Enterprise-Wide Clustering: High Throughput Computing For Business

Chris Hamilton, *Rock Linux Developer*

***Abstract*—Most of today's important computational problems require hundreds of thousands of CPU hours to solve. A typical High Performance Computing (HPC) cluster (such as a Beowulf-class cluster) can require a significant investment to attain an acceptable level of total processing power. High Throughput Computing (HTC) utilizes available networking and computing environments to perform the same tasks in a comparable time frame but without the investment in dedicated hardware. In a typical enterprise, workstations can be idle for 75% of each workday. HTC tools manage this unused workstation time in a dynamic cluster configuration to run the same applications as an HPC cluster. HTC therefore creates clusters that any enterprise can already afford. This paper will discuss what an HTC cluster is, what makes it different from HPC clusters, and what uses there are for HTC. The discussion will also cover requirements and strategies for implementing an enterprise-wide HTC cluster.**

***Index Terms*—cluster, High Throughput Computing, message passing, parametric execution, process load-balancing.**

I. Introduction

Complex problems requiring large amounts of processing power to solve are no longer confined to academia. Businesses can now harness the power of parallel computing to provide solutions in such diverse fields as financial management and resource planning as well as research and development. The use of parallel computing can help to increase efficiency and minimize risks by modeling both internal and market processes.

High Performance Computing (HPC) parallel clusters are becoming an increasingly common sight outside their traditional academic domain. With the advent of Beowulf-class cluster design, many problems that were too complex to properly model have now become more affordable to tackle. However, the Beowulf cluster design still requires a significant investment. Beowulf clusters require special network designs and the purchase of multiple dedicated computers. After adding the cost of software and training, a cluster can still be too expensive for some businesses.

High Throughput Computing (HTC) clusters offer a similar or higher processing capability while significantly lowering overhead costs per node. By utilizing a business's available computers and networks, new hardware costs are significantly reduced. For the same reason, the cost of administration is also significantly reduced.

In general, for any cluster or new computer system, there are costs associated with training administrators and users. During the training period, the new system is essentially non-operational and is not adding value to the enterprise. This is not the case with HTC. The HTC system software resides on desktop systems and workstations that are already in current use by the business, and therefore do not lie idle during the training period.

HTC can even be utilized as a test bed for HPC applications that do not require low network latency. The minimal installation and administration overheads allow for low-cost experimentation. Even if it were eventually decided not to fully implement the HPC, the use of the HTC cluster would have minimized the experimental system cost. Also, the HPC application can be kept on the HTC for future projects and is easily kept updated.

Workstation and desktops used by the HTC need not be adversely affected by the HTC software installation. Computers that run only interactive applications such as office and Internet applications are likely to use less than 25% of their processing capability during the day [1]. If a HTC application requires the entire workstation's processor or memory, the application can be run after work hours or during the times the workstation is idle.

Running HTC on a business's computers enhances resource efficiency. The cost of electricity consumed in keeping computers turned on throughout the day can now be justified. Computers experience wear and depreciation whether they are utilized or not; it is therefore in the interest of the business to utilize them as much as possible.

This paper will discuss what an HTC cluster is and compare it to other cluster solutions. The paper will also cover the requirements and options for a HTC system, as well as implementation in a business. We also present an example of a complete set of HTC software.

II. HTC Comparison

Broadly, a HTC environment is defined as one that can

Manuscript received February 26, 2001. This work was supported by Ambiguous Computer Company Ltd..

Chris Hamilton Author is General Manager of Ambiguous Computer Company Ltd., Singapore (telephone: 65-582-4907, e-mail: chris@ambigc.com).

deliver large amounts of processing capacity over very long periods of time [2]. An HTC system is designed to accomplish extremely large computing tasks. The idea is that most useful applications of this kind cannot be completed in minutes, hours, or possibly days. Instead, application time frames tend to be on the order of months or years. Below, we discuss the differences between HTC, HPC and High Availability (HA) clusters.

Before we begin, some terminology should be defined to aid comprehension.

- application – software that the user runs on an environment to solve a problem.
- environment – general term relating to the cluster tools being spoke of, it includes all nodes and software.
- job – an application with a wrapper to request scheduling by the batch system.
- node – a single computer in a cluster.
- platform – a specific operating system and processor type.
- process – an individual executable application component, usually what gets distributed across nodes.
- processor – the individual CPU(s) in a single computer.

