



Scaling Linux for Large Clusters

Submitted as partial fulfillment of Subcontract B503004

Revision 1.3 of 99/03/08

BitMover, Inc.

Tel: 415-722-5553

<http://www.bitmover.com>

1. Effort estimates

Each item has been amended with an estimation of the amount of effort required. There are 6 ranges of effort:

- [P] Unknown because of pending clarification.
- [0] No effort required, RedHat Linux provides this feature.
- [1] Small amount of effort required, 1-4 weeks.
- [2] 1-2 months of effort required.
- [3] 2-6 months.
- [4] More than 6 months.

2. Recurring themes

2.1. Cluster Communication

Since this is a cluster, there is a need to be able to do something, ask something, watch something, etc., over all nodes in the cluster. If each application needs to build this facility itself, we'll never get done. This implies that an additional requirement is to have a system level API which can essentially send an RPC to all nodes and get an answer in a short amount of time. RPC vectors are the brute force way and may be good enough,

especially if it is a cluster of large SMP boxes. If it is a cluster of 10,000 Uniprocessor systems, RPC vectors are not the right answer. A system like that needs a log based fanin/fanout substrate.

2.2. Communication vectors

Many of the API's which are asked for in the ASCII SOW are essentially extensions of existing API's to a cluster (and/or to an SMP; many of the API's had uniprocessors in mind and haven't been updated).

A facility which used some combination of vectors, lists, and/or regular expressions to describe the set of nodes on which the API's should be called would seem to be the right sort of answer. In general, it would be nice to be able to tell programmers that "it is just like setrlimit(2) except that you wrap it up in this cluster wide layer." The two problems are orthogonal and should stay that way.

Assume 4 way machines, 10K CPUs = 2500 nodes (this is purposefully conservative, it's more likely to be 8 way machines). If we have a single machine gathering statistics, with a payload of 3KB/host, that's 7.5MB/sec. A P5@166 talking to a K6@233 can easily do 10MB/sec using less than 50% of the CPU on both ends.

3. Required features

§ 4.2.1.1 SMP OS

POSIX 1003.1-1990 and 1003.2 (system calls, library calls, and user level commands) as well as 1003.4 (pthreads).

Intent: basic Unix functionality including pthreads.

Question: hasn't 1003.4 been renamed to 1003.1c?

Question: is 1003.4 just for the pthreads interfaces?

Effort: 1-2 due to verification of correctness and completeness. Could expand into greater effort if verification uncovers unexpected gaps in conformance.

§ 4.2.1.3 Networking protocols

Standard Internet protocol suite.

Effort: 0 - exists today.

§ 4.2.1.5 Group routing

Routing tools which segregates network traffic.

Intent: unclear. It sounds like it might just be a grouped routing command.

Effort: P.

Question: What is this?

§ 4.2.2.1 OSF DCE

DCE v1.1 and client for DFS

Intent: unclear. It would be useful to have these sections restated as requirements rather than a solution to the requirement. DCE is unlikely to ever be a standard part of Linux.

Note: See <http://www.bu.edu/~jrd/FreeDCE> for status of DCE on Linux.

Question: For the security component, is readily-available Kerberos 5 enough?

Effort: 4 if you want an open source version. Probably 4 no matter what, what is available now is AFS client

side, no server.

§ 4.2.2.1.1 DFS Server

OSF DFS file server.

Question: Is it possible to do this with a non-cluster translator machine?

Intent: export data from the cluster securely, with ACL's.

Effort: 4. There is no DFS server for Linux today. It may be possible to get IBM to release a version on Linux, they are much more interested in this space at this point.

§ 4.2.2.1.2 Cluster wide service security

All services done securely.

Intent: security between nodes in the cluster.

Question: When comparing a cluster to an SMP, the SMP authenticates at the boundary - once you're in, you're in - it doesn't authenticate each process creation, etc. Wouldn't it be better to make security work at the boundary and then have AUTH_UNIX style security within the cluster?

Question: For the boundary, would Kerberos 5 be sufficient?

Effort: P.

§ 4.2.2.1.3 Transarc Encina

Transaction system for High Performance Storage System API.

Question: What is the HPSS and why does it need Encina transactions?

Note: IBM can probably be talked into providing this.

Effort: 1 if IBM does, else 4.

§ 4.2.3.4 Job management

Nicely stated cluster process group definition and requirement. Basically describes an extension of a process group which spans host boundaries, and requires that the cluster process group can be used like a regular process group (signaled, etc).

Intent: extend process semantics to

have meaning in a cluster.

Effort: 3-. Do this by making processes objects much like files are objects. Have local process instance, remote process instance, debugged process instance, etc.

