In Proc. of the 5th IEEE International Conference on Cluster Computing (Cluster 2003), Hong Kong, Dec. 1-4, 2003.

A Case Study of Parallel I/O for Biological Sequence Search on Linux Clusters

Yifeng Zhu, Hong Jiang, Xiao Qin and David Swanson
Department of Computer Science and Engineering
University of Nebraska - Lincoln
Lincoln, NE 68588-0115
{yzhu, jiang, xqin, dswanson}@cse.unl.edu

Abstract

In this paper we analyze the I/O access patterns of a widely-used biological sequence search tool and implement two variations that employ parallel-I/O for data access based on PVFS (Parallel Virtual File System) and CEFT-PVFS (Cost-Effective Fault-Tolerant PVFS). Experiments show that the two variations outperform the original tool when equal or even fewer storage devices are used in the former. It is also found that although the performance of the two variations improves consistently when initially increasing the number of servers, this performance gain from parallel I/O becomes insignificant with further increase in server number.

We examine the effectiveness of two read performance optimization techniques in CEFT-PVFS by using this tool as a benchmark. Performance results indicate: (1) Doubling the degree of parallelism boosts the read performance to approach that of PVFS; (2) Skipping hot-spots can substantially improve the I/O performance when the load on data servers is highly imbalanced. The I/O resource contention due to the sharing of server nodes by multiple applications in a cluster has been shown to degrade the performance of the original tool and the variation based on PVFS by up to 10 and 21 folds, respectively; whereas, the variation based on CEFT-PVFS only suffered a two-fold performance degradation.

Keywords: *parallel I/O, CEFT-PVFS, PVFS, BLAST*

1. Introduction

Linux clusters have quickly gained popularity in the bioinformatics community as a cost-effective, high-performance computing platform in the past few years. The development in molecular biology has led to an unprecedented explosion in the volumes of generated biological data. For example, among the dozens of publicly accessible database centers, the European Bioinformatics Institute (EBI) holds more than 120 biological databases [1] and the National Center for Biotechnology Information (NCBI) stores approximately 31×10^9 base pairs in 24 million sequences [2]. In such cases, Linux clusters usually need to input and output large amounts of data from the storage subsystems. Unfortunately, the performance of storage subsystems has increasingly lagged behind the performance of computation and communication subsystems. This increasing performance disparity frequently prevents the effective utilization of the teraflop-scale computing capability that a modern cluster can offer.

To exploit the collective storage capacity existing among the local storage devices on cluster nodes, a wide variety of parallel I/O systems have been proposed and developed. One notable example of such systems is PVFS [3][4], which is a RAID-0 style high performance file system providing parallel data access with cluster-wide shared name space. While it addresses I/O issues for the low-cost Linux clusters by aggregating the bandwidth of the existing disks on cluster nodes, PVFS has two main disadvantages. It does not provide any fault tolerance in its current version and thus the failure of any single cluster node renders the entire file system service unavailable. In addition, the system resource contention on data servers can significantly degrade the overall I/O performance. A Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS) [5][6][7], extends PVFS from a RAID-0 to a RAID-10 style parallel file system to meet the critical demands on reliability and to minimize the performance degradation due to resource contention by taking advantages of the data and device redundancy.

In this work, we investigate the I/O access patterns of the MPI-based implementation of a well-known biological sequence search tool, called Basic Local Alignment Search Tool (BLAST) [8][9]. This parallel BLAST uses conventional I/O interfaces to access local disks. We implemented two new variations that run on the PVFS and CEFT-PVFS through their parallel I/O interfaces. The performance of the original and the two new implementations is measured and compared, based on the results and measurements of executing the three programs within the same environments. Through this real application, we examine the relationship between the degree of parallelism and the I/O performance. In addition, we study the impact of the system resource utilization on

the I/O performance and point out some important issues that must be addressed in parallel I/O designs.

Our experimental results indicate that the I/O performance experienced by the parallel BLAST stops increasing after the degree of parallelism in parallel I/O accesses increases to a certain level, due in part to the diminishing gains of I/O parallelism as computations (as opposed to I/O accesses) on the worker nodes become dominant. In addition, the parallel I/O performance is heavily influenced by the system resource utilization on all the data servers. When data server nodes are severely unevenly utilized (e.g., with one node much more heavily loaded than others), due to the sharing of resources by multiple applications running simultaneously on server nodes, our experiments showed that both the original parallel BLAST and the PVFS-based parallel BLAST suffered drastic performance degradations, by a factor of 10 and 21, respectively, because of their inabilities to avoid any hot-spot node. On the other hand, it is found by our experiments that the performance degradation caused by such severe load imbalance can be minimized in CEFT-PVFS by its ability to skip one or more hot-spot nodes. We believe that these findings not only benefit the bioinformatics research community, but also provide useful insights into the parallel I/O research in modern clusters.

