A k w | r ^ | r ^ \ v ∀

Systems Administration Considerations

for Ab Initio Applications

October 2003 > Part Number AB0407

Ab Initio Software Corporation
201 Spring Street > Lexington, MA 02421 > Voice 781.301.2000 > Fax 781.301.2001 > support@abinitio.com

GEFINANCE-DOC1813

INTELLECTUAL PROPERTY RIGHTS & WARRANTY DISCLAIMER

CONFIDENTIAL & PROPRIETARY

COPYRIGHTS

TRADEMARKS

RESTRICTED RIGHTS LEGEND

WARRANTY DISCLAIMER

Table of Contents

GEFINANCE-DOC1813

Systems administration considerations for Ab Initio applications

This document consists of brief discussions of some of the issues that should be considered prior to installing Ab Initio on Unix or Windows operating systems.

In each section, mention is made of some of the more complex situations that may exist in specific installations. Most of these situations are rare, but they are mentioned so that they can be understood in advance. The typical installation will have to worry about only a very small fraction of the types of issues noted here.

Audience

The range of tasks touched on in this document will probably involve application architects as well as system administrators. For example, determining the amount of space needed for application files (see "Application file space ($SERIAL, $MFS, etc.)" on page 5) involves knowledge of the applications themselves, how they are designed, and how they are to be run. If you are using the Ab Initio Meta>Environment (see "Ab Initio Meta>Environment datastore ($AB_AIR_ROOT)" on page 4), many of the decisions involved in its setup are determined by how it is to be used, how the EME datastore is organized, and so on.

Introduction

Ab Initio provides enterprise wide computing capabilities for any type of data or metadata access across multiple servers and multiple organizations in a completely portable and scaleable fashion.

Ab Initio does a good job of shielding the average developer from all of these complexities. However, some of these need to be considered by the systems administrators when setting up Ab Initio and assigning some of the resources such as disk space, memory, and connectivity that Ab Initio graphs (applications) will use. And because of the cross-platform and high performance aspects, some of these issues may be a little different from what systems administrators are used to dealing with in smaller or less sophisticated applications.

Disk space, disk configuration, and backups

Following are discussions of the different categories of space used by Ab Initio applications. Each section consists of a brief description of the approximate size required, the usage characteristics at runtime, and some implications for what types of storage may be better or worse for each usage. Typically, each of these areas will be located on a different filesystem, for reasons described in the separate sections.

Install space ($AB_HOME)

A default value for AB_HOME, which specifies the Ab Initio Co>Operating System's installed location, might be something like:

```
/usr/local/abinitio-V2-10-15-m23
```

The Ab Initio code requires approximately 200 megabytes. The area specified by the AB_HOME path is static at runtime, but is read many, many times as applications that are running many parallel processes all start up simultaneously. It should therefore generally be local storage, not NFS mounted. This space is needed on every machine on which Ab Initio is installed.

Ab Initio working storage ($AB_WORK_DIR)

A default value for AB_WORK_DIR might be something like:

```
/var/abinitio
```

The path specifies working storage for Ab Initio itself, not space used as temporary working space by the applications. Size requirements are dependent on many factors, such as the number of simultaneous applications, frequency of application failure, etc. We usually recommend approximately 1 to 10 gigabytes of space. This space is needed on every machine on which Ab Initio is installed.

This space keeps track of the information needed to recover applications and restart from checkpoints in case of failure. Also, very large numbers of very small files (as opposed to large blocks of data) tend to be written to this space. For both reasons, we recommend RAID-1 or mirroring for the filesystem used for Ab Initio working storage.

Backups

The data in this area is extremely transient, so frequent backups are not at all critical. It is worth noting that a restored backup of this area while an application is in process would not be useful for recovering a failure from that application, which makes it preferable that this area be backed up while no Ab Initio applications are running. However, that situation is extremely unlikely, so in practice, this is not a restriction.

Networked filesystems

Certain networked filesystems can be inappropriate for use for Ab Initio working storage. Some networked filesystems will claim, for performance reasons, that they have flushed data to disk when Ab Initio requests that they do so, when in fact it is merely cached. Since this space is used for checkpoint recoverability, the Co>Operating System needs to know that data has actually reached disk before starting the next operation. In our experience, all locally attached storage we have encountered and most networked storage meets these requirements, but please check with Ab Initio before using non-local storage for Ab Initio working storage.

