SLAM: Scalable Locality-Aware Middleware for I/O in Scientific Analysis and Visualization

Jiangling Yin, Jun Wang, Xuhong Zhang, Junyao Zhang
EECS, University of Central Florida
Orlando, Florida 32826
{jyin,jwang,xzhang,Junyao}@eecs.ucf.edu

Wu-chun Feng
Department of Computer Science
Virginia Tech
Virginia Tech, Blacksburg, VA 2406
wfeng@vt.edu

ABSTRACT

Whereas traditional scientific applications are computationally intensive, recent applications require more data-intensive analysis and visualization. As the computational power and size of compute clusters continue to increase, the I/O read rates and associated network cost for these data-intensive applications create a serious performance bottleneck when faced with the massive data sets of today's "big data" era.

In this paper, we present "Scalable Locality-Aware Middleware" (SLAM) for scientific data analysis applications. SLAM leverages a distributed file system (DFS) to provide scalable data access for scientific applications. To reduce data movement and enforce data-process locality, a data-centric scheduler (DC-scheduler) is proposed to enable scientific applications to read data locally from a DFS. We prototype our proposed SLAM system along with the Hadoop distributed file system (HDFS) on two well-known scientific applications. We find in our experiments that SLAM can greatly reduce I/O cost and double the overall performance, as compared to existing approaches.

Keywords

MPI/POSIX I/O; HDFS; Parallel BLAST; ParaView

1. INTRODUCTION

Modern technological advances have led to scientific instruments and computer simulations that create or collect extremely large and diverse datasets. To readily analyze and interpret this data, many scientific analysis/visualization applications have been designed. For instance, gene analysis tools such as parallel BLAST have been developed to help researchers to better understand the functionality of biological entities and processes [5]. Also, visualization applications such as ParaView [1] can interpret and graphically represent raw simulation/scientific data. These applications are developed with MPI programming model, in which the shared dataset is stored in a network accessible storage system like NFS, PVFS, or Lustre, and transferred to a parallel MPI process during execution. However, in today's big data era, rapidly growing data sets are too heavyweight to be moved efficiently over the network due to limited resources.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HPDC'14, June 23–27, Vancouver, BC, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2749-7/14/06 ...$15.00.
http://dx.doi.org/10.1145/2600212.2600709.

Distributed file systems, constructed from machines with locally attached disks, can scale with the problem size and number of nodes as needed. For instance, the Hadoop system employs a DFS for MapReduce applications and allows map tasks to access data locally. As the number of cluster nodes increases, the Hadoop system can scale-out, expanding storage and launching MapReduce tasks on the additional nodes. Compared to the MPI programming model,however, the MapReduce programming model lacks the flexibility and efficiency to implement the complex algorithms executed in scientific applications such as parallel BLAST [5], FLASH physics, or visualization applications.

In this paper, we propose "Scalable Locality-Aware Middleware" (SLAM), which allows scientific analysis applications to benefit from data-locality exploitation with the use of HDFS, while also maintaining the flexibility and efficiency of the MPI programming model. SLAM employs a process-to-data mapping scheduler (DC-scheduler) to transform a compute-centric mapping into a data-centric one so that a computational process always accesses data from a local or nearby computation node. We realize a SLAM prototype system using mpiBLAST and ParaView to demonstrate the efficiency of SLAM. In our work, SLAM runs on the Hadoop distributed file system (HDFS). Our experiments show that the I/O cost of data movement is highly reduced when SLAM is incorporated.

2. SLAM DESIGN AND IMPLEMENTATION

The objective of SLAM is to allow scientific analysis programs to benefit from data locality exploitation in HDFS. Since the data is distributed in advance within HDFS, the default task assignment may not allow parallel processes to fully benefit from local data access without considering data distribution. Thus, we need to intercept the original tasks scheduled and re-assign the tasks so as to achieve maximum efficiency on a parallel system with a high degree of data locality and load balancing.