The rest of this paper is organized as follows. Section 2 overviews some biological sequence search tools. Section 3 presents two parallel I/O implementations in a parallel sequence search application. In Section 4, the experimental performance results are discussed, along with an evaluation and comparison of the three parallel implementations of BLAST. Studies in the literature related to the current work are briefly discussed in Section 5. Finally, Section 6 concludes the paper with comments on current and future work.

2. Application Overview

The development of modern biological research has created the need for intensive biological sequence comparisons. When new biological sequences are discovered, biomedical researchers would search the existing databases of genes or proteins for similar or related sequences. Such analyses have great scientific value since the structure and function of new sequences may be implied from the known characteristics of similar or related sequences. The following introduces several powerful software tools for similarity searches.

2.1 BLAST

BLAST [2] has been continuously developed and refined since its initial release by NCBI in 1990 and is now the mostly widely used tool for a sequence similarity search. Given a query sequence and a sequence database as inputs, BLAST searches all entities in the database for those with high-scoring gapped alignment to the query, where the deletion, insertion and substitution are allowed in sequence comparison and the alignment scores are determined statistically and heuristically based on expert-specified scoring matrix.

The BLAST search tool contains a set of five programs: *blastn, blastp, blastx, tblastn and tblastx*. *blastn* and *blastp* perform a nucleotide or peptide sequence search in a sequence database of the same type respectively. In contrast, the *blastx* and *tblastn* can perform a sequence search of one type with a database of the other type by taking advantage of the fact that nucleotide and peptide sequences can be translated into each other in living cells. Specifically a nucleotide sequence can be translated into a peptide sequence and then compared with a database of peptide sequences, and vice versa. Finally, the *tblastx* differs from all four programs above in that it conducts six-frame translations on both the query and the database entities before comparing them. NCBI provides *blastall* as a single interface allowing the access to the five different comparison programs.

2.2 Parallel BLAST

Much work has been done to parallelize the BLAST programs in two approaches: query segmentation and database segmentation. In the first approach, the entire database is replicated to all worker nodes but the query sequence is split into several pieces. Each worker searches the entire database using one query piece. In the latter approach, the whole query sequence is copied to all workers while the database is divided into multiple segments. Each worker searches one database fragment using the entire query.

With the explosion of the database size, the first approach becomes less attractive due to large I/O overhead. Nowadays the size of sequence databases is exploding and can easily exceed the available memory capacity. If the sequence databases cannot fit into the physical memory, BLAST is forced to page into disks, forcing the CPU to sit idle while waiting for the memory page in and page out [10][11].

WU-BLAST [12] implemented both parallel approaches, but it runs on its own libraries. This causes a difficulty to keep up with new versions of BLAST distributed by NCBI. TurboBLAST [11] and mpiBLAST [10] implemented the second approach. They both directly deploy NCBI BLAST library without any modification. Since TurboBLAST is a commercial program and its source code is not publicly accessible, we choose the open-source mpiBLAST for our study.

The mpiBLAST algorithm involves a master and a number of workers. The master is responsible for

assigning the search tasks to the idle workers and for merging the results from all workers according to their alignment scores. Each worker copies the assigned database fragments to its local storage device and then executes the NCBI *blastall* to search through the database fragments.

3. Parallel I/O Implementations

The mpiBLAST does not employ any parallel I/O facilities and each worker accesses its own local disk. We implemented two parallel I/O variations over PVFS and CEFT-PVFS. Figure 1 shows the software stack of our implementation, with the mpiBLAST being a parallel wrapper around the BLAST library. While the original BLAST uses the conventional memory mapped I/O access, we intrusively modified the I/O subcomponent of BLAST and replaced its conventional I/O system calls with the PVFS and CEFT-PVFS native interfaces. As a result, the databases are striped on all data servers in a round robin fashion and each worker accesses the data servers in parallel. In our implementation, the stripe size is set at 64KB.

