Accelerating MPIBLAST on System X by using RAMdisks

Vivek Venugopal, Kevin Shinpaugh, Geoff Zelenka, Luke Scharf

Advanced Research Computing,
Virginia Tech,
Blacksburg, VA 24061 USA
{vivekv, kashin, gzelenka, lscharf}@vt.edu

Abstract. Genome sequencing is a major contributor to the high performance computing workload worldwide. Scientists use Basic Local Alignment Search Tool (BLAST) from the National Center for Biotechnology Information (NCBI) and perform their searches against databases from GenBank, the worldwide repository for genome sequences. With GenBank reporting exponential growth rate, more emphasis is given to the development of faster search algorithms and better access techniques for the huge databases. Even though parallel implementations, such as mpiBLAST, have emerged to address this problem, the performance of mpiBLAST is dependent on a lot of factors including the storage, memory and algorithm to architecture mapping. This paper addresses a Random Access Memory(RAM)disk implementation on Virginia Tech's System X Linux nodes that includes modifications to the shared storage of the mpiBLAST program with fragments stored in a RAM-based filesystem before the scheduled mpiBLAST execution. The user specified genome database is pre-formatted, fragmented and stored in the local storage e.g. */tmp* partition of the compute nodes. The fragmented pieces are transferred to the RAMdisk partition from the local storage partitions of the compute nodes. The initial pre-formatting and distribution of databases results in an average time savings of 90% and a speedup of 8x over the normal Network File System (NFS)-based shared storage mpiBLAST implementation with sustained results over 16 and 32 cluster nodes for the various query files.

1 Introduction

The demand for computational horsepower has increased with today's highly intensive computational applications such as genome sequencing, fluid mechanics modeling and astrophysics simulations. The majority of the compute cycle time in the field of bioinformatics is spent on the Basic Local Alignment Search Tool (BLAST) [1] sequence database search algorithms. BLAST characterizes an unknown sequence by comparing it against a database of known sequences. The various BLAST algorithms provide different flavors such as *blastn* for nucleotide searches, *blastp* for protein searches, etc. [2]. The report generated from the program execution includes the statistical computation of the similarities between the sequence and the database, that helps in identifying the

function of the new sequence. BLAST is used extensively world-wide and is popular due to its powerful, flexible and fast implementation. It is also used for phylogenetic profiling and bacterial genome annotation. However, the existing sequence databases are growing at an exponential rate with over 65 billion nucleotide bases in more than 61 million sequences. The growth of the GenBank database is almost exponential, as the database doubles itself every 18 months [3]. In this scenario, it is very difficult to search databases that cannot fit in the main memory during the BLAST operation. The sequence search time can be considerably reduced by splitting the database and parallelizing the sequence search. The BLAST algorithms are embarrassingly parallel with no interdependency involved in the pair-wise sequence comparisons. The mpiBLAST implementation [4] uses the portable communication library Message Passing Interface (MPI) [5] as an add-on and is available as an open-source package with binaries for different platforms.

The mpiBLAST approach splits the BLAST database into equally sized fragments and then distributes the fragments over the cluster nodes. This way, each node searches only a unique portion of the database for the same sequence. The search tends to be very I/O demanding as the size of the query and database increase over the span of the research effort. Small query searches can be performed in a matter of seconds and minutes. However a search of the nucleotide (NT) database with the same database as the query describes the difficulties associated with huge databases [6]. The objective is to accurately use the available resources along with the software to increase the time savings and the speedup factor.

This paper is organized as follows: section 2 describes the motivation behind this work and the current implementation of mpiBLAST. Section 3 describes the proposed implementation technique with respect to the RAM disk and the different access methods. The results and performance analysis are presented in section 4. The conclusions and the future work are discussed in sections 5 and 6 respectively.