SLAM system consists of two important parts: a data centric load-balanced scheduler called DC-scheduler and a translation I/O layer called SLAM-I/O. The DC-scheduler determines which specific data fragment is assigned to each node to process, thus minimizing the number of fragments pulled over the network. The SLAM-I/O will allow parallel MPI processes to directly access fragments treated as chunks in HDFS from local hard drive, which is part of the entire HDFS storage.

2.1 SLAM-I/O: A Translation Layer

Current scientific parallel applications are mainly developed with the MPI model, which employs either MPI or POSIX-I/O to run on a network file system or a network-attached parallel file system. SLAM uses HDFS to replace these file systems, which entails handling the I/O compatibility issues between MPI-based programs and HDFS.

We implement a translation layer, SLAM-I/O, to handle the incompatible I/O semantics. The basic idea is to transparently transform high-level I/O operations of parallel applications to standard HDFS I/O calls. We elaborate how SLAM-I/O works as follows. SLAM-I/O first connects to the HDFS server using hdfsConnect() and mounts HDFS as a local directory at the corresponding compute node. Hence each cluster node works as one client to HDFS. Any I/O operations of parallel applications that work in the mounted directory are intercepted by the layer and redirected to HDFS. Finally, the correspondent hdfs I/O calls are triggered to execute specific I/O functions *e.g.* open /read /write /close.

In the experiments to follow, we prototype SLAM-I/O using FUSE, a framework for running stackable file systems in a non-privileged mode. An I/O call from application to Hadoop file system is illustrated in Figure 1. Firstly, the Hadoop file system is mounted on all participating cluster nodes through the SLAM-I/O layer. Then the I/O operations of scientific applications are passed through a virtual file system (VFS), taken over by SLAM-I/O through FUSE and then forwarded to HDFS.

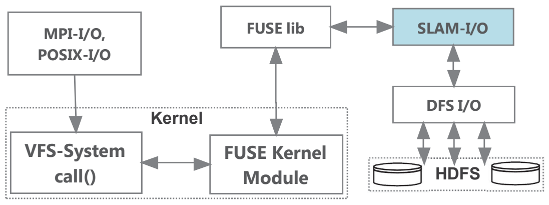

Figure 1: The I/O call in our prototype. A FUSE kernel module redirects file system calls from parallel I/O to SLAM-I/O. SLAM-I/O wraps HDFS clients and translates the I/O call to DFS I/O.

2.2 A Data Centric Load-balanced Scheduler

Scalability and high performance in data intensive scientific applications relies on data locality and load balance. However, heterogeneity issues exist that could potentially result in load imbalance. For instance, in parallel gene data processing, the global database is formatted into many fragments, and the data processing job is divided into a list of tasks corresponding to the database fragments. On the other hand, HDFS random chunk placement algorithm may distribute database fragments unevenly within the cluster, leaving some nodes with more data than others.

We implement a fragment location monitor as a background daemon to report unassigned fragment locations to the DC-scheduler. At any point of time, DC-scheduler always tries to launch a *local task* of the requesting process, that is, a task with its corresponding fragment available on the node of the requesting process. In practice, a high degree

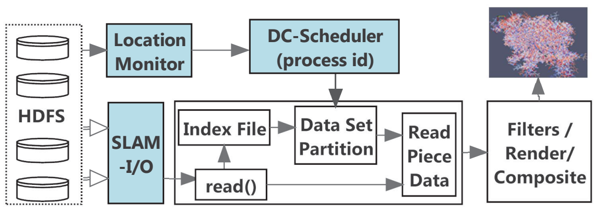

Figure 2: Proposed SLAM for ParaView. The DC-scheduler assigns data processing tasks to MPI processes such that each MPI process could read the needed data locally.

of data locality could often be achieved as each fragment has three physical copies in HDFS, leaving three different node candidates available for scheduling.

Upon an incoming data processing job, the DC-scheduler invokes the location monitor to report the physical location of all target fragments. If a process from a specific node requests a task, the scheduler assigns a task to the process using the following procedure. First, if local tasks exist on the requesting node, the scheduler will evaluate which local task should be assigned to the requesting process in order to make other parallel processes achieve locality as much as possible. Second, if no local task exists on the node, the scheduler will assign a task to the requesting process by comparing all unassigned tasks in order to make other parallel processes achieve locality. The node will then pull the corresponding fragment over the network.