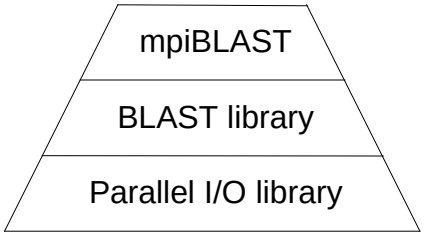

Figure 1. Software stack of mpiBLAST utilizing parallel I/O

To fairly compare the performance of the original mpiBLAST and the two new implementations with parallel I/O, we designed our experiment under the condition that they use equal amounts of hardware resources whenever possible. To achieve this, we artificially place the mpiBLAST master node and the metadata server of PVFS or CEFT-PVFS on the same node, and all mpiBLAST workers and the data servers on the same nodes if they are equal, as shown in Figure 2. If the data server number is not equal to the worker number, we make them overlap to the maximum degree.

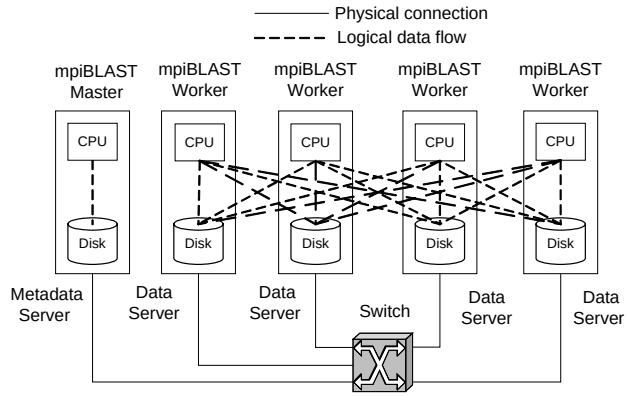

Figure 2. mpiBLAST over PVFS or CEFT-PVFS

CEFT-PVFS is a RAID-10 style parallel file system, which combines striping with mirroring by first striping among the primary group of server nodes and then duplicating all the data in the primary group to the mirror group to provide fault tolerance. The read operations in CEFT-PVFS have been designed to double the degree of parallelism: reading the first half of a file from one storage group and the second half from the other group in parallel if the desired data has already taken residence on both groups. In this way, for a single read operation, all data servers are involved. In addition, the metadata server is not only responsible for providing the striping information to the client nodes for parallel accesses; it also periodically collects the system resource utilization information from all data servers and determines the I/O service schemes. Figure 3 shows an example where the disk of one data server has already been extremely loaded by other applications running on the cluster. The metadata server detects the hot spot and informs the clients to skip that server node and read the whole data from its mirror node. Section 4 will further explain this in detail.

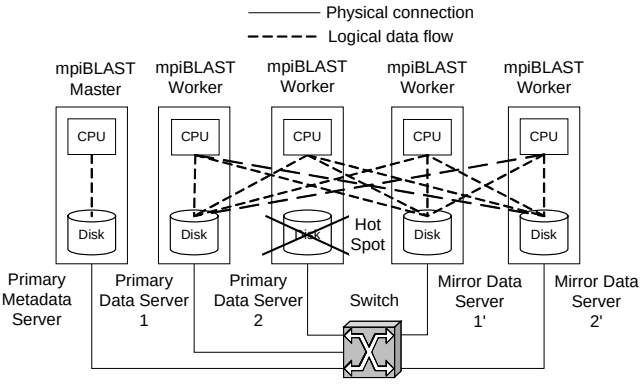

Figure 3. mpiBLAST over CEFT-PVFS with one hot spot disk skipped.

4. Performance Measurement and Evaluation

4.1 System Environments

In this work, we run the original mpiBLAST and its two new parallel-I/O based implementations on the PrairieFire cluster [14] at the University of Nebraska-Lincoln. At the time of our experiment, each cluster node was equipped with two AMD Athlon MP 2200+ processors, 2 GByte of RAM, a 2 gigabits/s full-duplex Myrinet [15] card, and a 20GB IDE (ATA100) hard drive. The Netperf [16] benchmark reports a TCP bandwidth over Myrinet of 126.51 MBytes/s with 47% CPU utilization. The disk write and read bandwidth is 32 and 26 MBytes/s respectively, as measured by Bonnie [17].

We used the sequence database *nt*, a nucleotide sequence database in non-redundant form, freely available for download at NCBI web site. Currently the *nt* database is the largest database available at NCBI and it has 1.76 million sequences, with a total file size of 2.7 GB. Our experiments are designed to model and measure the typical I/O behaviors of BLAST. Previous research has shown that the length of 90% of the query sequences used by biologists is within the range of 300-600 characters [13]. Thus in this work, a nucleotide sequence with a length of 568 characters, extracted from *ecoli.nt* database, is chosen as the query sequence.