To properly understand HTC, one should understand how it compares to other classes of clusters. Many people are already confused about the differences between High Availability (HA) and HPC. Therefore we first discuss HA and HPC.

A. HA Clusters

Firstly, HA clusters are not intended to share the same workload among several computers. What a HA cluster is meant to do is to increase availability of a service [3] (as its name indicates). This means that a HA cluster performs the same jobs and at the same approximate performance of a single computer doing the task. The advantage it provides is fault tolerance and this is typically implemented for a client/server model application. Depending on what mechanism is employed, an HA server can have several redundant processing nodes hooked to the same data resource (such as two servers connected by the same SCSI channel to a redundant disk array). The servers are redundant and the disks are redundant, so that any faulty system can be replaced or altered without interrupted service. These systems are designed for data storage and retrieval.

HA clusters can also be designed with redundant servers that operate simultaneously with their counterparts to increase throughput. The line between HA and parallel computing can blur in the implementation of such systems. In this configuration the redundant systems are actually serving separate transactions simultaneously. In cases where clients are shared between redundant systems, there will also have to be some sort of state-data sharing rather than just resource access. This is usually facilitated by remote procedure calls or some similar method that is used in parallel message passing.

It should be noted that in any case, HA does not provide the resources required to run large problems or distribute arbitrary computations and data among computers. It can, however be an important part of any HPC or HTC system for data storage as the HPC/HTC system can be client to an HA server.

B. HPC Clusters

HPC computing does provide mechanisms to run large problems and distribute arbitrary computations and data among computers. The environments are built around dedicated nodes that are designed with high processor and memory resources and low network latency [4]. This is so they can process data and communicate with the other nodes as fast as possible. Network latency is of paramount importance for intense processing such as this, as a single network request can halt the whole node while the data is awaited. These systems can require a large amount of careful design and redesign to remove bottlenecks, reduce inefficiencies and achieve peak performance for each application.

By means of hardware and system software tuning, HPC allows its users to configure its resources to solve specific problems in the most efficient way for the cluster resources available. This is ideal for repetitive problems that have critical time requirements. Most realistic time critical projects that need this level of optimization run for less than a week.

The types of applications that can realistically be run on the system can be limiting. If the system is utilized by a large number of groups, the system will need to be shared using a queue or batch system. Either time limits are applied per job or the system is partitioned into smaller groups of nodes (making smaller clusters). This limits the clusters usefulness for very large problems. It should be obvious that single-process, long-runtime applications will not fit well into the cluster batch system.

C. HTC Clusters

HTC, on the other hand, does provide arbitrary distributed computations and data while not being as susceptible to the problem of fitting applications into the batch queue. By utilizing the idle processes of a business' computers, HTC can have a much larger pool of nodes than, say, a 16-node Beowulf cluster or even a 128-node Beowulf cluster, depending on the business's size. Thus there are many more options for queue or batch configuration and more applications can be

accommodated. The HTC can use whatever resources are available to accommodate any queued process. In fact, a HPC cluster model can be configured as a resource that can be run under a HTC batch queue. A HTC system can also benefit from the utilization of retired ("obsolete") computers running as dedicated HTC nodes as well. Requesting months of resource utilization in a HPC environment can be nearly impossible, as can allocating funds for a new specialized HPC cluster. HTC does not require nearly the same investment.

HTC comparisons to HPC often do not convey the true advantages of HTC over HPC. The cost factor should be viewed in context with the specific implementation of the environment. If HTC utilizes already-present computers in a business, the hardware savings are clearly apparent. However, the cost of entirely new nodes can also favor HTC systems.

With the decrease in price of today's desktop systems, both HPC and HTC nodes are quite cost-effective. The anatomy of a new HPC node consists of a processor, memory, motherboard, video, networking, and casing. (Some motherboards can be purchased with video and networking on board.) Although video is not required for the node's use, most motherboards will still require an installed video card to boot. A completely new HTC desktop/node would consist of the same components, but would optionally have extra resources for an interactive user. This user would utilize the system as a workstation and would possibly require a hard drive, keyboard, mouse, and, monitor. (The hard drive is optional depending on network data sources, but in general the HTC node can be considered a standard desktop system). The differences in cost per node are minimal with approximately US$500 difference between the HPC node and the HTC desktop components. Yet the desktop allows for interactive user login for normal day-to-day desktop use as well as providing a large percentage of the system resources as a HTC node. The HTC node is available for all normal computing uses, while a HPC is too specialized for more than cluster use. As individual computer components get cheaper, a standard desktop will become more economical faster than a specialized HPC node would. This is because HPC nodes usually have added costs associated with their design as they are non-standard items. This means the cost difference between competitive HPC and HTC nodes is minimal.

