PERFORMANCE IMPROVEMENT BY TEMPORARY EXTENSION OF AN HPC CLUSTER

Pil Seong Park

Department of Computer Science, University of Suwon, Hwasung, Korea

ABSTRACT

Small HPC clusters are widely used in many small labs since they are easy to build and cost-effective. When more power is needed, instead of adding costly new nodes to old clusters, we may try to make use of the idle times of some servers in the same building, that work independently for their own purposes, especially during the night. However such extension across a firewall raises not only some security problem but also a load balancing problem caused by heterogeneity of the resulting system. In this paper, we devise a method to solve such problems using only old techniques applicable to our old cluster systems as is, without requiring any upgrade for hardware or software. We also discuss about how to deal with heterogeneity and load balancing in application, using a two-queue overflow queuing network problem as a sample problem.

KEYWORDS

HPC clusters, Security, NFS, SSH tunnelling, Load balancing

1. INTRODUCTION

A variety of architectures and configurations have appeared according to the desire of getting more computing power and higher reliability by orchestrating a number of low cost commercial off-the-shelf processors. One such example is a computer cluster, which consists of a set of loosely or tightly coupled computers that are usually connected to each other through a fast LAN and work together so that they can be viewed as a single system in many respects.

Many kinds of commercial clusters are already on the market. However the technologies to build similar systems using off-the-shelf components are widely known (e.g., see [1]), and it is easy to build a low-cost small cluster [2]. Many small labs commonly build their own cluster, and gradually grow it by adding more dedicated nodes later. However, instead of adding dedicated nodes, if there are some nodes that are being used for other purposes on the same LAN, we may try to utilize their idle times as long as they are not always busy enough, especially during the night. The Berkeley NOW (Network of Workstations) project is one of such early attempts to utilize the power of clustered machines on a building-wide scale [3]. However such extension gives rise to difficulties in security, administering the cluster, and load balancing for optimal performance.

In this paper, we consider some technical issues arising in extending a small Linux cluster to include some non-dedicated nodes in the same building across a firewall. Of course there are already various new good methods that can be used to avoid such technical issues. However we do not adopt such state-of-the-art technologies, since the sole purpose of this paper is to achieve our goal with our old HPC cluster as is, without any hardware or software upgrade. Some results of the experiments to evaluate the performance of the extended system are given at the end.

2. BACKGROUND

2.1. HPC Clusters

Computer clusters may be configured for different purposes. Load-balancing (LB) clusters are configurations in which nodes share computational workload like a web server cluster. High-performance computing (HPC) clusters are used for computation-intensive purposes, rather than handling IO-oriented operations. High-availability (HA) clusters improve the availability of the cluster, by having redundant nodes, which are then used to provide service when system components fail. The activities of all compute nodes are orchestrated by a cluster middleware that allows treating the whole cluster system via a single system image concept.

Well-known HPC middlewares based on message passing are the Message Passing Interface (MPI) [4] and the Parallel Virtual Machine (PVM) [5], the former being the de facto standard. Many commercial or non-commercial implementation libraries have been developed in accordance with the MPI standard. LAM/MPI, FT-MPI, and LA-MPI are some of widely used non-commercial libraries, and their technologies and resources have been combined into the on-going Open MPI project [6].

An HPC cluster normally consists of the dedicated nodes that reside on a secure isolated private network behind a firewall. Users normally access the master/login node (master, for short) only, which sits in front of compute nodes (slave nodes, or slaves for short). All the nodes in an HPC cluster share not only executable codes but also data via insecure NFS (Network File System). It is perfectly all right as long as the whole cluster nodes are on a secure LAN behind a firewall. However, to extend the cluster to include other nodes outside of the firewall, we will be confronted with many problems with security and data sharing among nodes. Moreover, such growth results in a heterogeneous cluster with nodes of different power, possibly running different Linux versions.

