ENHANCING PERFORMANCE OF AN HPC CLUSTER BY ADOPTING NON-DEDICATED NODES

Pil Seong Park

Department of Computer Science, University of Suwon, Hwasung, Korea
pspark@suwon.ac.kr

ABSTRACT

Persona-sized HPC clusters are widely used in many small labs, because they are cost-effective and easy to build. Instead of adding costly new nodes to old clusters, we may try to make use of some servers' idle times by including them working independently on the same LAN, especially during the night. However such extension across a firewall raises not only some security problem with NFS but also a load balancing problem caused by heterogeneity. In this paper, we propose a method to solve such problems using only old techniques applicable to old systems as is, without requiring any upgrade for hardware or software. Some experimental results dealing with heterogeneity and load balancing are presented using a two-queue overflow queuing network problem.

KEYWORDS

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

1. INTRODUCTION

The desire to get more computing power and higher reliability by orchestrating a number of low cost commercial off-the-shelf computers has given rise to a variety of architectures and configurations. A computer cluster is composed 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.

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.

Many kinds of commercial clusters are 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

David C. Wyld et al. (Eds) : WiMo, ITCSE, ICAIT, NC - 2015
pp. 49–59, 2015. © CS & IT-CSCP 2015 DOI : 10.5121/csit.2015.51005

low-cost personal-sized cluster [2]. Many small labs first build their own personal-sized cluster, and gradually grow it by adding some more dedicated nodes later. Instead, if there are some other 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 early attempts to make use of 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 forecasting for optimal performance.

In this paper, we deal with some technical issues in extending a personal-sized Linux cluster across a LAN. We do not use state-of-the-art technologies, since the sole purpose is to achieve our purpose using our old HPC clusters as is, with no hardware or software upgrade. Some results of the experiments to evaluate the system are given at the end.

2. BACKGROUND

2.1. HPC Cluster Middlewares

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).

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. According to the standard, many commercial or non-commercial implementation libraries have been developed. 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].

All the nodes in an HPC cluster share not only executable codes but also data via NFS (Network File System), which is not encrypted in general. It is perfectly all right as long as the whole cluster nodes are on a secure local area network behind a firewall.

To extend the cluster to include other nodes outside of the firewall, we will be confronted with some 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. File Sharing Protocols

NFS, created by Sun Microsystems in 1984, is a protocol to allow file sharing between 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 to use 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.

2.3. SSH Tunnelling

Encryption of NFS traffic is necessary for secure extension across a firewall. One of the common techniques that are ordinarily 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. EXTENSION OF AN HPC CLUSTER

We would like to extend our old cluster to include some other nodes across the firewall, as shown in Figure 1. In general, the master node has attached storage that is accessible by diskless slave nodes using insecure NFS. Since NFS relies on the inherently insecure unencrypted UDP protocol (up to NFSv3), IP spoofing is possible. Moreover, firewall configuration is difficult because of the way NFS daemons work.

Figure 1. Cluster extension to include the non-dedicated node EXT across a firewall.

3.1. Fixing NFS Ports for SSH Tunnelling

SSH tunnelling, which makes use of SSH port forwarding, is widely 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 cluster satisfy this condition, we can make NFS work over TCP by specifying the option "-o tcp" in the mounting command. The following is an example of mounting *server*'s */nfs_dir* directory on the client's *mount_pt*.

mount -t nfs -o tcp *server*:*/nfs_dir* *mount_pt*

The daemons essential for NFS operation are 2 kinds. Portmapper (port 111) and rpc.nfsd (port 2049) use fixed ports, but 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].

The ports of the first three 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 new port numbers in /etc/services, for example

rquotad 4004/tcp
rquotad 4004/u에

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 */nfs_dir*,

/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.