NOTE: It is important that the AB_WORK_DIR space be located on a filesystem physically separate from any other filesystem that running applications might be filling up. Otherwise, applications that fail because they run out of space (one of the more likely causes for application failure) will also almost certainly be unrecoverable.

Ab Initio Meta>Environment datastore ($AB_AIR_ROOT)

The AB_AIR_ROOT path specifies the storage space (usually known as the *datastore)* used for the Ab Initio Meta>Environment, otherwise known as the EME. If you will be using the EME as the primary point of control for all Ab Initio software projects, this single datastore instance will be accessed by all Ab Initio projects across your entire installation. As a result, it has the most stringent possible requirements for robustness. We recommend RAID-1 or mirroring and nightly backups, as a minimum.

Backups

The Ab Initio files in this area comprise a database. They must be backed up in a method that guarantees the multiple files are in sync, just like any other database. Ab Initio provides a command that allows a backup command to be run within the context of a database lock, so that it is guaranteed to run on a synchronized set of files. We recommend that the backup command used be a relatively quick command, such as a tar and compress of the files to another directory, so that the datastore lock is freed as soon as possible. Lengthier steps like migrating the backup to tape can then occur while the datastore is unlocked.

Separate filesystem

We recommend that the EME datastore files be located on their own separate filesystem, so that no application is likely to fill it up. We also recommend that this filesystem be monitored in such a way that alerts will occur if it becomes nearly full.

Filesystem dependability

As mentioned above for Ab Initio working storage, the datastore needs to be able to be certain that, when the filesystem asserts that something is committed to disk, it really has been. In addition, the EME makes extensive use of features such as file locking and memory mapping

which, for performance reasons, are sometimes not supported or not supported correctly on some types of networked filesystems.

For example, some networked filesystems say they have given you an exclusive lock on a file when in fact they have not. Some do not support memory-mapped files at all. The EME will not run correctly on a filesystem of this type. So far, all locally attached filesystems we have encountered run fine, but certain networked filesystems do not. Please check with Ab Initio before using non-local storage for an EME.

The size of EME datastores varies dramatically with the number of users and the number of applications, and it grows over time. You might start with a size of approximately 50 gigabytes, with the understanding that this figure would be reassessed once the actual usage patterns become clearer.

Note that there need be only one instance each of the EME for development, for QA, and for production in your entire installation. The space described in this section is *not* required separately for every server running Ab Initio.

Application source code ($DML, $XFR, etc.)

This is the space that the applications' source code is installed in, and by default it is where applications are executed—though that is not a requirement. There are few special requirements involved with this type of space. It is typically small, reads and writes tend to be low volume, performance is not much of an issue, and there are no specific requirements for locking, memory mapping, and the like.

Application file space ($SERIAL, $MFS, etc.)

Obviously, the amount of space needed for applications cannot be estimated without considering data volumes and application design isssues for the applications themselves—and these will vary widely from site to site. However, certain characteristics of how this space is used are worth noting.

In general, the Co>Operating System will be reading and writing these areas in large contiguous blocks. This means that high disk transfer rates here are good, while high seek rates are more optional, with the net positive result that the largest portion of disk space required can often be on the least expensive types of disk.

Paradoxically, some of the most expensive storage solutions can have the worst performance. For example, some network attached storage options have relatively slow transfer rates, due to the network bottlenecks, and try to make up for this with large cache spaces. However, Ab Initio applications processing many megabytes or terabytes of sequential data often get no benefit at all from caching. So locally attached storage offers usually the best performance.

Since this is usually your largest storage requirement by far for an Ab Initio application, you will typically want it protected by at least RAID-5 to avoid down time caused by single disk failures. And since the reads and writes tend to be large, performance is good with RAID-5 as well. Higher levels of RAID would typically not be required.

Multiples

One special consideration for systems administrators in assigning this space is centered around an Ab Initio construct called *multiflles.* Multifiles are essentially a method of dividing a physical file into many separate pieces stored in different locations, allowing multiple processes each to operate on their own piece, in order to gain parallelism and take advantage of multiple CPUs. See the Ab Initio documentation, and especially the Ab Initio *Co>Operating System Administrator's Guide* for a more in-depth treatment of this topic. The remainder of section will focus on the systems administration implications of this technique, such as performance tuning issues and backup strategies.

Multifiles and physical resources

To maximize the performance gains from data-parallel applications, it's desirable to have the applications spread across the maximum hardware resources available. If SMP systems are what is used at your site, sharing of CPUs and memory is not much of an issue. With an Ab Initio application providing data parallelism, the underlying Unix O/S will manage allocation of physical CPUs and memory. However, allocations of disk resources may or may not require more thought on the part of the systems administrators.

