# A Brief Introduction To Commodity Clustering

Ryan Kaulakis
05/05/04
Pennsylania State University, McKeesport Campus
A Research Study for the Dept. of IST

## Abstract

This paper explains the basics of commodity clustering, and gives examples of its application in both business and academia. Four different varieties of clustering technologies are addressed, their merits discussed, and alternatives presented.

## I. Introduction to Commodity Clustering

Well, to begin to understand the concept of commodity clustering, it is useful to look at the words that make it up. A “commodity” in the sense that it is used in “commodity clustering” is used in the economist’s usage meaning “mass-produced or unspecialized” (m-w.com). Examples of commodity goods include grain, steel, and, of particular interest to this paper, personal computers. Non-commodity goods include very specialized items, such as custom car-parts, industrial plants, and Cray super-computers. Now, “cluster” is a general term for an aggregate or group. That said, a “commodity cluster,” as used in this paper refers to a group of commodity grade PCs which have somehow been made to work together. Instances of which include Beowulf clusters, such as those used by NASA, and computing grids such as used by the SETI Project. Both groups use their clusters to perform the work of supercomputers; as a matter of fact, as of the writing of this paper, the world’s fastest supercomputer (according to http://www.top500.org/list/2003/11/) is Japan’s Earth Simulator, which has been clocked at 4.1 TFLOPS—one trillion floating point operations per second.

Now, why would anyone want to use a cluster? Well, aside from the fact that they can be made to operate faster than many traditional supercomputers, they also have another, considerably more tangible benefit: they are cheap. Due to the fact that clusters are made from commonly available parts, they can be constructed as frugally as desired. One can go so far as to invest in multiple racks of high-quality, top of the line equipment, and hook it all together with dedicated high-speed network connections, or you could take so computers out of the trash, and stick them together with some old phone cable. By their very design, commodity clusters are very adaptable, and scale well to nearly any environment. There are additional benefits, as well; since clusters are made of many independently running machines, the failure of individual machines is not problematic. The upshot of which is that, you don’t need to worry overly much about expensive backup and failsafe mechanisms to maintain continuous uptime; if there is a problem, you can just yank out the device in question, and fix it, without interrupting the overall system operation. Given all of these qualities of commodity clusters, it’s easy to see why they could be considered very desirable.

“But how do they work?” you might be wondering. Well, that is a very simple question with a very hard answer. Clusters can be constructed using any number of technologies, and can provide any number of different services, so there is no general answer to “how do they work?” There are, however, several categories of cluster out there, with purposes ranging from distributed computing, to sharing memory, and even to sharing services, with each having its own set of associated problem domains and technologies.

In general, a single individual machine in a commodity cluster is not as fast as a single special-purpose supercomputer, and therefore will probably not be able to crunch numbers as fast. Yet, somehow commodity clusters can out-perform many of the fastest supercomputers. It’s no secret how conventional supercomputers achieve their speed; they have extremely optimized hardware that operates at very high speed. So, how do the clusters out-perform them? Simple, they cheat. While the supercomputer is optimized in hardware, the cluster is so in software. The cluster breaks do a problem in a special way so that it can distribute all the little pieces to its constituents. That way the overall problem gets solved very efficiently. Examples of distributed computing clusters are discussed in detail in sections II through IV.

Shared memory systems, on the other hand, do not have much in the way of non-cluster analogs. Though there are some special purpose storage systems, all of those of any significant size incorporate clustering technology at some level. The idea behind a shared memory cluster is simple enough: you want to be able to reliably store as much data as possible, as efficiently as possible. Many special purpose solutions exist for how to achieve this, but this paper addresses some general approaches in sections II and V.

Finally, service cluster may be the most vaguely defined of all of the cluster categories that I've discussed thus far. They all revolve around the center theme of reliably providing a service, such as a website or a database. There are as many different approaches to doing this as there are services, some are specialized distributed database solutions, while others are more general, and focus on being able to simply provide a very reliable service, regardless of what the actual service is. This paper addresses the second, since they are more widely useful, and more detail may be found in sections II and VI of this paper.

In section VII of this paper, I will show how clusters can be applied in a variety of settings, incorporating the specific technologies noted in sections III through VII.