We only performed the *blastn* search in our experiments, which compares a nucleotide sequence against a nucleotide database, although the experiments can also be carried out with other BLAST programs without any modification.

4.2 I/O Access Patterns

We instrumented the source code of NCBI BLAST library and collected the I/O traces at the application level. To eliminate the influence of the trace collection facilities on the completion time, this trace collection function is turned off during other measurements.

While the I/O access pattern of mpiBLAST constitutes both small and large reads with small writes, it is dominated by large reads with large variations in the temporal and spatial accesses. Figure 4 shows an example of the traces of the original mpiBLAST when 8 workers searched against 8 *nt* database fragments simultaneously. Among 144 I/O operations, 89% were reads ranging in data size from 13 bytes to 220 MB, with a mean of 31.29 MB. The remaining I/O operations were 16 small write operations for recording temporary results and synchronizing the multithreads on the same nodes. The data size for write operations has a minimum of 50 bytes and a maximum of 778 bytes, with a mean of 690 bytes.

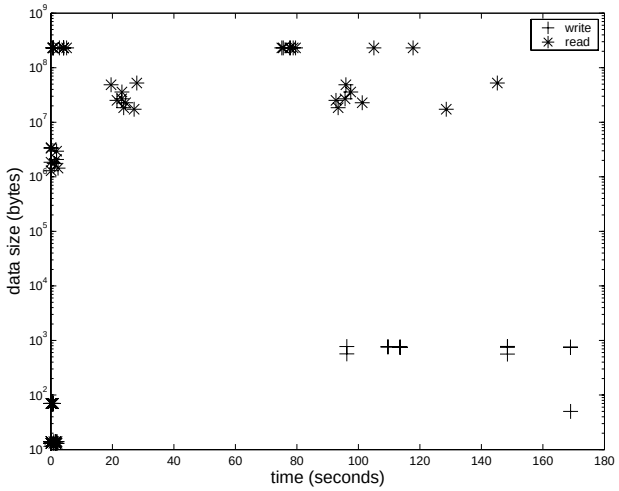

Figure 4. The overall I/O access pattern of mpiBLAST with 8 workers at the application level (master not included). A query sequence of 568 bytes from *ecoli.nt* is searched against the *nt* database with 8 fragments by using *blastn*.

4.3 Parallel I/O Comparisons

Recall that mpiBLAST utilizes the distributed storage devices and it needs to take two steps. Each worker first copies the database fragments to its local disks and then performs similarity searching independently. Since PVFS and CEFT-PVFS provide cluster-wide shared name space and mpiBLAST runs directly on them through parallel I/O interfaces, the worker does not need the copying procedure in the parallel I/O implementations. To fairly evaluate the parallel I/O performance, we measured the times for the database copying and subtracted the average copying time from the total execution time of the original mpiBLAST in each measurement.

The performance of the original mpiBLAST and mpiBLAST-over-PVFS is compared under the condition that they use the same number of cluster nodes. Since cluster nodes serve both as workers and as data servers in the mpiBLAST-over-PVFS, the amount of resources that the original and new mpiBLAST programs used is exactly identical. As Figure 5 shows, when the number of worker nodes is 1, mpiBLAST-over-PVFS performs worse than the original approach. This is not surprising since all the BLAST workers carry the overhead of an additional layer, the TCP/IP stack, that all data has to go through, and of the need for the workers (or clients in PVFS) to access the metadata server. When the number of cluster nodes increases to 2, PVFS starts to show advantage over the original mpiBLAST. Its advantage persists as the number of cluster nodes increases to 4 and 8. However, the amount of gains of PVFS over the original mpiBLAST

tends to be smaller as the number of cluster nodes becomes larger.

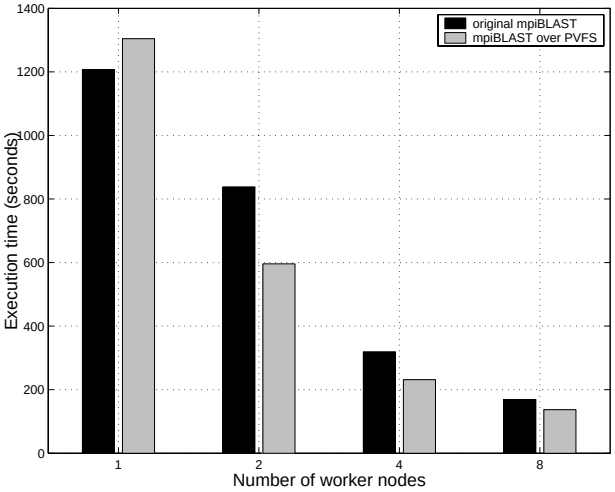

Figure 5. Performance comparison between original mpiBLAST and mpiBLAST-over-PVFS under the same amount resources. In mpiBLAST-over-PVFS, nodes are both data servers and mpiBLAST workers.