If disk space being used is on some sort of disk array or SAN that already stripes logical volumes across physical controllers and physical spindles, allocation of disk across multifiles is also simple. In this case, you should assign the control partition and each physical data directory to any logical volume or volumes, and let the underlying disk subsystem take care of the hardware allocation. In fact, since these systems often hide these details from the systems administrators, there is often no choice.

But if the systems administrators are responsible for allocating physical disk resources directly to applications, care needs to be taken to spread the physical data directories for each partition of a multifile as evenly as possible across the disk controllers and spindles available. This avoids disk contention on a single controller or a single spindle becoming the performance bottleneck in the application.

Multifile partition sizes

It is important that the filesystems created for use as multifile system data partitions be uniform in size. Data tends to be partitioned very evenly across the partitions of a multifile system. So if one partition is much smaller than the rest, the entire multifile system will effectively be full when the smallest partition is exhausted, and space on other filesystems will be wasted.

Similarly, you should not put other large data, such as a serial data partition, on a multifile mount point, as that partition will be exhausted first, wasting space on the other partitions.

For example, if an application is to run 12 ways parallel in a production system using space on six spindles across two controllers, the systems administrator might create six equal sized mount points for the data partitions, one on each spindle and three on each controller. Then the 12 data partitions would be spread out two per filesystem.

Multiples and backups

Another thing to be careful about with multifiles is consistency across backup sets. A multifile is a single logical entity composed of multiple parts spread across multiple disk volumes. So, in addition to the normal considerations that arise from running backups while applications are running and files may be incomplete, you must also keep in mind that a file may be spread across many volumes.

For example, if a multifile consists of 12 pieces spread across 12 data volumes, and one of these volumes is lost and restored from a week-old tape, it is possible that the one restored partition will now be out of sync with the other 11. You can either decide simply to understand this issue and live with it when doing restores, or use application level shell scripts or generic Ab Initio applications to back up and compress consistent snapshots of multifiles to a single volume for backups.

Continuous Flow queues

In addition to supporting connections to other queueing mechanisms, such as MQ or JMS queues, the Ab Initio Co>Operating System with the Continuous Flows option installed also supports its own high speed disk based queues.

Even if the primary use of Ab Initio continuous flows at your site will be to access data from external queues and deliver it to other external queues or database tables, it is common for multiple Ab Initio applications doing different portions of queue processing to communicate with each other internally through Ab Initio queues. These queues have similar requirements to the Ab Initio EME datastore space mentioned above: hence file locking and flushing *must* work correctly. The size of this space, and whether it experiences large sequential reads and writes or many smaller ones is dependent on application design.

CPU and memory allocation

CPU and memory allocation is performed by the underlying operating system, not by Ab Initio. However, there are certain facets of Ab Initio which systems administrators should be aware of when configuring environments for Ab Initio applications.

One of the main reasons clients buy Ab Initio is to be able to make sure their applications are making maximum possible use of all hardware resources to complete their tasks as quickly as possible. A properly tuned Ab Initio application is likely to create enough work to keep all available CPUs 100% busy, and often (under developer control) will use large fractions of available memory for high speed in-memory operations as well. This can have many consequences.

Scheduling

High-volume Ab Initio batch applications and user queries may interact poorly when run on the same set of processors at the same time. Users on such systems will likely see unacceptably long response times when Ab Initio jobs are being run.

There are many ways to manage this; for example, the Ab Initio applications can be made to use less than 100% of the available resources. However, as using 100% of the resources to get the batch jobs done as quickly as possible is usually the goal, it is often more productive to schedule user queries and high-volume applications to run either on different servers or at different times of day.

Sharing server memory resources

Similarly, consideration needs to be given to sharing memory resources on a server that may permanently tie up large areas of physical memory, such as Oracle's SGA segment. This is certainly possible, and in fact is often done.

On the other hand, you may choose to make different memory tuning choices on a system shared between Oracle and Ab Initio, as compared to one devoted solely to Oracle. Processors these days on servers used for Ab Initio applications usually have 1 gigabyte or more of physical memory available per CPU. This is usually enough to allow developers to use many optimization techniques in their graphs that take advantage of this memory.