Since mpiBLAST adopts a master-slave architecture, the DC-scheduler could be directly incorporated into the master process, which performs dynamic scheduling according to which nodes are idle at any given time.

2.3 ParaView with SLAM

We show how SLAM is implemented in Paraview in this section. ParaView employs reader modules on data server processes to interpret data from files. To process a dataset, the data servers running in parallel will call the reader to read a meta-file, which points to a series of data files. Then, each data server will compute a designated fragment of the data assignment according to the number of data files, number of parallel servers, and server rank. Data servers will read the data in parallel from the shared storage and then filter/render.

In order to achieve locality computation for ParaView, the default task assignments need to be intercepted to use our proposed DC-scheduler to assign tasks for each data server at run time. Specifically, we illustrate the SLAM framework organization for ParaView in Figure 2.

Our proposed DC-scheduler strategy in Section 2.2 is very suitable for applications with dynamic scheduling algorithms, such as mpiBLAST, in which scheduling is determined by which nodes are idle at any given time. However, since the data assignment in ParaView uses a static data partitioning method, the work allocation is determined beforehand; no process works as a central scheduler. For this kind of scheduling, we adopt a round-robin request order for all data servers. Until the task set is empty, the data server process with a specific process ID can get all the data pieces assigned to it. The data servers will read the data in parallel and then filter/render.

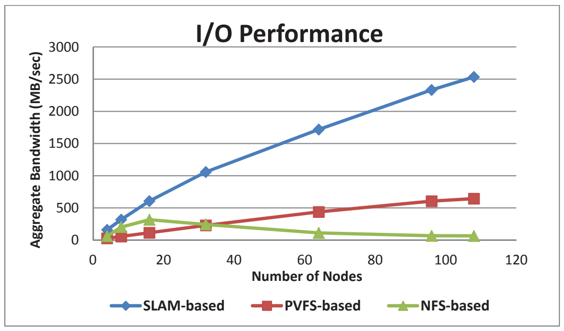

Figure 3: Read bandwidth comparison of NFS, PVFS and SLAM based BLAST schemes.

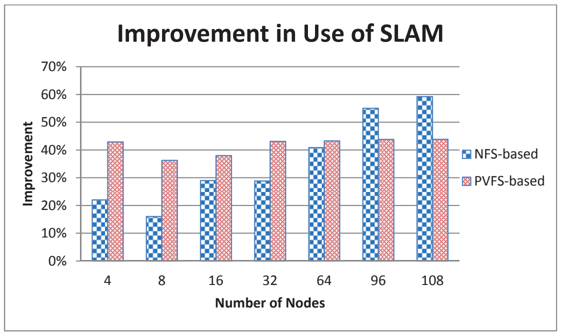

Figure 4: Performance gain of BLAST execution time when searching the *nt* database using SLAM, compared to NFS and PVFS-based.

3. EXPERIMENTS AND ANALYSIS

3.1 Experimental Setup

We conducted comprehensive testing on our proposed middleware SLAM on Marmot. *Marmot* is a cluster of the PRObE on-site project [4] and housed at CMU in Pittsburgh. The system has 128 nodes / 256 cores and each node in the cluster has dual 1.6GHz AMD Opteron processors, 16GB of memory, Gigabit Ethernet, and a 2TB Western Digital SATA disk drive.

In our experiment, MPICH [1.4.1] is installed as parallel programming framework on all compute nodes running CENTOS55-64 with kernel 2.6. We chose Hadoop 0.20.203 as the distributed file system, which is configured as follows: one node for the NameNode/JobTracker, one node for the secondary NameNode, and other compute nodes as the DataNode/TaskTracker. For comparison to SLAM, we run experiments with two conventional file systems—NFS and PVFS2. We choose NFS as it is the default shared file system on most clusters. Additionally, we installed PVFS2 version [2.8.2] with default setting on the cluster nodes.

