CCF Transactions on High Performance Computing (2021) 3:242–251
https://doi.org/10.1007/s42514-021-00073-w

REGULAR PAPER



Improving bioinformatics applications performance via active storage systems

Zhiyang Ding[1] · Xiao Qin[2] · Shu Yin[3]

Received: 2 April 2021 / Accepted: 9 July 2021 / Published online: 26 August 2021
© China Computer Federation (CCF) 2021

Abstract
Since large-scale and data-intensive applications have been widely deployed, there is a growing demand for high-performance storage systems to support data-intensive applications. Compared with traditional storage systems, next-generation systems should embrace a dedicated processor to reduce the computational load of host machines and may have hybrid combinations of diverse storage devices. We present a pipelining technique for active storage systems and evaluate the design on a widely used bioinformatic application called pp-mpiBLAST. Experimental results indicate that the proposed technique can reduce overall execution time by up to 50% and retain better scalability of clusters.

Keywords Active storage systems · I/O parallelism · BLAST · Bioinformatics

1 Introduction

Processing massive amounts of data have resulted in a mushrooming of data-intensive applications like bioinformatics data processing. Evidence shows that the collective amount of genomic information doubles every 12 months (Heshan et al. 2010; Yang et al. 2014; Huang et al. 2017). Most bioinformatics applications have to deal with the I/O bottleneck issue. Processing huge datasets in a high-performance cluster normally require copying data from storage nodes to compute nodes, thereby leading to a large number of I/O operations (Xie et al. 2016). Active storage is an effective technique to improve applications' end-to-end performance by offloading data processing from compute nodes to storage nodes (Zhang et al. 2020).

Active storage brings three key advantages. First, the amount of data moved back and forth between compute nodes and storage nodes in clusters can be significantly reduced, since large datasets can be locally processed by storage nodes before being forwarded to compute nodes. Second, data-intensive applications run faster, because active storage nodes accelerate data processing operations. If compute nodes and active storage nodes efficiently coordinate, both compute nodes and storage nodes can perform data processing in parallel. Third, network performance in clusters can be improved due to reduced amounts of data moved into and out of storage nodes.

There are two main challenges in applying active storage to support data-intensive applications on clusters. The first challenge is to partition a parallel application into compute-intensive and data-intensive tasks. If such a partition is successfully created, compute nodes will handle computation-intensive tasks whereas active storage nodes will run data-intensive tasks. The second challenge lies in the coordination between compute nodes and active storage nodes. When it comes to applications where computation-intensive tasks are independent of data-intensive tasks, compute nodes and active storage nodes are non-blocking to each other, meaning that computing and storage nodes can easily operate in parallel. However, if compute nodes have to wait for storage nodes to catch up, the blocked compute nodes could slow down data-intensive applications.

In this paper, we address the partitioning and synchronization issues in the context of active storage supporting bioinformatics applications. To solve the blocking problem

✉ Shu Yin
yinshu@shanghaitech.edu.cn

Xiao Qin
xqin@auburn.edu

[1] State Development and Investment Corporation, Beijing, China

[2] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA

[3] School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China

incurred by synchronized computing and storage nodes, we developed a pipelining mechanism that exploits parallelism among data processing transactions in a sequential transaction stream. We report the effectiveness of the pipelining mechanism that leverages active storage to maximize the throughput of data-intensive applications on a high-performance cluster. To demonstrate the effectiveness of the pipelining mechanism designed for active storage, we implemented a pipelined application called pp-mpiBLAST, which extends mpiBLAST, which is an open-source parallel BLAST tool. pp-mpiBLAST deals with sequential data processing transactions, each of which contains a filtering/formatting task and a mpiBLAST task. The mechanism overlaps data filtering/formatting in active storage with parallel BLAST computations in compute nodes, thereby allowing clusters and their active storage nodes to perform data processing in parallel. The pp-mpiBLAST application relies on active storage to filter unnecessary data and to format databases, which are then forwarded to the cluster running mpiBLAST.

We develop an analytic model to study the scalability of pp-mpiBLAST on large-scale clusters. This model allows us to study the performance of pp-mpiBLAST on a cluster using active storage. This performance model is ideal for application developers who have limited computing resources to test the scalability of their parallel applications using active storage. Furthermore, programmers can use the analytic model to explore the design space related to the number of compute nodes, active storage speed, data processing capacity, and input data size. The model shows the behavior of pp-mpiBLAST under different configurations of the cluster coupled with active storage. Measurements made from a working implementation and a modeling study suggest that this method not only improves mpiBLAST's overall performance by up to 75%, but also achieves high scalability on clusters coupled with active storage.

The rest of the paper is organized as follows. Section 2 summarizes the related work. In Sect. 3, we describe the design and implementation details of the active storage node. The analytical model is presented in Sect. 4. Experiment results and performance evaluation are discussed in Sect. 5. Finally, Sect. 6 provides conclusions and future research directions.