In cases where, for example, the available memory is around 250 megabytes per CPU or less, most of it may be tied up just in executing applications, leaving little or none available for in-memory optimizations. In extreme cases, especially when multiple applications are running simultaneously on servers with limited available memory, it is possible for memory to be exhausted, resulting either in paging (and dramatically reduced performance) or swap space running out and applications failing.

Mixed use servers

Finally, there is a somewhat more subtle performance consideration involved in running multiple different types of applications simultaneously on the same server.

One of the performance benefits Ab Initio gets over other types of applications, such as databases, is that it tends to do its disk I/O in large contiguous blocks, since the Co>Operating System is oriented towards processing streams of data. This type of operation is typically very fast, even on inexpensive disks.

The random I/O used by databases, on the other hand, results in the disk head being moved frequently, which is often a slow operation even on very expensive disks. The result is that many database operations tend to be disk-bound.

The CPUs of database servers are thus often used much less than 100% of the available time because they are waiting for disk seeks, whereas Ab Initio applications typically get 100% CPU utilization, with most of that time spent actually working on your application.

Sharing a server between a database and Ab Initio

Now consider what can happen when an application such as a database is run at the same time as an Ab Initio application, on the same server.

The database will be issuing lots of reads all over the disk, while Ab Initio is trying to do big block reads. The net result can be that since the database is constantly moving the disk head, while Ab Initio's applications' reads subsequently need to move it back again, you can end up with overall disk performance as poor as the database's.

Thus, if Ab Initio is sharing space with a database, it is desirable to try to make sure that heavy database access and heavy Ab Initio access are scheduled at different times, or just in different "phases" of the application.

For example, one of the best practices recommended for developers at your site might be that they typically land load-ready files *after* all their heavy transformation processing, then load as a separate phase of the application.

Shared filesystems

Clearly, none of this is an issue if Ab Initio and the database are on separate servers. It may or may not be an issue if Ab Initio and the database are on *separate filesystems,* as in some cases the underlying storage arrays may use the same spindles for multiple mount points. Since these mappings are often hidden in several layers of indirection, this situation may be difficult or impossible to control or even diagnose.

Multiple installed versions of Ab Initio

It is usually possible, and often desirable, to have multiple versions of the Co>Operating System installed simultaneously on the same server. Which version is used by applications can then be determined by settings within the applications, by the setting of the $AB_HOME environment variable in the environment (and having **$AB_HOME/bin** in the PATH), and/or a default link set by the systems administrators.

When an administrator installs Ab Initio on a server, for example in the default location of **/usr/local,** Ab Initio will create a unique directory (e.g.

```
/usr/local/abinitio-V2-10-15-m23
```

) for that version. It will also prompt you to specify whether or not it should set a default link (by default, **/usr/local/abinitio)** to be a symbolic link to that version, or to leave it at the old version.

There are several reasons to want to support multiple versions of Ab Initio at the same time. While upgrading, for example, you may want to leave as your default version the one that is in production, and do testing on the new version. Once testing is completed, the defaults can be changed to the new version.

Compatibility between versions on different servers

Ab Initio can launch applications cross-platform across multiple servers. For this to work, compatible versions of the Co>Operating System that can talk to each other must be installed on both servers. Usually, releases within the same "minor version" (e.g. 2-10-X and 2-10-Y) can talk to each other, and releases from two different "minor versions" cannot. The release notes that ship with Ab Initio releases lay out very clearly which versions can interoperate.

Protocol number

You can also interrogate the releases themselves about compatibility. If you run the Ab Initio command "m_env -v" to get the Ab Initio version number, you will see output like this:

```
ab initio version 2.10.15.m23  P2_10_l
```

In this example, **P2_10_l** is the *protocol number.* Two different Ab Initio releases with the same protocol number will be able to interoperate. When an Ab Initio application needs to run across two servers, the servers must have versions of the Co>Operating System installed that share the same protocol number, and the application must run under these versions on both platforms.

Multiple platfforms

It is worth noting that not all of the approaches described above work on all platforms. NT platforms, for example, do not have symbolic links. However, by choosing a path that includes the version number at install time, and by using different settings for AB_HOME instead of a

global setting for the whole system, this can be worked around. On OS/390, only a single version can have its libraries installed in the "LPA library" for maximum performance, though again there are ways to deal with this that are documented separately.