It should now be clear what an HTC cluster is and how it can benefit a business. HA does provide important solutions for businesses, but in a perpendicular manner to HPC and HTC. While HPC and HTC can be used for some similar applications and in the same environment, there are some specific differences in how they operate. In the worst case for a HTC running an HPC-optimized application, one would have an incredibly inefficient HPC solution. In the worst case for a HPC running a large HTC application, one would have to "hog" the cluster and use all the dedicated nodes for a long time. With all this in mind, HTC clearly provides a lower cost, more general-purpose solution.

III. HTC CLUSTER REQUIREMENTS

Requirements for a HTC system as specified by the Condor Team [5] are defined for their particular, restricted-license solution. Similar cluster systems can however be designed with some limitations to their implementation. For this paper, most of the tools included will be Open Source-based tools. It was a goal of the research this paper is based upon that all software licenses allow redistribution and release the source code openly. OpenPBS [6] will be included, as it will be released under a BSD-style license in 2002. (It is currently under a commercial distribution restricted open-source license.) Condor will be discussed as an example of what can be accomplished and hopefully what will become more available for Open Source in the future. Condor provides a complete HTC cluster solution. This paper will cover alternative Open Source solutions, but they may lack the maturity of the Condor equivalent. Please consult these software's licenses and documentation for further information [5,6].

A. Processor and Memory Resource Sharing

The main requirement of any cluster technology is some sort of resource-sharing scheme. For parallel processing, this refers to memory and processor sharing. In most cases, files must also be shared. There are several approaches to processor and memory sharing in a cluster. Many are very similar to one another, and may simply consist of an application programming interface (API) sitting on top of another sharing system that does the actual work. Most of the differences are in how the cluster is presented to the user application. The first technology that will be discussed involves sharing resources by simple remote execution. One can schedule or run any type of process in a batch across a cluster using remote execution. The second, process load-balancing, is where individual processes are migrated dynamically in the cluster, so the application and user do not necessarily need to be aware of remote execution. The third technology discussed is message passing--an interface to open data communications between nodes that are executing multiple processes of an application. These technologies are all unique in their complexity and in what problems they are intended to solve.

Remote execution can consists of a simple remote login shell script. It will login and run the same application with either the same data or different data for each node. Schemes can be implemented on any stock networked computer to accomplish this task. However, applications can also be run this way on HTC using queues and scheduling processes. With these tools it is possible to generate large execution schemes and what is referred to as parametric execution. Parametric execution is the running of an application multiple

times, varying the input (or parameters) of the single application each time. Execution scripts may be developed by hand or by using specialized software interfaces. Using this scheme, for example, one could easily launch a visual rendering application across the cluster and have each node processing a different frame. One could run a computation application with various function inputs across the cluster. With this type of execution, it is necessary to manually collate the resultant data (if any) at the end of execution.

Process load-balancing is a scheme for homogenous (same computer platform) clusters to pass processes. It will dynamically migrate processes off high-load systems on to less utilized systems. It has been designed for workstations to pool resources for any processes, not just special parallel or scheduled programs. There are limits to what can be done in load balancing--any I/O access outside memory such as network, user input, or storage access may require too much network traffic if a application is moved off its original node. File access without a special file system for process migration will require excess network traffic with the original node. Also, the processes cannot use more than a single node's available memory, so large problems may still need a solution like message passing on top of load-balancing.

Message passing in its simplest incarnation is a remote execution of an application across the cluster using the message-passing interface to communicate data between processes running on remote nodes. The reasons why this is better than using a standard remote procedure call or doing independent network connections are portable and stability. Safe and reliable procedures for communicating have been designed into the interface. The interface is also consistent across all the platforms it supports and generates compatible binary messages. Message passing allows for parallel applications to be released that will work on a variety of platforms. It also allows a cluster to be heterogeneous (made out of multiple platforms types).