§ 4.2.3.5 Job scheduling

A request for more advanced scheduling features which are cluster wide.

Question: This is really two requirements, right? The first is that you can schedule across the cluster. The second is that the scheduler takes into consideration several metrics.

Note: GNU Queue can do quite a bit of the this but does not have the resource feedback/tuning you want.

Effort: 2+

§ 4.2.4.1 Administration

Requires a mechanism for administering and monitoring the entire cluster from a single node.

Question: Does the Sandia CPLANT work on this solve the problem?

Effort: 3-.

§ 4.2.5.1 Languages

Requires Fortran90, C, and C++ with parallel features.

Question: (to self:) What is the status of Portland Group front ends for Linux?

Effort: Unknown, pending contacting the Portland Group.

§ 4.2.6.1 Debugger

Cluster wide debugger with GUI interface.

Question: What are the requirements for debugging multiple-process-instance cluster apps? Debuggers that can handle multiple threads in a single process are available.

Effort: Depending on how well the remote process stuff is integrated, this could be trivial (if the local multi threaded debugger is good enough) or a huge effort.

§ 4.2.6.4 Profiler

Profiler with thread support.

Question: What are the requirements for multiple-process-instance apps? (Multiple thread profiling is easy.)

Effort: 2- if I understand this properly.

§ 4.2.6.5 Event tracer

Dynamic event tracing including support for threads.

Question: Is this just an extension to strace/ltrace? Is it a requirement to support application-defined event types?

Effort: 2+. This can best be done by putting the event entry points in a shared library. Then use ltrace to watch those events. The only difficult part is the cluster wide aspects of this.

§ 4.2.6.6 Performance tools

Tools for gathering and storing performance statistics at both the hardware and the OS level, including processes and threads.

Note: Some statistics mentioned may not be maintained by the hardware.

Effort: See next section.

§ 4.2.6.8 Cluster development tools (GUI)

Tools which will display information such as performance statistics mentioned in other sections.

Effort: 3.

§ 4.2.7.1 Linker/library

Standard Unix linker/loader/library support.

Effort: 0

§ 4.2.7.2 make

Standard Unix make

Effort: 0

§ 4.2.8.1 MPI library

MPI library support for both networks and shared memory.

Question: What level of optimization is required?

Effort: 3 if heavily optimized SMP

versions are needed. SGI had to change their OS in some ugly ways to get the performance they wanted.

Note: It is probably not necessary to optimize this for the SMP case under Linux. Linux is so light weight that the optimizations are likely to be a wash. The one exception is large data transfers and that probably should be done with explicit page flipping.

§ 4.2.8.6 X & Motif

X11R6 and Motif.

Effort: 0 - exists today.

§ 4.2.9.2 Audit

Login/logout and denial record for each login attempt.

Effort: 0 - exists today.

§ 4.2.10 DOE security mandates

Translates to a requirement for all source code and/or vendor supplied fixes.

Effort: 0 - exists today.

§ 4.2.12 On site support

At least two bodies with functioning brains on site.

Question: For how long?

3.1. *Optional features*

§ 4.2.1.1.1 X/Open

XPG4 brand.

Effort: Unknown. X/OPEN may get a lot more interested in this due to increased support for Linux from traditional vendors such as SGI, HP, and IBM. On the other hand, Linux has more Unix seats than all of those guys put together so X/OPEN trying to evolve Linux is somewhat like the tail trying to wag the dog.

§ 4.2.1.1.2 Single system image

Support for making the cluster look somewhat like one machine to outside users and to jobs on the cluster.

Effort: 3. This seems to be a repeat

of similar material elsewhere. So this effort level is a duplicate.

§ 4.2.1.1.3 Alternative threads packages

May propose a lighter weight threads package.

Question: If the package was found to be acceptable, is it in lieu of or in addition to pthreads?

Effort: 1 - the suggestion would be to use clone() and user level locks. The locking needs some work but clone() already is there. pthreads is built on top of clone.

§ 4.2.1.1.4 GNU utilities

GNU utilities desirable.

Effort: 0 - already done.

§ 4.2.1.1.5 Job check point / restart

Check point / restart desirable.

Question: We've discussed restrictions on this, such as no socket support. How does that fit with the MPI requirement? Can this feature be more clearly specified?

Effort: 3- assuming some restrictions such as the entire set of processes on all nodes are check pointed and restarted on the same nodes.

§ 4.2.1.2 Fault tolerance

A request for some level of fault isolation and tolerance.

Question: If this took the form of not affecting jobs unless they were on the node that failed, and subsequent jobs would just ignore the failed node, is that acceptable?

Question: In other sections, job scheduling seems to imply mechanisms which will specify which nodes the jobs will run on. What happens when a node is unavailable? Shall the entire job be stalled?