Interoperation with other software

Ab Initio applications run under specific operating system versions. They also often access third party software, most commonly databases, which have their own software versions. Whenever considering an upgrade of the operating system Ab Initio is running under, or of third party software such as databases being accessed by Ab Initio, you should always check with Ab Initio (781-301 -2000 or support(5),abinitio.com) whether your current Ab Initio release supports the new other software releases. If the software release you are moving to is a major new release and is very new, please give Ab Initio as much advance warning as possible to make certain we support the release before you need it.

Job scheduling

Each Ab Initio application, when deployed to a production environment, executes as a single ksh script. This script is the single point of control and contact as far as scheduling is concerned. The script returns a single error code: zero for success and non-zero for failure.

Of course, Ab Initio applications can report multiple errors. And these error codes can come from anywhere: the Co>Operating System itself, the operating system, third party software such as databases, or even wrapped user code from other arbitrary applications. Since there is no way to get all this into one error code, the Ab Initio script-returned error code indicates only the success or failure of the application as a whole. Individual error messages generated during execution are reported to the scheduler via stderr, and tracking information and any additional output are available through stdout.

Note that a single Ab Initio application may start many components on many different servers that run Ab Initio. However, all monitoring and error reports will funnel back to the launching point and be returned in stdout and stderr to the scheduler.

Scheduling and performance

There are certain considerations to take into account regarding performance when scheduling Ab Initio applications:

- Ab Initio applications can be configured to consume 100% of the resources of a server (and thus finish execution as quickly as possible) in any situation where they are handling more than the most trivially small amounts of data.
- An Ab Initio application that has access to all or most of the resources of a server can be tuned (for example) to use large amounts of memory to optimize certain operations using in-memory techniques.

Because of these two considerations, it is almost always the best performance choice to schedule only one application to run at a time, with each such application tuned to use all resources to finish execution as soon as possible.

If multiple applications are run at the same time, the performance of each of them will be at best equal to the others—if no resource constraints are exceeded. However, with the knowledge that multiple applications can be run at the same time, developers will have to be conservative and use only a fraction (instead of virtually all) of available resources such as main memory for their applications. This will in turn limit the optimization techniques available to each application, and it almost guarantees that running multiple applications simultaneously will result in worse performance than running one at a time.

Simple connectivity

There are three types of connectivity to think about when configuring a server for Ab Initio:

- Connectivity of the client desktop development environment (the Graphical Development Environment or GDE) to the Ab Initio server
- Multiple-server connectivity — which allows a single application to start pieces of itself across multiple servers (if they have the Co>Operating System installed) from a single "launching node"
- Connectivity to the Ab Initio Enterprise Meta>Environment datastore

GDE connectivity

Connectivity of the GDE to the server is set up through the **Run > Settings** dialog in the GDE. Users specify the hostname on which the application will be launched, their own Unix (or NT or MVS) user ID, their own password, the location where the Co>Operating System is installed on that machine, and their working directory.

Users also specify a "connection method," the protocol the GDE will try to use to connect to the host, transfer files, and start processes. The main connection method choices are:

- **telnet** (plus **ftp** for file transfers)
- **rexec** (plus **ftp** for file transfers)
- **telnet** only (file transfers via Ab Initio's server)
- **rexec** only (file transfers via Ab Initio's server)
- ssh only (file transfers via Ab Initio's server)

(There is another choice called "Local NT Host" for the situation where the GDE and the server software are both installed on the same NT box. This is not usually used, but the situation would occur, for example, if someone were developing remotely on a laptop, or for demos.)

telnet

Suppose you use telnet with file transfers via Ab Initio's server (the third method in the list above). This means that only telnet ports need to be available. Also, telnet picks up settings from the user's login environment. You can use umask within the .cshrc file to allow users to share write access to files within Unix groups.

"Best practices" should require users not to put any definitions of application variables within any of their login shell definitions (.login, .profile, .cshrc, .kshrc). If applications were to depend on such settings during testing, they might not run in a production environment where the variables were not set. The recommendation should be that all variables required for an application be defined in the Ab Initio project settings or the graph (application) parameters.

The only added complexity of telnet over rexec is that, since telnet is also used for terminal access, it needs to be able to recognize when telnet is prompting for a user ID, prompting for a password, or has issued a prompt and is waiting for a command. There are simple scripting commands available in the advanced telnet settings if the users' prompts do not match the default patterns, but it is usually simpler to have Ab Initio users use a common profile to set prompts in a standard way that meets the default patterns.