B. Batch System

The next important tool for a fully usable HTC cluster is some sort of queuing or batch system. Most resource-sharing systems come with their own execution systems that may run immediately on the cluster if used. To implement fairness to cluster users and the interactive users it may quickly become apparent that some sort of job batching is necessary. No matter what sharing system is utilized, if there are many users wanting to use the cluster, a batching or queuing system is essential. A batch system usually consists of a queue that holds submitted jobs, and some sort of scheduling mechanism. Scheduling policies can include anything from specific time allotments, allocating the next available resource, or some resource balancing mechanism. This is entirely dependent on the batch system's implementation.

C. Monitoring

Another important part of an HTC for most users are the monitoring tools. Most batch systems and resource-sharing schemes have their own tools for monitoring. Some are very complex and may help profile the actual performance of the application on the cluster, while others just acknowledge graphically that the job is still running. With most monitors, the user should be able to get a reliable report of how much memory and processor utilization a job or jobs have.

D. Security

On a HTC environment, even more than for a HPC, security in network transactions may be necessary. Even if the environment is exposed only on an intranet, there could be malicious use to allow improper access to data or modify cluster scheduling. Some sharing systems and schedulers do provide data and login security. In the case of message passing, development has only recently started in securing the actual messages. Clusters may span several sites across the Internet or a WAN by using encrypted IP tunneling. One may also wish to provide internal cluster resources to outside groups or users. If that is the scheme used, user encrypted credentials as well as secure data encryption will be necessary. Currently, Globus [7] software provides secure credential mechanisms for certifying authorized outside users of the cluster. Secure shell (*ssh*) or secure http (*https*) may be used to provide secure remote access and data transfer.

E. Fault Tolerance

One important feature that is lacking in current Open Source solutions is fault tolerance. This is not the same kind of fault-tolerance as provided by a HA cluster. Here the term refers to the guarantee of a scheduled job completing regardless of outside interference. The software Gnu Queue does provide for some limited functionality with special versions of older Linux kernels. The non-open source Condor however, provides a complete solution for most remote execution and PVM (a type of message passing discussed later) programs. Fault tolerance is important for long-running applications on a volatile network such as an HTC. With fault tolerance, if a node goes down, the jobs running on the node can be restarted on another node. If the job's queue time is up, but the execution is not complete, Condor can also *checkpoint* or save the current state and run the application from that point on the next execution. Condor also uses this procedure to limit the use of workstations by running only when the user is inactive; it migrates the saved state to another inactive node when the user returns. Some queuing systems can provide limited fault tolerance by running multiple executions of the same application and returning data from the first fully completed run. The only recommended Open Source solution for fault tolerance is to not have a fault. Many parallel applications cannot be made fault-tolerant independently without significant code modification. Of course, if applications run on

the cluster are single-process execution, then fault tolerance is of little concern and the application is just re-executed.

F. Network File System

HTC systems can optionally use distributed or network file system technologies. There are several available, some specific to the type of memory and processor resource-sharing tools, others generic to all computer systems. It is important to look into compatibility, stability, and scalability for any proposed distributed file system for each cluster environment. Many may not work with certain types of network topologies or require extra hardware.

There also other tools available that may provide different interfaces or additional useful capabilities. Tools such as High Performance Fortran as a parallel language extension to Fortran and the data visualizer OpenDX are can be used.

A large percentage of the required mechanisms are modular in nature. In fact, one can mix and match one's preferred software or run competing configurations (within reason). Almost every conceivable configuration (using both commercial and non-commercial tools) has been done before, and there is information available on the Internet to help set them up. For the purposes of this paper, an example set of mechanisms and tools will be discussed later. The software provided in the example may be reconfigured to suit a business' needs.

IV. HTC Implementation In An Enterprises

A HTC system inside an enterprise has some very critical technical and social aspects that must be considered. For a HTC system to be used and favored by the business, it must satisfy the needs of the owners of the nodes [3]. The owners in this case are the interactive users who have been allocated the actual computer by the business. They have specific tasks to perform for the company. If they feel that they are losing productivity or are in any way uncomfortable with the HTC cluster use of the computer, it could mean more than the HTC project being terminated. It is important to find willing owners, technical staff or other employees that will find the system useful as cluster users themselves. It will take time to tune the HTC to a business' needs and therefore should be deployed initially on a small test-bed anyway. Once a solution has been found that minimizes the interactive user interference and there is demonstrable usefulness of the HTC, a business-wide rollout can begin.

