Scientific Data Services -- A High-Performance I/O System with Array Semantics

Kesheng Wu, Surendra Byna, Doron Rotem, Arie Shoshani

Lawrence Berkeley National Laboratory

One Cyclotron Road Berkeley, CA 94720

9/21/2011

DISCLAIMER

Scientific Data Services
-- A High-Performance I/O System with Array Semantics

Kesheng Wu, Surendra Byna, Doron Rotem, Arie Shoshani
Lawrence Berkeley National Lab
{KWu, SByna, D_Rotem, AShoshani}@lbl.gov

ABSTRACT
As high-performance computing approaches exascale, the existing I/O system design is having trouble keeping pace in both performance and scalability. We propose to address this challenge by adopting database principles and techniques in parallel I/O systems. First, we propose to adopt an array data model because many scientific applications represent their data in arrays. This strategy follows a cardinal principle from database research, which separates the logical view from the physical layout of data. This high-level data model gives the underlying implementation more freedom to optimize the physical layout and to choose the most effective way of accessing the data. For example, knowing that a set of write operations is working on a single multi-dimensional array makes it possible to keep the subarrays in a log structure during the write operations and reassemble them later into another physical layout as resources permit. While maintaining the high-level view, the storage system could compress the user data to reduce the physical storage requirement, collocate data records that are frequently used together, or replicate data to increase availability and fault-tolerance. Additionally, the system could generate secondary data structures such as database indexes and summary statistics. We expect the proposed Scientific Data Services approach to create a "live" storage system that dynamically adjusts to user demands and evolves with the massively parallel storage hardware.

1. INTRODUCTION

In the past few decades, the computing power of high-performance computers (HPC) has increased at a much faster pace than that of I/O systems. By a straightforward extrapolation of current hardware trends, an exascale computer will not be able to write its state to a checkpoint file onto disks before the next fault appears somewhere in the system [6]. Therefore, the whole system is not able to make any progress. Such dire predictions are spurring a number of new research efforts. Here we focus on addressing the underlying I/O issue by proposing a new design that combines the best features of parallel file systems and database systems.

There are a number of research efforts addressing the checkpoint issue [3][2]. A key objective in these efforts is to write the checkpoint file as fast as possible, typically by allowing each processor to write independently and producing log-structured files. Though these techniques can achieve good writing performance, the resulting checkpoint files cannot be read nearly as quickly, which makes it difficult to restart the failed jobs. We propose to take a more holistic approach with the aim of achieving high performance for both read and write operations.

Our approach combines strengths of both parallel file systems and database systems. In the next section, we present the key characteristics of these two commonly used data storage systems. In Section 3, we briefly review a number of state-of-the-art storage systems to provide contrast for the proposed design. The new design is presented in Section 4.

2. DATA STORAGE SYSTEMS

Existing data storage systems generally fall into two categories: file systems and database systems. A file system stores the user data as a series of files where each file is identified by a file name and stores user data as a sequence of bytes. This simple data organization makes it easy to construct a file system, but requires the user to handle all the semantic structure of the data themselves. There are a number of efforts to produce high-level file formats such as HDF5[1] and ADIOS BP[2]. A file system typically organizes these files into a hierarchy of directories. As file systems grow in size, this centralized hierarchy is an impediment to overall system performance [16].

[1] http://www.hdfgroup.org/HDF5/
[2] http://www.olcf.ornl.gov/center-projects/adios/

File systems were originally designed for a computer with a single CPU and a single disk. Over the previous decades, they have evolved to encompass many disks, servers and clients. Some of the initial design choices are limiting the overall system performance in the parallel environment. The hierarchical file organization is one such limitation. Another more commonly cited limitation is the POSIX consistency semantics that requires complex lock management to make parallel file systems behave as if they are on the original one-CPU-one-disk environment.

A number of newer file systems have relaxed their adherence to the POSIX semantics and achieved good performance [9][7]. Despite improvements, data accesses to file systems are slow and are predicted to perform even worse with increased number of clients at the extreme scale. The main reason behind poor performance of file systems is the static nature of the data under the current file system design. Once data is accepted by a file system, the byte sequence cannot be changed unless the user explicitly initiates the modification. This design reduces the burden on the system administrators and file system developers, but contributes to the poor performance for applications that read and write their data in different access patterns.

