Evaluating BLAST Runtime Using NAS-Based High Performance Clusters

Sadiq M. Sait, M. Al-Mulhem
College of Computer Science and Engineering
King Fahd University of Petroleum and Minerals
{sadiq, mulhem}@kfupm.edu.sa

Raed Al-Shaikh
EXPEC Computer Center
Saudi Aramco
raed.shaikh@aramco.com

***Abstract* - The Basic Local Alignment Search (BLAST) is one of the most widely used bioinformatics programs for searching all available sequence databases for similarities between a protein or DNA query and predefined sequences, using sequence alignment technique. Recently, many attempts have been made to make the algorithm practical to run against the publicly available genome databases on large parallel clusters. This paper presents our experience in evaluating both the serial and parallel BLAST algorithms onto a large Infiniband-based diskless High Performance Cluster (HPC) that offers lower hardware cost and improved reliability, as opposed to traditional diskfull clusters. The paper also presents the evaluation methodology along with the experimental results to illustrate the scalability of the BLAST algorithm on our HPC system. For our measurement and comparison, we considered cluster sizes up to 32 compute nodes. Our results show that BLAST runtime can still be retained with the use of the diskless clusters, while improving the runtime reliability.**

Keywords: Diskless clusters, HPC Reliability, Infiniband, Experimental Performance, Genomes, BLAST.

I. INTRODUCTION

Biomedical Informatics is an interdisciplinary science which combines knowledge of biology, computer science and information engineering to manage health and biomedical information. Recently, it is playing leading role in research successes such as the mapping of human genes (also known as the human genome project) [1] and the comparison of primary biological sequence information of different proteins or DNA sequences, primarily performed by the Basic Local Alignment Search Tool algorithm (BLAST).

BLAST [19], which is our main tool in this research, is a suite of programs designed to search all available sequence databases for similarities between a protein or DNA query and known sequences, using sequence alignment technique. Sequence alignment provides an accurate mapping between the elements in the two strings. Given a pair of strings, there are many possible alignments, and each one can be assigned a quality score; by giving positive scores to exact letter matches and negative scores to substitutions (i.e., where one letter in a sequence is mapped to a different letter in the second sequence) and gaps (i.e., where mapped letters are the same, but they occur in different positions in the sequences). This scoring system for matches and substitutions is normally done in the form of a “scoring matrix” in which the entries reveal the biological impact of the corresponding matches or substitutions between two inputs [2]. The global similarity score of a pair of sequences is simply the score of the best (i.e., highest-scoring) alignment of the two sequences. It is important to mention that even if the global similarity score is low, there may still be portions of the sequences that match extremely well, and such local alignments are often of higher biological interest than the best global alignment.

Clearly, projects such as BLAST and other Biomedical Informatics projects in general, require data analysts and computing expertise as well as medical research talents to analyze and manage billions of data elements. On top, it has been estimated that the collective amount of genetic information doubles every twelve to eighteen months. This increased volume of information boosts the amount of computation required when comparing an unknown sequence to the databases of known sequences.

Though the growing interest in optimizing the parallel and distributed computing solutions using scaled-out hardware designs is matching this need, the increase in the number of cores – and hardware components in general - is drastically increasing the probability of hardware failures in such systems. On average, the normal failure rate of a 512 nodes cluster with dual sockets in each node is around 1-2 nodes per week [23]. Although sounds insignificant, every single failure on such a cluster would cause the whole running application to abort, causing tens of hours of computations to be wasted. As a result, one the attempts to increase the reliability of such platforms is researching the diskless HPC systems.

Diskless HPC clusters consist of compute nodes with no local disks. Instead, the compute nodes get their OS image during boot-up by using a centrally located device (or disk node) over a local LAN. In some designs, an internal network (e.g. 1 Gbps Ethernet) is used to provide not only inter-processor communications (IPC) among compute nodes but also a medium for booting and file transmission. In other advanced designs, as exhibited in Section III, the IPC communication is carried out on a separate extremely high-speed interconnect technology such as Infiniband or Myrinet. Each diskless compute node boots through the NIC's boot ROM with a small bootstrap, and then use either protocols such as BOOTP, DHCP, or NIC's Preboot Execution Environment (PXE) to get the OS image from a remote machine (in our case the disk node). Typically, a broadcast BOOTP request is first sent to a DHCP server to obtain an IP address. Then, the compute node sends a request to the TFTP server to get the boot image, point to the OS image, and start the booting process. During booting, all the necessary system files get transmitted through the network. The compute node completes the bootup when the remote file system is mounted as root file system (NFS_ROOT).