Any size rollout should also deal with the sensitivity of the data that is used on the cluster and on the individual nodes. Most tools do provide adequate user-level security. However, when adding features such as distributed file systems and remote login there are increased variables that the administrators must master. Depending on how the applications are run on a node, it might be a security hazard to let user-owned processes execute, for example. Some computers may not make sense as a node when security issues are fully taken into account. Network security is also an important issue. As mentioned earlier, malicious users might attempt to compromise these new tools both while running on nodes and by "sniffing" the network. These security problems must be clearly dealt with by the cluster implementation. It is important to have trust worthy owners because they can wreak havoc on, at the very least, processes that execute on their system. A cluster must therefore have both trusted users and trusted owners.

Other technical issues need to be addressed as well. In most organizations there may be a mix of platforms running. There could be Macs, PCs, and Unix workstations all running various operating systems and using various processors. Not all of these platforms may be supported by the resource-sharing tools. Process load-balancing by its very nature must be run on only one operating system and one processor type. A cluster may have several separated process load-balancing groups--one for each platform if they are available. Batch systems and other tools can have these limitations as well, so it is important to research a business' needs with regard to what is available. Message-passing applications will generally need to be pre-compiled for each platform that they will run on. The best solution may be to intermix what tools are available and test until a workable solution is found. That is why Open Source tools are important, they make porting to different platforms possible, and limit the upfront cost of trying new tools. Some commercial solutions provide trial software as well.

It is important to keep cluster users informed of what tools are available. Once a few of the users or administrators have had success with the cluster, they should demonstrate its uses. Tools come with examples and tutorials that provide most of the required knowledge to program with the tool. It should not be assumed that users of any type have had parallel computing experience. Once users have a grasp on the new tools, it is important to try to integrate the tools they previously used with the cluster. It should be possible to adapt any computational task, at a minimum running it on less-used computers.

Development for specific tools comes with a need to understand their specific capabilities. Message passing can do its best work on large data sets that do not fit in a single node's memory or that may be self-modifying. Parametric execution, the parallel extension of remote execution, is good for running small, computation-intensive applications with multiple data sets. The process of developing for message passing and parametric execution systems can be simplified as below:

Message Passing Application Development Process

1. Identify serial code that may become parallel.
    a. Multiple execution on pre-partitioned static data (collating result by using message

passing gathering when finished).
   b. Large data set that may be partitioned.
   c. Self-modifying large data sets.
2. Search for similar parallel routines on Internet, etc..
3. Learn the message-passing application programming interface (API) in preferred language.
4. Add message-passing commands into code and rewrite parallel portions. Compile and link to the message-passing library.
5. Learn queue/batch system interface (simple).
6. Write a batch scheduling routine for job.
7. Run job.
8. View results.

Parametric Execution Application Development Process

1. Locate serial binaries or write serial code that can vary data across interested values for a specific problem.
   a. Multiple execution on pre-partitioned static data.
   b. Multiple execution on static data sets using arrays or ranges of specific parameters.
2. Learn queue/batch system interface.
3. Write a batch scheduling routine for job including parameter varying code.
4. Run job.
5. Collate results.
6. View results.

These are just examples of what will be required to use the new tools the cluster will bring. It is important to monitor progress on learning and using the cluster. Cluster usage can be monitored by the cluster's own monitoring software. A cluster can be very helpful to a business or it can end up being completely ignored. It is important to encourage usage as it will save time and result in more competent users.

If all else fails, an otherwise successfully set up cluster may be made available to other groups of users. Globus provides a way to give the cluster time to other users by a process called resource brokering.

If, however, the HTC does turn out to be a boon to the business then its continued existence is assured. HTC clusters scale very easily with additional computers or computer upgrades. It may become necessary to partition resources due to popular demand. This may also be recommended depending on available platforms, tools, and geography.

The previous remark highlights the only true limitations of an HTC system. Cluster size and cluster network latency are critical to what can be run on the cluster. It may be possible to partition the system based on this and provide multiple solutions across the business that do provide decent performance. If all else fails, HPC-like clusters can be added to the HTC environment by making a separate partition for them and running them on their own network.