The second form of a commonly used data storage system is the database system. Unlike file systems, a database system typically adopts the relational data model for user data [12]. Compared to the simple sequence of bytes data model, the relational data model provides more information for the database system to optimize the physical storage and makes it possible for the system to add secondary data structures such as summary statistics and indexes. A database system typically also supports the Structured Query Language (SQL) for users to access the data and perform common analysis tasks, while a file system only provides an Application Programming Interface (API) for users to read and modify their data files. More importantly, the relational data model allows the database system to plan and optimize the requested data accesses and analysis computations.

Despite its strengths, very few scientific applications make use of database systems. A key reason for this is that scientific data don't easily fit into the relational data model. Even in the cases where the data could fit into the required data model, the user community still is unwilling or unable to use databases. For example, in the High-Energy Physics (HEP) community where many petabytes of data records are produced yearly, they have experimented with an object database system, but have abandoned that effort at a significant cost. This community eventually developed an object-oriented analysis system called ROOT [4]. One major reason for this choice is that SQL was not expressive enough for the desired analysis tasks. Another common reason for avoiding database systems is cost. Major commercial database systems are expensive to use and free systems typically have limited functionality, while file systems typically come with the computer system for free.

3. PROMISING SOLUTIONS

The advantages of merging database systems and file systems were first recognized in 1990s. For example, Olson implemented the Inversion file system on top of Postgres [13]; and Gifford et al. [10] proposed semantic file systems to provide associative accesses by automatically indexing the files and directories. Next, we briefly review a number of ideas and systems in the current literature and discuss how they might address key aspects of the scientific data management challenges.

3.1 High-Level Data Models

One strong feature of database systems is the high-level data model, such as the relational data model. In order to provide a high-level data model to scientific applications, a number of self-describing file formats have been developed. Among them, HDF5 and netCDF are the most widely used ones. The common feature among these scientific file formats is that they all treat user data as arrays. These software libraries are generally known as supporting an array data model. Most scientific computing packages such as LAPACK, MATLAB, and R work with arrays. Therefore, the array data model is a good starting point for developing advanced data management capabilities.

One prominent example of data management system with the array data model is SciDB [17]. SciDB is a new database system being developed by leaders of the database research community. The design of SciDB treats arrays as first class objects in the same way as existing relational database systems treat tables. The developers are expanding SQL to support queries on arrays. This extended version of SQL is called the Array Query Language (AQL). At the lower level, SciDB will also support a language called Array Functional Language (AFL). Anticipating that many users would rather keep their data in the original format instead of loading them to SciDB, the designers also plan to operate on HDF5 files directly, a feature dubbed "in situ data processing."

Instead of developing a new database system and then adding on some features to support in situ data processing, there are also efforts to modify file systems with high-level semantics [5][11][16]. While these efforts are more likely to be accepted by scientific communities, we believe that the array data model needs to be supported as a first class citizen instead of being supported through layers of metadata.

3.2 Recent File Systems

Earlier we have mentioned the key shortcomings of POSIX file systems, we now review efforts to mitigate them in newer file systems.

Current file systems are designed to work with many disks, servers and clients. Such parallel file systems are required to behave as if they are on a machine with one CPU and one disk following the POSIX standard. PVFS is a widely used file system that relaxes this consistency requirement while maintaining the same POSIX API [7]. It departs from the traditional POSIX file system standard only slightly.

Another well-known Non-POSIX file system is the Google File System (GFS) [9]. It has spawned a number of widely used derivatives such as Hadoop Distributed File System[3] and CEPH [18]. Key characteristics of GFS include integrating computing nodes with data storage nodes, replicating data for resilience and availability, and a tight integration with the MapReduce data processing system [8]. The integration of data storage and computation onto the same compute node reduces the physical distance between computation and data. It improves overall data access speed and reduces the access latency. A number of proposed data storage efforts have taken this approach to improve the I/O throughput [15][16].

Data replication inside the file system clearly improves the resilience of the system; however, to effectively improve the throughput of the data analysis, additional semantic information is needed. To this end, the MapReduce system used with GFS assumes the data stored in the files can be viewed as name-value pairs. This data model allows analysis tasks to be defined on name-value pairs, and executed on each data block concurrently. Because the input and output of each task is well defined, the MapReduce system is able to easily recover from failures of individual nodes or data analysis tasks. This again demonstrates the importance of having a clearly defined high-level data model.