2.2. Linux/Unix File Sharing Protocols

NFS, created by Sun Microsystems in 1984, is a protocol to allow file sharing between Linux/UNIX systems residing on a LAN. Linux NFS clients support three versions of the NFS protocol: NFSv2 (1989), NFSv3 (1995), and NFSv4 (2000). However the NFS protocol as is has many problems in extending the cluster, since its packets are not encrypted and due to other shortcomings to be discussed later.

Other alternatives to NFS include AFS (Andrew File System), DFS (Distributed File System), RFS (Remote File System), Netware, etc. [7]. There are also various clustered file systems shared by multiple servers [8]. However we do not adopt such new technologies since they are not supported by our old cluster's hardware and software.

2.3. SSH Tunnelling

Encryption of NFS traffic is necessary for secure extension across a firewall. One of the common techniques that are commonly used is known as cryptographically protected tunnelling. An IP-level or TCP-level stream of packets is used to tunnel application-layer segments [9]. A TCP tunnel is a technology that aggregates and transfers packets between two hosts as a single TCP connection. By using a TCP tunnel, several protocols can be transparently transmitted through a firewall. Under certain conditions, it is known that the use of a TCP tunnel severely degrades the end-to-end TCP performance, which is called TCP meltdown problem [10].

The SSH protocol allows any client and server programs to communicate securely over an insecure network. Furthermore, it allows tunnelling (port forwarding) of any TCP connection on top of SSH, so as to cryptographically protect any application that uses clear-text protocols.

3. TEMPORARY EXTENSION OF AN HPC CLUSTER ACROSS A FIREWALL

We would like to extend our old cluster (consisting of the master and slave nodes in Figure 1) to include some other nodes, say Ext1 and Ext2, outside of a firewall, as shown in Figure 1. The master node generally has attached storage that is also accessible by diskless slave nodes using insecure NFS. Since NFS relies on the inherently insecure unencrypted UDP protocol (up to NFSv3), transactions between host and client are not encrypted, and IP spoofing is possible. Moreover, firewall configuration is difficult because of the way NFS daemons work.

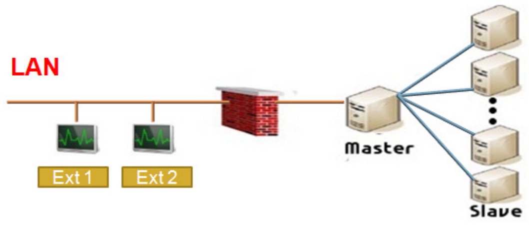

Figure 1. Cluster extension to include external nodes Ext1 and Ext2 across a firewall.

3.1. NFS over TCP and Fixing NFS Ports

SSH tunnelling, which is based on SSH port forwarding, is commonly used to encrypt some unencrypted packets or to bypass firewalls, e.g., see [9,11]. SSH tunnels support only TCP protocols of fixed ports, but NFS uses UDP protocols by default, and the ports of some daemons essential for the operation of NFS are variable.

Fortunately NFS over TCP protocols are also supported from the Linux kernel 2.4 and later on the NFS client side, and from the kernel 2.4.19 on the server side [12]. Since all the nodes of our old cluster satisfy this condition, we can make NFS work over TCP just by specifying the option "-o tcp" in the mounting command. The following shows an example of mounting the NFS *server*'s */nfs_dir* directory on the client's *mount_pt*.

```
# mount  -t nfs  -o tcp  server:/nfs_dir   mount_pt
```

For tunnelling, we need fix NFS ports. The daemons essential for NFS operation are 2 kinds. Portmapper (port 111) and rpc.nfsd (port 2049) use fixed ports, while rpc.statd, rpc.lockd, rpc.mountd, and rpc.rquotad use ports that are randomly assigned by the operating system. However the ports of the latter can be fixed by specifying port numbers in appropriate configuration files [13], as shown below.