There is a number of obvious advantages to diskless clusters. First, the cost per cluster node becomes lower. Nowadays, the average cost of a server-level hard drive is about $200 [3]. This translates to $102,400 for a 512 nodes cluster. Second, diskless clusters have smaller footprints, i.e., lower power and cooling requirements. Third, cluster configuration and setup are consistent. In a diskfull cluster, system administrators spend considerable amount of time in developing and running script to ensure identical installations of OS images and files for all individual cluster nodes. In diskless cluster, since all nodes bootup over a network from a centralized disk server, identical OS images and installation files are ensured, thereby achieving system and file consistency across all compute nodes.

The real advantage to diskless clusters, however, is the reduced maintenance, or downtimes. With diskless systems, all mechanical parts – apart from the internal fans – are eliminated. For example, the mean time between failures (MTBF) of an internal disk is reported to be 300,000 hours, or 34 years of continuous operation [23]. Thus, if there is a cluster of 100 nodes, 3 to 4 disks will be replaced every year. If there is a cluster with 12,000 nodes, then on average, a disk fails every 25 hours, or around every day.

On the other hand, there are clearly obvious drawbacks associated with diskless HPC. The most obvious drawback is the added network traffic. Since the compute nodes load their OS image by using a centrally located device over a local LAN, a diskless HPC cluster configuration generates more network traffic than a diskfull HPC cluster by reading the image over LAN. Moreover, if the network connection or the centralized OS image is not available, none of the compute nodes will be accessible. Fortunately, solutions exist for these drawbacks [3], such as creating a RAM disk on each compute node by allocating part of the compute node's main memory as a partition for the file system. The RAM disk will be used for storing the most frequently accessed files. Therefore, the compute node can access some files from local memory instead of through the network.

Our primary objective in this paper is to evaluate both the serial and parallel BLAST algorithms on a diskless Infiniband-based High Performance Cluster. The motivation behind this work is to eliminate a major source of failure in the HPC environment and provide a reliable platform for BLAST's long runtimes, while retaining performance. To the best of our knowledge, this is the first paper that discusses BLAST performance when using diskless HPC based on the latest Intel's Westmere technology and Infiniband QDR interconnect.

The rest of the paper is organized as follows. In Section II, we present other related work in enhancing BLAST performance, generally either by running multiple sequential searches in parallel or by parallelizing the serial search algorithm. In Section III, we present our diskless cluster environment and detailed configurations. The performance and experimental results are discussed and analyzed in Section IV. Section V concludes the study and identifies future work.

II. RELATED WORK

Before discussing the related work on parallelizing BLAST, it is imperative to present the two basic methods that BLAST is based on, as indicated by Stephen Altschul et al. [25] in their work. These methods are:

(1) The maximal segment pair measure

Sequence similarity determination can be identified as either global or local. Global similarity technique focuses on optimizing the overall alignment of two sequences, which may include large stretches of low similarity. The local similarity algorithm, on the other hand, seeks only relatively conserved subsequences, and a particular comparison may result in several distinct subsequence alignments, whereas unconserved regions do not contribute to the measure of similarity. Generally, local similarity measures are preferred for database searches.
Many similarity measures begin with a matrix of similarity scores for all possible pairs of residues. Identities and conservative replacements have positive scores, while unlikely replacements have negative scores. For DNA sequence comparisons, identities are scored with +5, and mismatches -4; other scores are of course possible. A sequence segment is a contiguous stretch of residues of any length, and the similarity score for two aligned segments of the same length is the sum of the similarity values for each pair of aligned residues. Given these rules, we define a maximal segment pair (MSP) to be the highest scoring pair of identical length segments chosen from 2 sequences. The boundaries of an MSP are chosen to maximize its score, so an MSP may be of any length. The MSP score, which BLAST heuristically attempts to calculate, provides a measure of local similarity for any pair of sequences.