## II. Summary of Research

To begin with, when I started on this project four months ago the purpose of my research was to construct a working cluster. However, after my colleagues and I had accomplished that—within a month of starting out—I have focused more on adding in additional capabilities to our system. As of the writing of this paper, not all systems are complete. This section details efforts of this author and his colleagues, and may safely be skipped over without any loss to the reader. It is also of tantamount importance that the reader understands that, though the cluster discussed in this section is still very much a work in progress, future developments will be reported as they arise, and that this is neither definitive nor authoritative source on any issue discussed herein.

In section I of this paper, it is claimed that clusters can be much cheaper than their conventional non-commodity counterparts. Though there is evidence to support this claim widely available (for instance **[1][2][3]**) I would like to take the opportunity to both share with you our hardware setup for this experiment, as well as the cost for said equipment.

| Quantity | Hardware | Cost |
|---|---|---|
| 1 | Compaq Server-grade PC with 2 Pentium II CPUs | $0.00 |
| 8 | Dell Optiplex GXa PC with Pentium II CPU | $0.00 |
| 4 | IBM PC with Pentium II CPU | $0.00 |
| 1 | 10/100 MB Ethernet Switch | $0.00 |
| 1 | 10/100 MB Ethernet Hub | $0.00 |
| 25 | HP 50/75/100 Pentium I PC | $0.00 |
| | Total Cost | $0.00 |

From this table it should be obvious that the costs for hardware can be kept to a minimum. For clarity, all equipment listed in this table was donated over the last few months for use in the cluster. In particular, the Dells, HPs, and IBM PCs were all reclaimed from the trash, while the Compaq and the Ethernet equipment are on loan from the campus' Engineering and IST departments. In short, this cluster—which will be referred to by its project name (CID) from here on out—was constructed out of stuff that was just lying around, unused. Further, since Linux was used as the underlying operating system, the software costs for CID were also nothing, as only free and open-source software was used in its making.

At this point, it is helpful to know a little bit more about the workings of the project that I referred to above.

To begin with the name that was decided upon for our cluster was CID—which stands for Cluster In Development, a title which seemed quite appropriate for our system, given the near constant alteration and expansion that was performed upon it. Unlike many clusters, CID is designed to be completely dynamic in its configuration, and from the very beginning it has been constructed in such a way as to allow the dynamic addition and removal of individual machines—with one important exception: due to hardware limitations, only one computer was used as a master for the cluster, as well as a server. The system's setup if explained in greater depth below.

Here is how CID works (in the general sense):
There is a central server which acts as a "head" for the cluster—allowing users a way to interface with the cluster. There are also additional computers called "nodes" which have no physical interfaces (e.g. no keyboard, mouse, or IDE support). These nodes are set up in such a way as to only make use of the computer's CPU, NIC, and RAM, without leaving any lasting information on the hardware that it is running on. The nodes boot from a floppy disk, which contains scripts that retrieve a custom OS and filesystem from the head server, and all of this information is stored in RAM—taking up about 8MB of space in all. Every member of the cluster runs a program called omdiscd which allows every member of the cluster to contact every other member. Processes can be started anywhere in the cluster, and they will automatically migrate to the most efficient location for them to run. This holds true both for the head, as well as for all nodes, though the head is the only computer in the cluster that would have any processes that are CPU intensive enough such that migration would really help. Thus, if you had a job which would benefit from large scale migration, with our setup you would pop the boot disk into a couple labs worth of computers—which would add themselves into the cluster—and then you'd have a considerably larger cluster to run the job on. After the job is finished, simply rebooting the nodes (sans the boot-disk) will return them to their normal operational mode—which, in the case of my campus' labs, means that the nodes reboot into windows, and stop being nodes.

As of 5/5/2004 9:50 am CID consists of thirteen separate machines, all Pentium II or Celeron processor based, with a clock-speed of 333MHz for each CPU. Additionally, each machine has at least one 100BaseT Ethernet NIC and a 3.5-inch floppy disk-drive. There is one computer which has two CPUs, three 15GB IDE hard-disks, four VGA video cards, three 10/100 Ethernet NICs, an IDE CD-ROM drive, and a 3.5-inch floppy drive. This machine was constructed out of spare parts, and is used a "head" for the cluster, as it acts serves as a multi-purpose server for the rest of the cluster. This computer, given the hostname Neo, bridges three networks, one of which is an internal network which connects all of the "persistent nodes" (those nodes which are always a part of the CID) in the cluster, while the other two connect to public IP addresses, allowing authorized individuals to access CID from around the globe. Apropos, on each external network that CID is connected to it gets its address via DHCP, which would normally complicate remote access, if it weren't for a dynamic DNS address which resolves the name "cheshire.ath.cx" to CID's current public interface. Neo runs a number of services, including a TFTP service which is used to distribute the nodes' OS and file-systems. In all, when not handing out system images, the average network load of CID is around 25kbps, which is negligible on modern Ethernet networks.