GDE connectivity and the production environment

GDE connectivity may or may not be available to production machines for use in product support. However, the GDE can be very useful for debugging in production, even if it is restricted to read only access. Without access to the GDE, production support users may have to use text based log files and Ab Initio facilities from the Unix command line (which may be less familiar than the GDE to them) to collect execution and performance data.

Multiple-server connectivity

The second connectivity issue that can arise is the possibility that a single application may run across multiple machines, so long as they all have Ab Initio installed.

If the single user ID that launches an Ab Initio job has rexec access to all the machines and directories the job will run across, and if Ab Initio's install directories are the same across all the machines, running over multiple servers requires no special configuration. You simply start the job under the specified ID. When a component in the application is specified to start on a remote server, Ab Initio starts it there (under the same ID) with rexec, and assumes that the various pieces of an Ab Initio installation will be found at the same locations on the remote server as they occupy on the launch server.

Testing multiple-server connectivity

A simple way to test whether multiple-server access is configured properly is to do the following:

1. Log into the ID in question on the machine that will be launching the applications.
2. Make sure AB_HOME is set to the location where Ab Initio is installed and that $AB_HOME/bin is in your PATH.
3. Issue the following command:

   m_ls //computername/ for each computer (specified by *computername)* that the application is to execute on.

   This should list the root directory of the machine in question. It will fail if the configuration is not correct to access that machine.

If this fails, one (or both) of the two following things is true about the machine for which the failure occurred:

- The ID does not have rexec access to the machine.

You may be able to correct this by granting the ID access via the **.rhosts** file.

- The locations of the Ab Initio install directories (such as $AB_HOME and $AB_WORK_DIR) on the machine are different from those on the launch machine.

These problems can be corrected by editing a configuration file called:

$AB HOME/config/abinitiorc

In the same directory is a file called **abinitiorc.examples,** which gives examples of how to edit the file. More detail can be found in the Ab Initio *Co>Operating System Administrator's Guide.* (It is also possible to make these settings on a user-by-user basis with a **$HOME/.abinitiorc** file, using the same syntax.)

Enterprise Meta>Environment connectivity

The third connectivity issue to consider is access to the Ab Initio EME, which can occur in three ways:

- from the GDE
- from a web browser
- from executing applications

Access to an EME from the GDE

This is set up through a **Project > EME Datastore Settings** dialog similar to the GDE's own **Run > Settings** dialog mentioned earlier.

The dialog appears to offer fewer choices, but that's only because the question of whether file transfer happens via ftp or not is irrelevant here. All transfer occurs via the selected protocol of telnet, rexec, or ssh. Otherwise, the choices and issues are identical to those for the GDE Run Settings described earlier.

Access from a web browser

From their web browser, users connect to a login screen that offers, via a web form, the same choices available via the EME Datastore Settings dialog. Users thus specify an ID, a password, and the location of the EME (which can include the hostname if the EME is on a different host than the web server). Hidden options, which become visible when the user presses a "show" button, include additional connection method settings and telnet scripting.

Note that if the EME is to be located on a host other than the web server, both the web server and the EME server will need to have the Co>Operating System and the EME software licensed and installed.

From applications

For running applications, the environment variable AB_AIR_ROOT specifies the EME datastore location, including the machine name if the EME is on a different server. As for all other accesses from running applications, these will use either default connection methods or those specified in **$AB HOME/config/abinitiorc or $HOME/.abinitiorc,** as described above, to access the EME server.

Summary

In each of the cases described above, the connection protocols establish connections to the EME from a different starting point:

- The GDE connects directly from the client desktop to the EME.
- The web browser uses web connection to the web server, but then uses the connection methods, ID, etc. supplied to connect from the web server to the EME server.
- Running applications access the EME from the "launching node" where the application is started.

Usually, all of these can use the same access methods, but it's possible that firewalls may further complicate the picture if different protocols are available from different points.

Business users

Access to a production EME via a web browser for business users, operations users, and the like, is a topic in itself, and is covered in the Ab Initio *Guide to Managing Enterprise Metadata.* Business metadata access can be through Unix permissions, but usually only the web server ID is used, which gives limited access. The user ID and password entered on the login screen by the user refer to a list of Ab Initio maintained IDs and passwords within the EME, and grant or deny access only for browsing metadata within the EME itself.

Database and other third party connectivity

For databases local to the machine Ab Initio is installed on, there is typically no issue. For connectivity to remote databases, there are two possibilities.

- The remote database server has Ab Initio installed

  In this case you simply instruct the Ab Initio application to run the database components locally on the database server, and the Co>Operating System handles data transport transparently.