2 Motivation and the current implementation of mpiBLAST

System X is Virginia Tech's supercomputing cluster constructed from 1,100 Apple Xserve G5 nodes, each with dual 2.3-GHz PowerPC 970FX CPUs, 4-GB ECC DDR400 RAM, one 80-GB SATA local hard disk, one Mellanox Cougar InfiniBand 4x host channel adapter, and a Gigabit Ethernet (GbE) network interface card [7]. This supercomputing cluster is currently used to solve problems in a variety of fields including fluid mechanics modeling, structural analysis, chemistry and molecular dynamics, astrophysics calculations and biological sequence database search. The time allocated for the various projects as of September 2007 is shown in Figure 1. Biological applications, such as mpiBLAST and EpiSim, are allocated 18% of the total computation time on System X. Most of the mpiBLAST jobs are of a longer duration as compared to the EpiSim jobs and therefore mpiBLAST warrants a closer look. The main idea is to explore the different ways to improve the execution time of the mpiBLAST routine on most general purpose clusters with effective management of available storage resources. This also results in faster queue throughput on such clusters as a side effect.

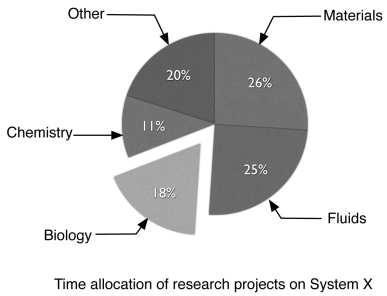


Fig. 1. Time allocation of research projects on System X

mpiBLAST reduces the time required for the computationally intensive sequence alignment process with the addition of more processors for job execution. However, there are a lot of initial steps in the mpiBLAST routine that consume a lot of time. Before a mpiBLAST search routine is executed, the raw sequence database is formatted and partitioned into fragments. The mpiBLAST package provides an application called *mpiformatdb*, that integrates both the database formatting and partitioning tasks. Depending on the number of nodes available, the user can specify the number of fragments, and the database is partitioned into an equally sized set of files. The user is expected to define a configuration file that specifies the local and shared storage paths for the database. The fragmented set of files are copied by the mpiBLAST program from the shared storage path to the local storage path during run time. The disadvantages associated with this formatting and partitioning task are as follows:

(a) this task needs to be repeated if the number of nodes change,
(b) large number of small files are generated, which is difficult to manage and copy along the nodes.

Even if all the users in the queue are using the same database, the database is formatted and partitioned for each user before the job execution, since the database is not available in a common path on the general purpose cluster. Moreover each user can specify a different parameter for the number of nodes for the mpiBLAST job.

The mpiBLAST routine shown in Figure 2 is based on database segmentation and organizes the different parallel processes into a master or head node and a set of worker or compute nodes. The head node uses a greedy algorithm for assigning the fragments to the compute nodes. The compute nodes copies these fragments to the local storage path specified in the configuration file and executes the mpiBLAST search routine. The local storage path is usually the local disk of the compute node. Each compute node reports it local results to the head node after finishing the search and match procedure for each fragment. This procedure is repeated until all the fragments have been searched. After the head node receives the results from all the query sequences, it formats and prints out the results to an output file using the standard NCBI BLAST output function.

After a close examination of the mpiBLAST routine, the time allocation is done for the following steps:

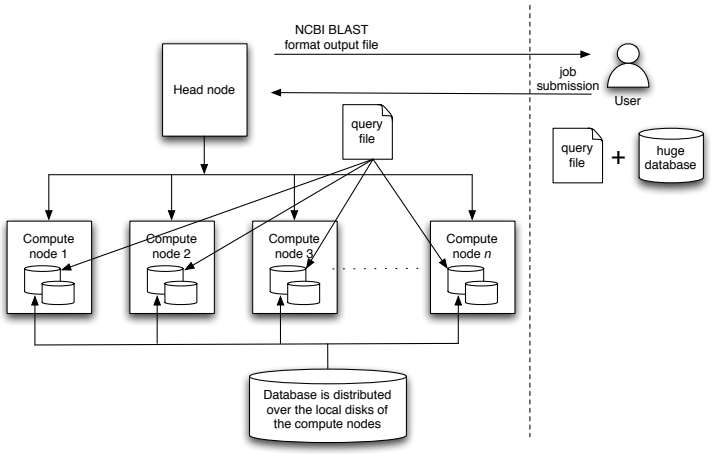


Fig. 2. Current implementation of mpiBLAST on SystemX

(i) formatting and partitioning of all the fragments,
(ii) copying all the data from the shared storage path to the local storage path on all the specified compute nodes,
(iii) disk to memory reads and writes for the sequence matching,
(iv) head node collects the results from all the compute nodes and merges them into a single output file.