2 Related work

Active storage: Current practice for data-intensive applications on high-performance clusters often results in high I/O communication overhead between compute nodes and storage nodes in the cluster. When massive amounts of data must be transferred back and forth between parallel compute nodes and storage systems, cluster computing applications' performance can suffer from network bandwidth saturation. One efficient approach to reducing network traffic caused by moving data between compute nodes and storage systems is to incorporating computing capacities into storage systems, thereby offloading some data-intensive computing tasks from clusters to their storage nodes. That inspired researchers to make the storage more "active" , or "smart" . Active storage can be implemented at storage node levels (Tang et al. 2004; Felix et al. 2006; Sivathanu et al. 2002). ASF Fitch et al. (2009) and FAWN Andersen et al. (2009) are two examples of active storage implementations at storage node levels. Fitch et al. developed the Active Storage Fabrics (ASF) model to address petascale data-intensive challenges (Andersen et al. 2009). ASF is aimed at transparently accelerating host workloads by close integration at the middleware data/storage boundary or directly by data-intensive applications (Andersen et al. 2009). FAWN - developed by Andersen et al. couples embedded CPUs to small amounts of local flash storage (Andersen et al. 2009). Andersen et al. used FAWN as a building block to construct a cluster, in which computation and I/O capabilities are balanced to improve the energy efficiency of the cluster running data-intensive parallel applications. Active storage has also been explored in terms of handling unstructured data (Chen and Chen 2012) and working in a Lustre system (Piernas et al. 2007). Many studies have explored data staging and caching to buffer data temporarily for performance optimizations of data access in the near future (Tian et al. 2012). A line of work has studied the efficient data reorganization for multidimensional data organization. Sarawagi et al. categorized the strategies for efficient organization of large multidimensional arrays (Sarawagi and Stonebraker 1994). Xu et al. proposed a large-scale object-based active storage platform for data analytics on the internet of things (Xu et al. 2016). Xie et al. provided an active storage framework for an object-based storage platform to run data-intensive applications locally (Xie et al. 2016). In this study, we focus on active storage like ASF and FAWN. We implemented the pipelining technique in an active storage testbed that is similar to ASF and FAWN.

Parallel computing can improve the performance of bioinformatics applications like sequence database search tools. Given a database and query sequences (e.g., DNA, amino-acid sequences), the search tools search for similarities between the query sequences and known sequences in the database. The tools enable scientists to quickly identify the function of newly discovered DNA sequences or to accurately identify species of a common ancestor (Lin et al. 2005). Among many sequence search tools, BLAST is one of the most popular tools used on daily basis by bioinformatic researchers. MpiBlast Hajibaba et al. (2019) is a promising, open-source, parallel implementation of the BLAST toolkit (Altschul et al. 1990). Like other bioinformatics

applications, mpiBlast has to pre-process (e.g., format data) databases before searching for similarities between query sequences and known sequences in the preprocessed databases.

We implement a pipelined bio-application called pp-mpiBlast to demonstrate a way of employing active storage to improve the performance of data-intensive bioinformatics applications. Our pp-mpiBlast incorporates a data-processing pipeline with mpiBlast on a high-performance cluster.

3 Design and implementation

In this section, we describe the system from the top down: an overview of the system, a hybrid mix of storage devices, and the parallel pipelined processing. Hereinafter, the active storage node is referred to as ASN for short.

3.1 Active storage for clusters

A typical high-performance cluster consists of compute nodes and storage nodes. Data-intensive applications on clusters can cause heavy I/O traffic between the computing and storage nodes in the cluster. To achieve high performance for data-intensive applications, we aim to reduce network traffic caused by moving massive amounts of data from/to storage systems in clusters. This goal can be accomplished by offloading data-intensive computations from compute nodes to active storage nodes.

Figure 1 illustrates a cluster that involves a collection of compute nodes coupled with active storage nodes. Storage nodes become active if they can handle application-level data processing offloaded from compute nodes. The rectangles in Active Storage Node represent processing units that handle application-level data. The squares represent data offloaded from the compute nodes.

The storage devices can be further divided into two categories: mass storage and buffer storage. Benefiting from the non-volatile memory store, Solid-state disk (SSD) is a popular option to fill the latency gap, which is around 5-order-of-magnitude, between main memory and spinning disks (He et al. 2010). Thus in our system, SSDs are used as buffer disk drives: large-scale data is moved from mass storage to buffer drives before processing. The results of experiments in Sect. 5 show that SSDs not only speed up the I/O but also provide a better scalability performance. The advantage of using the hybrid mix of both the solid-state disk and the magnetic hard disk is mutual complementarity: the fast and expensive cooperates with the large-capacity and cost-efficiency.

In this paper, we use a commodity computer as the compute end. The "channel" is formed by the compute nodes of a cluster and the ASN is considered as a pipeline or assembly line. In other words, by applying the active storage, applications containing multiple stages are capable to extend to a parallel pipelined implementation. The exploration of parallelism improves the performance of data-intensive applications. As a case study, we extend the mpiBlast, a well-known parallel BLAST application, to a parallel pipelined implementation, called pp-mpiBlast.

3.2 Parallel pipelined system

Native mpiBlast application can be considered as two steps: format the raw database file (corresponding to the query request) and run the parallel BLAST functions to do the comparison. Thus, the pre-cook (i.e. format) phase and the parallel computation phase are handled by the ASN and compute nodes, respectively.