The ports of the first three (i.e., rpc.statd, rpc.lockd, and rpc.mountd) can be fixed by adding the following lines in the NFS configuration file /etc/sysconfig/nfs, for example,

```
STATD_PORT=4001
LOCKD_TCPPORT=4002
LOCKD_UDPPORT=4002
MOUNTD_PORT=4003
```

The rest ports can be fixed by defining a new port number 4004 in the configuration file /etc/services, for example

```
rquotad 4004/tcp
rquotad 4004/udp
```

3.2. Setting up an SSH Tunnel

For tunnelling of NFS, it is necessary for the server to mount its own NFS directory to be exported to clients. Hence on the server side, the configuration file /etc/exports has to be modified, for example, for exporting its */nfs_dir* to itself,

```
/nfs_dir    localhost (sync,rw,insecure,root_squash)
```

where "insecure" means it allows connection from ports higher than 1023, and "root_squash" means it squashes the root permissions for the client and denies root access to access/create files on the NFS server for security.

Now an SSH tunnel has to be set up from the client's side. On the client side, to forward the ports 11000 and 12000, for example, to the fixed ports 2049 (rpc.nfsd) and 4003 (rpc.mountd), respectively, we can use the command

```
# ssh nfssvr -L 11000:localhost:2049  -L 12000:localhost:4003 -f sleep 600m
```

where "*nfssvr*" is the IP or the name of the NFS server registered in the configuration file /etc/hosts, and "-f sleep 600m" means that port forwarding is to last for 600 minutes in the background.

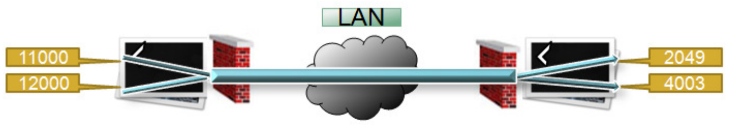

Figure 2. Local port forwarding for SSH tunnelling.

When connected to the NFS server, an SSH tunnel will be open if the correct password is entered. Then the NFS server's export directory can be mounted on the client's side. The following command is an example to mount the */nfs_dir* directory of the NFS server on the */client_dir* directory of the client.

```
# mount -t nfs -o tcp,hard,intr,port=11000, mountport=12000 localhost:/client_dir
```

3.3. Suggested Structure

Even though the NFS through an SSH tunnel is encrypted, it has a serious drawback if we cannot completely trust the local users on the NFS server [14]. For example, if some local user on the NFS server can login on the server and create an SSH tunnel, any ordinary user on the server can mount the file systems with the same rights as root on the client.
One possible solution might be prohibiting local users' direct login to the NFS server to prevent creating an SSH tunnel. One simple way is changing all local users' login shells from /bin/bash to /sbin/nologin in the file /etc/passwd. However it causes another problem. In general, the master node normally works as the NFS server too in small HPC clusters. However local users should be allowed to login the master node to use the cluster, which is dangerous when using NFS through an SSH tunnel, as was pointed out previously.

Figure 3 is an alternative of the structure of an extended HPC cluster that takes everything into account. The structure has a separate NFS server which does not allow local users' login. An SSH tunnel can be created by the root user on Ext1 or Ext2, and local users just login on the master node to use the cluster.

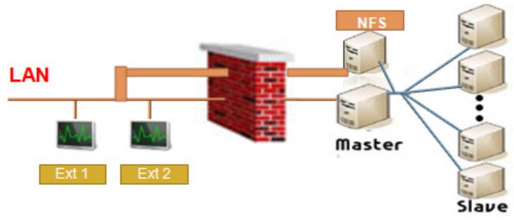

Figure 3. The suggested structure

The firewall setting of the master node does not need any modification. However, since the NFS service is provided through SSH tunnelling, it is required for the NFS server to open the port 22 only to the computation nodes outside of its firewall.

The NFS server should be configured so that it releases as little information about itself as possible. For example, services like portmapper should be protected from outside. And it is suggested to specify explicitly the hosts that are allowed to access all the services on the NFS server. This can be done by setting ALL:ALL in the configuration file /etc/hosts.deny, and listing only the hosts (or their IPs) together with the services which are allowed to access, in the file /etc/hosts.allow.