(2) Rapid approximation of MSP scores

When searching a DNA (or protein) database containing multi-thousands sequences, usually only a few, if any, will result in a match to the query sequence. For this reason, the examiner is interested in identifying only those sequence entries with MSP scores over some threshold score S. These sequences include those sharing highly significant similarity with the query as well as some sequences with borderline scores. This latter set of sequences may include high scoring random matches as well as sequences distantly related to the query. Surprisingly, the biological significance of the high scoring sequences may be inferred almost exclusively on the basis of the similarity score, while the biological context of the borderline sequences may be helpful in distinguishing biologically interesting relationships [25].

Based on these two methods, there have been several attempts to model and enhance the performance of BLAST algorithm, generally either by running multiple sequential searches in parallel or by modifying the serial search algorithm to become a parallel algorithm [1,2]. Although executing multiple sequential searches in parallel increases throughput by an amount that is proportional to the number of simultaneous sequential searches, the latency of each search will continue to increase at the rate that sequence databases are growing. On the other hand, transforming a serial algorithm into a parallel code requires knowledge in parallel programming, an area that has traditionally been confined to high-performance computing.

The serial BLAST was extensively evaluated using C, C++, C#, Java, Perl and Python programming languages [1], under both Windows and Linux operating systems. They also compared the memory usage and speed of execution for three standard bioinformatics methods, and one of which is BLAST algorithm. As expected, the implementations in C and C++ were the fastest and used the least memory. However, programs in these languages generally contained more lines of code. Java and C# appeared to be a compromise between the flexibility of Perl and Python and the fast performance of C and C++. The relative performance of the tested languages did not change from Windows to Linux and no clear evidence of a faster operating system was found. The authors however, did not explore the performance when parallelizing the code using the mentioned programming languages.

Two tightly coupled optimizations to mpiBLAST, the parallel version of BLAST algorithm, were proposed by Lin et. al [2]. These are: 1) fine-grained and dynamically load-balanced task scheduling and 2) scalable, asynchronous output processing. With fine-grained and dynamically load-balanced task scheduling, they allow flexible placement of master and worker processes to balance their computation loads. In addition, different worker processes can dynamically re-group and thus avoid long idle periods caused by the large difference in the computational times for different tasks. With asynchronous output processing, they obtained the performance benefit of parallel I/O without the synchronization overhead imposed by traditional collective I/O techniques. They presented extensive evaluation of these designs on large scale IBM BG/P systems. Their experimental results demonstrated that these two enhancements allow the application to scale almost linearly (93% efficiency) to 32,768 cores of BG/P. They also showed that those optimizations allow tackling real computational biology problems, i.e., sequence searching a microbial genome database against itself to support the discovery of missing genes in genomes.

The approach of Dynamic BLAST, an application to exploit the benefits offered by a distributed and heterogeneous system such as the Grid was also exploited [26]. The authors focused on using existing parallel versions of BLAST and extending those, particularly query segmentation. Rather than limiting execution of the search to any one resource, regardless of how large it is, they distributed the workload across different and independent resources found in the Grid, resulting in a larger pool of worker nodes and, thus, faster turnaround time. One of the key components of Dynamic BLAST is a file parsing module which examines user submitted query file containing any number of BLAST queries, and parses it to create a number of files or fragments of approximately equal size.

Further, a new parallelization approach to execute BLAST in distributed and parallel environments, considering load balancing was also proposed [22]. They used a replicated allocation of the (sequences) database, where each copy is physically fragmented. Their work is somewhat distinct, in a way that they focused on the database-level (i.e. database distribution design, I/O parallelism and query execution) to improve BLAST evaluation in clusters or Grids.

A high-throughput BLAST system based on Web services was also developed in 2003 [27]. Their solution provides an alternative BLAST service and allows users to perform multiple BLAST queries at one run in a distributed, parallel environment through the Internet. Soap-HT-BLAST is implemented using Perl and its module SOAP::Lite for Web services. The hardware architecture of Soap-HT-BLAST includes one head node (machine) and three compute nodes. Each compute node has four Ultrasparc III 900MHz CPUs and 8GB memory. The CPU load of a compute node is the summation of four CPUs' loads and the full load of each CPU is 25%.