The performance of mpiBLAST-over-PVFS is also compared with that of the original mpiBLAST when allowing the number of data servers to vary. Figure 6 shows execution time with different number of workers (1, 2, 4, 8) and different number of PVFS data servers (1, 2, 4, 6, 8, 12, 16). Noticeably, PVFS does not perform as well as mpiBLAST under any of the four worker group sizes when it has only one data server. When the number of PVFS servers increases to 2, it outperforms the original mpiBLAST when the worker group size is 1, 2, and 4. The performance of PVFS improves further as the number of data servers becomes 4 and starts to show advantage over that of the original mpiBLAST under all four sizes of worker group. This advantage, while persistent, does not increase any further as the number of data servers continues to increase. In fact, the performance of mpiBLAST-over-PVFS does not consistently follow a continuous growth curve. When the number of data servers changes from 12 to 16, PVFS performance has no significant gain or even slight deterioration. We observed that the utilization of CPU on the worker node is kept close to 99% most of the time and the I/O time only occupy a very small portion of the overall execution time when the number of data servers is large, suggesting that computations (for sequence comparison), as opposed to I/O accesses, have become the absolute dominating factor in execution time. For example, when the number of worker nodes was 2, the time spent on I/O operations was measured to be around 11% of the total execution time on one worker node when running the original mpiBLAST. This, according to Amdahl's Law, implies that improving the I/O performance will have little influence on the overall execution time. Although we used the largest database available at NCBI during our experiments, its size is only serveral GBs, only twice or three times larger than the size of the RAM on any server node of the cluster used for the experiments. With the rapid increase of the biological database, it is highly likely that when the size of the database is in the order of hundreds of GBs or several TBs, the performance gain due to the increase of the number of data servers will be much more significant.

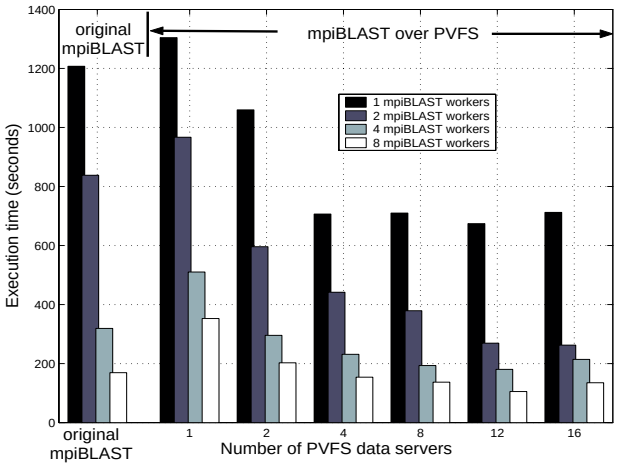

Figure 6. Performance comparisons between the original mpiBLAST and the mpiBLAST-over-PVFS under different number of workers with multiple server configurations.

4.4 Comparisons between PVFS and CEFT-PVFS

In the previous section, the experimental measurements of mpiBLAST-over-PVFS clearly showed the efficiency of parallel I/O. In this section, we compare the performances of PVFS and CEFT-PVFS. Recall that mpiBLAST is a read-dominated application, thus the performance can be significantly influenced by read performance of the I/O subsystem. In CEFT-PVFS, the read operations are designed to read the partitioned data both from the primary group and mirror group simultaneously. Thus when using the same number of cluster nodes as data servers, CEFT-PVFS and PVFS have the same degree of parallelism for read operations. In Figure 7, the PVFS is configured with 8 data servers while CEFT-PVFS is configured with 4 mirroring 4 data servers. In these measurements, all these cluster nodes are dedicated and there is no other application running on them. The performance of mpiBLAST-over-CEFT-PVFS is slightly worse than the mpiBLAST-over-PVFS. This

performance degradation is acceptable since CEFT-PVFS needs to manage slightly larger amount of metadata.