Effort: P.

§ 4.2.1.4 Third party transfer

The ability to move data between disks without involving the CPU other than initiation.

Question: Does this mean at the block level or at the file level? I assume file level.

Effort: 1+ if we are talking about using the new smart network disks about to appear from the disk drive folks.

§ 4.2.1.6 File systems

Cluster wide file systems, with support for at least terabyte files. Also journaling for fast recovery after crashes.

Effort: 4. This is hard, but doable.

§ 4.2.2.2 Object request broker CORBA support.

Effort: 0 - exists today.

§ 4.2.3.1 Cluster wide resource management GUI view and administration of the resources in the cluster.

Effort: 2.

§ 4.2.3.1.1 Cluster wide resource limits

Standard Unix resource limits applied to the cluster.

Question: We talked about this - as I recall there were additional limits beyond the standard ones. I also believe we had almost come to the agreement that if the resource limits were somewhat more fine grained, the standard Unix API would work. Is this correct?

Note: Standard Unix API's for this are `getrlimit(2)` and `setrlimit(2)`.

Effort: 2. Depends on the cluster communication substrate.

§ 4.2.3.1.2 Resource management API

API for cluster wide resource management.

Note: This one and the previous one seem to be two part: one part is finer grained control over the resources, down to the CPU, disk, file system, memory, network level; the second part is an extended API for manipulating the resource limits and setting

policy on actions taken when resource limits are exceeded. We need to spend some time defining exactly what the resources are and what the actions are.

Effort: 2.

§ 4.2.3.2 Cluster wide accounting

Job accounting across the cluster.

Note: Seems like this is also of the form (a) define the local API, (b) apply the local API to each node and aggregate the results. The only difference between the local version and the remote version is the addition of node identifiers in the data in the cluster case.

Effort: 2-.

§ 4.2.3.3 Non-user accounting

Notes that it would be nice to have the accounting system complete enough that all resources could be added up to 100%.

Effort: 2-.

§ 4.2.3.5.1 Gang scheduling

Requests the ability to schedule all processes/threads in a distributed job at the same time.

Question: This should just be part of the cluster process group operations vector, right?

Effort: 1+ - depends on cluster communication substrate.

§ 4.2.3.5.2 Cluster Job Scheduling / monitoring / admin

GUI interface to scheduling, monitoring, and administering cluster jobs. Also monitoring history.

Effort: 2.

§ 4.2.4.2 Cluster monitoring

Cluster wide monitoring and tuning of system parameters.

Note: This seems like yet another thing that ought to work on a per node basis and then be aggregated back to a GUI for the cluster.

Effort: 2

§ 4.2.4.3 Centralized repository

One place to go look for all system parameters.

Question: Is there any reason this can't be `/gproc`, an aggregation of all the `/procs`?

Effort: 2+.

§ 4.2.4.4 user maintenance

Localized user add/delete/etc tools need to work across the cluster.

Effort: 2- Part of sys admin interfaces.

§ 4.2.5.1.1 Preprocessor

C preprocessor; usable for all languages.

Effort: 0 - exists today.

§ 4.2.5.1.2 Cross referencing

Sounds like `cxref/cflow` except for all languages.

Effort: Unknown, I suspect these exist already.

§ 4.2.5.1.3 Assembly listing

Sounds like `gcc -S`.

Effort: 0 - exists today.

§ 4.2.5.1.4 C++ templates/exceptions

Request for semi standard C++ features.

Effort: 0 - egcs. with Cygnus.

§ 4.2.5.1.5 Cray pointers

Cray style pointers in Fortran77.

Effort: Unknown. Did you say that this was available in from a vendor?

§ 4.2.5.2 High performance Fortran

HPF v1.1 or v2.0.

Effort: Unknown. Did you say that this was available in from a vendor?

§ 4.2.6.1.1 Visual representation of data

Ability to visualize data being debugged.

Effort: 3 - this one sounds like you want to watch all data while the system is running. We need to talk about what that implies - is there a generic way to visualize what it is you want to

see or is this LLNL specific (or worse, application specific)?

§ 4.2.6.1.2 Conditional breakpoints

Fast conditional tests added to a program being debugged.

Question: Have you see the UPS debugger? It has a lot of what you are asking for and is free.

Effort: 0 - UPS does this and is free.

§ 4.2.6.1.3 Memory leak debugging

Purify like memory debugger.

Effort: 2-3 - Rational does not have a Linux version yet.

§ 4.2.6.1.4 Debugging optimized code

Should still have stack trace back, etc.

Effort: 0 - gdb/gcc do this today.