How to utilize the active storage device is another critical issue. In general, all cases can fall into two scenarios. The first is that tasks are independent (i.e. compute nodes and

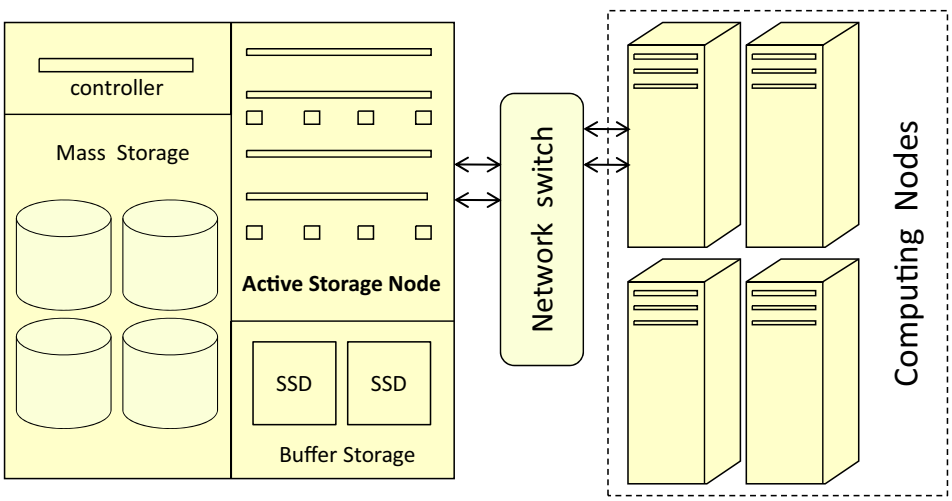

Fig. 1 A cluster consists of compute nodes and active storage nodes

active storage nodes are non-blocking to each other), meaning that computing and storage nodes can operate in parallel. However, if compute nodes have to wait for storage nodes to catch up, the blocked compute nodes can slow down data-intensive applications. For instance, in terms of mpiBlast, the parallel comparison step requires the formatted database file from the previous step. The assembly line pattern is one of the solutions for the second scenario.

We extend the mpiBlast to a parallel pipelined implementation (hereinafter referred to as pp-mpiBlast). The pp-mpiBlast system consists of two tasks: (1) raw database formatting, and (2) genome or protein sequences comparison. The first task can be offloaded to the active storage node, where prepare the formatted database file for partitioning from the previous step. The second task will be performed on the compute nodes as it requires advanced computing resources for fast sequence comparison. Original pipeline patterns can be divided into inter-and intra-application pipeline processing while pp- mpiBlast is in charge of intra-application parallel processing. As the name "intra-" suggests, one native sequential transaction is partitioned into multiple parallelled pipelined transactions. The system performance is improved by fully exploiting the parallelism. The workflow of pp-mpiBlast is depicted in Fig. 2.

3.3 Intra-application pipeline processing

As mentioned in the previous section, to extend a sequential transaction to multiple pipelined parallel transactions, both partition and merge functions are introduced in the pp-mpiBlast system. Figure 3 illustrates a two-staged pipeline processing workflow for a two-task scenario. The figure shows that the pipeline pattern not only improves the performance by exploiting the parallelism but also solves the out-of-core processing issue, which requires a larger amount of data that can not fit in the ASN's main memory. In pp-mpiBlast, the partition function is implemented within `mpiformatdb` function running on ASN, while the merge function is a separate one running on the front node of the cluster.

When partitioning the source data, an assistant function called integrity-check automatically returns the additional displacements by scanning the return or the symbol defined by the programmer. The reason we integrate the integrity-check procedure to the partition function is that there exists the consistency issue of partitioned data files; the content of the source data file could be broken in shatters (e.g. a sequence could be cut and placed into two split fragments, not on purpose). Figure 4 describes the integrity-check workflow.

Pipeline parallelism is an important processing pattern and we are interested in providing models and guidance for tuning the scalability and the performance using this pattern. In Sect. 4, we develop a mathematical analysis model.

3.4 Preliminary result

To prove the feasibility of the partition-based intra-application pipeline design, a preliminary test on a single-node with a 2GB memory environment is performed. We extended Word-Count and String-Match benchmarks of Phoenix system (Ranger et al. 2007), which is a shared-memory