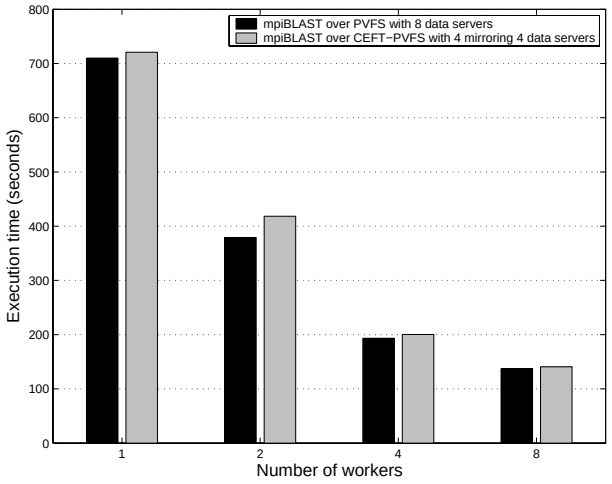

Figure 7. Performance comparisons between the mpiBLAST over CEFT-PVFS and the mpiBLAST over PVFS when the total number of data servers is 8 and the number of mpiBLAST workers varies.

4.5 Avoiding Hot Spot in CEFT-PVFS

As an integral part of a cluster, all the server nodes usually serve as computational nodes, too. The system resources of these nodes, such as CPU, memory, disk and network, can be heavily stressed by different scientific applications running on these nodes, thus potentially degrading the overall I/O performance of the parallel file system. This degradation cannot be avoided in the original PVFS since there is only one copy of any data stored in the system. However, in the CEFT-PVFS, where each piece of desired data is eventually stored on two different nodes, the redundancy provides an opportunity for the clients to skip the hot-spot node that is heavily loaded and read the target data from its mirroring node. This is not only possible for a single-node hot spot, but also possible for multi-node hot spots as long as no two nodes of any mirroring pair become hot spots.

In our experiments, we artificially stress the disk on one data server by a simple program, shown in Figure 8, to simulate a scenario where other I/O-intensive applications sharing the same node are running simultaneously (thus overloading the local disk). In this program, the synchronous write is guaranteed to always have a disk access. As we have measured, when only this program is running, both CPUs on the stressed node are nearly 95% idle and therefore will likely have little or no negative impact on the write performance. All three mpiBLAST implementations were executed under exactly the same workload, namely, the disk of one of the server nodes was artificially stressed using the program in Figure 8 while the rest of the nodes were evenly loaded by the mpiBLAST programs.

```
1.  M = allocate(1 MBytes);
2.  Create a file named F;
3.  While(1)
4.      If(size(F) > 2 GB)
5.          Truncate F to zero byte;
6.      Else
7.          Synchronously append the
8.          data in M to the end of F;
9.      End of if
10.End of while
```

Figure 8. Program to stress the disks

Figure 9 shows the performance results of the three implementations, with and without disk stressing, respectively. In CEFT-PVFS, all the clients skipped the hot spot node whose disk was being stressed by using our program and read all the desired data from its mirror node. While the performance of the original mpiBLAST and mpiBLAST-over-PVFS degraded by a factor of 10 and 21, respectively, under the disk stress, the mpiBLAST-over-CEFT-PVFS degraded only by a factor of 2.

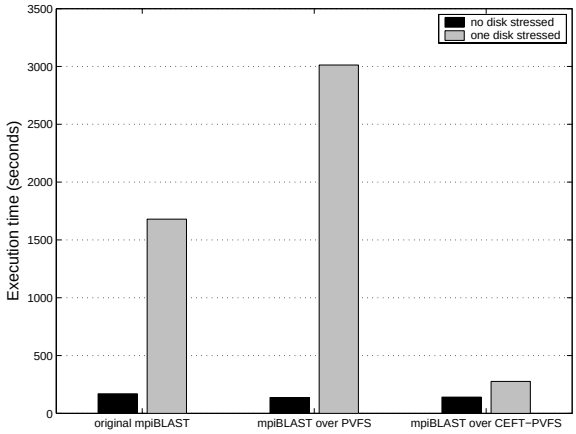

Figure 9. Performance comparisons between the original mpiBLAST, the mpiBLAST over PVFS and the mpiBLAST over CEFT-PVFS under 8 data servers and 8 mpiBLAST workers when stressing the disk on one node

5. Related Work

In the field of parallel I/O, most applications use micro-benchmarks to evaluate the overall performance of parallel I/O system [3][6][18][19][20][21][22][23]. In these benchmarks, multiple clients write or read simultaneously in a simple pattern. They aim to measure

the peak I/O performance. These benchmarks provide useful insights into the parallel I/O systems. However, in real scientific applications, the I/O access patterns are much more complicated and these applications rarely achieve the peak performance.