III. THE CLUSTER DESIGN

To perform our BLAST benchmarks, a DELL cluster of PowerEdge M610 Blade Servers was used. It is the same cluster we utilized earlier to perform our diskless vs. diskfull HPC evaluation using the High Performance LINPACK (HPL) tool [12]. The cluster consisted of 32 nodes with dual sockets and Intel QuadCore X5570 (Nehalem) 2.93GHz processors. The operating system running on the nodes was RedHat Enterprise Linux Server 5.3 with the 2.6.18-128.el5 kernel. Each node was equipped with an Infiniband Host Channel Adapter (HCA) supporting 4x Double Data Rate (DDR) connections with the speed of 16Gbps, and 1Gbps Ethernet connection. The Infiniband connection was used for the actual inter-process communication while the Ethernet connection was mainly used for the OS image boot-up and remote access. Each node also had 12 GB (6 x 2GB) DDR3 1333MHz of memory, therefore, the total amount of memory the system had was around 384GB.

A disk node in the cluster was sharing a Linux ext3 file system as a network file system (NFS) among the 32 nodes of the cluster. This file system contained the Intel MPI libraries as well as the BLAST binaries and genomes databases. This helped in providing shared access to all of the 32 nodes of the cluster instead of having to propagate multiple copies of these items to all of the nodes. The disk node was also hosting the NFS_ROOT file system containing the operating system that will be shared among the diskless clients via network. All nodes had SSH trust keys between them so no password was needed for access. In addition to the 32 nodes, one management node was used to monitor and maintain the cluster and two subnet managers for the Infiniband interconnect network were available in an active/passive setup to manage the Infiniband network.

In case of diskless configuration, we had to increase the number of concurrent NFS threads running on the disk node hosting NFS_ROOT to handle the diskless clients' NFS requests, by adjusting the default value of "$RPCNFSDCOUNT" variable in the NFS process file. This value depends greatly on the IO load pattern, the network speed, the concurrency in the access and similar things. Red Hat recommends that systems administrators do a dynamic sizing of the number of *nfsd* threads. During our tests, 64 concurrent threads was found to be the optimal number. Nevertheless, the NFS I/O load on the disk node went up to 55 (*uptime* command figure), or:
[(55 / 8 cores) – 1 * 100 = 587%] over-utilization when the diskless cluster first booted.

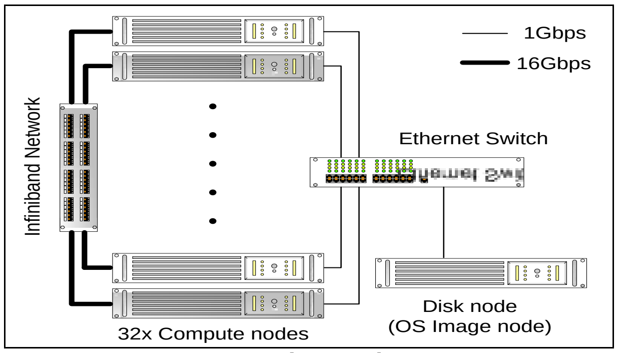

Figure 1. Experimental setup and communication

From each compute node we had a 4x-DDR Infiniband connection going to a central 32-port Qlogic Infiniband switch. Figure 1 shows the Infiniband interconnection design as described. It is important to mention that this design is considered non-blocking as each node guarantees to have the full 4x DDR 16Gbps interconnect speed. This fast interconnect would drive the cluster to a higher utilization, which in theory, may affect the diskless concept.

When we first booted our diskless cluster, the internal network first high sending burst took place at when the kernel image was being sent to the booting nodes to be loaded into their memory, while the second sending burst took place a few seconds later, which was caused by loading the actual OS files. These two bursts maxed to approximately 118MB/s, which is the maximum throughput of a 1Gbps connection. These bursts indicate a clear network contention on the disk node while the diskless nodes were booting up. In between the two bursts, the network activity goes down as the kernel image (initrd) scans for hardware in these diskless nodes, in which it does not need much of network activity. On the other hand and after completing the bootup process, the network activity decreased for both network send and receive down to 500KB/s, as the diskless nodes had the OS image loaded into memory, and minimal access to disk node would be required. Such minimal access is primarily caused by Linux activities related to /proc and other virtual file systems for collecting and reporting system statistics.

IV. EXPERIMENTAL RESULTS

In this section, we present our experimental results when benchmarking BLAST using both diskless and diskfull nodes.