Based on these observations, a mpiBLAST implementation on a general purpose cluster can benefit from optimizations in some of the above steps resulting in considerable time savings.

3 Proposed implementation of a mpiBLAST server on a general purpose cluster

While considering a different implementation of mpiBLAST on our cluster, one of the initial steps consisted of exploring query segmentation in addition to database segmentation. However, query segmentation has already been explored in some previous works [8], [9], [10]. In the query segmentation procedure, the query is fragmented into small files depending on the available compute nodes and each query is searched against the fragmented database stored in the local storage path of the compute node. However, with ever increasing sizes of databases, the query segmentation method results in huge overheads on the I/O and poor scalability on large systems. Since the query fragmentation was not viable for a general purpose cluster with large databases, the scope of optimization shifted to the efficient utilization of existing resources for the mpiBLAST routine.

3.1 Memory partition-RAMdisk

Linux provides several varieties of Random Access Memory (RAM)-backed filesystems, for various system-level tasks. These filesystems are very fast, since neither a physical hard drive nor a network is involved, but they are volatile; the contents are lost whenever the system is rebooted or shut down. A tmpfs style of RAM-backed filesystem is mounted by default in */dev/shm/* on many modern Linux distributions, including Yellow Dog Linux (YDL), Centos, Fedora, and Redhat. The tmpfs filesystem is backed by dynamic kernel-space memory, and it may be used as fast temporary scratch space for scientific applications such as mpiBLAST, provided that sufficient RAM is available to store the database (on the tmpfs filesystem) and also to perform the calculations.

The RAMdisks have been around for more than a decade, but they are generally not cost-effective. Hard drives and now tape drives are much cheaper per megabyte, but they are slower than the memory access speeds. So after a lot of number crunching, people often end up with the idea of multiple levels of caching to validate a hierarchical storage management type system. This type of management is used in our proposed implementation with pre-formatted databases stored on the local disks of the compute nodes and are transferred to the respective RAMdisks. On the System X nodes, the */dev/shm* partition is a dynamic mount partition capable of allocating 2 GB per processor.

3.2 Access techniques

On most general purpose clusters, the user submits the job using a batch queuing system. For a computationally intensive task such as mpiBLAST with common databases, all the users invest the same amount of time in partitioning and fragmenting the database. The number of fragmented databases and the node requirement is decided by the user. A considerable amount of time savings can be obtained by pre-formatting and distributing the fragmented databases to the local storage path on the compute nodes. The proposed mpiBLAST server shown in Figure 3 consists of the following features and procedures:

(i) All the databases are pre-formatted and split into a specified number of fragments, which is same as the number of nodes allocated for the mpiBLAST server. The format of the fragmented database now dictates the number of nodes used by any job that runs against this set of files.

(ii) The fragmented databases are then distributed over the RAMdisk partitions of all the compute nodes. Also, a copy of the database is stored on the local hard disk of the compute node, in order to decrease the amount of time required to bring the node back into service after a reboot.

(iii) Each user submits a mpiBLAST job that consists of the query file to be matched against a specific database. The mpiBLAST routine starts the alignment process on all the compute nodes directly as the initial tasks have already been performed.

(iv) At the end of the mpiBLAST routine, the head node merges the results and generates the output file in the NCBI BLAST format.