Ross et al. [24] studied the I/O characteristics of the FLASH astrophysics, a parallel scientific application on Linux clusters and simulated its checkpoint and plotfile that were basically write-only operations. One important observation made in that study suggests that the existence of recursion in the interfaces of parallel file systems substantially degrades the overall I/O performance, pointing to the extreme importance of the efficiency of high-level I/O interfaces.

Li et al. [25] studied the I/O behavior of an AMR cosmology application and compared the performance of MPI-IO with parallel HDF version 5. They discussed the advantages and disadvantages of both systems and concluded that there was a mismatch between the user access pattern and the file distribution and striping pattern. Some of their experiments are conducted on a cluster connected by a slow communication network that limited the scope of their observation.

Purakayastha et al. [26] measured file system workloads of some scientific applications on MPPs, particularly on the CM-5 architecture. They observed that small I/O requests dominated I/O operations, that the write traffic was consistently higher than read traffic, and that files were not shared between jobs. Smirni et al. [27][28] showed that there were significant variations in temporal and spatial I/O patterns across applications. They optimized the I/O performance by applying qualitative access pattern classification based on trained neural networks and hidden Markov models, flexible policy selection using fuzzy techniques and adaptive storage formats based on redundant representations [29]. Bennett et al. [30] improved the parallel I/O performance by using collective I/O that aggregates multiple I/O requests from different processors into one request.

6. Conclusions

In this paper, we study the I/O behavior of the parallel BLAST tools with three different I/O access schemes: 1. using conventional I/O interfaces on local disks, 2. using parallel I/O interfaces on PVFS, and 3. using parallel I/O interfaces on CEFT-PVFS. Based on our extensive experiments, we investigated the performance impacts of the degree of I/O parallelism and the contention of the I/O resource on parallel BLAST.

While the incorporation of the parallel I/O interfaces substantially improves the performance of parallel BLAST, we found that a higher degree of I/O parallelism may not lead to better performance, depending on whether I/O accesses remain dominant in execution time and how big the data set is. The performance of the parallel BLAST improves consistently in the initial growth in the number of data servers in PVFS. However, this improvement becomes insignificant when this growth continues beyond a certain level. The seemingly counter-intuitive result is due to the fact that the I/O time gradually becomes a very small portion of the overall execution time so that the improvement in I/O performance becomes very insignificant relative to the overall performance, a conclusion consistent with Amdahl's Law. In the case of CEFT-PVFS, doubling the degree of I/O parallelism for read operations provides a comparable read performance with respect to that of PVFS when the same number of data servers is used for both systems.

It is found that skipping the server node with a heavily loaded disk improves the parallel I/O performance in CEFT-PVFS. In a cluster, most nodes are typically time and space shared by multiple applications and thus the local disks of some nodes can be much more heavily loaded than those on others. The existence of the I/O resource contention can substantially deteriorate the performance of the original parallel BLAST and the one based on PVFS. While mirroring of disks provides data redundancy for fault tolerance in CEFT-PVFS, this redundancy can be exploited to improve the I/O performance by skipping one or more hot-spot server nodes and accessing the desired data from their mirror nodes. Our experiments showed that, with the existence of an artificially generated hot-spot data server, the parallel BLAST-over-CEFT-PVFS greatly outperformed the original parallel BLAST and the one based-on PVFS.

In this work, we only examined the impact of I/O resource contention. Clearly, the load conditions of the memory, network and CPU can also influence the I/O performance. We will further study the impact of contention of these resources in related ongoing work.

7. Acknowledgment

This work is supported by an NSF Grant (EPS-0091900), a Nebraska University Foundation Grant (26-0511-0019) and an Academic Priority Grant of University of Nebraska – Lincoln (UNL). Work was completed using the Research Computing Facility at UNL. We are grateful to our anonymous reviewers.

REFERENCES

[1] E. M. Zdobnov, R. Lopez, R. Apweiler, and T. Etzold, "The EBI SRS server – new features," *Bioinformatics*, 18(8):1149-1150, Aug. 2002.

[2] National Center for Biotechnology Information (NCBI), ftp://ftp.ncbi.nih.gov/, 2003.

[3] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in

Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, Oct. 2000, pp. 317-327.

[4] W. B. Ligon III, "Research directions in parallel I/O for clusters," *keynote speech*, in *Proceedings of 2002 IEEE International Conference on Cluster Computing*, Sept. 2002.

[5] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, "Design, implementation, and performance evaluation of a cost-effective fault-tolerant parallel virtual file system," *International Workshop on Storage Network Architecture and Parallel I/Os, in conjunctions with 12th IEEE International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, LA, Sept. 2003.