The actual steps involved in setting up CID were fairly simple:
1) We installed Gentoo Linux on the Compaq server, and set it up as a stable Linux system that used 2.24-openmosix kernel. The steps for doing this can be found on the Gentoo website **[4][5]**. This machine will be referred to as Neo from now on.
2) A TFTP service was installed on Neo.
3) Using the ramdisk howto **[6]** we constructed a complete open-mosix based linux system that was 8MB in size.
4) Using GRUB **[7]** we constructed boot disks for the diskless computers.
5) After wiring all of computers into a bus topology, we installed DHCP on Neo **[8][9]** and inserted the boot-disks into each node.
6) After insuring that all required services were running on Neo, we booted up all of the nodes, and after a few minutes, we had a fully functioning OpenMosix Cluster.

In all, the process took my colleagues and me the better part of a month, though, considering that we only worked on CID during our free time, and that we were learning all of this as we went along, it should be possible to get a cluster, such as CID operational in a matter of days. Certainly, simpler clusters such as will be described in sections III through VI take much less time to setup.

From the building of CID, my colleagues and I learned a great deal about the inner workings of Linux, and I would recommend projects like CID to anyone who would be interested in trying their hand at a practical application of Linux. This should be particularly interesting to schools and other groups, who would value such a low-cost training aid. Professional administrators, on the other hand, would most likely see something such as CID as a cheap and powerful way to handle a heavy user base, such as a number of Matlab users.

In the future, significant additional capabilities will be added to CID, as it is planned to add on features like OpenAFS, to enable a sharing of hard-disks across the cluster, and Samba to enable Windows users to store files on the shared disks. Additionally, to add to CID's growing number of practical applications, we plan on adding many components of OSCAR, to allow for software-level clustering on top of our OpenMosix based kernel-level clustering; this will enable us to run many special-purpose scientific applications, as the need arises. Further, we are working on integrating the individual nodes more closely into the control of the cluster, as well as allowing for increased security and expansion across multiple LANs. All of this working toward the ultimate goal of creating a cluster that can run across multiple campuses, creating an extremely large-scale super-computer.

That's a long way off, however, though again I encourage anyone who might be interested, to try CID out.

## III. The Basics of OpenMosix

There are a number of different clustering techniques out there, and OpenMosix happens to be fine example of kernel-level clustering, meaning that OpenMosix works at the Operating System level.

That said, OpenMosix is a kernel extension to the standard 2.4 Linux kernel. Which means that one can make an OpenMosix-ready kernel out of a vanilla stock kernel by applying the appropriate patch, and then going through the normal kernel compile cycle (I personally used this: **[5]**). In addition to a kernel patch, one needs to install a few programs, as well as configure a couple files, and that's pretty much it. It's all explained well in the documentation.

As I said before, OpenMosix is a kernel extension, which means that it is the lowest level of clustering technology you can get in an OS. It has a number of advantages over user-space clustering method, such as speed and transparency, but there are also a couple caveats to take into account when using it as well.

OpenMosix is fast; it acts as a general load-balancer within the kernel, and uses criteria that only the kernel has access to (such as the IO usage statistics for a particular process) and uses that information intelligently to decide what to migrate and what to keep. Now, when it migrates a process, that means that it sends the process to a separate computer—one which must also be running the same version of OpenMosix—which will execute the process. Additionally, each member of an OpenMosix cluster has the ability to migrate processes amongst the cluster, so if any given node becomes burdened too heavily, it can send off processes to other members to lighten its own load. The cluster members are in constant communication to maintain all relevant connections, so that processes do not just get bounced from one over-burdened node to another. Additionally, processes can be expelled from a node, so that process data is not lost when you shut-down the computer or the like. In general, OpenMosix does things the Right Way to maintain the integrity of data as well as the overall efficiency of the cluster itself, and there is extensive documentation on the particulars of process migration here: **[10]**

OpenMosix also has the added advantage of transparency, which means that a user (and hence any program run by a user) should see no difference between a process that has been migrated and one that has not. Since OpenMosix is kernel-based, it can migrate any process on the system that would benefit from migration. Even processes that require access to the hard-disks of its home node can access directly whatever files it needs, using OpenMosix's MFS filesystem. The MFS filesystem allows every node in the cluster to have full access (respecting permissions) to the filesystems of every other node, which solves many problems that can be encountered with simple process IO.

