgrid/agent

```
-rw-r--r--  1 root  wheel  1252 Nov 25  2007 com.apple.xgrid.agent.plist.default
-rw-------  1 root  wheel    16 May  8 13:51 controller-password
```

- /etc/xgrid/controller

```
-rw-r--r--  1 root  wheel  687 Nov 25  2007 com.apple.xgrid.controller.plist.default
```

- /var

```
drwxr-xr-x  4 root  wheel  136 Dec 10 22:30 /var/xgrid
```

- /var/xgrid

```
drwxr-xr-x  6 _xgridagent       _xgridagent  204 May  8 13:51 agent
drwxr-xr-x  3 _xgridcontroller  wheel        102 Nov 25  2007 controller
```

- /var/xgrid/agent/

```
drwx------  2 _xgridagent  _xgridagent  68 May  8 13:51 controllers
drwxr-xr-x  2 _xgridagent  wheel        68 Nov 25  2007 cookies
srwx------  1 _xgridagent  _xgridagent   0 May  8 13:51 status
drwxr-xr-x  2 _xgridagent  _xgridagent  68 May  8 13:51 tasks
```

- /var/xgrid/controller

```
drwxr-xr-x  2 _xgridcontroller  wheel  68 Nov 25  2007 blobs
```

- /var/run

```
-rw-r--r--  1 root  daemon       3 May 20 17:16 xgridagentd.pid
```

Contains the process id of the currently running xgridagentd

- /Library/Preferences/com.apple.xgrid.*.plist

```
-rw-rw-r--  1 root  admin  489 May  8 13:51 com.apple.xgrid.agent.plist
-rw-rw-r--  1 root  admin  337 Dec 10 22:30 com.apple.xgrid.controller.plist
```

- /System/Library/LaunchDaemons/com.apple.xgrid*.plist

```
-rw-r--r--  1 root  wheel  423 Jul 28  2009 com.apple.xgridagentd.plist
-rw-r--r--  1 root  wheel  435 Jul 28  2009 com.apple.xgridcontrollerd.plist
```

- /usr/libexec/xgrid/

```
-rwxr-xr-x  1 root  wheel  39248 Feb 11 02:30 IdleTool
-rwxr-xr-x  1 root  wheel  48096 Feb 11 02:31 xgridagentd
-rwxr-xr-x  1 root  wheel  48432 Feb 11 02:30 xgridagentexec
-rwxr-xr-x  1 root  wheel  66208 Feb 11 02:30 xgridagenthelper
-rwxr-xr-x  1 root  wheel  48144 Feb 11 02:31 xgridcontrollerd
```

- /Library/Logs/

```
drwxr-xr-x  3 root  admin  102 May  8 13:51 /Library/Logs/Xgrid/
-rw-r--r--  1 _xgridagent  admin  639 May  8 13:51 /Library/Logs/Xgrid/xgridagentd.log
```

- /usr/share/sandbox/x*

```
-rw-r--r--  1 root  wheel  522 Jul 28  2009 xgridagentd.sb
-rw-r--r--  1 root  wheel  687 Jul 28  2009 xgridagentd_task_nobody.sb
-rw-r--r--  1 root  wheel  590 Jul 28  2009 xgridagentd_task_somebody.sb
-rw-r--r--  1 root  wheel  775 Jul 28  2009 xgridcontrollerd.sb
```

Undocumented executable operation

- IdleTool Appears to return the HDIdletime (no-mouse/kb). It does not seem to take simple args. It contains the following strings in the binary: IOHIDSystem and HIDIdleTime. However it has two unexpected features. When it runs it tries to read from every mounted disk (seen from the seatbelt debug log) . Thus using this for idle time calculations would likely keep all disks spun up. Second it's unix permissions include a setuid root which empirically appears to be unnecessary as it still functions when setuid is removed. The purpose of this executable is not known. It's plausible it is used when the xgrid agent is set to be unavailable when not idle. If so the disc touching aspect would seem to be a bad mistake when used for that purpose..

- xgridagentexec execs any supplied command with args. What else it does in not known. xgridhelperd uses to spawn jobs as user nobody.

```
97711 root /usr/libexec/xgrid/xgridagentexec --uid -2 --priority 20 --directory /var/xgrid/agent/tasks/i
```

```
usage: /usr/libexec/xgrid/xgridagentexec [--uid <uid>] [--priority <priority>] [--fd <fd>] [--directory <
```

When a job runs Emprically, a command like

sudo /usr/libexec/xgrid/xgridagentexec --uid -2 --directory /tmp/foo `which touch` bar

will run as user -2 (nobody) and cd the the directory /tmp/foo where it will touch the file bar. It does not appear to chroot to that directory.