Future tools and parallel mechanisms may be rolled out on top of current implementations as well. Parallel evolution is for from over. In fact there are several new solutions on the horizon. Globus is continuing work on their grid technology that provides all other computing resources as well as clusters inside a cohesive user interface [8]. The group that currently maintains PVM [9] is also working on an Open Source PVM/MPI fault-tolerant message-passing system called Harness [10].

V. The Rock Linux Example

Over the past year, Rock Linux has been developing a mix of parallel tools to develop a cross-platform Open Source parallel operating system under the Massively Parallel Processing Project. This system includes a broad array of tools and examples to get a cluster up and demonstrate its performance. It is currently only available for the Intel x86 compatible platform, but does include some significant features for the particular platform. It is sad however, that Rock Linux cannot incorporate such a "perfect" solution as Condor due to license restrictions. Condor, however, can be added to Rock Linux, which makes an excellent combination of all available software. What will be discussed here is a sample mix of available tools that meet an HTC's requirements. Comparisons of Condor to similar solutions as well as suggestions of workarounds will be made.

A. Processor and Memory Resource Sharing

Remote execution is facilitated by a variety of tools in Rock Linux. There are several batch systems available to easily disperse processes across the cluster. There are also tools that allow for a scripted application to be ran using *ssh* across specified nodes.

Rock Linux's resource-sharing systems include message passing, process load-balancing, and remote execution. Message passing is provided by the two main standard systems MPI [11] and PVM. MPI is a standard created after PVM, but is much more open to implementation variation since PVM is released by only one group. The MPI available comes from either MPICH [12] or LAM [13]. Either system is available for use, though MPICH is much more integrated with the other available packages. Both MPICH and LAM are developed as Open Source, and have a large amount of complementary tools. All three tools provide heterogeneous message passing between processes.

Mosix [14] is also included in Rock Linux. Mosix is a process load-balancing system currently under development. It provides a clean and safe way to dynamically transfer process between nodes. Though it may not scale very well, the benefit of process migration has important implications for quality of service. The biggest social issue with HTC clusters

is that the interactive user (owner) does not appreciate sharing his/her computer with a cluster application. With Mosix, processes can be forced to migrate--even interactive user processes can be balanced. Using Mosix migration, it may be possible to get even better quality of service than under Condor's checkpointing for the cluster processes and the interactive user processes.

B. Batch System

The queuing system that Rock Linux recommends is the OpenPBS [6] system. Though Generic NQS [15] and Gnu Queue [16] are provided as truly Open Source solutions. OpenPBS has a much wider range of features and is in current development. Generic NQS is a simple job queue utility that is missing most features of a full batch system. Gnu Queue is more than just a queue, but also a load balancing system. It provides preemptive remote execution on a cluster using its own shell system (non-migrating). The shell provides checkpointing through Linux kernel patches. The patches however, are not current with Linux development so they have limited use. OpenPBS only provides limited fault tolerance by allowing applications to execute multiple times simultaneously. This gives a higher probability of completion and works for a larger set of problems than remote execution and PVM-specific message passing (Condor).

The OpenPBS batching system contains four important components: the submitter, server, scheduler, and executer. What these components provide, together with OpenPBS's portability, are the reasons why it is the recommended Rock Linux solution.

The submitter can perform several tasks such as generating a batch job script for the user or reporting where in the queue a job is. However, the most important task is to notify the server so that job can be put in the queue.

The server contains all current waiting jobs in a queue for the batch system. It communicates with all other components and orchestrates the batch system. The server protects the current jobs and notifies the scheduler when resources are available.

The scheduler is a simple FIFO (first in first out) but it can be replaced by Maui [17] to provide a variety of ways to determine how to run jobs. The scheduler runs jobs according to site policy that is defined by the administrator. This includes user group priorities, time limits, node limits, and others. The Maui schedulers allow for execution using advanced resource algorithms to determine "best fit" execution.

The executer is usually a proxy device that runs the job's process on each node. It is responsible for monitoring its execution and reporting its completion or forcing its termination. It will communicate progress to the server so that other jobs may use its resources when the current job is finished.

C. Monitoring

Rock Linux utilizes several monitoring tools. OpenPBS integrates *xpbs*. *xpbs* allows users to submit and modify jobs as well as monitor and view results. Mosix has contributed applications that work much like a Unix or Linux *ps* or *top*. There are also simple tools for monitoring the message-passing processes across the cluster.