Some warnings and considerations:

- OpenMosix is quite stable, and normally works well with nearly anything that is thrown at it, however there are some conditions (documented here: **[11]**) that can cause untoward bugs to arise. However, OpenMosix has a very active developer community, and problems, even rather minor ones, are usually solved in a timely manner.
- OpenMosix has one important requirement to migrate a given process: it must be able to be migrated in its entirety. This means that running Mozilla on one node will not magically split the load of Mozilla amongst all the members of the cluster; since Mozilla is a single process, it cannot be broken up. Similarly, OpenMosix can migrate forked processes, but may opt not to, if it would lower system efficiency. However, threads cannot be migrated separately, since they are not normally independent processes, which means that the entire threaded application would need to be migrated as a whole.
- While OpenMosix's MFS can handle simple, intermittent filesystem access, it becomes unwieldy with processes which are very heavy on IO. There is an optimized version of MFS called DFSA (that stands for Direct File-System Access) that can considerably speed up IO across the cluster. However, certain processes are simply so heavy on IO that the migration algorithm will simply not migrate them, to prevent slowdown.
- OpenMosix is not designed to normally scale across multiple LANs, and as such some steps may need to be taken to enable this to occur (see here for details: **[13]**)

All that said, OpenMosix is extremely flexible, and is fact linearly scalable, enabling large numbers of nodes to coexist without undue overhead. It is not uncommon to "roll-out" OpenMosix over a given facilities' unused computers to make a temporary super-computer for overnight use. Further, since no program needs to be written specifically to work with OpenMosix, that means that any process may be migrated—thus if a given cluster has only one busy user, his processes can be picked up by other nodes, and he will experience a performance increase. OpenMosix can be made to fit transparently into nearly any installation, and given the performance benefits of its process migration, there is great incentive to do so.

There are very few real alternative to OpenMosix, as it fills its niche quite well, but if one was interested there are at least two alternatives to consider:

- Mosix: The original project which OpenMosix forked from some years ago. It is not covered under the GNU General Public License, since it has its own unique licensing terms to regulate its usage. The project seems to have stopped active development, and is considered to have been quite thoroughly eclipsed by OpenMosix. Many originally Mosix-based installations have switched to OpenMosix, and very few seem keen on adopting it in the future.
- Condor: A user-space "high-throughput computing" management system. It appears to run on both UNIX and some flavors of Windows. Though I have heard comparatively little about this project, if the claims on its website (http://www.cs.wisc.edu/condor/) are to be believed, then it is a staunch competitor for the transparent distributed computing niche. It appears to have many features in common with OpenMosix, including the ability to migrate unmodified jobs intelligently. However, Condor is covered under a unique liberal license, called the Condor Public License, which is incompatible with the GNU General Public License.

- Additionally there may be some proprietary clustering technologies which could deliver performance comparable to an OpenMosix cluster, such as Sun's Grid Engine, or IBM's plethora of clustering methods.

## IV. Introduction to OSCAR

OSCAR stands for Open Source Cluster Applications Resources, and is made available by the Open Cluster Group **[12]**. It is not a single application, but rather a bundle of interrelated special purpose programs, each of which is the current best practice representative of its type. Just as OpenMosix functions in kernel-space, OSCAR works in user-space, which means that OSCAR—and therefore all of the component programs that it contains—does not require any kernel operations to run, and should work fine on both the 2.4 and 2.6 kernel series. To discuss OSCAR, one must look at its individual components:

- C3: A condensed acronym for Cluster Command Controls, C3 is a set of tools that enable a cluster administrator to execute system wide commands, such as cluster-wide shutdowns or reboots, and system-wide file operations. Further it enables the administrator to have an arbitrary command executed system-wide (for example have each node execute a script).
- HDF5: The Hierarchical Data Format package consists of a specification and a supporting library, and is a commonly used format for scientific data. This package is not one that the average user would have considerable cause to see, but it is vital for the use of the format.
- LAM/MPI: This package, the Local Area Multi-computer/Message Passing Interface, is a set of cluster friendly libraries which are often used when writing parallel-run scientific software. This is also a package primarily of interest to programmers.
- OPIUM: This package, the OSCAR Password Installed and User Manager, is a handy little program which facilitates the management of user accounts across the cluster. It handles SSH keys for users, allowing them to traverse the cluster without being bothered to repeatedly enter passwords.
- PBS: The Portable Batch Scheduler is a set of programs that enable the running of batch jobs across a cluster. This is mainly handy for system administrators.
- PVM: The Parallel Virtual Machine package is a set of libraries and associated programs that allow distributed computing, and also work on both UNIX and Windows systems. Again, this is mainly used by programmers.
- SIS: The System Installation Suite, is a package that enables administrator to install cluster images across the network, allowing for the remote installation and/or updating of cluster nodes' OS.
- Switcher: This package contains a utility that helps end users to manage their environmental configuration files, without getting their hands dirty.

Obviously, OSCAR is not something that the end-users of a cluster would normally see or use directly, as it is more back-end than anything else. However, to scientific programmers, and to cluster administrators OSCAR can be quite a boon, making a number of otherwise tedious or hairy tasks convenient and automated. However, it should be noted that OCSAR is just a set of tools, and requires significant configuration to form a cluster. Additionally, OSCAR does not include any general purpose process migration, so, in order to actually have a cluster, you need to use applications which were written specifically to take advantage of the libraries and other capabilities offered by OSCAR. There are some open-source programs that will use OSCAR natively, however, if you really have need of those sorts of programs—which are typically highly specialized and scientific in nature—then you already probably know how to write the code yourself.

OSCAR can be used in conjunction with many other systems, since it lives entirely in user-space, and many of its components can be used on a number of platforms, allowing clusters to include heterogeneous groups of operating systems.

The Open Cluster Group's OSCAR package uses many industry standard packages, and this author would be hard-pressed to find many viable alternatives to use in place of OSCAR's packages for their purposes.

## V. Introduction to OpenAFS

If OpenMosix and OSCAR are ways to share CPU power between computers, then OpenAFS is a way to share storage space. Derived from the Andrew Filesystem, OpenAFS was developed originally at Carnegie Mellon University, and was later extended by IBM, and it is touted by its advocates as a "stable, secure, enterprise-quality product," as it began its life as a closed-source project, and was later opened. In truth, OpenAFS is a convenient way for an administrator to allot storage space for user accounts in UNIX and Windows systems.

How is it convenient? To begin with, OpenAFS runs entirely in user-space, and requires no kernel patching to use. This makes it ideal for integration with systems like OpenMosix, above, which have stringent requirements for their kernels, and may occasionally reject arbitrary patching. Plus, existing entirely in user-space makes OpenAFS as easy to install as an other given program (see http://www.openafs.org/doc/index.htm for general documentation or http://www.gentoo.org/doc/en/openafs.xml for Gentoo specific instructions). Additionally, OpenAFS is considerably more convenient than the standard UNIX method of sharing files over a network, NFS. OpenAFS maintains compatibility with NFS (see http://www.openafs.org/pages/doc/UserGuide/auusg010.htm#HDRWQ80 for details), while at the same time, providing a uniform namespace in which users can store and retrieve data with relative transparency. Systems administrators should enjoy the additional control capabilities offered by OpenAFS (as compared to NFS), including the ability to limit the size of volumes allotted to each user (volume quotas), Access Control Lists—which are fine tuned file permissions that allow complete control over access to a given file or set of files, and the additional transparent migration of volumes, just to name a few.