§ 4.2.6.1.5 Debugger language

Scripts on break points should be possible; compiled is better than interpreted.

Effort: 0 - gdb does this.

§ 4.2.6.1.6 Parallel break points

Defines a kind of break point which has meaning across multiple processes and/or threads.

Question: Isn't this a repeat of an earlier debugging feature?

Effort: easy if this is just thread support, harder if it is cluster wide.

§ 4.2.6.1.7 Post mortem debugging

Core dumps.

Effort:

§ 4.2.6.1.8 Symbol table

15 MB symbol should not impact debugger startup time by more than 1 minute.

Question: Number of symbols?

Effort: 0 - probably works today, need to verify.

§ 4.2.6.1.9 Data aggregation

Question: Is this about per thread data?

Effort: P.

§ 4.2.6.2 Stack traceback

Runtime ability to generate a stack trace.

Note: I believe this will be part of the next glibc release.

Effort: 0 - if you can wait for next glibc. 1 if you can't, I can extract a prerelease and make sure it works.

§ 4.2.6.3 Lightweight corefile

Lightweight corefile API.

Effort: Unknown.

§ 4.2.6.5.1 message passing event tracing

Trace send/receives/syncs for message passing libraries.

Question: Does this include sockets? Or just MPI?

Effort: 0-1 - as long as these are shared libraries, this is just some options to ltrace(1) and some code to aggregate it.

§ 4.2.6.7 Cluster API

API for turning on profiling, etc.

Note: This is another example of something which should be built on top of the communication substrate. The API should be just like the local API except in cluster form.

§ 4.2.6.9 Timer API

Parallel tools consortium timers.

Effort: Unknown.

§ 4.2.7.3 Parallel make

Something like the GNU **make -j**.

Effort: 0 - exists today.

§ 4.2.7.4 Source code control system

Something like BitKeeper.

Effort: 0 - BitKeeper will work just fine.

§ 4.2.7.5 Dynamic processor allocation

Dynamically specify the number of processors/threads without recompilation.

Effort: Unknown, this seems like it is a function of the MPI/PVM library.

§ 4.2.8.2 MPI2 library

MPI2 library when it gets standardized.

Effort: 1.

§ 4.2.8.3 Visual display of messages

Like XPVM.

Effort: 2.

§ 4.2.8.4 PVM library

Within 6 months of the release from ORNL.

Effort: 1.

§ 4.2.8.5 Parallel I/O API

Parallel I/O API.

Question: Is there a more detailed definition of this API.

Effort: P.

§ 4.2.8.7 Visualization API

OpenGL & PHIGS.

Effort: 0 - OpenGL is getting much easier with SGI's interest in Linux.

§ 4.2.8.8 Math libraries

Optimized.

Effort: 1-3 depending on how bad they are.

§ 4.2.9 Security

Something like ssh.

Question: This is just for login, correct?

Effort: 0.

§ 4.2.9.1 Login notice

Standard Unix login.

Effort: 0.

§ 4.2.11 Documentation

Online and hard copy.

Effort: 0.

3.2. Additional features

Cluster communication

There needs to be some sort of scalable, reliable cluster communication mechanism. Broadcast doesn't work well enough. What we have in mind

is some sort of N-way tree, where each node is responsible only for its children.

Unix API extensions

Almost all services desired by the University can fit in the existing Unix API, provided that there is some method of calling that API on all nodes and aggregating the results back to the originator. This is sort of a meta-service.

Logical file support

As the file systems get large, a single file system model doesn't work, nor does a location dependent model. There needs to be a level of indirection which will allow, among other things, the ability for a logical file to span multiple file systems, and a logical file name space which is location independent.

Parallel I/O API

The parallel I/O API described in the ASCII SOW appears to be incomplete. Combined with the logical file support, there needs to be a way to create a cluster process group with each process close to a portion of the logical file. In other words, the placement and number of the processes in a distributed computation is a function of the locations and number of chunks in a logical file. The placement process should include a mechanism for telling each process which chunk of the file it owns.

Network disks

Many of the I/O problems can not be solved well with locally attached disks. High speed remote disks may be needed. There needs to be some effort to make this sort of thing standard. The process of doing so is a multi year effort; if the University wishes to have such disks available in a usable manner in the next 3 years, the time to worry about it is now.

Scheduled bulk data transfer

Both the disks and network cards need to be taught how to do high volume bulk data transfers over the network.

Global file system on network disks

Given network disks, a small amount of NVRAM on the disks, scheduled transfer, and ST aware networking interfaces, a globally shared coherent file system is possible.

4. Summary

This pass includes estimates of effort for each item. I would like to reach agreement on the easy ones and essentially put them behind us. Future meetings will address each of the hard items in turn.