On the client side, to forward the ports 10000 and 20000, for example, to the fixed ports 2049 (rpc.nfsd) and 4003 (rpc.mountd), respectively, we can use the command

ssh *nfssvr* -L 10000:localhost:2049 -L 20000:localhost:4003 -f sleep 600m

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

Once 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=10000, mountport=20000 localhost:*/client_dir*

4. DEALING WITH HETEROGENEITY

Data partitioning and load balancing are important components in parallel computation. Since earlier works (e.g., see [14]), many authors have studied load balancing using different strategies on dedicated/non-dedicated heterogeneous systems [15,16,17,18], but it is nearly impossible to find works on the security problems arising in cluster expansion, which is more technical rather than academic.

Even though the original tightly-coupled HPC cluster may be homogeneous, the extended cluster inevitably will behave like a heterogeneous system. First, the power of the newly added node EXT may be different and its workload may change continually since it is not a dedicate node. In addition, the communication speed between the node EXT and the original cluster is slower than that among the original cluster nodes. Hence we need to use a dynamic run-time load balancing strategy [19], while assigning equal amount of work to the nodes of the original cluster.

4.1. The Sample Problem

Strategies of dynamic load balancing depend on problems, and we need introduce the problem we consider first. We deal with the old famous two-queue overflow queuing problem given by the Kolmogorov balance equation

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

where $p_{i,j}$ 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, and 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, say $A\mathbf{x} = 0$ 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 x 20,000.

4.2. Job Assignment by Domain Decomposition

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 1.a) shows the grid of a two-queue overflow model with 9 x 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, 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 computed by 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 to store the values to be received from adjacent nodes.

Figure 1.b) shows partitioning of the corresponding matrix A of size 45×45, each submatrix A_i being of size 5×45. For better load balancing, workload will be assigned in units of one vertical line (that corresponds to the computation with one submatrix A_i). Since we deal with a huge matrix problem of size $20{,}000 \times 20{,}000$, one unit workload is computation with $100 \times 20{,}000$ submatrix.

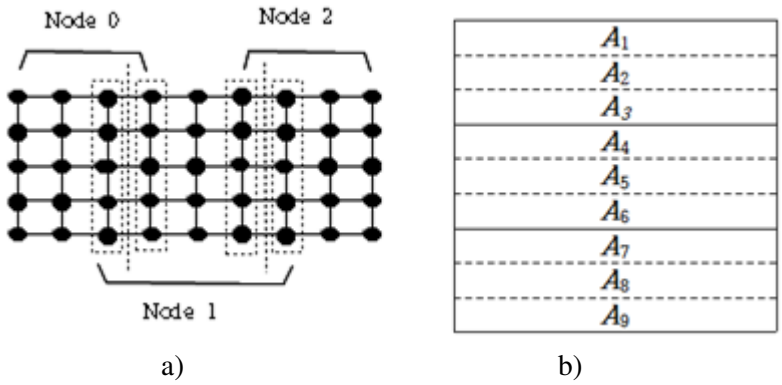

Figure 2. Examples showing a) job assignment to 3 nodes, and b) matrix partitioning for a $9 \times 5 = 45$ state two-queue overflow queuing model.

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

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

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 solution as that of serial computation. During computation, each node should communicate with its adjacent nodes to exchange boundary point values, and all nodes should synchronize computation.

The workload of the added node EXT may change dynamically all the time, and it is unpredictable and uncontrollable. If the node EXT is busy doing some other jobs, it cannot communicate with its adjacent nodes in time, degrading overall performance by making all other nodes wait.

Asynchronous algorithms appear naturally in parallel systems and are 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 1990's, some theories on asynchronous iteration has been developed, e.g., see [20]. Fortunately, the matrix *A* in our problem satisfies the necessary condition for a matrix to satisfy for convergence. 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 two adjacent nodes should be done through the master node, and the master node should keep the most current values of 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 according to compute nodes' request.

Figure 3 is the master's algorithm using MPI functions to handle other nodes' request. The master 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. 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 a 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 number of data received.