The use of “rsh” command, which is not encrypted, is common on old HPC clusters with old middlewares. Fortunately LAM/MPI v.7.1.2 allows SSH login with an authentication key but without a password, e.g., see [15].

4. DYNAMIC LOAD BALANCING

The original HPC cluster may be homogeneous, i.e., the power of all nodes may be the same. However, the extended cluster may be heterogeneous or may behave like heterogeneous one even if all the nodes have the same power, since the communication speeds between nodes are not the same depending on various factors: the types of network, existence of firewall, and encryption. Moreover the workload of the non-dedicated nodes outside of the firewall may change continually, and the communication speed may change continually because of the traffic on the outside LAN. Hence we need to use a dynamic run-time load balancing strategy [16], while initially assigning equal amount of work to the original slave nodes of the cluster.

Data partitioning and load balancing have been important components in parallel computation. Since earlier works (e.g., see [17]), many authors have studied load balancing using different strategies on dedicated/non-dedicated heterogeneous systems [18,19,20,21], but it was not possible to find related works on the security problems arising in cluster expansion, which is more technical rather than academic.

4.1. The Sample Problem

Dynamic load balancing strategies depend on the problem we deal with, and we need introduce our problem first. We would like to solve the old famous two-queue overflow queuing problem given by the Kolmogorov balance equation

$$\begin{aligned}&[\lambda_1\left(1-\delta_{i,n_1-1}\delta_{j,n_2-1}\right)+\lambda_2\left(1-\delta_{j,n_2-1}\right)+\mu_1\min(i,s_1)+\mu_2\min(j,s_2)]\,p_{i,j}\\&=\lambda_1\left(1-\delta_{i,0}\right)p_{i-1,j}+\mu_1\left(1-\delta_{i,n_1-1}\right)\min(i+1,s_1)\,p_{i+1,j},\quad i,j=1,2,\end{aligned}$$

where $\delta_{i,j}$ is the Kronecker delta, $p_{i,j}$ is the steady-state probability distribution giving the probability that there are i_j customers in the j-th queue. It is assumed that, in the i-th queue, there are s_i parallel servers and n_i-s_i-1 waiting spaces. Customers enter the i-th queue with mean arrival rate λ_i, and depart at the mean rate μ_i. The model allows overflow from the first queue to the second queue if the first queue is full. The total number of possible states is $n_1 \times n_2$.

It is known that no analytic solution exists, and the balance equation has to be solved explicitly. We can write the discrete equation as a singular linear system $A\mathbf{x}=0$, where $\mathbf{x}$ is the vector consisting of all states $p_{i,j}$'s. Even for systems with relatively small numbers of queues, waiting spaces, and servers per queue, the size of the resulting matrix is huge. The numbers of waiting spaces in our problem are 200 and 100 in each queue, respectively, and the resulting matrix size is $20{,}000 \times 20{,}000$.

4.2. Job Assignment by Domain Decomposition

It is easy to see that the graph of the resulting matrix of a k-queue problem is the same as the graph of the matrix corresponding to the k-dimensional Laplacian operator. Thus the graph of the matrix A is a two-dimensional rectangular grid.

As an example, Figure 4.a) shows the grid of a two-queue overflow model with $9 \times 5 = 45$ states, where each dot corresponds to some state $p_{i,j}$, i=1,2,…9 and j=1,2….5. If we use 3 computation nodes (or processes, one at each node), we may assign 15 grid points to each node. However to compute the values at boundary points (in vertical dotted boxes), each node needs the values at the boundary points to be computed by its adjacent nodes. As a result, node 0 and node 2 should have enough memory to store 20 grid points (or 4 vertical grid lines) and node 2 needs to store 25 grid points (or 5 vertical grid lines), even though each node updates only 15 grid points (or 3 vertical grid lines). The extra grid points are for storing the values to be received from adjacent nodes.