Using OpenAFS, a regular user can have full read-write access to all his files, and is able to access them, as if they were in a local directory—in Linux, that means the user could keep his files in an OpenAFS volume called "/afs/mycell.com/usr/myname"—and thanks to an aggressive caching mechanism, file access speed is normally not noticeably slower than access to, for instance, a floppy disc or CD-ROM. Obviously though, since OpenAFS is a distributed filesystem that runs over a network, it is affected by that underlying network, and an unreasonably slow network will degrade filesystem performance. This raises some concerns about losing one's network connection, will it affect OpenAFS? Yes, a client that loses his connection to the cell's (discussed later) servers, will be affected thus: if a file is located on a single server, which is rendered unreachable, then a client will not be able to access nor alter the remote file, but if there are multiple copies (mirrors) of said file within the cell, then service will not be interrupted, as the available version is used in place of the disconnected one.

As was mentioned before, OpenAFS gives system administrators a number of handy tools; however it is easiest to explain these tools in relation to the structure of an OpenAFS system: The fundamental unit in and OpenAFS system is the "volume." This is a convenient container for related files and directories—for instance, user John Q. Public might store all his files in a single volume—that the OpenAFS administrators have given a set maximum size called a quota—similar to a regular disk quota. These volumes are stored in physical partitions on hard-disks—giving a practical reason to set limits on a given volume's size, so that it does not exceed the size of the physical storage medium. A group of administered systems running OpenAFS is called a cell, and in each cell, all volumes may be uniformly accessed regardless of their physical location. This enables an administrator to do several things: he may mirror a given volume by placing multiple copies of it throughout the cell, thereby allowing easier access to the files, and guarding against individual machine failure; or, he may move a volume to another location transparently, in case an individual machine needs to be serviced. Now, given the high visibility of volumes in a given cell, the security of said volumes becomes rather important; OpenAFS has built-in support for password protection as well as fine-grain file permissions, which go beyond the standard "Read-Write-Execute" to four directory-level permissions, "Lookup-Insert-Delete-Administer," three file-level permissions, “Read-Write-Lock,” and eight user-defined auxiliary permissions. Additionally, OpenAFS maintains support for UNIX mode-bits, and groups.

There are few arguments against using OpenAFS that are not also arguments against distributed filesystems in general, and so they will not be discussed here. In stead, alternative distributed filesystems—ones which may be better suited to individual purposes than OpenAFS—will be listed:

- MRAFS: The Multi-Resident AFS system, is a version of AFS developed by the navy, that automates many AFS tasks, such as the migration of volumes based on usage, as well as extending AFS to allows for true mass storage. It appears to be quite handy, but is only available by direct request from either the Pittsburgh Supercomputing Center, or from the Navy (whose contact site is currently unreachable), and its licensing is also unknown to this author, thanks to the scarcity of information available on MRAFS.
- CODA: The direct descendant of OpenAFS, CODA is a very reliable mass-distributed-storage systems, which requires a little bit of kernel patching, and is available under the terms of the GNU GPL. This author highly recommends the use of CODA for systems which do not mind kernel recompilation, mostly because of its superb documentation, but also because of its superb fault-tolerance.
- Intermezzo: Originally a speed-oriented rewrite of CODA, Intermezzo has extended itself into a high-availability distributed filesystem. Like CODA, Intermezzo involves recompilation of the kernel; however, it does not seem to have quite as extensive of documentation of its antecedent.
- Lustre: A filesystem developed by Cluster Filesystems Inc., it aims at being an extremely high-end distributed filesystem, capable of maintaining 10,000+ nodes concurrently, and transferring data at rates exceeding 100Mb/s. This system is beyond that normally needed for small clusters—of say one hundred nodes—but should outperform other filesystems for extremely large clusters.

## VI. Introduction to LVS

The final variety of clustering which this paper will address is what this author refers to as “Service-level” clustering. Unlike the previously discussed varieties of clustering, which use networks indirectly, Service-level clustering makes direct use of the network, exploiting its features to provide performance benefits. LVS, the Linux Virtual Server, is a fine example of Service-level clustering. It is not a single application, but rather a set of kernel modifications and utility programs that work together. What LVS does is fairly simple to describe: it makes a number of independent machines appear as though they were a single server to a client. The LVS “cluster” tricks a client into communicating with any one of a number of machines that are hidden behind a uniform access gateway.

To get a better idea of what LVS can do, it is handy to look at an example: Imagine for an instant that the ABC Corporation has a website, which they host and conduct all their business through. They obviously want their website to have 24/7 uptime and they want uniform performance out of it—that is, no slowdown during busy times. If their website is on a single machine, it is possible that their entire business might be brought down by any problems with that machine. LVS allows them to have one or more “gateway” machines that will send any requests for their website out to a pool of servers, thereby allowing fault tolerance and distributing the load.