In order to evaluate the performance, the benchmarks were run on the first diskless node for the serial BLAST tests, and ranging from one node and up to 32 nodes for the MPI experiments. We used both the NCBI BLAST [24] and mpiBLAST packages [26] for the tests. Both implementations are freely available, whereas mpiBLAST is the parallel implementation of the tool. The main benefit to using mpiBLAST versus the serial BLAST is performance. mpiBLAST can increase performance by several orders of magnitude [26] while still retaining identical results as output from the serial BLAST. Particularly, through the use of database fragmentation, mpiBLAST performs a BLAST search in parallel. Database fragmentation partitions a database into multiple fragments and by distributing the fragments across many computational-resources (e.g. cluster-nodes), where each fragment can be searched simultaneously. Furthermore, by segmenting the query into multiple, independent searches, multiple BLAST searches can be simultaneously performed.

In our experiments, we used two different databases against our two 560 and 1,410 nucleotides input sequences to examine the scalability of our BLAST runs, namely: the Drosoph database for having the Drosophila sequences with a size of 120MB, and the human genomes database with around 9.8GB of sequence records. Table 1 shows the performance benchmark of the serial BLAST using both the diskless and diskfull configurations.

Table 1. Serial BLAST comparison using the two cluster configurations

Cluster Type	Database	Elapsed Time (560 nucleotides sequence input)	Elapsed Time (1,410 nucleotides sequence input)
Diskfull	Drosoph	6.5 seconds	10.5 seconds
Diskless	Drosoph	6.3 seconds	10.1 seconds
Diskfull	Human genomes	221 seconds	289 seconds
Diskless	Human genomes	212 seconds	280 seconds

In this serial BLAST benchmark, the diskless runs slightly superseded the diskfull configuration in all iterations. Specifically, the diskless cluster performed around 3% better than the diskfull version. The rationale behind this slight increase is the elimination of disk accesses when referencing the OS was needed.

Table 2 shows the performance of the mpiBLAST code using the Drosoph database. In this test, the mpiBLAST was compiled using MVAPICH while retaining all the default options. In addition, the database was fragmented prior in each run to a number of partitions that is equal to the number of the cluster nodes, in order to achieve the optimal performance.

Tables 2. mpiBLAST performance benchmark using the Drosoph database

Nodes	Diskless (560 nucleotides)	Diskfull (560 nucleotides)	Diskless (1,140 nucleotides)	Diskfull (1,140 nucleotides)
	time	time	time	time
1	7.2 sec	7.6 sec	11.7 sec	12.4 sec
2	3.9 sec	4.0 sec	9.0 sec	9.5 sec
4	2.5 sec	2.6 sec	7.9 sec	8.1 sec
8	0.7 sec	0.73 sec	3.0 sec	3.1 sec
16	0.3 sec	0.31 sec	1.5 sec	1.5 sec
32	0.2 sec	0.2 sec	0.2 sec	0.2 sec

It is noticeable that the diskless-runs on a single node took around 7.2 seconds when using the 560-nucleotides sequence, whereas it took only 6.3 seconds when using the serial BLAST (in fact this observation applies to all single MPI-node runs vs. serial BLAST runs). This effect is due to the fact that the MPI-based BLAST code has more routines and functions to call, making the code more complex, and thus more time to run. Another observation is the degradation in performance increase rate when reaching 6 nodes. This degradation is related to the additional communication overhead with respect to the computation time. This communication is lessened in the Human genome database runs as the computation time gets larger with respect to the communication overhead. Similar to the serial BLAST tests, the diskless cluster outperformed the diskfull setup in both database runs.

Table 3: mpiBLAST performance benchmark using the Human genome database

Nodes	Diskless (560 nucleotide seq.)	Diskfull (560 nucleotide seq.)	Diskless (1,140 nucleotide seq.)	Diskfull (1,140 nucleotide seq.)
	time	time	time	time
1	230 sec	239 sec	296 sec	303 sec
2	121.7 sec	127 sec	164 sec	168 sec
4	85 sec	88 sec	105 sec	108.1 sec
8	22 sec	25 sec	35 sec	37.2 sec
16	15 sec	16.1 sec	23.2 sec	24 sec
32	4.0 sec	4.3 sec	5.0 sec	5.3 sec

Table 3 presents the performance using the 9.8GB Human genome database. Again, both the 560 and 1,140 nucleotides sequences were used in the runs. Overall, the performance of the diskless setup supersedes the diskfull cluster by around 2-4%. This percentage was lessened when exceeding 12 nodes as the MPI communication overhead became the main contributor to the running time. It is also noticeable that that performance scalability is somewhat linear when using up to 12-15 nodes.