3.2 Evaluating Parallel BLAST with SLAM

To make comparison with the open source parallel BLAST, we deploy mpiBLAST version [1.6.0] on all the nodes in the clusters. Equipped with our SLAM-I/O layer at each cluster node, HDFS can be mounted as a local directory and used as shared storage for parallel BLAST. BLAST itself can then run on HDFS without recompilation. For clarity, we labeled them as NFS-based, PVFS-based and SLAM-based BLAST. During the experiments, we mount NFS, HDFS and PVFS2 as local file systems at each node if a BLAST process is running on that node.

We select nucleotide sequence database *nt* as our experimental database. The *nt* database contains the GenBank, EMB L, D, and PDB sequences. At the time when we performed experiments, the *nt* database contained 17,611,492 sequences with a total raw size of about 45 GB. The input queries to search against the *nt* database are randomly chosen from *nt* and revised, which guarantees that we find some close matches in the database. We used the same input query in all running cases and fixed the query size to be 50 KB with 100 sequences, which generated a same output result in the amount of around 5.8 MB. The *nt* database was partitioned into 200 fragments.

To test scalability we collected results of aggregated read bandwidth for an increasing number of nodes as illustrated in Figure 3. The bandwidth is based on the total read time and overall amount of data processing. We find SLAM to be a scalable system, since the read bandwidth greatly increases as the number of nodes increase. However, the NFS and PVFS based BLAST schemes have a considerably lower overall bandwidth, and as the number of nodes increases, they do not achieve the same bandwidth increase. This indicates a large data movement overhead exists in NFS and PVFS over the network that bars them from being efficiently scalable.

When running parallel BLAST on a 108-node configuration system, we found the total program execution time with NFS, PVFS and SLAM based BLAST to be 589.4, 379.7 and 240.1 seconds, respectively. We calculate the performance gain as Equation 1, where $T_{\text{SLAM-based}}$ denotes the overall execution time of parallel BLAST based on SLAM and $T_{\text{NFS/PVFS-based}}$ is the overall execution time of mpiBLAST based on NFS or PVFS.

$$improvement = 1 - \frac{T_{\text{SLAM-based}}}{T_{\text{NFS/PVFS-based}}}. \quad (1)$$

As seen from Figure 4, we conclude that SLAM-based BLAST could reduce overall execution latency by 15% to 30% for small-sized clusters with less than 32 nodes as compared to NFS-based BLAST. Given an increasing cluster size, SLAM reduces overall execution time by a greater percentage, reaching 60% for a 108-node cluster setting. This indicates that NFS-based setting is not scaling well. In comparison to PVFS-based BLAST, SLAM runs consistently faster by about 40% for all cluster settings.

3.3 Evaluating ParaView with SLAM

To test the performance of ParaView with SLAM, ParaView [3.14] was installed on all nodes in the cluster. To enable off-screen rendering, ParaView made use of Mesa 3D graphics library version [7.7.1]. The DC-scheduler is implemented with VTK MultiBlock datasets reader for data task assignment. A multi-block dataset is a series of sub datasets, together they represent an assembly of parts or a collection of meshes.

For our test data we use the Macromolecular datasets that was obtained from a Protein Data Bank containing a repository of atomic coordinates, information describing proteins and biological macromolecules. The processed output of

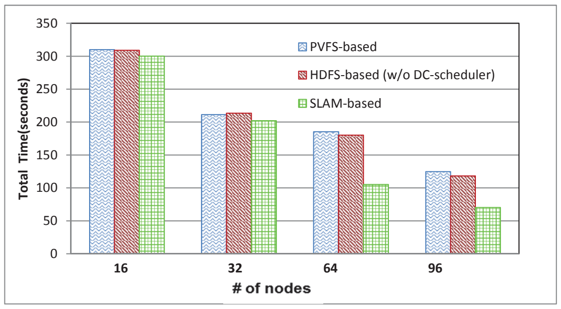

Figure 5: Execution time of PVFS, HDFS and S-LAM based ParaView.

