Abstract Storage: Moving File Format-Specific Abstractions into Petabyte-Scale Storage Systems

Joe B. Buck Noah Watkins Carlos Maltzahn Scott A. Brandt
Computer Science Department, University of California, Santa Cruz, California, USA
{buck,jayhawk,carlosm,scott}@cs.ucsc.edu

ABSTRACT

High-end computing is increasingly I/O bound as computations become more data-intensive, and data transport technologies struggle to keep pace with the demands of large-scale, distributed computations. One approach to avoiding unnecessary I/O is to move the processing to the data, as seen in Google's successful, but relatively specialized, MapReduce system. This paper discusses our investigation towards a general solution for enabling in-situ computation in a peta-scale storage system. We believe our work with flexible, application-specific structured storage is the key to addressing the I/O overhead caused by data partitioning across storage nodes. In order to manage competing workloads on storage nodes, our research in system performance management is leveraged. Our ultimate goal is a general framework for in-situ data-intensive processing, indexing, and searching, which we expect to provide orders of magnitude performance increases for data-intensive workloads.

Categories and Subject Descriptors

E.2 [**Data Storage Representations**]: Contiguous representations; E.2 [**Data Storage Representations**]: Object representation; E.2 [**Data Storage Representations**]: Composite structures; H.3 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Design, Performance

Keywords

Active storage, structured storage, data-intensive computing

1. INTRODUCTION

High-end computing environments (HEC) have become an indispensable tool in the scientific community as simulations and computational models become increasingly complex. The use of massively parallel computing environments have allowed scientists to tackle problems which would otherwise be intractable. However, as the time and space complexities of scientific applications grow, the ability for computing environments to scale is essential to maintaining the utility of HEC systems. Traditional scientific computations have largely been CPU bound, benefiting from contemporary, high-end computing environments which scale well for workloads with relatively small I/O requirements. However, emerging problem domains are requiring data-intensive computations whose workloads are not well suited for today's HEC systems. According to a recent DARPA study [17], exa-scale computing is coming, and has the potential to require zetta or yotta bytes of storage. This same study also predicts that the cost of exa-scale computing will be dominated by data movement, in terms of time, money, and energy. The predicted I/O demands of next-generation, extreme-scale computations is our motivation for a new system design which combines storage and processing, thereby offering a high-performance, scalable execution environment for data-intensive computations.

Our approach to scaling HEC environments for data-intensive computations is to reduce, and where possible, eliminate data movement between computations and storage. In-situ computations, or application-specific, co-located computations executing on storage nodes in service of distributed applications, can be utilized to achieve significant reductions in data movement. However, current HEC systems are ill-equipped to achieve these optimizations because the byte stream abstraction commonly used in file system interfaces hides the application-specific structure of data and complicates any storage-side execution of operations that depend on these structures. This paper proposes a new design for HEC systems which use specifications of the structure of data, and knowledge of application behavior, to intelligently distribute data in support of in-situ computations, and reductions in data movement.

The evolution of file formats, and application-level functions on distributed data, has motivated the creation of structured storage abstractions. Examples of such systems are Google's BigTable[9], Amazon's SimpleDB[1], Apache's CouchDB[2], and Facebook's Cassandra (an open-source alternative to BigTable)[3], each of which are designed for a specialized purpose, and to scale beyond petabyte capacities. The confluence of application-specific file formats and interfaces in these systems demonstrate a real and present need for such features in large scale systems, and

DADC'09, June 9–10, 2009, Munich, Germany.

likely reveal opportunities for advancements in scientific HEC systems.

Today, parallel file systems stripe data across storage nodes in order to support application agnostic features such as replication and scalability. The resulting data partitioning creates application data dependencies that involve multiple storage nodes, and thus require multiple network connections. The design of our system makes use of structured storage, which supports more efficient use of network resources by minimizing or eliminating inter-storage node data partitioning. However, even with optimal efficiency in the usage of network resources, interconnects will remain a bottleneck for data-intensive computations as transport technologies fail to keep pace with the demands of next-generation workloads.

In-situ computation is a promising approach to dealing with data transport bottlenecks by exploiting parallelism within clustered petescale systems, such as Blue Gene/P. The work of the Active Disks[23] project utilized available parallelism between disk controllers to achieve computation speed-ups. However, prognostications made in the Active Disk papers that on-disk processing capacity would scale with CPU performance have not materialized. While the performance of past systems warranted the use of inter-disk parallelism, current technology trends have created alternative sources of exploitable parallelism. Today we are seeing a prevalence of large data centers composed of thousands of low-cost, rack-mounted PCs with gigabytes of main memory, multi-core CPUs, disk drives providing terabytes of local storage, and the possibility to achieve teraflops of local computational power through the addition of special-purpose GPU units. Making use of available processing power present in HEC environments will project the original ideas developed by the Active Disk project into the context of modern data centers, and the resources they make available. While in-situ computations introduce additional requirements of a storage system, we are beginning to see acceptance of system designs that make use of storage node processing resources in order to achieve better performance. For example, the SNIA OSD working group's OSD-3 draft[12] assigns new levels of responsibility to storage nodes for the placement of data.