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

    }
} until all compute nodes finish computation
```

Figure 3. A master's algorithm to handle compute 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 is 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 requests of EXT, 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. That is, when one of these conditions is satisfied, we just decrease/increase two boundary grid lines of EXT 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 our matrix is very sparse (just 5 nonzero entries in each row) and can be generated by each node instantly whenever necessary.

5. PERFORMANCE TESTS

Our old cluster we want to extend has 2 nodes 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. The non-dedicated node EXT which sits on the outside fast Ethernet has a 2.4 GHz Intel® Core™ i5 CPU with 8 GB memory, and runs on Fedora 13.

The performance of the NFS through an SSH tunnel will inevitably drop due to encryption overhead. Tests were performed between the master node and the node EXT, using UDP or TCP, and with or without an SSH tunnel across the firewall. The times it took to read or write a file of size 1GB from EXT were measured, at varying NFS block sizes. Since NFSSVC_MAXBLKSIZE (maximum block size) of the NFS in Fedora Core 4 is 32*1024 (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 they 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 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 we see, 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.

As long as the NFS block size is taken as large as possible, the tunnelling overhead may not be large even though NFS service is done through SSH tunnelling, since the non-dedicated nodes outside of the firewall need not read or write so often through NFS, which is common in high performance parallel computing.

Table 1. Performance 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

Figure 4 shows the comparison results. The solid line marked with 1 is the synchronous computation result using only the dedicated nodes of our old cluster which has 4 cores in total. The dotted line marked with 2 is the result when we add the node EXT (we created just 1 process on EXT). 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 size less than 10^{-5} were 956.75 sec for curve 1 and 854.14 sec for curve 2, respectively. Hence the speedup by the added EXT node is 1.12.

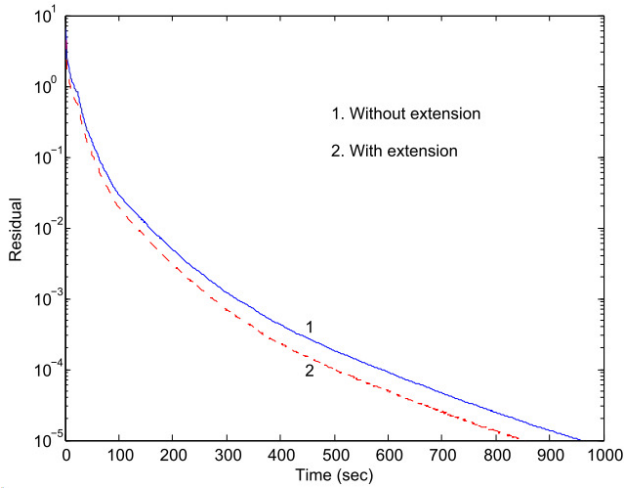

Figure 4. Comparison results.

The result is somewhat disappointing, since the speed of the CPU core on EXT is at least twice as fast as that of the old cluster. Probably it is mainly due to much slower network speed between the cluster and the node EXT and the extra workload for some other jobs.

6. CONCLUSIONS

Personal-sized HPC clusters are widely used in many small labs, because they are easy to build, cost-effective, and easy to grow. Instead of adding costly new nodes, we can extend clusters to include some other servers on the same LAN, 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.

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 an 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 to many of small home-made cluster systems. We also devised a dynamic load balancing method which is based on domain decomposition and asynchronous iteration, using a two-queue overflow queuing network problem of matrix size $20{,}000 \times 20{,}000$. The speedup obtained by using one extra process on a much faster non-dedicated node is somewhat disappointing, mainly because of the slow fast Ethernet speed between the cluster and the extra node. We expect much higher speedup if the outer network is upgraded.

ACKNOWLEDGEMENTS

This work was supported by the GRRC program of Gyeonggi province [GRRC SUWON2014-B1, Cooperative Recognition and Response System based on Tension Sensing].

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] 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.
[15] 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.

[16] 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.

[17] 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

[18] 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.

[19] 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.

[20] 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 head of Computer 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.