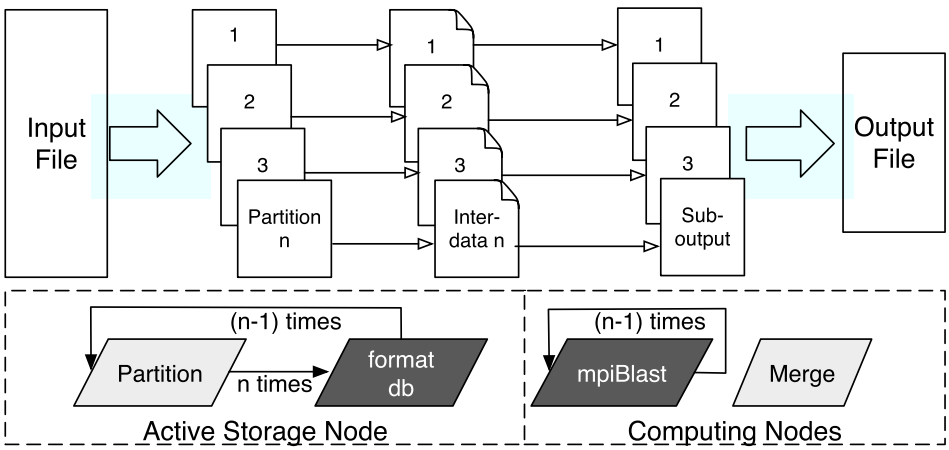

Fig. 2 pp-mpiBlast Workflow

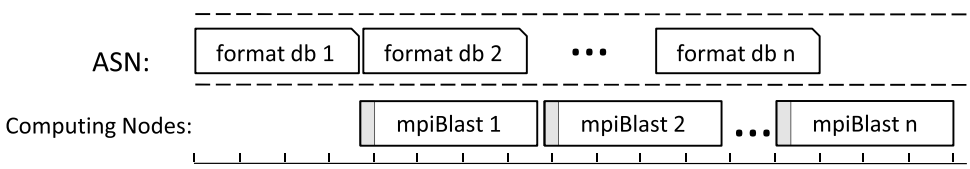

Fig. 3 Pipeline tasks scheduling

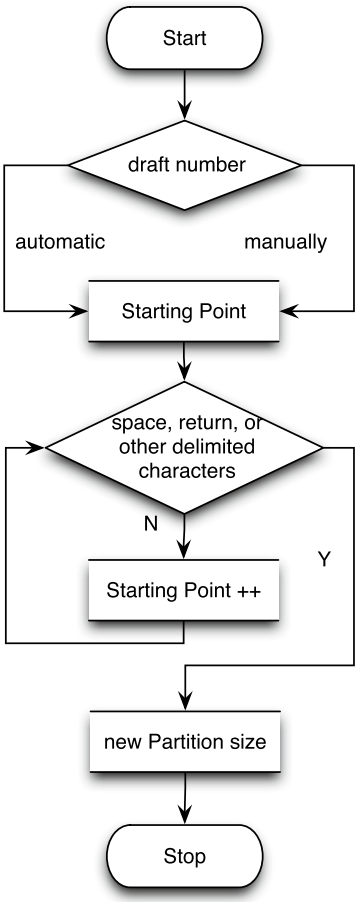

Fig. 4 The Workflow diagram of sequence-integrity checking

implementation of MapReduce, to intra-application pipeline editions using partition and merge runtime functions we developed under the Phoenix system. Figure 5 depicts the workflow of the extended approach. After the input data is partitioned into fragments in size of 600 MB, each of them is processed sequentially using native word-count or string-match applications. And then, the generated sub-results are merged at the end. The control test is that we run the native applications to process the input data without the partition and merge functions.

Table 1 shows the results. We can observe that in terms of data-intensive, especially memory-intensive, applications, partitioning can significantly reduce the running time. For example, in terms of word-count application results, an average 2.4x speedup of time consumption can be achieved. On the contrary, partitioning does not benefit the speed of the string-match application. But, it can make the large-scale data-intensive applications run- ning on limited memory machines. For example, when executing the string-match application without the help of the partition function, the native system does not support the case that the input data size is more than two times the local memory size. Thus, the preliminary results show that the partition-enabled design can (1) improve data-intensive applications' performance, (2) adapt the data-intensive applications to limited memory machines or both.

Fig. 5 The Workflow of the extended Phoenix model

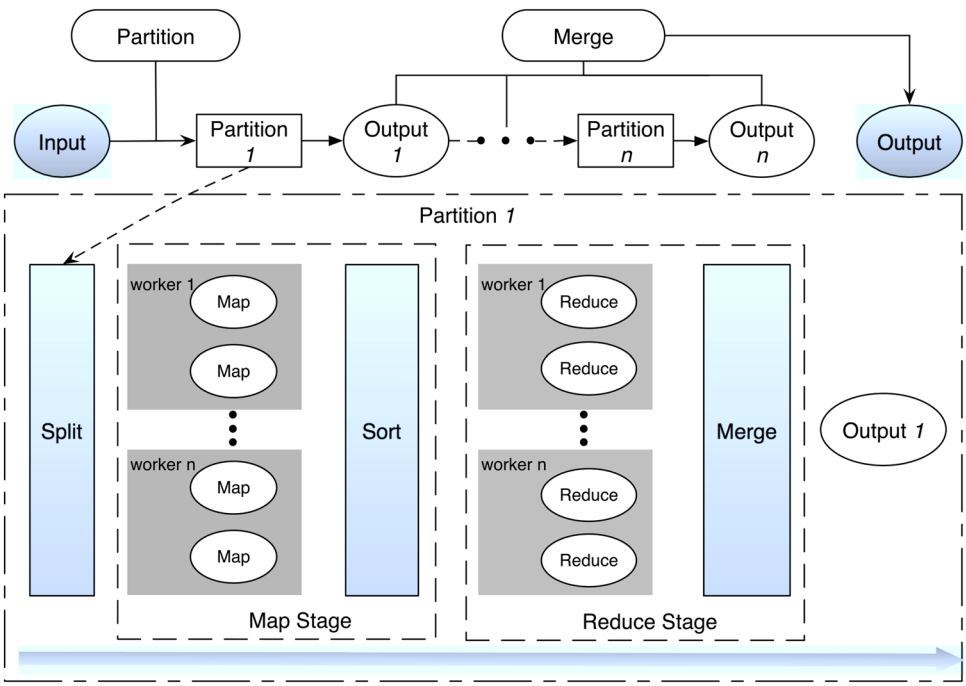

Table 1 Execution time of Word-Count and String-Match benchmarks with and without partition function under two input data sizes (in GB) on a single node. The testbed machine is equipped with 2 GB main memory