As storage nodes are assigned increasing levels of responsibility, and applications are decomposed into sub-computations running at arbitrary locations within the system, security and quality of service become key factors in our design. Recent advancements in virtualization and performance management provide the services necessary to control distributed computations with a heterogeneous set of security and quality of service requirements. Virtualization techniques support host-based performance isolation, as well as many low-level security features necessary to implement higher level policies. End-to-end performance management controls distributed applications whose associated computations cross system boundaries, and where virtualization-based performance isolation fails to offer adequate levels of control (e.g. disk throughput reservations).

In light of these observations and realizations, it is apparent that changes in HEC environments must be made in order to support the demands of next-generation, data-intensive computations. The remainder of this paper focuses on our proposal for a new HEC system design which offers scalability for data-intensive workloads. In Section 2 we present our system requirements, and a detailed design of our system is explored in Section 3. Our current prototype is described in Section 4. Related work is covered in Section 5, followed by future work in Section 6, and finally the conclusion in Section 7.

Definitions

Before proceeding we will define several terms and state a few assumptions that we hold. When we talk about parallel file systems we assume a file system running on a large cluster of servers that can be directly accessed by clients which run applications that access and store data on the parallel file system.

storage node refers to a server in a parallel file system that is responsible for managing and offering access to some part of the storage managed by the parallel file system.

in-situ computation means executing computations on data where that data is stored, as opposed to transferring data from its current location to the location of a separate processing resource. In the context of a parallel file system this means the storage node executing computations which access, and possibly alter, data resident on the storage they manage.

end-to-end is used in our performance management discussion to mean all shared resources on a client, all shared resources on a server, and the communications channel between them.

application-specific data structures is used to refer to the structures and format of data specific to an application. Examples are astrological data being organized into arrays that correspond to sections of the sky, or performance monitoring software storing point-in-time performance values in an array indexed by time.

data element is a specific instance of an application-specific data structure. In the case of the performance monitoring software example, a data element could be a vector that stores power consumption of a CPU in 100 millisecond intervals for the duration of a given test.

2. SYSTEM REQUIREMENTS

The key to optimizing the performance of data-intensive computations is to minimize data movement. Data movement can be reduced by using in-situ computations. This section explains the components that we feel will enable in-situ computations.

Structured Storage

Structured Storage refers to the ability of a distributed storage system to use application and user supplied knowledge to impose structure on, and intelligently place data in a distributed environment. New functionality is made possible, for example our system can use this new found information to eliminate inter-storage node data partitioning by requiring that application specific data elements be managed by single storage nodes. For example, if applications store array-based data sets, storage systems

with knowledge of this structure can take measures to ensure array elements are stored contiguously. This will require taking into consideration the trade-off between throughput and latency, in that current parallel file systems partition data over many spindles to increase throughput while also increasing latency when applications access specific data elements that span spindles, due to having to execute multiple accesses which each have fixed time costs. Therefore, it may not be possible to maintain reasonable throughput speeds and also guarantee application data elements are contained on a single storage node.

Programming Models

Programming models are used by developers to express computations in terms of functionality, as well as policy. When programming models expose interfaces customized for particular applications, developers can take advantage of a more natural way of expressing their computations, easing application development, and facilitating more accurate reasoning about program behavior. For example, when only a byte-stream interface is available, developers of HEC applications must create custom code or use libraries to map their application logic onto the byte-stream interface. The use of libraries to hide this complexity is common, but doesn't allow any of the knowledge of the structure of data to be used to support lower-level optimizations. Programming models and structured storage are complementary components of the system. While programming models provide an interface through which data is accessed, and semantics and policy are expressed, structured storage facilitates the efficient implementation of programming models which may require specialized fragmentation policies in order to support in-situ computations, or optimize for access patterns.

Quality of Service

Contemporary HEC environments support simultaneous execution of multiple computations, each accessing services and resources provided by the storage system. Providing quality of service guarantees to applications is difficult because storage nodes are required to service unpredictable application requests, and may not have knowledge of application-level policies. QoS in storage systems is also challenging because of disk arm positions and seek times that can easily dominate retrieval latencies, making performance isolation between users non-trivial. This problem becomes more difficult with the introduction of in-situ computations that compete for the resources of a storage node whose policy also includes serving external data requests. In order to provide quality of service guarantees to applications in a distributed system, which may contain an arbitrary mix of in-situ computations and policies, end-to-end performance management is essential. To support performance guarantees made to applications, resources must be controlled at all levels of the system with a unified view of system policy.