these protein datasets are polygonal images, and ParaView is used to process and display such structures. In our test, we take each dataset as a time step and convert it to a subset of ParaView's MultiBlock file with extension ".vtu". Due to the need to download multiple datasets to the test system, we duplicate some datasets with a little revision and save them as new datasets in "binary" mode. For each rendering 96 subsets from 960 datasets were selected. As a result, our test set was approximately 40 GB in total size and 3.8 GB per rendering step. We use a python script to setup the visualization environment and needed filters to create a reproducible test. The script was submitted to the ParaView server via the provided pvbatch utility to produce a test run on a given node count.

Figure 5 illustrates the overall execution time of a ParaView analysis for an increasing number of nodes with the use of PVFS, HDFS and SLAM. With a small cluster size, the total time of the ParaView experiment did not greatly differ because the available network bandwidth is sufficient to deliver the data needed by the computational processes and there is no network contention. At 64 nodes however, the SLAM based ParaView shows it's strength in large clusters seeing a major reduction in total time when compared with the PVFS and HDFS based ParaView, being nearly 100 seconds quicker in execution for a total execution time of 110 seconds. In a 96 node cluster, the difference between SLAM and the other filesystems is lessened, but still a great improvement is observed with SLAM based ParaView executing in 70 seconds, a reduction almost twice over PVFS and HDFS based ParaView.

4. RELATED WORK

The data locality [7] or in-situ computation is a desirable technique to improve I/O performance. VisIO [6] obtains a linear scalability of I/O bandwidth for ultra-scale visualization but requires hard coding effort to rewrite the ParaView read methods. Janine *et. al.* [2] develop a platform which realizes efficient data movement between in-situ and in-transit computations that perform on large-scale scientific simulations. The Hadoop Distributed File System (HDFS) is an open source community response to the Google File System (GFS), specifically for the use of MapReduce style workloads [3]. Dryad and Spark are two other frameworks to support data locality computation. The idea behind these frameworks is that it is faster and more efficient to send the compute executables to the stored data and process in-situ rather than to pull the data needed from storage. Different from these approaches, our SLAM uses an I/O middleware to allow existing MPI-based parallel applications to achieve scalable data access with an underlying distributed file system.

5. CONCLUSIONS

In this paper, we developed a scalable locality-aware middleware to dramatically improve the I/O performance of scientific analysis applications. A SLAM-I/O layer is implemented to allow traditional MPI or POSIX based applications to run using Hadoop distributed file system. To exploit data-task locality computation, we proposed a novel data-centric load-balancing scheduler. The scheduler is independent of specific applications and could be adopted for other MPI-based programs that benefit from some form of data locality computation. By conducting experiments over two real scientific application, we found that SLAM can greatly reduce the I/O cost and double the overall execution performance as compared with existing schemes.

6. ACKNOWLEDGMENTS

This work is supported in part by the US National Science Foundation Grant CNS-1115665, CCF-1337244 and National Science Foundation Early Career Award 0953946.

This work is conducted at a PRObE staging cluster-128-node Marmot cluster, which is supported in part by the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE).

7. REFERENCES

[1] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. *The Visualization Handbook*, 717:731, 2005.

[2] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 49:1–49:9, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd. Probe: A thousand-node experimental cluster for computer systems research. volume 38, June 2013.

[5] H. Lin, X. Ma, W. Feng, and N. F. Samatova. Coordinating computation and i/o in massively parallel sequence search. *IEEE Trans. Parallel Distrib. Syst.*, 22(4):529–543, Apr. 2011.

[6] C. Mitchell, J. Ahrens, and J. Wang. Visio: Enabling interactive visualization of ultra-scale, time series data via high-bandwidth distributed i/o systems. In *IPDPS, 2011 IEEE International*, pages 68–79, May.

[7] S. Sehrish, G. Mackey, J. Wang, and J. Bent. Mrap: A novel mapreduce-based framework to support hpc analytics applications with access patterns. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 107–118, New York, NY, USA, 2010. ACM.