D. Security

MPICH is configured to login securely using *ssh*. OpenPBS uses the secure copy command to transfer job data. On top of all of this, Rock Linux as uses Globus [7]. Globus is linked to use OpenPBS from installation. To get a site or users certified one should contact the Globus Team and apply for credentials.

E. Fault Tolerance

As stated before, Rock Linux can only provided limited workarounds for fault tolerance. If fault tolerance is critical or if the job may need to migrate often, the addition of Condor to the Rock Linux installation will probably be the best solution.

F. Network File System

Rock Linux has several files systems to choose from. Each one will most likely require different directory topologies. Common NFS and Coda are included with stock Rock Linux. Included with the parallel packages are three Open Source file systems--Parallel Virtual File System (PVFS) [18], Global File System (GFS) [19], and the Mosix File System (MFS). PVFS was designed for HPC systems as a way for diskless nodes to efficiently communicate with a file server. GFS was designed to implement a network encompassing a fault-tolerant filing system. MFS was designed to provide Mosix systems with a unified file system. Both GFS and MFS can support Mosix's Direct File–System Access (DFSA) that provides Mosix processes with direct file access while on remote nodes. This provides a simpler mechanism than Condor remote procedure call-based file access.

There are too many other parallel packages to describe here--in fact there are over fifty available. In addition to special parallel packages, Rock Linux has many features that are complementary to clusters and general system administration. Using Rock Linux's build system, for instance, it can be very easy to rebuild the entire set of included parallel tools using a higher performance compiler than *gcc* and with more optimization. Please refer to the Rock Linux website [20] for more information.

VI. Conclusion

HTC clusters can be an important addition to any enterprise. They are affordable and provide a large array of solutions. Current Open Source software provides a very low cost of entry while providing a good selection of tools. The addition of Condor or future resource-sharing mechanisms can provide fault-tolerant parallel processing as well. Rock Linux provides a set of integrated tools that provide an easy to rollout HTC cluster.

Acknowledgment

The author would like to thank the Condor Team and all the Open Source groups from which the Rock Linux Massively Parallel Processing Project have been drawn.

References

[1] Condor Team, "Idle Machine Statistics," http://www.cs.wisc.edu/condor/goodput/idle.html .
[2] M. Livny, J. Basney, R. Rama, and T. Tannenbaum, "Mechanisms for High Throughput Computing," http://www.cs.wisc.edu/condor/doc/htc_mech.ps .
[3] SGI, "Linux Failsafe Project" http://oss.sgi.com/projects/failsafe/index.html .
[4] T. Sterling, D. Becker, M. warren, T. Cwik, J. Salmon, Bill Nitznerg, "An Assessment of Beowulf-class Computing for NASA Requirements: Initial Findings from the First NASA Workshop on Beowulf-class Clustered Computing," http://www-hpc.jpl.nasa.gov/PUBS/BEOWULF/report.pdf .
[5] Condor Team, "Condor Project," http://www.cs.wisc.edu/condor/ .
[6] Veridian Systems, "OpenPBS," http://www.openpbs.org .
[7] Globus Team "Globus," http://www.globus.org .
[8] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," to be published in Intl. J. Supercomputer Applications, 2001.
[9] PVM Team, "Parallel Virtual Machine," http://www.epm.ornl.gov/pvm/pvm_home.html .
[10] Harness Team, "Harness," http://www.csm.ornl.gov/harness/ .
[11] MPI Forum, "MPI," http://www.mpi-forum.org/ .
[12] MPICH Team, "MPICH," http://www-unix.mcs.anl.gov/mpi/mpich/ .
[13] LAM Team, "LAM," http://www.mpi.nd.edu/lam/
[14] Mosix Project, "Mosix," http://www.mosix.cs.huji.ac.il/ .
[15] Stuart Herbert, "Generic NQS," http://www.gnqs.org .
[16] W. G. Krebs, "Gnu Queue," http://bioinfo.mbb.yale.edu/~wkrebs/queue-development.html .
[17] Maui Project "Maui," http://www.supercluster.org .
[18] PVFS Project "The Parallel Virtual File System," http://parlweb.parl.clemson.edu/pvfs/ .
[19] Sistina Software, "Global File System," http://www.sistina.com/gfs/ .
[20] Rock Linux Team "Rock Linux," http://www.rocklinux.org .