Benchmarks	Word count	Word count	String match	String match
Data Size	1GB	1.25GB	1GB	1.25GB
w/ partition	40.50s	50.91s	17.76s	20.61s
w/o partition	85.71	139.54s	17.62s	21.00s

4 Modeling and analysis

We develop in this section an analytic model to study the performance and scalability of the parallel pipelined BLAST on large-scale cluster computing platforms. The analytic model has the following three key advantages:

- **Performance evaluation:** The model allows us to study the performance (i.e., speedup and throughput) of parallel pipelined BLAST system with active storage.
- **Scalability analysis:** The performance model is ideal and suitable for bioinformatics application developers who have limited local computing resources to test the scalability of their parallel applications using active storage.
- **Design-space exploration:** Application programmers can use the model to explore the design space related to the number of compute nodes, active storage speed, data processing capacity, and input data size. The model shows the behavior of the parallel pipelined BLAST under different configurations of a high-performance cluster coupled with active storage.

Section 3 describes the parallel pipelined implementation of a basic local alignment search tool or BLAST. Our pipelined BLAST has two data processing stages: (1) formatting data and (2) retrieving and processing formatted data. The first stage of the pipeline pre-processing is the input of a data set before passing on to the second stage that runs mpiBLAST - a parallel implementation of BLAST.

Response time, speedup, and throughput are three critical performance measures for the pipelined BLAST. Denoting T_1 and T_2 as the execution times associated with the first stage and second stage in the pipeline, we can calculate the response time $T_{response}$ for processing each input data set as the sum of T_1 and T_2. Thus, we have

$$T_{response} = T_1 + T_2 \tag{1}$$

The throughput (see Eq. 2) of the two-stage pipelined system is inversely proportional to the maximum of the two execution times T_1 and T_2.

$$Throughput = \frac{1}{max(T_1, T_2)} \tag{2}$$

The speedup for the pipelined BLAST is:

$$Speedup = \frac{T_{un-pipelined}}{T_{pipelined}} \tag{3}$$

where $T_{un-pipelined}$ is the data processing time for the un-pipelined BLAST and $T_{pipelined}$ is the processing time of the pipelined BLAST. If n is the number of input data sets to be processed by a cluster with active storage, then processing time of the un-pipelined BLAST is the product of n and $T_{response}$ (see Eq. 1), leading to

$$T_{un-pipelined} = n \times T_{response} = n \times (T_1 + T_2) \tag{4}$$

The processing time for the pipelined BLAST is:

$$\begin{aligned} T_{pipelined} &= T_1 + (n-1) \times max(T_1, T_2) + T_2 \\ &= T_{response} + (n-1) \times max(T_1, T_2) \end{aligned} \tag{5}$$

where T_1 is the processing time of stage 1 for the first data set, T_2 is the execution time of stage 2 for nth data set, $(n-1) \times max(T_1, T_2)$ is the time spent on $(n-1)$ data sets when the two stages are carried out is parallel. Applying Eqs. 4 and 5 to Eq. 3, we obtain speedup as:

$$Speedup = \frac{n}{1 + (n-1) \times \frac{max(T_1, T_2)}{T_{response}}} \tag{6}$$

Now we are positioned to model execution times T1 and T2 for the two stages in the pipeline. The processing time of the first stage is the sum of (1) data input/output times and (2) filtering/formatting time $T_{1,comp}$. Input time is proportional to unformatted input data size s_u and inversely proportional to disk read bandwidth b_i. Similarly, output time is proportional to formatted output data size s_f and inversely proportional to disk write bandwidth b_o. This leads to:

$$T_1(s_u, s_f, b_i, b_o) = \frac{s_u}{b_i} + T_{1,comp}(s_u) + \frac{s_f}{b_o}. \tag{7}$$

The execution time T_2 of stage two is the sum of (1) input time of formatted data and (2) processing time $T_{2,comp}$. The input time depends on data size s_f, disk input bandwidth b_i, and the number m of compute nodes in a cluster. Assuming that the formatted data size s_f is uniformed distributed among the m compute node, we can express the input data as $\frac{s_f}{m \times b_i}$. Thus, the execution time T_2 for the second stage is given below:

$$T_2(s_f, b_i, m) = \frac{s_u}{m \times b_i} + T_{2,comp}(s_f, m) \tag{8}$$

where $T_{2,comp}(s_f, m)$ is affected by the formatted data size s_f and the cluster size (i.e., number of compute nodes m).