- xgridhelperd run form the command line outputs the following when run, then takes input from the keyboard in full duplex.

```
RPY 0 0 . 0 94
Content-Type: application/beep+xml

<greeting><profile uri="xgridagenthelper"/></greeting>
END
```

xgridhelperd calls xgridagentexec

- xgridagentd when run from a normal user terminal command line outputs:

```
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Error>: Error: could not open logfile "/Library/Logs
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Error>: Error: agent could not change permissions of
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Error>: Error: agent could not add user "_xgridagent"
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Notice>: Notice: agent will only accept tasks when i
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Notice>: Notice: agent maximum task count is 1
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Notice>: Notice: agent processor speed is 2400
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Error>: Error: agent password file "/Users/foo/Libra
Sat May  8 14:33:52 Gantor.local xgridagentd[3403] <Notice>: Notice: agent failed
```

- xgrdicontrollerd when run from a normal user command line outputs:

```
Sat May  8 14:38:18 Gantor.local xgridcontrollerd[3427] <Error>: Error: could not open logfile "/Library
Sat May  8 14:38:18 Gantor.local xgridcontrollerd[3427] <Error>: Error: controller missing password file
Sat May  8 14:38:18 Gantor.local xgridcontrollerd[3427] <Notice>: Notice: controller failed
```

Searching these for strings shows that it appears that xgridagentd have very few strings and looks like a simple wrapper that launches helperd. xgridhelperd has a lot of strings many of which appear to be messages that might get sent to a controller.

Missing functionality

core functionality

Xgrid is intentionally a thin interface for a grid controller and thus lacks bells and whistles. Users who want these can implement their own using the cocoa api for this. However, listed here are major functionality missing from the xgrid command cli that is particularly difficult to work around.

Resource hinting

tasks should be allowed to hint how much resources they desire (number of processor slots and amount of memory). The controller could use this to correctly count multi-threaded tasks as though they were multiple tasks on the agent or to avoid putting multiple large-memory jobs on the same machine. Currently the controller can overburden a machine and the maxTaskCount plist item is not only too course grained, but it cannot be changed dynamically or without admin access.

Require tasks to run on single Agent

There is syntax to require tasks to start concurrently but does not allow requiring tasks to be placed on a single agent. (note: The scoreboard -art interface allows narrowing the jobs to a specifically named agent, but then one has to name this agent rather than letting the scheduler choose.) For example, highly parallel jobs may prefer to run on the same node rather than use MPI to chat across the network.

SSH-like authentication

The requirements for kerberos authentication is unacceptable to many kinds of agents, and password authentication with it's shared plaintext passwords is flimsy. Xgrid needs to replace the shared password system with something more secure (modeled after ssh-keys or some other public-key exchange authentication) that would allow the controller to have a different password or ssh-key for every agent if desired. passwords and keys should be stored in a keychain rather than in plain text.

Wish List

SuspendWhenNotIdle

By suspend I don't mean not accepting pending tasks but actually freezing the running tasks in a way that allows them to restart. A light weight implementation of this need not use checkpointing or task migration. Instead, when agents are used on desktops it would sometimes be desirable to have running jobs halt as though they received a POSIX sigSTOP. By having a long resume-lag (minmum idle time before restart) the jobs will get paged out by the VM if they are in the way. If this is implemented it would be desirable to allow the resume lag time to be variable across the day so that, for example, the restart lag time can be large during business hours and short after close of business. This should be trigger-able either from a non-idle signal or when the load from non-nobody processes was approaching 1. Even with jobs set to nice -20 the jobs can overburden a machine in their memory and network usage, and should not resume instantly after the load is low like unix process priority scheduler allows.
while_idle

Suspect Bugs

- xgrid -job run

1. If you control-c this command it does not stop the running job, and since you don't know it's job-id you can't kill it either.
2. Conversely, if you kill the job backing the run-command from the xgrid admin, then the run command silently waits forever.
3. If the job fails to start due to no suitable nodes (by art or by taskerrors) the run command silently waits forever.
4. leaving out one of the three art flags causes the art not to run but the job to run anyhow (i.e. art failes siliently)

- Launchd

1. When the Xgrid Agent crashes, launchd fails to kill the orphaned tasks. Launchd then relaunches the xgridagent daemon and the controller resends the tasks. (This process can repeat till the computer collapses under the load if the reason for the crash was caused be the jobs.)

- Specification size bug: apparently you can submit a job larger than you can later retrieve the specification for.

```
xgrid -job specification -id 8501
xgrid(12630,0xa04d1720) malloc: *** mmap(size=391450624) failed (error code=12)
            error: can't allocate region
            set a breakpoint in malloc_error_break to debug

    the job was created like this:
    xgrid -job submit -in foo /bin/sleep 50
    where foo is local directory with a single file in of size 133MB.
```

Frequently Asked Questions

The FAQ now have their own document.