During the mpiBLAST run, we also measured the effect of the diskless cluster environment in terms of temperature of both CPUs, the motherboard's temperature, and the power consumption for all 32 nodes. DELL's version of Intelligent Platform Management Interface (IPMI) tool [15] was used to collect such readings while the benchmarks were running on the nodes and fully utilizing the CPU and memory.

Table 4. Temperature and power consumption for diskfull vs. diskless HPC

#Nodes/State	Avg. Node Temp. (^{o}C)	Avg. Node Power (Watts)
126 Nodes/diskfull	21	282
126Nodes/diskless	21	279

In terms of temperature and heat dissipation, the diskfull and diskless readings were about the same at 21^{o}C while performing the HPL test. In terms of power consumption, however, the diskless nodes operated with an average of 279 Watts per node, compared to 282 Watts per node for the diskfull configuration. That is about 2% saving in power. This difference in power saving matches the hardware specifications of the published DELL internal disks power consumption [15] where they consume around 5 Watts per node. According to the United States' Department of Energy statistics for 2010, the average price for electricity in the USA is 14.53 cents per kW hour [19]. This would translate to an annual saving of U.S. $45,772 for a diskless cluster consisting of 12,000 nodes compared to a diskfull cluster of the same size.

Furthermore, selected tests were conducted to examine the behavior of diskless system under various conditions, one of which is the Gigabit Ethernet network utilization and disk I/O activities at the disk node throughout the experiment run time. It was noticed that obvious network and disk I/O activities occurred only when all the 32 nodes were booting up and loading the OS image via the network, reaching up to 115MB/s aggregate. However, after bootup, minimal activities were observed. Such observation is expected as access to OS node is needed only during the bootup of the 32 compute node, and later the inter-process communication among compute nodes is carried out by the Infiniband links. On the other hand, disk writes continued as the diskless nodes were writing their states on the disk node, such as system and kernel logs (e.g. /var). These writes, however, did not exceed 5MB/s aggregate.

We also measured the effect of losing the head node crash while the cluster is being utilized. Particularly, the NFS service was stopped (i.e. NFS_ROOT) on the disk node that was serving the OS image to the compute nodes. The NFS crash caused the compute nodes to completely stall with no network access. This is expected because during BLAST run, there was still access to disk node, but minimal in the range of 300-400 KB/s. The system was back to normal operation, however, when the NFS service was back online. This behavior is expected as the NFS protocol is stateless. That is, the NFS server should not need to maintain any protocol state information about any of its clients in order to function correctly. With stateless servers, a client needs only retry requests until the server responds; it does not even need to know that the server has crashed, or the network temporarily went down.

V. CONCLUSION AND FUTURE WORK

The Basic Local Alignment Search (BLAST) is a computation-intensive tool for searching all available sequence databases for similarities between a protein or DNA query and predefined sequences, using sequence alignment technique. Recently, many attempts have been made to make the algorithm practical and reliable to run against the publicly available genome databases. One of these attempts is to research the feasibility and performance of diskless HPC systems. In this paper, we evaluated the performance of serial and parallel BLAST algorithms using a state-of-the-art diskless HPC cluster. Our results show that BLAST runtime can still be retained with the use of the diskless clusters, while improving the runtime reliability.

As a future study, we plan to expand the size of the Infiniband diskless cluster to include 512 compute nodes, and then investigate BLAST performance. We also plan to evaluate diskless cluster performance when using other popular benchmarks such as the Pallas MPI benchmarking tool [17] which gives more insight on MPI behavior and performance. We are also considering measuring the performance of diskless clusters when using 10 Gbps Ethernet for IPC communication instead of Infiniband.

ACKNOWLEDGMENT

The authors would like to thank King Fahd University for Petroleum and Minerals (KFUPM) and the EXPEC Computer Center (ECC) at Saudi Aramco for their invaluable support and contributions to conduct this research.

REFERENCES

[1] Fourment,M. and Gillings,M.R. (2008) A comparison of common programming languages used in bioinformatics. BioMed Central Ltd Bioinformatics (BMC). 2008; 9: 82.

[2] Lin,H. et al.(2008) Massively parallel genomic sequence search on the Blue Gene/P Architecture. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, USA, pp.1-11.