Figure 4.b) shows partitioning of the corresponding matrix A of size 45×45, where each non-zero entry is marked by a dot. Each submatrix A_i is of size 5×45, and for better load balancing, workload will be assigned in units of one vertical line (that corresponds to the computation with one submatrix A_i) on the 2D grid. Since we deal with a huge matrix problem of size $20{,}000 \times 20{,}000$, one unit workload is computation with a $100 \times 20{,}000$ submatrix A_i.

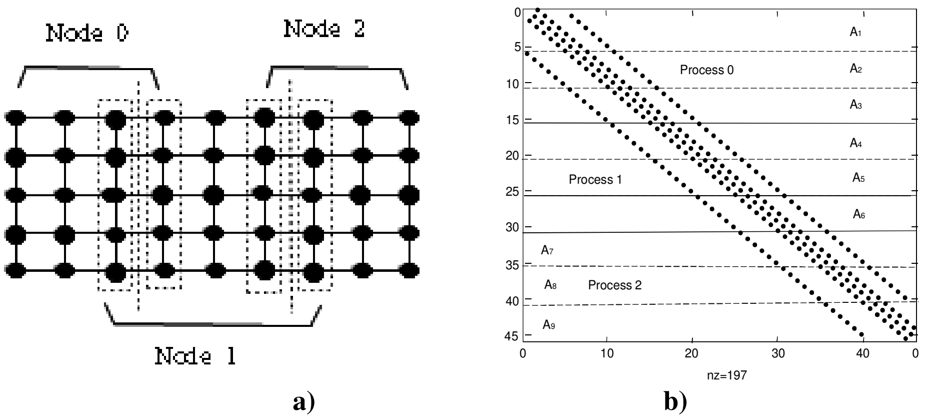

Figure 4. Examples showing a) job assignment to 3 nodes, and b) matrix partitioning for a 9 x 5 = 45 state two-queue overflow queuing model.

To solve the singular system $A\mathbf{x} = 0$, we use a splitting method *A=D-C* where *D* is the diagonal matrix that consists of all diagonal entries of *A*, and *C=D-A*. Then we can rewrite $D\mathbf{x} = C\mathbf{x}$ which leads to a convergent eigenvalue problem $D^{-1}C\mathbf{x} = \mathbf{x}$. For convergence test, we use the size of the residual defined by

$$\| \mathrm{r} \|_2 = \| (I - D^{-1}C)\mathbf{x} \|_2 \quad \text{where } \| \mathbf{x} \|_2 = 1.$$

4.3. Communication for Asynchronous Iteration

In normal synchronous parallel computation, each node updates the values of the grid points assigned to it in lockstep fashion, so that it can give the same result as that by serial computation. During computation, each node should communicate with its adjacent nodes to exchange boundary values, and all nodes should synchronize computation.

The workload of the added nodes Ext1 and Ext2 may change dynamically at all times, and it is unpredictable and uncontrollable. If any of these is busy doing some other jobs, it cannot communicate with other nodes immediately, degrading overall performance by making all other nodes wait. Asynchronous algorithms appeared naturally in parallel computation and have been heavily exploited applications. Allowing nodes to operate in an asynchronous manner simplifies the implementation of parallel algorithms and eliminates the overhead associated with synchronization. Since 1990s, extensive theories on asynchronous iteration have been developed, e.g., see [22]. Fortunately, the matrix *A* in our problem satisfies the necessary condition for a matrix to satisfy for convergence of asynchronous iteration. Hence we adopt asynchronous iteration freely, but we need some more work for implementation details.

We assume the master node is always alive and is not busy so that it can immediately handle communication with other nodes. Data exchange between any two nodes should be done via the master node, and the master node should always keep the most recent values at all grid points. But all communication should be initiated by compute nodes (including the added non-dedicate nodes), and the master node should just respond to compute nodes' request.