Now, the details of how LVS works can be rather confusing for those who do not have extensive prior knowledge of computer networking (particularly the middle layers of the OSI model). However, it can be explained part-by-part without much difficulty. Following a request packet from a client machine, the first thing the packet will encounter is the gateway. The gateway is a machine that has some method of intelligently forwarding the packet to a particular machine in the server pool. The gateway sends the packet off again (with some minor alterations) into the server pool, where it is received by the particular server that the gateway addressed the packet to. At this server, the packet is read, and the appropriate action is taken by the server, based on the contents of the packet. Any kind of services that run over TCP/IP can be handled by LVS, as it is only concerned with the delivery of packets, and not with their actual contents.

Gateway machines are the key parts of an LVS cluster, as they perform the essential load-balancing functions that distinguish an LVS system from a simple “Round-Robin” setup, in which there is no intelligent method of ensuring that individual machines are not over-burdened. Gateways are distinguished from the rest of the LVS system by the fact that they, themselves, do not provide any services to external clients (though, they usually are configured with various services to monitor the cluster they serve). There are a myriad of ways to connect to a gateway from the outside, as it appears as a single machine to the outside world, but, it is the only way to connect to the inside of the cluster. Gateways can use NAT, IP Tunneling, or even Direct Routing to control the cluster, they are quite flexible. Further, there may be backup gateways setup to assume the responsibilities of the normal gateway, in case of failure in the original. Add to this the ability to masquerade a second LVS system as a single server inside the server pool of another, and it is easy to see exactly how flexible LVS systems are.

Given the great potential for fault-tolerance that LVS systems possess, it is not surprising that they are typically used in situations where high-availability is important. There are however, some caveats that must be mentioned:

- LVS is only concerned with the availability of servers, it is left up to the system administrator to ensure that the servers will behave appropriately when they receive connections. This means that, in the case of web-servers, the administrator needs to ensure that each server has identical copies of the website, and that they can share whatever common CGI and databases they may need. LVS provides some tools that may aid in this, but other technologies are required to address the problems of controlling the content of a number of servers.
- LVS does allow some fault tolerance, such as for the removal of individual servers from the pool for maintenance, without interruption of overall service. However, any clients connected to a particular server when it goes down will experience all the effects of a server outage, at least until they try to reconnect, and are redirected to another server in the pool. This means that session data is not inviolate, and unless the administrator takes steps to prevent it, can be lost between connections.
- LVS is a truly powerful networking tool, and, as such, its documentation should be read thoroughly before trying to use it in anything other than an experimental setting.

Given the general nature of LVS, it is easy to see that it can be made to work with a number of different technologies, though one should be aware that blatant alteration of normal network traffic may cause rather cryptic errors, is not used appropriately—much like any other networking tool (e.g. just think of how many errors can be generated by an inappropriate firewall). There are a number of application specific alternatives to LVS, far too many to be detailed here, though a simple search will turn up as many as one cares to read.

## VII. Example Applications

Now, it may be well and good to know the basics of the technologies discussed in this paper, but it is important to see how they can be made to work together. For that reason two example applications will be discussed, a “toy” university in which the individual technologies are simply used tighter, and a second example, this time of a business. Please remember that neither of these places exist, and that they are only examples.

**Wibni University:**

Wibni University has a broad curriculum, teaching many different courses and topics. The student body is diverse and rarely agrees about anything, but one thing is for certain though, everyone wants more out of the school’s IT systems; they want the lab computers to be faster, they want more storage, less downtime, and, most importantly, they don’t want to pay any more. Wibni-U’s IT department is filled with people who are eager to please, and so they used the technologies discussed in this article to enhance the performance of their systems. They used OpenMosix on their batch-compilation computers for their computer-science department, and they also stuck Yafray on there for the graphic-design students to play around with. Using LVS, they set up load balancing for the internal servers, so that they could take some down and perform the regular maintenance without interrupting anybody's sessions. Finally, they set up everyone's home directories so that they would store their files on an OpenAFS system, and thus be able to access them from anywhere on the network, while also allowing the IT staff to do mirroring of the important files (like the MSDN libraries) so that people would be able to access it better, without having to install it on every computer-science student's terminal.