Instead of taking a radical departure from the traditional design, many file system efforts attempt to incrementally modify the design [5][11][16][19]. The basic approach is to add additional attributes about files and make these attributes searchable through a querying interface such as the Spotlight from Mac OS.

3.3 NoSQL

In many applications, only a small fraction of the functionality of SQL is required. To simplify the system design and to scale up to larger applications, a class of data processing systems known as NoSQL was developed in the past few years[4]. A prime example of NoSQL is Hive[5] based on Hadoop MapReduce programming paradigm. By performing a handful of filtering operations on the data, such as locating the number of records satisfying simple conditions, NoSQL systems do not need a full-fledged database system and the complexity that comes with it. By limiting the supported operations, NoSQL can scale to massive amounts of data on a large number of nodes. NoSQL systems also focus on fault tolerance by replicating data. These systems however are not sufficient for scientific data, because they typically assume the user data is primarily text and can be captured with key-value pairs. In contrast, most scientific data contains much more complex relationships.

3.4 I/O Forwarding and Staging

In most parallel file systems, the I/O operations are performed through many layers of servers and networks. For example, many systems have I/O forwarding layer and staging layer [1][12]. The I/O forwarding software generally only concerns itself with aggregating I/O operations to improve the system throughput [12]. This is typically performed as part of the I/O system operation. However, the staging software such as ADIOS is run by users and must be programmed by users for different data analysis tasks [1]. In the literature, this way of conducting data analysis while the data is en route to disk is also known as "in situ data analysis." This approach has the advantage of removing the need to read the data to perform the analysis computation.

---

[3] http://hadoop.apache.org/hdfs/

[4] http://nosql-database.org/

[5] http://hive.apache.org/

The existing staging systems require users to explicitly manage the data flow from the computation program to the storage systems. It also requires the user to program all necessary computation. We could simplify this process by having the file system support MapReduce style of analysis computation.

4. SCIENTIFIC DATA SERVICES

The Scientific Data Services (SDS) combine the merits of database systems with file systems, without pushing the burden involved in either system on to users. We plan to follow the general principle established by the database research community to clearly separate the logical organization from the physical organization of user data. The logical organization is what is visible to the user; on this level, we plan to adopt the array data model instead of the sequence of bytes model used by the existing file systems. This data model is widely used in scientific data formats and matches well with the data model supported in most programming languages. The data access functions will be expressed using this data model at the high-level without dependence on the underlying physical organization. This is in the same spirit as the SQL interface to the database system, but without requiring the users to rewrite their analysis codes to use a new data access language, as SciDB demands. By cleanly separating the logical interface from the underlying physical data layout, the system implementers are free to innovate and adapt techniques to parallelize lower level disk operations and to automate performance tuning as demonstrated in many parallel database systems.

Our initial implementation will leverage existing file system technology and be presented to the operating system as a virtual file system similar to many recent file system efforts such as PVFS2 [7] and CEPH [18]. We will have an extensible framework to allow additional functionality to be added as we progress. We plan to optimize for checkpointing I/O in the first phase, followed by developing automated data reorganization algorithms, before addressing other types of I/O traffic patterns and services. This data service approach combines the key concepts from file system research and parallel database techniques to provide a clear and efficient data access interface as well as advanced data services for exascale data.

Figure 1 shows an overview design of the SDS system. We assume a parallel architecture with hundreds of thousands of clients, where each client is on a compute node and has many CPU cores producing and consuming data. The SDS will adopt the MPI-IO and HDF5 data access API. We believe that HDF5 high-level API is capable of accessing data in terms of regions, sub-regions, inquiring for data with specified attributes, etc. We will extend the API based on user needs, but conform to the HDF5 or MPI-IO APIs to keep it simple and easy for programmers to adopt without a steep learning curve. The SDS system contains various services including indexing, data reorganization, local and remote data mover services, data reduction, data enhancement, feature extraction, and access optimization services. We also plan to provide an easy way to extend data services using templates. The SDS daemons stay alive to perform reorganization of data in order to optimize writes and reads depending on the directives provided by applications. More specifically, we plan to provide the following services.