We do not present our full asynchronous parallel algorithm, but show only the key algorithm that allows asynchronous computation. Figure 5 is the master's algorithm using MPI functions to handle other nodes' request. The master node continually checks if any message has arrived from any other nodes by calling the MPI_Iprobe( ) function. If there is any message arrived, it then identifies the sender, tag, and sometimes the number of data received if necessary. Based on the tag received, the master takes some appropriate action. For example, if the received tag is 1000, it means "Receive the boundary data I send, and send the boundary data I need.", etc. Then any compute node can continue its work without delay.

```
do {
  MPI_Iprobe(MPI_ANY_SOURCE,MPI_ANY_TAG,communicator,&flag,&status);
  if (flag) {                          // TRUE if some message has arrived.
    sender=status.MPI_SOURCE;       // Identify the sender
    tag=status.MPI_TAG;             // Identify the type of the received message.
    If ( some data has been received )
      MPI_Get_count(&status,MPI_datatype,&n); // Identify no. of data received.

      // Depending on the value of "tag", take an appropriate action.

   }
}  until all other nodes finish computation
```

Figure 5. A master's algorithm to handle other nodes' request.

We also adopt a dynamic load balancing strategy by measuring the time interval between two consecutive data requests by the same node. It is necessary only for non-dedicated nodes, since the others are dedicated and have the same computational power. The workload of the non-dedicated node should be reduced if

$$n < \alpha \frac{N t}{T}$$

where N is the average workload (in units of the number of vertical grid lines), T is the average time interval between two consecutive data requests by the same dedicated node, t is the time interval between two consecutive requests of non-dedicated nodes, and α is some tolerance parameter less than 1 (we used 0.8). Similarly, the workload of the non-dedicate node is increased if

$$n > \beta \frac{N t}{T}$$

for some β (we used 1.2).

In adjusting workloads, we used a simpler strategy. When one of these conditions is satisfied, we just decrease/increase two boundary grid lines of non-dedicated nodes and assigned them to/from adjacent nodes. For this, to each non-dedicated node, we assigned the grid region which is between the two regions assigned to dedicated nodes. It takes very little time to re-assign grid lines to some other node, since the matrix is very sparse (just 5 nonzero entries in each row) and any submatrix A_i can be generated by each node instantly whenever necessary.

5. PERFORMANCE TESTS

Our old HPC cluster we want to extend has 2 nodes alive, each with dual 1GHz Pentium 3 Xeon processors running Fedora Core 4, with LAM/MPI v.7.1.2 installed. The nodes are interconnected via a Gigabit LAN, and NFS is used for file sharing. Both of the non-dedicated nodes Ext1 and Ext2 that are on the outside fast Ethernet LAN have a 2.4 GHz Intel® Core™ i5 CPU with 8 GB memory.

The performance of the NFS protocol through SSH tunnelling will inevitably drop due to encryption overhead. Communication speed tests were performed between the NFS server node and the node Ext1, using UDP or TCP, and with or without SSH tunnelling across the firewall. The times it took to read or write a file of size 1GB from the exterior node Ext1 were measured at varying NFS block sizes. Since NFSSVC_MAXBLKSIZE (maximum block size) of the NFS in Fedora Core 4 is 32*1024 (e.g., see /usr/src/kernels/2.6.11-1.1369_FC4-i686/include/linux/ nfsd/const.h), tests were performed at block sizes of 4k, 8k, 16k, and 32k, respectively, 3 times for each and the results are averaged. In addition, to delete any remaining data in cache, NFS file system was manually unmounted and mounted again between tests.

The following are example commands that first measure the time taken to create the file /home/testfile of size 1GB on the NFS server and read it using block size 16KB.

```
# time dd if=/dev/zero of=/home/testfile bs=16k count=65536
# time dd if=/home/testfile of=/dev/null bs=16k
```

The results of the NFS performance test, with or without SSH tunnelling over the fast Ethernet LAN are given in Table 1. For the common NFS without tunnelling, the figures in parentheses are the times it took when TCP is used, and others are when UDP is used.