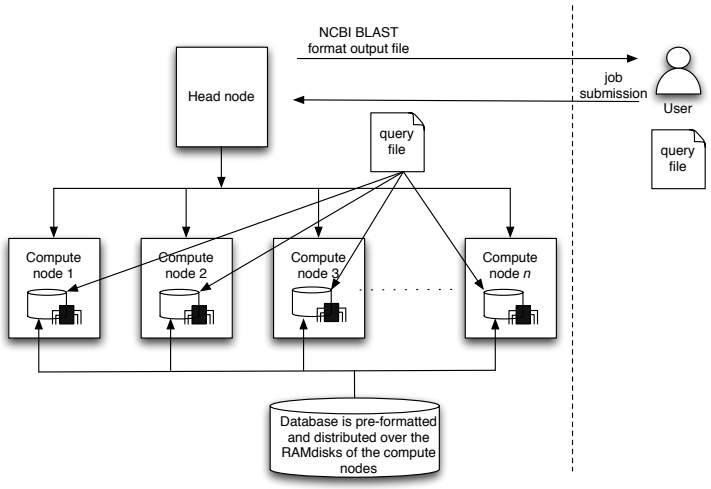


Fig. 3. Proposed implementation of mpiBLAST using RAMdisk

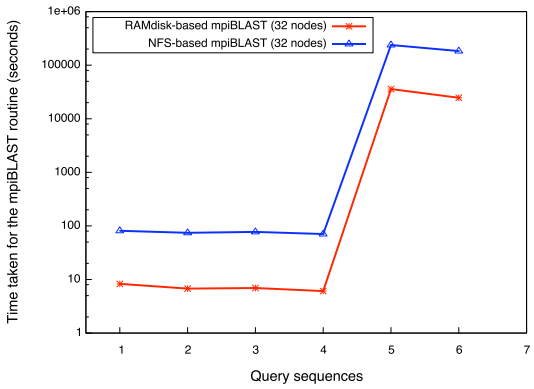
Table 1. Comparison of mpiBLAST for both RAMdisk and NFS based implementations

Test run conducted for 32 node cluster					
Query sequence	**Size**	**Time taken for RAMdisk-based execution (secs)**	**Time taken for NFS-based execution (secs)**	**%age time savings**	**Speedup factor**
Query1	4KB	8.288	81.157	89.78	8.79
Query2	4KB	6.766	74.343	90.89	9.87
Query3	4KB	6.935	77.357	91.03	10.15
Query4	4KB	6.073	70.6414	91.4	10.63
Query5	9.3MB	35886.724	238216.97	84.935	5.64
Query6	9.3MB	24673.634	183637.101	86.563	6.442

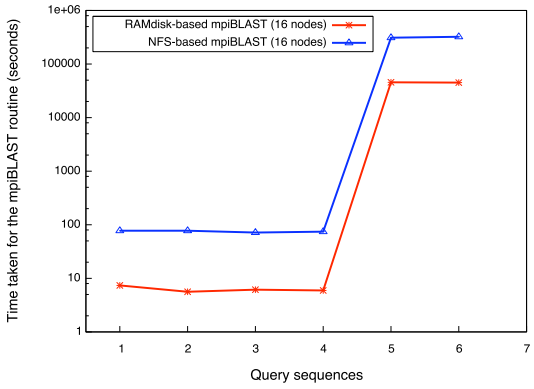
Test run conducted for 16 node cluster					
Query sequence	**Size**	**Time taken for RAMdisk-based execution (secs)**	**Time taken for NFS-based execution (secs)**	**%age time savings**	**Speedup factor**
Query1	4KB	7.381	77.515	90.477	9.50
Query2	4KB	5.615	77.4286	92.747	12.79
Query3	4KB	6.142	71.564	91.41	10.65
Query4	4KB	5.939	74.326	92	11.51
Query5	9.3MB	45581.589	310178.836	85.30	5.804
Query6	9.3MB	45022.973	322654.86	86.04	6.166

4 Results and performance analysis

The experimental setup consisted of setting up two sets of clusters for testing the proposed mpiBLAST server. We used a 16 compute node and 32 compute node cluster for obtaining the results. We selected the nucleotide (NT) database containing GenBank, EMB L, D and PDB sequences. This database is an unordered ASCII file that is updated daily and is about 22 GB as of July 2007. The randomly selected query files are of different sizes and have been sequenced against the NT database on both sets of the clusters. Queries 1-4 belong to the *e.chrysanthemi.fas* sequence and was used in most of the original mpiBLAST work [4]. Queries 5 and 6 are large query files with more than thousand sequences and belong to *EST.unigenes.fasta* and *peronospora.FinalTranscript* categories respectively.