External Links

What documentation is available for Xgrid?

Official Apple Docs

man pages for the command-line tools contain useful examples:

- man xgrid (http://developer.apple.com/mac/library/documentation/Darwin/Reference/ManPages /man1/xgrid.1.html) # client submits and monitor jobs
- man xgridctl (http://developer.apple.com/mac/library/documentation/Darwin/Reference/ManPages /man8/xgridctl.8.html) # start/stop xgrid daemons

The primary Xgrid documentation is the Xgrid Administration manual for Mac OS X Server:

- http://images.apple.com/server/pdfs/Xgrid_Admin_v10.6.pdf

The ADC Developer library contains a reference description of the Xgrid Foundation API for Cocoa developers:

- http://developer.apple.com/documentation/Performance/Conceptual/XgridDeveloper/index.html

Apple's FAQ has some good info, but it is often too terse and has not been groomed of stale info or broken links.

- official Apple FAQ (http://lists.apple.com/faq/pub/xgrid_users/)

Apple Marketing docs.

- http://www.apple.com/macosx/features/xgrid/
- http://www.apple.com/server/macosx/features/xgrid.html
- http://developer.apple.com/hardware/hpc/

Non-Apple web sites that deal with Xgrid

MacResearch has a superb set of tutorials (http://www.macresearch.org/the_xgrid_tutorials) by Charles

Parnot.

- The Xgrid Tutorials (Part I): Xgrid Basics (http://www.macresearch.org /the_xgrid_tutorials_part_i_xgrid_basics)
- The Xgrid Tutorials (Part II): GridStuffer Basics (http://www.macresearch.org /the_xgrid_tutorials_part_ii_gridstuffer_basics)
- The Xgrid Tutorials (Part III): Running Batch Jobs (http://www.macresearch.org /the_xgrid_tutorials_part_iii_running_batch_jobs)
- The Xgrid Tutorials (Part IV): Submit Jobs with Ruby (http://www.macresearch.org /the_xgrid_tutorials_part_iv_submit_jobs_with_ruby)

Another introductory tutorial can be found at MacDevCenter

- Distributed Tiger: Xgrid Comes of Age (http://macdevcenter.com/pub/a/mac/2005/08/23 /xgrid.html)

Other helpful sites that discuss Xgrid:

- http://www.macos.utah.edu/documentation/administration/xgrid/xgrid_presentation /general_info.html
- http://www.macos.utah.edu/documentation/administration/xgrid.html
- http://hammonds.scu.edu/~classes/pyxg.html
- http://cmgm.stanford.edu/~cparnot/xgrid-stanford/
- http://kellerfarm.com/kfsproducts/opensource/agentidler/index.html
- Xgrid Batch Editor (http://kellerfarm.com/kfsproducts/yesfree/xgridbatcheditor/index.html)
- Xgrid.rb (http://bitbucket.org/j05e/xgrid/src)
- PyXG = Python + Xgrid (http://pyxg.scipy.org/)

References and Footnotes

1. ↑ Xgrid for Linux was at http://unu.novajo.ca/simple/archives/000026.html but that link is now only on the google cache (http://cc.bingj.com/cache.aspx?q=xgrid+for+linux&d=4848159179736361&mkt=en-US&setlang=en-US&w=66a1fe92,1c3500de) . An alternative set of links can be found here http://lists.apple.com/archives/xgrid-users/2005/Apr/msg00056.html
2. ↑ 2.0 2.1 GridStuffer is available at http://cmgm.stanford.edu/~cparnot/xgrid-stanford/html/goodies /GridStuffer-info.html
3. ↑ xgridFuse is available at http://cmgm.stanford.edu/~cparnot/xgrid-stanford/html/goodies /XgridFUSE-info.html
4. ↑ RxGrid is the ruby client API http://www.tslab.se.shibaura-it.ac.jp/index.php/Software/rxgrid
5. ↑ pyXG is the Python client API project http://pyxg.scipy.org/
6. ↑ The PyXG project files are hosted at https://launchpad.net/pyxg
7. ↑ xgridbatcheditor is available for free from http://kellerfarm.com/kfsproducts/yesfree /xgridbatcheditor/index.html
8. ↑ Xgrid.rb is available at http://bitbucket.org/j05e/xgrid/src
9. ↑ xgridstatus is a compiled binare available for free at http://cmgm.stanford.edu/~cparnot/xgrid-

stanford/html/goodies/xgridstatus-info.html
10. ↑ http://support.apple.com/kb/HT4020

- .
- .