- The remote database server does not have Ab Initio installed

  In this case the necessary client software for remote connection must be installed on a server running Ab Initio. The Co>Operating System can then run the database components on that server, and the database client software (for example, SQL*Net) handles data transport. This usually requires some configuration of the database client (for example, TNS names for Oracle), but no additional work is necessary to set up Ab Initio for a remote versus a local database.

  The same applies to certain other third party interfaces. For example, for the Ab Initio Continuous Flows product to access a queue, the Co>Operating System must either be licensed and run on the system on which the queue is defined, or there must be a middleware client installed that gives the Co>Operating System remote access.

Remote access to files

Remote access to files is possible directly through the Co>Operating System if the files are located on a machine where Ab Initio is licensed and installed, or via the Ab Initio ftp components, which ftp data directly into or out of a running Ab Initio application without requiring landing the file someplace first.

Multiple connectivity paths

In the simple world of desktop PCs, most computers have one host name, one IP address, and one speed of network connection to choose from. In the world of multiprocessor servers, there can often be many names and many IP addresses for a given particular system, with each such name or IP address bound to a particular interface. There may be multiple interfaces, each with a different speed. (Note however that this is a fairly unusual circumstance in the SMP world until you reach the point where a single SMP is not sufficient and a cluster is required.)

A simple example

Suppose servers alpha, beta, and gamma are part of a "cluster" of E6500 Sun servers. They are connected to each other by a fast fiber-channel that operates at speeds of gigabytes per second. They are all also connected to the corporate intranet via a standard 10 or 100 megabyte per second ethernet. The fiber channel connection is faster, but it connects *only* to alpha, beta, and gamma.

Each machine would have at least two host names. For example **alpha_eO** might be alpha's ethernet name, while **alpha_fO** would be its name on the fiber channel. In addition, suppose omega is a server someplace else at the site that is not on the cluster. It has only one name, **omega_eO,** on the corporate ethernet WAN.

Distributed applications in Ab Initio

Ab Initio applications can be fully distributed across any servers on which Ab Initio is installed. The Co>Operating System takes care of moving data between platforms as needed, using TCP sockets. Because these are generally high volume applications designed to execute as quickly as possible using whatever resources are available, it is desirable to have the data that Ab Initio needs to move across platforms use the highest speed path available. However, this path may vary depending on the source and target of the data.

Taking the example described above, let's say data files in an application land on omega, are sent to alpha, and then are partitioned and processed in parallel across alpha, beta, and gamma. We need to be able to specify that:

- between omega and alpha, data uses the **omega_eO** to **alpha_eO** interfaces
- among alpha, beta, and gamma, data uses the **alpha_fO, beta_fO and gamma_fO** interfaces

Layouts

Data routes are specified in Ab Initio using *layouts.* Each component in an Ab Initio graph has its own layout, which tells the Co>Operating System "where" the component runs and "how parallel" it runs. For dataset components, the layout specifies the location(s) and parallelism of the data.

For example, a component layout might specify "serial (no parallelism) on **alpha_eO."** Another might specify "72 ways parallel—24 each on **alpha_fO, beta_fO, and gamma_fO."** Every component in an Ab Initio application has its own layout, although most of these usually are not explicitly specified because they are propagated from components where they are explicitly set.

- A layout for a multifile specifies where each piece of the multifile lives. This includes which machine, and which directory on that machine, the piece goes in.
- A layout for a component in a graph specifies the node that the component runs on and the directory on that machine in which the component will run.

Multifiles are typically built by the systems administrators. For a complete overview of multifiles, see the Ab Initio *Co>Operating System Administrator's Guide.* In the following discussion we will concentrate on what all this means in terms of multiple connectivity paths.

Layouts and multiple connectivity paths

Because you specify host names as part of the location information in a layout, you are also specifying which interface will be used to get at each partition of the data, if it is located on some other platform. Therefore, when you build multifiles for systems with multiple interfaces, you must be careful to specify separate multifile systems for intra-cluster communication versus communication outside the cluster.

In the example mentioned earlier, data might flow two ways parallel from omega into a two-way multifile system on alpha built by a command like the following:

```
m_mkfs
    //alpha_eO/diskO/control/landing/mfs2way \ //
    alpha_e0/diski/data/landing/lof2 \
    //alpha_eO/diski/data/landing/2of2
```

We'll call this "layout E".