(a) 32-node mpiBLAST implementation



(b) 16-node mpiBLAST implementation

Fig. 4. Execution time for mpiBLAST implementations

The results obtained for the RAMdisk and the NFS-based mpiBLAST routines have been tabulated in Table 1 for the 32 node and 16 node cluster implementations. The execution time for all the mpiBLAST routines are obtained from an average of 5 values. After each NFS-based mpiBLAST execution an automated script was used to clean up all the files in the local storage path. This setting simulates each unique user logging onto the system and submitting their mpiBLAST job. On the RAMdisk-based implementations, automated scripts were used to store the pre-formatted and fragmented databases on the RAMdisk partition before the scheduled mpiBLAST execution. The time taken for both the mpiBLAST implementations on the 16-node and 32-node clusters is shown in Figure 4. The performance results show an overall time savings of 90% and an average speedup of 10x for the RAMdisk-based mpiBLAST implementation.

5 Conclusions

This paper proposes a diskless mpiBLAST server using the RAMdisk based implementation. We have shown that the proposed RAMdisk-based implementation provides a considerable time savings by the distribution of the pre-formatted fragmented databases. This also results in an overall speedup for the query search using the modified mpiBLAST routine. However, the results vary for the different queries based on their sizes. The proposed RAMdisk implementation shows an 6x improvement for the large query files (i.e. queries 6 and 7). The different queries have different access patterns and a particular working set of sequence alignment. This mpiBLAST modification is inexpensive in terms of system resources and can be easily implemented on general purpose clusters. Based on the results, we recommend the usage of a RAMdisk based implementation for mpiBLAST as there is no cost involved in setting up the RAMdisk partition on the compute nodes.

6 Future work

The varying results for the queries with different sizes opens the possibility of doing a query segmentation in addition to the database segmentation over the clusters. However, this may not effectively scale well and therefore is currently being tested for a better performance analysis. The characterization of the query files for a specific database can yield more information about its access patterns and this needs to be studied to determine a linear relationship between the execution time versus query size. Another aspect that merits further investigation is that the current mpiBLAST program reads the data from the local storage into the memory and does the sequence alignment. In our proposed implementation, we use the RAMdisk partition as the local storage, which is essentially the memory. Therefore, two copies of the database reside in the memory partition of the clusters and this can be avoided by setting a parameter to avoid reading the database into memory due to the modified local storage path. The job submission queue can also be parsed to group the users with same database requests for efficiently allocating the databases over the memory partitions.

7 Acknowledgments

We would like to acknowledge Jeremy Archuleta and Heshan Lin for all their assistance and also for the fruitful discussions and suggestions. We also thank Sucheta Tripathy from Virginia Bioinformatics Institute (VBI) for assisting us with the query selections for testing the mpiBLAST implementation.

References

1. National Center for Biotechnology Information: NCBI BLAST. http://www.ncbi.nlm.nih.gov/BLAST/ (Last Accessed: December 2006)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215**(3) (1990) 403–410
3. NCBI: GenBank statistics. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html (Last Accessed: December 2007)
4. Darling, A., Carey, L., Feng, W.: The design, implementation, and evaluation of mpiBLAST. In: ClusterWorld Conference and Expo, in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003, San Jose, California (June 2003)
5. MPI:: The Message Passing Interface standard. http://www-unix.mcs.anl.gov/mpi/index.htm (Last Accessed: December 2007)
6. Gardner, M.K., chun Feng, W., Archuleta, J., Lin, H., Ma, X.: Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications. In: Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference. (2006) 22–22
7. Advanced Research Computing: System X. http://www.arc.vt.edu/arc/SystemX/index.php (Last Accessed: December 2007)
8. Braun, R.C., Pedretti, K.T., Casavant, T.L., Scheetz, T.E., Birkett, C.L., Roberts, C.A.: Parallelization of local blast service on workstation clusters. Future Generation Computer Systems **17**(6) (2001) 745–754
9. Lin, H., Ma, X., Chandramohan, P., Geist, A., Samatova, N.: Efficient data access for parallel blast. In: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. (2005) 72b–72b
10. Chi, E., Shoop, E., Carlis, J., Retzel, E., Riedl, J.: Efficiency of shared-memory multiprocessors for a genetic sequence similarity search algorithm. Technical Report TR97-005, University of Minnesota Computer Science Department (1997)