5 Evaluations

5.1 Evaluation environment

We implemented the pp-mpiBlast in a 14-node cluster (1 node works as ASN), whose configuration is outlined in Table 2. The nodes in the cluster are connected by Ethernet adapters, Ethernet cables, and one 1Gbit switch. We choose an Intel X-25M 80GB solid-state disk and a SATA Raid tower with four WD50000AAKS disks as a RAID 0 array. The *MPICH2-1.0.7* is chosen as the message passing interface (MPI) in the cluster. The pp-mpiBlast is extended from mpiBlast-1.5.0, in which NCBI Blast 2.2.24 is the comparison tool. All applications are compiled with *gcc 4.4.1*.

5.2 Single node evaluation

We perform *mpiformatdb* program under different storage disk schemes: w/ SSD as the buffer disk and w/o buffer disk. Since the data pre-fetching is out of the scope of this chapter, when we test the SSD cases, we modify the program to move the data from mass storage devices to the SSD and then trigger the format function. That means the time consumption of SSD contains both the data-transfer and data-format phases. Table 3 shows the results. As observed from the table, the case of SSD (e.g.the second row) outperforms the HDD since it benefits from a large number of random reads and writes when the function is reordering the sequence in a descending order based on entry length. After balancing the disk capacity, storage reliability, I/O speed, and random w/r speed, the hybrid mix of mass storage and buffer disk is a promising choice. Figure 6 shows the comparison of trends of using SSD and HDD. We can observe that the scalability of using HDD for *mpiformatdb* function is not good, compared with the SSD one. Thus in the following experiments, we use the SSD as the buffer disk.

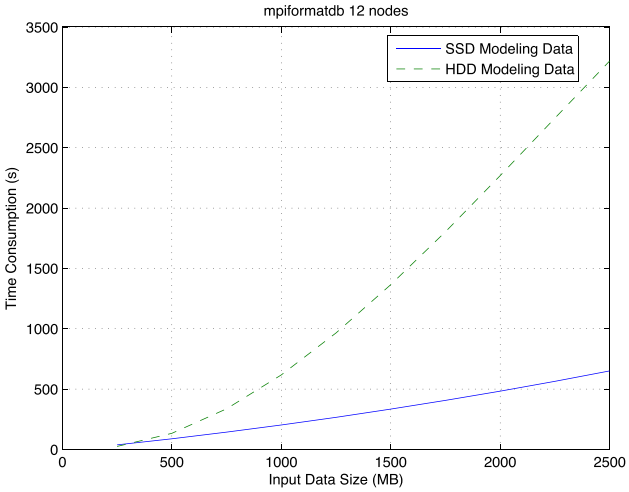

Fig. 6 Time consumption trends comparison: SSD vs. HDD

5.3 System performance evaluation

4.5.3 System Performance Evaluation Fig. 7 shows the system performance evaluation. The pp-mpiBlast testbed, which is configured by 12 compute nodes and 1 ASN (follows the pipelined processing pattern), is compared with two control experiments (native systems with a different number of nodes): the native mpiBlast running on 1) 12 nodes cluster (equals to the number of compute nodes in pp-mpiBlast testbed), and 2) 13 nodes cluster (equals to the total number of nodes in pp-mpiBlast testbed). The reason to choose two competitors is to present the performance improvement comprehensively.

In Figs. 7 and 8, the testbed with native mpiBlast contains 12 and 13 nodes, respectively. In each figure, results generated by pp-mpiBlast are using 3 different partition sizes: 250 MB, 500 MB, and 750 MB, which are presented by three sub-figures, respectively from left to right. Observed from the figures, the time consumption comparison results show that the performance of pp-mpiBlast beat both control testbeds: averagely reducing the execution time by 50% (i.e. 2x speedup). And Fig. 9 shows that: the improvement is greater when the input data size increase within a certain range of the input size, which is relative to the main memory size.

We also compare the performance in terms of the different partition sizes. Figure 10 presents that the 500 MB

Table 2 Experiment platform

	Compute node	ASN
CPU	Intel Xeon X3430	Intel Q9400
Memory	2GB	2GB
OS	Ubuntu 9.04	Ubuntu 9.04
Kernel Version	2.6.28-15	2.6.28-15
Network	1Gbps Ethernet	1Gbps Ethernet

Table 3 Time cost (in seconds) of performing mpiformatdb program under different input data size (in MB) on an ASN

Data size	500 MB	750 MB	1 GB	1.25 GB	1.5 GB	1.75 GB
HDD	108.4s	369.5s	639.1s	945.6s	1,385.0	1,845.1 s
SSD	101.9s	164.4s	225.9s	291.1s	369.5s	441.1 s

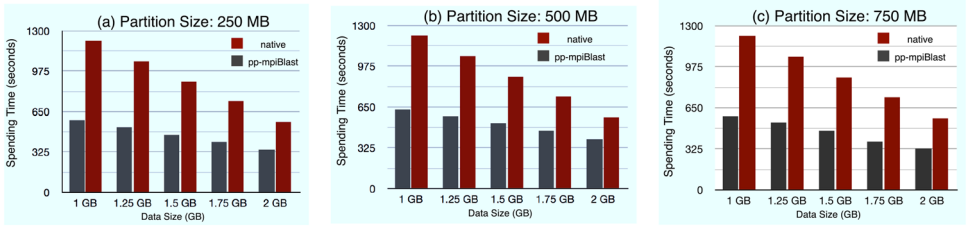

Fig. 7 Execution time comparison between pp-mpiBlast system and native system (running on a computer cluster of 12 nodes). The pp-mpiBlast system contains a twelve nodes computing cluster and one ASN. Results are generated under 3 different partition sizes: 250 MB, 500 MB, and 1.25 GB, which are presented by three sub- figures, respectively from left to right