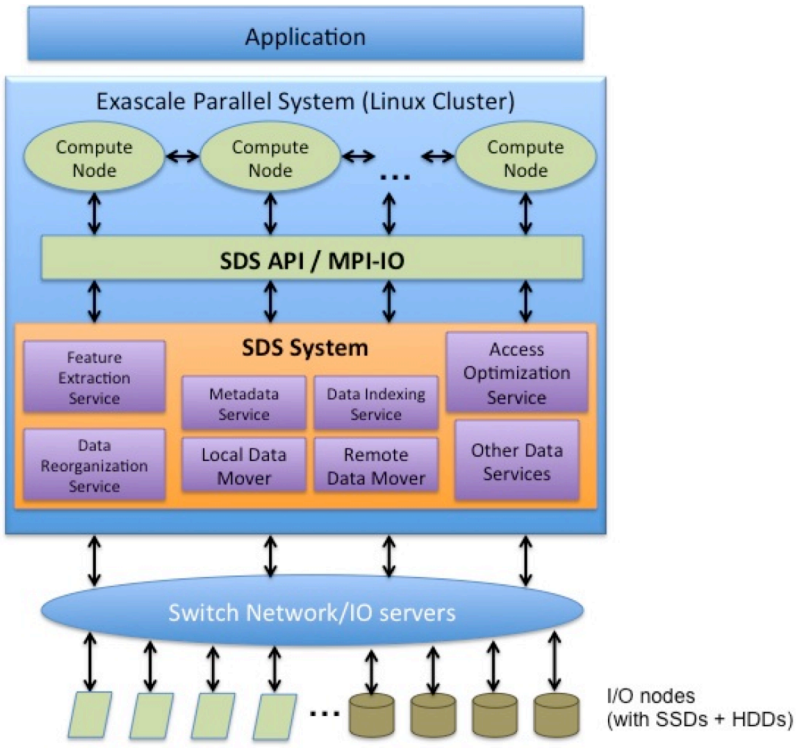

Figure 1: Overview design of SDS

Data Reorganization: Based on data access characteristics, there are various efficient possibilities to store data in the storage hierarchy. For example, storing checkpointing data from N

processes in N files or as N pieces is more efficient than storing all the data in one large file. PLFS [3] and ADIOS BP file format [1] follow this strategy by creating sub-files for data from each process and managing metadata related to the sub-files. SDS will manage more metadata than simple sub-file information for relating data to processes that generate the data. Our metadata management contains attributes of data, such as geographical locations, ranges of data, etc. Based on the attributes, the data reorganization service dynamically chooses data placement for achieving the best write and read performance.

Indexing: To support efficient accesses, we plan to employ state-of-the-art indexing methods. For accessing and placing data objects, we will use hash-based techniques to provide the maximum concurrency possible [WSS10]. For lookups on a local storage system, we will use a B-tree variant because it has proven useful in many file systems for similar tasks [OS10]. For queries based on values inside the arrays, we plan to use compressed bitmap indexes [WOS06], as they are likely to be the most efficient option.

Metadata Service: Instead of just locating bytes of data, the proposed metadata service is able to locate data based on attributes of the data. This service is responsible for obtaining characteristics of data. As mentioned earlier, the metadata service manages attributes of data to enhance the quality of representing data. We initially plan to provide an interface that allows users to annotate data attributes.

Data Enrichment: Before dumping data onto storage devices, some processing can be performed to reduce the amount of data. In many cases, there are many "common" data records, which contain expected common cases. The data enrichment service provides a library to define functionality to identify low information content data and to obtain useful characteristics for improving future use of the data.

Data Access Acceleration: Data caching and prefetching are efficient ways to improve performance of data analysis. We propose to explore data access characteristics of exascale data and develop strategies for efficient prefetching strategies.

Feature Extraction: In pattern recognition and image processing applications, various analyses, such as principal components analysis, latent semantic analysis, partial least squares, independent component analysis, *etc.*, can be performed to extract useful features in data. This in situ analysis strategy is proven effective to reduce the amount of data stored.

Data Mover services: Since data movement is a significant performance bottleneck, the local and remote data mover services will analyze performance of the data transfer costs and schedule data transfers to minimize performance overheads. The local data mover caches and prefetches data depending on access patterns within a cluster. The remote data mover is an extension of the data service approach to distributed environments. This service enables sub-setting and aggregating data based on data requests.