[6] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, "Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS)," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Parallel I/O in Cluster Computing and Computational Grids Workshop*, May 2003, pp. 730-735.

[7] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, "Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS)," in *Proceedings of Cluster World Conference & Expo*, San Jose, California, June 2003.

[8] S. F. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, 215:403-410, 1990.

[9] S. F. Altschul, T.L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res.*, 25:3389-3402, 1997.

[10] A. E. Darling, L. Carey, and W. Feng, "The design, implementation, and evaluation of mpiBLAST," in *Proceedings of Cluster World Conference & Expo*, San Jose, California, June 2003.

[11] R. Bjornson, A. Sherman, S. Weston, N. Willard, and J. Wing, "Turboblast: A parallel implementation of blast based on the turbohub process integration architecture," in *IPDPS 2002 Workshops*, April 2002.

[12] WU-BLAST, http://blast.wustl.edu/, 2003.

[13] K. T. Pedretti, T. L. Casavant, R. C. Braun, T. E. Scheetz, C. L. Birkett, and C. A. Roberts, "Three complementary approaches to parallelization of local BLAST service on workstation clusters," in *Proceeding of Fifth International Conference on Parallel Computing Technologies*, 1999, *Springer Verlag in Lecture Notes in Computer Science Series*, Vol. 1662.

[14] Prairiefire Cluster at University of Nebraska - Lincoln, http://rcf.unl.edu, 2003.

[15] Myrinet, http://www.myrinet.com, 2003.

[16] Netperf, http://www.netperf.org, 2003.

[17] Bonnie, http://www.textuality.com/bonnie, 2003.

[18] H. Taki and G. Utard, "MPI-IO on a parallel file system for cluster of workstations," in *Proceedings of the IEEE Computer Society International Workshop on Cluster Computing*, Melbourne, Australia, 1999, pp. 150-157.

[19] R. Cristaldi, G. Iannello, and F. Delfino, "The cluster file system: Integration of high performance communication and I/O in clusters," in *Proceeding of the 2nd IEEE/ACM international symposium on Cluster computing and the grid, Berlin*, Germany, May 2002.

[20] S. A. Moyer and V. S. Sunderam, "PIOUS: A scalable parallel I/O system for distributed computing environments," in *Proceedings of the Scalable High-Performance Computing Conference*, 1994, pp. 71-78.

[21] F. Isaila and W. F. Tichy, "Clusterfile: A flexible physical layout parallel file system," in *Proceedings of 2001 IEEE International Conference on Cluster Computing*, 2001, pp. 37-44.

[22] S. Garg and J. Mache, "Performance evaluation of parallel file systems for PC clusters and ASCI red," in *Proceedings of 2001 IEEE International Conference on Cluster Computing*, 2001, pp 172-177.

[23] M. Vilayannur, M. Kandemir, and A. Sivasubramaniam, "Kernel-level caching for optimizing I/O by exploiting inter-application data sharing," in *Proceedings of 2002 IEEE International Conference on Cluster Computing*, 2002, pp. 425-432.

[24] R. Ross, D. Nurmi, A. Cheng, and M. Zingale, "A case study in application I/O on Linux clusters," in *Proceeding of Supercomputer*, Denver, Nov. 2001.

[25] J. Li, W. Liao, A. Choudhary, and V. Taylor, "I/O analysis and optimization for an AMR cosmology application," in *Proceedings of IEEE International Conference on Cluster Computing*, pp. 119-126, 2002.

[26] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor," in *Proceedings of the Ninth International Parallel Processing Symposium*, April, 1995, pp. 165-172.

[27] E. Smirni and D. A. Reed, "Lessons from characterizing the input/output behavior of parallel scientific applications," *Performance Evaluation*, 1998, Vol. 33, pp. 27-44.

[28] E. Smirni and D. A. Reed, "Workload characterization of input/output intensive parallel applications," in *Proceedings of the Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Springer-Verlag Lecture Notes in Computer Science*, June 1997, Vol. 1245, pp. 169-180.

[29] H. Simitci and D. A. Reed, "A comparison of logical and physical parallel I/O patterns," *International Journal of High Performance Computing Applications, special issue (I/O in Parallel Applications)*, 1998, Vol. 12, No. 3, pp. 364-380.

[30] R. Bennett, K. Bryant, A. Sussman, R. Das, and J. S. Jovian, "A framework for optimizing parallel I/O," in *Proceedings of the Scalable Parallel Libraries Conference*, 1994.