As is expected, the larger the NFS block size, the faster in all cases. For the common NFS without SSH tunnelling, write operation using UDP is slightly faster than TCP, but it is not the case for read operation. Moreover the NFS with SSH tunnelling takes 3.9%-10.1% more time for write and 1.4-4.3% more for read, than the common NFS using UDP.

Table 1. The speed of NFS, with or without SSH tunnelling (sec).

Block size	Common NFS		SSH tunnelled	
	Read	Write	Read	Write
4K	96.12 (95.31)	119.37 (120.95)	100.29	131.45
8K	95.07 (94.43)	115.74 (120.31)	98.12	122.13
16K	93.85 (92.21)	114.70 (118.32)	95.31	121.72
32K	92.51 (91.45)	112.35 (115.87)	93.89	116.83

As long as the NFS block size is taken as large as possible, the tunnelling overhead may not be large, since the external nodes outside of the firewall need not read or write so often through NFS, which is common in high performance parallel computation.

Figure 6 shows the decrease pattern of residual until it reaches the size less than 10^{-5}. The line marked with A is by synchronous computation using only the dedicated nodes of our old cluster which has 4 cores in total. The line marked with B is the result when we add the node Ext1, and that marked with C is when we add both Ext1 and Ext2 (we created just 1 MPI process on each node Ext1 or Ext2). The size of the residual and elapsed time were checked every time the first slave reports the intermediate result. The running times to converge to the result with residual less than 10^{-5} were 955.47 sec for curve A, 847.24 sec for curve B, and 777.08 sec for curve C, respectively. Hence the speedup by B and C are 1.129 and 1.231, respectively.

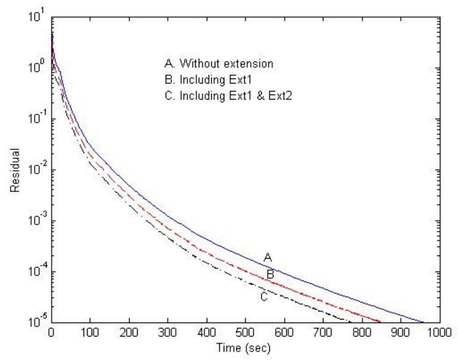

Figure 6. Comparison results.

The result is somewhat disappointing, since the speed of the CPU core on Ext1 or Ext2 is almost twice as fast as that of the old cluster. Probably it is mainly due to much slower network speed (fast Ethernet) of the outside network on which both Ext1 and Ext2 reside. Other reasons might be due to some extra workload imposed on them, and also due to the network traffic which influences the communication speed.

5. CONCLUSIONS

Small HPC clusters are widely used in many small labs, because they are easy to build, cost-effective, and easy to grow. Instead of adding new nodes, we can extend the clusters to include some other Linux servers in the same building, so that we can make use of their idle times. However, unlike a tightly-coupled HPC cluster behind a firewall, the resulting system suffers a security problem with NFS which is vital for HPC clusters. In addition, the resulting system becomes heterogeneous.

Of course there are many new good solutions using recent technologies. However we do adopt such strategy, because they require upgrade of hardwares and/or softwares including the operating system. Instead we devise a solution using SSH tunnelling, which can be applied to the old system as is. Probably this approach may be helpful for many small old cluster systems. We also considered a dynamic load balancing method which is based on overlapped domain decomposition and asynchronous iteration, using a two-queue overflow queuing network problem of matrix size 20,000 x 20,000. The speedup obtained by using extra processes on a much faster non-dedicated nodes is somewhat disappointing, mainly because of the slow network speed of the fast Ethernet between the cluster and the exterior nodes. However we can expect much higher speedup if the outer network is upgraded to a Gigabit LAN, for example.

ACKNOWLEDGEMENTS

This work was supported by the GRRC program of Gyeonggi province [GRRC SUWON2014-B1, Intelligent Context Aware & Realtime Security Response System].