At the same time, three-way parallel partitioning among alpha, beta, and gamma might take place in a multifile system created by a command like this:

```
m_mkfs //alpha_f0/diskO/control/work/mfs3way \
    //alpha_f0/diskl/data/work/lof3 \
    //beta_f0/diskl/data/work/2of3 \
    //gamma_f0/diski/data/work/3of3
```

We'll call this "layout F". Components running in layout E will communicate with components they are connected to on gamma using the corporate ethernet, because the name **alpha_eO** is used throughout. Components running in layout F will communicate with components running on other computers through the fiber-channel, since the _fO names are used throughout there.

Connecting components in different layouts

What happens when you connect components running in two different layouts such as the ones described above? —The answer is that if that's all you do, the application will of course fail. A component communicating through the ethernet cannot talk directly to a component on the

fiber-channel. They are using different addresses and the network has no way to get them to talk. How then do you connect data coming into alpha two ways parallel on the ethernet, using the layout E, to components running three ways parallel across the cluster, using layout F?

Getting the data from the first layout into the second is a two-step process.

1. You must first repartition locally, in another component also running on alpha, to a layout created by a command like this:

   ```
   m_mkfs //alpha_f0/diskO/control/if/mfs3way \
          //alpha_f0/diskl/data/if/Iof3 \
          //alpha_f0/diskl/data/if/2of3 \
          //alpha_f0/diskl/data/if/3of3
   ```

   A data flow between layout E and this one occurs entirely within alpha, and Ab Initio will not use TCP connections or IP addresses at all for it. Instead, it will use something local to alpha (by default, "named pipes") to connect these components. But note that the layout above, though on the same computer, is now in the fiber-channel.

2. Now you can connect the component with the above layout with components in layout F across the cluster. Only _fO names are being used, so the components can all talk successfully and at maximum possible speed.

The reason for going through all this is to make it clear that the systems administrator dealing with a cluster usually needs to create two different layouts (rather than the usual one) to implement a given level of parallelism:

- a "fast" connection layout that is used in 99% of the cases
- a "slow" layout using only ethernet addresses, all located on one node of the cluster, for communications outside the cluster

Specifying hostname synonyms

How did Ab Initio know that the flows from **alpha_eO** to **alpha_fO** were all actually the same node? The answer is that there's no way for Ab Initio to figure that out on its own. The systems administrator has to specify explicitly that these are all synonyms for the same node.

This is done through configuration variables defined in detail in the Ab Initio *Co>Operating System Administrator's Guide.* The variables are usually defined in a configuration file located at:

$AB HOME/config/abinitiorc

A second file located at

$AB_HOME/config/abinitiorc.examples

contains examples of how to set up several connectivity situations. In the case we are discussing here, the file would contain a line like this:

```
AB_HOSTNAME @ alpha_eO : alpha_fO
```

—which specifies that the hostname **alpha_eO** should be treated as the same node as **alpha_fO.**

The "launch" interface in a cluster

Typically, if an application is going to be launched on a cluster, it is launched from one of the nodes in that cluster. There has to be a specific location where the scheduler, such as AutoSys, kicks off the job. Once the job is started, Ab Initio can use the layouts for the individual components in the manner described above to determine where to launch the application's components and how they communicate with each other. But first the launcher needs to know which interface to use to launch the job itself, and which interface will be used to report errors and runtime statistics back to the launcher. What interface will the launcher use?

Unfortunately there's no way, either within Unix or with schedulers such as AutoSys, to specify that the Ab Initio launcher be started using a particular interface, as Ab Initio itself does for runtime components. So the default is that Ab Initio will use whatever interface is returned by the **hostname** command on the launching node. This is usually the ethernet address. The ethernet address generally has connectivity to everything, so this will virtually always work, but it may not always result in using the fastest available interface.

Specifying a launch interface

If the launching node and all components in the graph being launched are within a cluster that has a higher bandwidth interface available, you may want to use that interface. To tell this to Ab Initio, you set a variable called AB_HOST_INTERFACE in the environment of the job to the hostname on the higher bandwidth interface. This makes Ab Initio use the higher bandwidth interface to launch the components, and it also makes the components use the higher bandwidth interface to report back.

Since applications may be launched from many machines, this setting often cannot be hard-coded. However, there are usually regularities in naming conventions that allow a systematic method of determining the proper interface name. For the examples discussed above, the following might work:

```
export AB_HOST_INTERFACE =$(hostname | sed 's/_eO/_fO/')
```