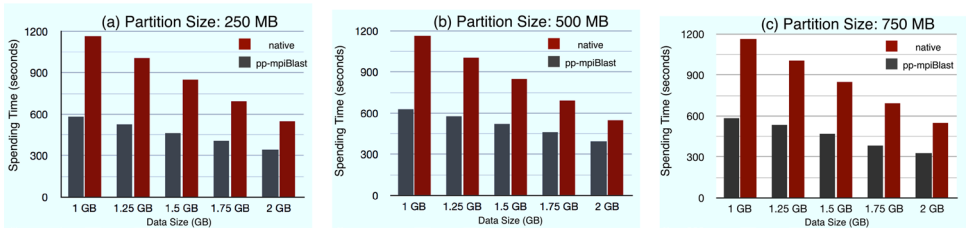

Fig. 8 Execution time comparison between pp-mpiBlast system and native system (running on a computer cluster of 13 nodes). The pp-mpiBlast system contains a twelve nodes computing cluster and one ASN. Results are generated under 3 different partition sizes: 250 MB, 500 MB, and 1.25 GB, which are presented by three sub- figures, respectively from left to right

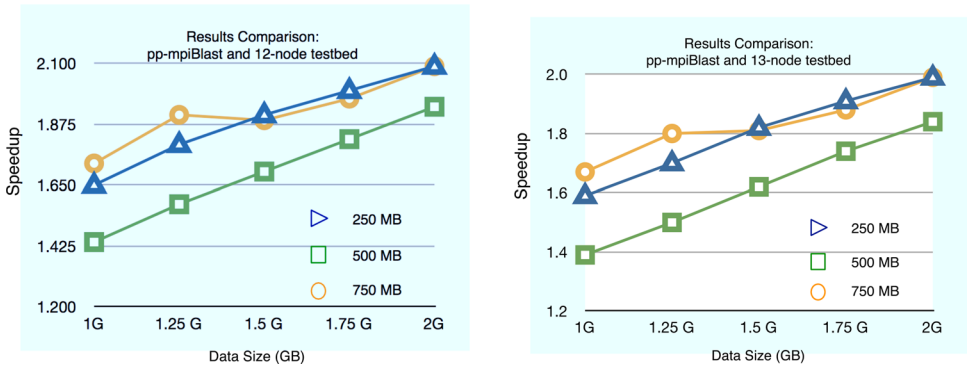

Fig. 9 Speedup trends: As input data size grows larger, the performance speedups of using pp-mpiBlast increase. The sub-figure on left is the comparison result between pp-mpiBlast and the 12-node testbed. And the right one is the result compared with the 13-node testbed

partition provides better performance in general. Based on the data, the reason can be summarized as follows: (1) The smaller the better is not true because small partitions always generate more overheads. And (2) the larger the better is also not convinced since then. It means that a partition-size-threshold exists for optimal performance. The issue of measuring the threshold in quantity will be dug into our future work.

Based on the test results, we can see that using an intra-application pipeline parallel processing model to extend mpiBlast improves performance and scalability. However, as we mentioned in the previous sections, the approach is not general; it requires that the target application can be decomposed into stages, such as streaming and RMS applications. Also, applications parallelized using pipeline models are very sensitive to load balancing. To avoid bubbles or reduce their side effect, how to balance the heterogeneity issue between the ASN and compute nodes will be our next topic.

6 Conclusion

We have presented a pipelining technique that allows compute nodes and active storage nodes in a cluster to perform CPU-intensive and data-intensive computing in parallel. The technique exploits parallelism among data processing transactions in a sequential transaction stream, where each transaction is sequentially handled by computing and storage nodes in the cluster.

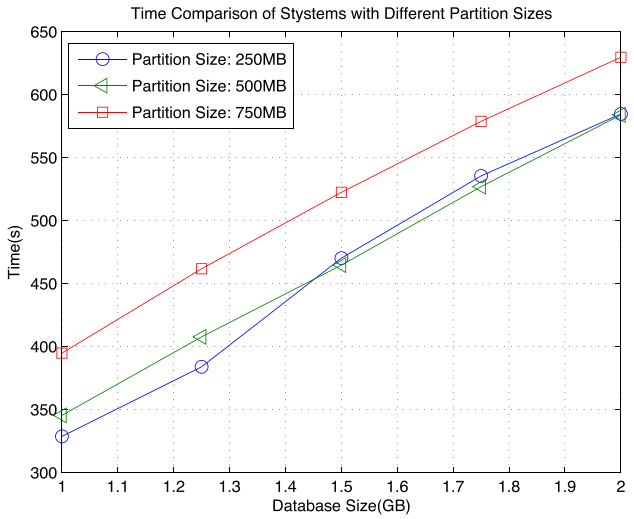

Fig. 10 Time consumption curves comparison: different partition size

One central trait of the pipelining technique is that it makes use of active storage to maximize the throughput of data-intensive applications (e.g., bioinformatic applications) on high-performance clusters. We show that the synchronizations among computing and storage nodes can hinder active storage nodes from accelerating data processing operations. Our technique solves the blocking problem that prevents active storage and compute nodes from processing data in parallel. The implementation of pp-mpiBLAST demonstrates that active storage is an effective approach to improving data-intensive applications' overall performance by offloading data processing to storage nodes in clusters.