As this is a simple "toy" example, it is likely that it suggested nothing new to the reader, however the following example may demonstrate better the capabilities of these technologies in a non-toy example.

**Xyzzy Incorporated:**

The Xyzzy corporation is in the business of map making, specifically, making special order maps of areas that are generally too small to get taken up by many larger companies. They have an ingenious approach to making maps, which avoids the troubles of getting their hands on satellites; they compose their maps using data gathered from drone aircraft. Due to the prohibitive cost of the aircraft, they hire out the drones from various contractors. One day, a call comes in for a map of an obscure island chain, which contains the current political, magnetic, and geological features of the area, and they need it ASAP. So, Xyzzy Inc. calls up their contractors, and their contractors send out drones. Each drone sends its data to an IP address, which resolves to an LVS cluster run by Xyzzy Inc. This cluster ensures continuous uptime for Xyzzy Inc.'s data-reception systems, even in the event of someone spilling coffee in the IT closets. From there the data is sent to each contractor's storage account, so that they know how much and what data comes from each when time comes for them to get their commission. The data-storage system is an OpenAFS cell that allows Xyzzy Inc. to easily backup and distribute the data without interrupting the storage processes. After all the data is stowed away, several in house programs utilizing the tools provided by OSCAR rip through all the relevant files, and process them into data suitable for the making of maps, entering coordinates and other such information into a MySQL database, which itself it set up via a combination of LVS and CODA as a fault-tolerant high-availability server. After that is finished, Yafray is used in conjunction with OpenMosix to render a full map out of the information contained in the database and images. The final map produced by this process is then made available both through the Xyzzy Inc. website—which itself is hosted on a high-availability LVS system—and in hard copy to the original party which had placed the delivery. Not bad for a week's work, and none of it required any human intervention, nor did the underlying technology cost Xyzzy Inc. a dime.

In this example it is easy to see how these seemingly disparate technologies can be made to function together in a business setting, even a turnkey one like this.

## VIII. Conclusion

It should be easy to see the promising power that commodity clustering brings with it; it's cheap, it's powerful, and—best of all—anyone can use it—provided they are willing to read the documentation. This author hopes that this paper will serve to whet the reader's appetite for clustering technologies, and that the reader will take it upon himself to read the actual documentation for the technologies discussed herein. Technologies like the ones discussed in this paper put amazing potential into the hands of the user, and all one needs to do is reach out and take advantage of them.

## IX. Appendix

**A test program used in the calibration of CID:**

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
      if(argc != 2){
            printf("Error: incorrect number of command arguments.
Terminating.\n");
            return 1;
      }
      int n = 0;
      char *x = *(argv +1);
      while(*x >= '0' && *x <= '9'){
            n = (n *10) + (*x - '0');
            x++;
      }
      for(int i = 0; i < n; i++){
            system("awk 'BEGIN {for(i=0;i<10000;i++)for
(j=0;j<10000;j++);}' &");
      }
      return 0;
}
//this program requires AWK
//to run, enter a number equal to 1+the_number_of_cpus
//as a command line argument
//(c)Ryan Kaulakis 2003, covered under GNU GPL v.2
```

**Citations**

1. http://www.cineca.it/streaming/openmosix/slides_moshe.php?radice=Diapositiva&last=32
2. http://openmosix.sourceforge.net/Business_Applications_2003.pdf
3. http://insight.zdnet.co.uk/hardware/servers/0,39020445,2125219,00.htm
4. http://www.gentoo.org/doc/en/handbook/index.xml
5. http://www.gentoo.org/doc/en/openmosix-howto.xml
6. http://www.linux-mips.org/linux-vr/ramdisk.html
7. http://www.gnu.org/software/grub/manual/grub.html
8. http://tldp.org/HOWTO/Clone-HOWTO/index.html
9. http://www.gentoo.org/doc/en/diskless-howto.xml
10. http://openmosix.sourceforge.net/linux-kongress_2003_openMosix.pdf
11. http://sourceforge.net/tracker/index.php?group_id=46729&atid=447171
12. http://oscar.openclustergroup.org/tiki-index.php
13. http://itsecurity.mq.edu.au/papers/White%20Paper%20-%20Security%20and%20openMosix.pdf