REFERENCES

[1] G. W. Burris, "Setting Up an HPC Cluster", ADMIN Magazine, 2015, http://www.admin-magazine.com/HPC/Articles/real_world_hpc_setting_up_an_hpc_cluster
[2] M. Baker and R. Buyya, "Cluster Computing: the commodity supercomputer", Software-Practice and Experience, vol.29(6), 1999, pp.551-576.
[3] Berkeley NOW Project, http://now.cs.berkeley.edu/

[4] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. J. Dongarra, MPI: The Complete Reference. MIT Press, Cambridge, MA, USA, 1996.

[5] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam, PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA 1994.

[6] Open MPI, http://www.open-mpi.org

[7] Linux NFS-HOWTO, http://nfs.sourceforge.net/nfs-howto/

[8] Clustered file system, http://en.wikipedia.org/wiki/Clustered_file_system

[9] M. Dusi, F. Gringoli, and L. Salgarelli, "A Preliminary Look at the Privacy of SSH Tunnels", in Proc. 17th International Conference on Computer Communications and Networks(ICCCN '08), 2008.

[10] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama, "Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency," in Proc. 2005 OpticsEast/ITCom, Oct. 2005.

[11] Port Forwarding Using SSH Tunnel, http://www.fclose.com/b/linux/818/port-forwarding-using-ssh-tunnel/

[12] Linux NFS Overview, FAQ and HOWTO, http://nfs.sourceforge.net/

[13] http://www.linuxquestions.org/questions/linux-security-4/firewall-blocking-nfs-even-though-ports-are-open-294069/

[14] Linux NFS Overview, FAQ and HOWTO, http://nfs.sourceforge.net/

[15] ssh-keygen: password-less SSH login, http://rcsg-gsir.imsb-dsgi.nrc-cnrc.gc.ca/documents/ internet/ node31.html

[16] L. V. Kale, M. Bhandarkar, and R. Brunner, "Run-time Support for Adaptive Load Balancing", in Lecture Notes in Computer Science, Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP) Cancun - Mexico, vol.1800, 2000, pp.1152-1159.

[17] M. Zaki, W. Li, and S. Parthasarathy, "Customized Dynamic Load Balancing in a Heterogeneous Network of Workstations", in 1996 Proc. 5th IEEE Int. Symposium on High Performance Distributed Computing.

[18] M. Eggen, N. Franklin, and R. Eggen, "Load Balancing on a Non-dedicated Heterogeneous Network of Workstations." in International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002), June 2002.

[19] J. Faik, L. G. Gervasio, J. E. Flaherty, J. Chang, J. D. Teresco, E.G. Boman, and K. D. Devine, "A model for resource-aware load balancing on heterogeneous clusters", Tech. Rep. CS-03-03, Williams College Dept. of Computer Science, http://www.cs.williams.edu/drum/, 2003.

[20] I. Galindo, F. Almeida, and J. M. Badia-Contelles, "Dynamic Load Balancing on Dedicated Heterogeneous Systems", Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol.5205, 2008, pp 64-74

[21] K. Lu, R. Subrata, and A. Y. Zomaya, "On the performance-driven load distribution for heterogeneous computational grids", J. of Computer and System Sciences vol.73, 2007, pp.1191-1206.

[22] M. P. Raju and S. Khaitan, "Domain Decomposition Based High Performance Computing", International J. of Computer Science Issues, vol.5, 2009,pp.27-32.

AUTHOR

Prof. Pil Seong Park received his B.S. degree in Physical Oceanography from Seoul National University, Korea in 1977, and M.S. degree in Applied Mathematics from Old Dominion University, U.S.A. in 1984. He received his Ph.D. degree in Interdisciplinary Applied Mathematics (with emphasis on computer science) from University of Maryland at College Park, U.S.A in 1991. He worked as the director of Computer & Data Center at Korea Ocean Research & Development Institute from 1991 to 1995. Currently, he is a professor of the Department of Computer Science, University of Suwon in Korea. His research interest includes high performance computing, Linux clusters, digital image processing, and knowledge-based information systems. He is a member of several academic societies in Korea.