We implemented the pipelining technique in a bioinformatic application called pp-mpiBLAST, which incorporates mpiBLAST—an open-source parallel BLAST tool. The pipeline in pp-mpiBLAST processes a sequential transaction stream, in which each transaction contains a filtering/formatting task and a mpiBLAST task. The pipelining technique overlaps data filtering/formatting in active storage with parallel BLAST computations in compute nodes. Measurements made from a working implementation and analytic model have been encouraging. The pipeline and active storage not only reduce mpiBLAST's overall execution time by up to 50%, but they also achieved high scalability on large-scale clusters.

There are three directions to pursue future work. First, we intend to explore the extensibility of data-processing modules and operations that are preloaded into active storage. The modules and operations are likely to be application domain-specific. Second, we will study the parallelisms among multiple active storage nodes. Third, we intend to investigate whether similar partitioning and synchronization schemes can be applied to both read-intensive and write-intensive applications.

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990)

Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V.: Fawn: A fast array of wimpy nodes. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 1–14 (2009)

Chen, C., Chen, Y.: Dynamic active storage for high performance i/o. In: 2012 41st International Conference on Parallel Processing, pp. 379–388. IEEE (2012)

Felix, E.J., Fox, K., Regimbal, K., Nieplocha, J.: Active storage processing in a parallel file system. In: Proceedings of the 6th LCI International Conference on Linux Clusters: The HPC Revolution, pp. 85 (2006)

Fitch, B.G., Rayshubskiy, A., Pitman, M.C., Christopher Ward, T.J., Germain, R.S.: Using the active storage fabrics model to address petascale storage challenges. In: Proceedings of the 4th Annual Workshop on Petascale Data Storage, pp. 47–54 (2009)

Hajibaba, M., Sharifi, M., Gorgin, S.: Data-parallel computational model for next generation sequencing on commodity clusters. In: International Conference on Parallel Computing Technologies, pp. 273–288. Springer, New York (2019)

He, J., Bennett, J., Snavely, A.: Dash-io: an empirical study of flash-based io for hpc. In: Proceedings of the 2010 TeraGrid Conference, pp. 1–8 (2010)

Heshan, L., Xiaosong, M., Wuchun, F., Nagiza, F.S.: Coordinating computation and i/o in massively parallel sequence search. IEEE Trans. Parallel Distrib. Syst. **22**(4), 529–543 (2010)

Dan Huang, Dezhi Han, Jun Wang, Jiangling Yin, Xunchao Chen, Xuhong Zhang, Jian Zhou, and Mao Ye. Achieving load balance for parallel data access on distributed file systems. IEEE Transactions on Computers, 67(3), 388–402, 2017

Lin, H., Ma, X., Chandramohan, P., Geist, A., Samatova, N.: Efficient data access for parallel blast. In: 19th IEEE International Parallel and Distributed Processing Symposium, pp. 10, IEEE (2005)

Piernas, J., Nieplocha, J., Felix, E.J.: Evaluation of active storage strategies for the lustre parallel file system. In: SC'07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, pp. 1–10. IEEE (2007)

Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating mapreduce for multi-core and multiprocessor systems. In: *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24. Ieee (2007)

Sarawagi, S., Stonebraker, M.: Efficient organization of large multidimensional arrays. In: Proceedings of the Tenth International Conference on Data Engineering, pp. 328–336, Washington, DC (1994). IEEE Computer Society

Sivathanu, M., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Evolving rpc for active storage. In: *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pp. 264–276 (2002)

Tang, H., Gulbeden, A., Zhou, J., Strathearn, W., Yang, T., Chu, L.: The panasas activescale storage cluster-delivering scalable high bandwidth storage. In: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, pp. 53–53. IEEE (2004)

Tian, Y., Klasky, S., Yu, W., Abbasi, H., Wang, B., Podhorszki, N., Grout, R., Wolf, M.: Smart-io: System-aware two-level data organization for efficient scientific analytics. In: 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 181–188 (2012)

Xie, Y., Feng, D., Li, Y., Long, D.D.E.: Oasis: an active storage framework for object storage platform. Fut. Gen. Comput. Syst. **56**(Supplement C), 746–758 (2016)

Xu, Q., Aung, K.M., Zhu, Y., Yong, K.L.: Building a large-scale object-based active storage platform for data analytics in the internet of things. J. Supercomput. **72**(7), 2796–2814 (2016)

Yang, X., Yin, Y., Jin, H., Sun, X.-H.: Scaler: Scalable parallel file write in hdfs. In: *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 203–211. IEEE (2014)

Zhang, J., Xie, T., Jing, Y., Song, Y., Hu, G., Chen, S., Yin, S.: Bora: a bag optimizer for robotic analysis. In: 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 144–158. IEEE Computer Society (2020)