[3] J. Laros and L. Ward, “Implementing scalable diskless clusters using the network file system”, Proceedings of the Los Alamos Computer Science Institute (LACSI) Symposium 2003, USA, October, 2003.

[4] B. Guler, M. Hussain; T. Leng, and V. Mashayekhi, “The Advantages of Diskless HPC Clusters using NAS”, DELL Inc., Nov. 2002.

[5] C. Yang and Y. Chang, “A Linux PC Cluster with Diskless Slave Nodes for Parallel Computing”, High-Performance Computing Laboratory, Department of Computer Science and Information Engineering, Tunghai University, Jan, 2003.

[6] C. Engelmann, H. Ong and S. Scott, “Evaluating the Shared Root File System Approach for Diskless High-Performance Computing Systems”, Proceedings of the 10th LCI International Conference on High-Performance Clustered Computing (LCI-09), Colorado, 2009.

[7] Terry Jones, Andrew Tauferner, Todd Inglett, et al., “HPC Colony: Linux at Large Node Counts Report from Experiments Conducted on Sixth BGW Day”, August 10, 2007

[8] J. Laros, C, Segura and N. Dauchy, “A Minimal Linux Environment for High Performance Computing Systems”, The 10th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida, July 2006, pp.130-138.

[9] C. Lu., “Scalable Diskless Checkpointing for Large Parallel Systems”, MSc. Thesis, University of Illinois at Urbana-Champaign, 2002.

[10] B. Maher, “Techniques to Build a Diskless Boot Linux Cluster of JS21 Blades”, IBM Red Book, 2006.

[11] T. Morgan JR., “DRBL: Diskless Remote Boot in Linux”, Master’s Capstone Project on High Performance Computing, April, 2006.

[12] HPL - High-Performance Linpack Benchmark. Available at: http://www.netlib.org/benchmark/hpl

[13] S. Frank and R. Haskin, “GPFS: A shared-disk file system for large computing clusters”, Proceedings of 1st Conference on File and Storage Technologies (FAST), USA, Jan., 2002, pp. 231–244.

[14] P. Reisner and L. Ellenberg, “Replicated storage with shared disk semantics”, Proceedings of the 12th International Linux System Technology Conference (Linux-Kongress), Germany, Oct, 2005, pp.111-119.

[15] DELL Blades Server for HPC M610. Available at: http://www.dell.com/us/en/enterprise/servers/server-poweredge-m610.

[16] C. Juszczak, “Improving the Write Performance of an

NFS Server", Proceedings of the USENIX Winter 1994 Technical Conference, USENIX, Association Berkeley, CA, USA, pp. 20-20, 1994.

[17] Pallas Benchmarking tools. Available at: http://people.cs.uchicago.edu/~hai/vcluster/PMB/

[18] J. Dongarra, J. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future", in the Journal of Concurrency and Computation: Practice and Experience, 2003, pp. 803-820.

[19] Energy Information Administration, USA Department of Energy http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_b.html

[20] NCBI BLAST, available at: http://blast.ncbi.nlm.nih.gov/Blast.cgi

[21] H. Lin, P. Balaji, R. Poole, C. Sosa, X. Ma, W. Feng, "Massively parallel genomic sequence search on the Blue Gene/P architecture", SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing.

[22] A. Darling, L. Carey, and W. Feng, "The Design, Implementation, and Evaluation of mpiBLAST", 4th International Conference on Linux Clusters: The HPC Revolution 2003 in conjunction with ClusterWorld Conference & Expo, June 2003.

[23] K Salah, R. Al-Shaikh, M. Sindi, "Towards Green Computing Using Diskless High Performance Clusters", submitted to Journal of Network and Computer Applications (JNCA), December, 2010.

[24] J. Sloan, "High performance Linux clusters with OSCAR, Rocks, openMosix, and MPI", O'Reilly Publication, 2005.

[25] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, "Basic local alignment search tool", J. Mol. Biol. 215:403-410, 1990.

[26] E. Afgan, and P. Bangalore, "Dynamic BLAST – a Grid Enabled BLAST", The International Journal of Computer Science and Network Security, Vol. 9, Issue 4, pp. 149-157, 2009.

[27] J. Wang, and Q. Mu, "Soap-HT-BLAST: High Throughput BLAST based on Web Services", Journal of Bioinformatics 19:1863-1864, 2003.