We plan to utilize in situ analysis for performing tasks such as indexing, feature extraction, data enrichment, etc. With the proposed features, we aim to achieve various goals including ease of use, scalability at extreme scale, portability, and extendibility while preserving data security and reliability.

5. SUMMARY

With the impending data deluge in exascale computing, we anticipate that access strategies with superior performance will have an enormous impact on many fields of science. Extracting information out of data is critical for scientific advancements. In this paper, we outlined a novel design for improving the pace of scientific discovery in the form of a "live" data service instead of the current static file system approach. The proposed data services could play a significant role of enabling scalable scientific data management and paving the way for a data management paradigm shift for dealing with very large volumes of data.

6. ACKNOWLEDGMENTS

This work was supported in part by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

7. REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng; K. Schwan, M. Wolf, S. Klasky. 2009. "Extending I/O through high performance data services," CLUSTER '09. pp.1-10. DOI= 10.1109/CLUSTR.2009.5289167.

[2] S. Al-Kiswany, M. Ripeanu, S. S. Vazhkudai, A. Gharaibeh. 2008, "stdchk: A Checkpoint Storage System for Desktop Grid Computing," International Conference on Distributed Computing Systems, pp. 613-624.

[3] J. Bent, G. A. Gibson, G. Grider, B. McClelland and P. Nowoczynski and J. Nunez and M. Polte and M. Wingate. 2009. PLFS: A checkpoint file system for parallel applications. In SC'09, ACM/IEEE.

[4] R. Brun, F. Rademakers. 1997. "ROOT: An object oriented data analysis framework," Nuclear instruments & methods in physics research, Section A 289, 1-2, 81—86.

[5] J. B. Buck, N. Watkins, C. Maltzahn, and S. A. Brandt. 2009. Abstract storage: moving file format-specific abstractions into petabyte-scale storage systems. In DADC '09, 31-40. ACM. DOI=10.1145/1552280.1552284.

[6] F. Cappello "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," International Journal of High Performance Computing Applications, 23(3): 212– 226, 2009. http://hpc.sagepub.com/cgi/reprint/23/3/212.pdf.

[7] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur. 2000. PVFS: a parallel file system for Linux clusters. In ALS'00, Vol. 4. USENIX.

[8] J. Dean and S. Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM. V51, 107-113. DOI=10.1145/1327452.1327492

[9] S. Ghemawat, H. Gobioff, and S/-T. Leung. 2003. The Google file system. SIGOPS Oper. Syst. Rev. 37, 5, 29-43. DOI=10.1145/1165389.945450.

[10] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole, Jr.. 1991. Semantic file systems. In SOSP '91. ACM, New York, NY, USA, 16-25. DOI=10.1145/121132.121138

[11] J. L. Naps, M. F. Mokbel, and D. H. C. Du. 2011. "Pantheon: Exascale File System Search for Scientific Computing". In SSDBM 2011.

[12] K. Ohta, D. Kimpe, J. Cope, K. Iskra, R. Ross, Y. Ishikawa. 2010. "Optimization Techniques at the I/O Forwarding Layer," Cluster 2010, pp. 312-321.

[13] M. A. Olson. 1993. The Design and Implementation of the Inversion File System. In Proceedings of USENIX Winter'1993. pp. 205~218.

[14] P. O'Neil and E. O'Neil. 2001. Database: Principles, Programming, and Performance. 2nd Ed. Morgan Kaufmann.

[15] I. Raicu, I. T. Foster, and P. Beckman. 2011. Making a case for distributed file systems at Exascale. In LSAP '11, 11-18. ACM. DOI=10.1145/1996029.1996034.

[16] M. Seltzer and N. Murphy. 2009. Hierarchical file systems are dead. In HotOS'09. USENIX, Berkeley, CA, USA

[17] M. Stonebraker, J. Becla, D. Dewitt, K.-T. Lim, D. Maier, O. Ratzeberger, and S. Zdonik. Requirements for science databases and SciDB. In CIDR 2009, Monterey, CA, 2009.

[18] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. 2006. Ceph: a scalable, high-performance distributed file system. In OSDI '06, 307-320. USENIX, Berkeley, CA, USA.

[19] J. Xing, J. Xiong, N. Sun, and J. Ma. 2009. Adaptive and scalable metadata management to support a trillion files. In SC '09. ACM, New York, NY, USA. Article 26. DOI=10.1145/1654059.1654086