Scalability, Availability, and Durability

Distributed storage systems offer many features and guarantees such as reliability, recovery, and snapshots that users have come to expect. However, supporting structured storage and in-situ operations in a safe, scalable, and feature rich system presents many challenges, which we discuss in Section 3.5.

3. SYSTEM DESIGN

3.1 Structured Storage

The layers of the system related to structured storage are presented in Figure 1. At the highest level, *Application Data* represents the abstraction at which applications interact with the system. An example of this is an application running a simulation that stores measurement data in an array format. At this level applications construct specifications of the data they interact with, and make these specifications available to the storage system. *Application dictated alignment* occurs within the storage system, and aligns data according to the specifications passed in by the application. An example of this is when the aforementioned simulation application writes out multiple arrays. In this case we want to make sure that each array is stored in its entirety on a single storage node, so the application data is aligned accordingly. Below that, logically, a similar process happens where *file system influenced alignment* occurs. In our example, the arrays may be realigned again, possibly on file system block boundaries, but this should not violate the application specified alignment. The file system based alignment is meant to complement the application based alignment. At the lowest level is the *storage node* which stores file system extents in some fashion. File system extents correspond to actual blocks on disk(s) located at storage nodes.

Communication costs are most often incurred when data is fragmented, and additional storage nodes must be contacted to satisfy data requests. Current parallel file systems are designed to partition data, and distribute that data according to a set of semantics which support general policy goals such as data balancing, redundancy, and reliability. However, applications logically align computations with application-specific knowledge of data structure, and when the logical alignment of a computation's access to data deviates from the placement of data in the underlying file system, such as when an application accesses a data element that spans nodes as a result of preceding data in the same file and not purely due to the data elements size, unaligned data accesses result and additional communication costs are incurred. Thus our system will take into account knowledge of the structure of application data when aligning and distributing data among storage nodes.

A function we will build into our system is the ability to enforce application data alignment in storage. This functionality comes at a cost in that data which was previously spread across multiple nodes in order to achieve higher total bandwidth at the point the data is read off of hard drives is now limited to the bandwidth of a single hard drive. In preventing fragmentation we minimize communication overhead and simplify the models for in-situ processing. This functionality will be enabled at the ADT level, described in Section 3.3, and we believe that the ability to align data in this fashion is a noteworthy capability of our system. As such, we explain the details and consequences of data alignment along application data element boundaries.

Eliminating fragmentation of application data elements requires knowledge of the structure of data being stored in the file system such that storage nodes manage only whole data elements. Assigning a data element to a storage node that results in a "remainder" being stored else where will prompt complete reassignment of the original element to an

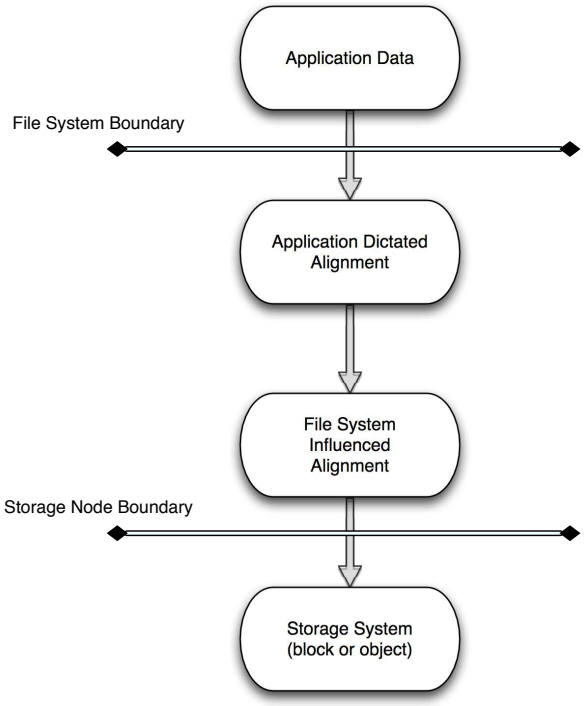

Figure 1: Stages of Data Management Within Application Write Stream

alternative node. The motivation for this requirement is to allow clients to access whole data elements with a single connection to a storage node. Of course this strategy will not work when data elements do not fit on a single node. However, many scientific workloads operate on structured data composed of smaller elements that do fit on single nodes. We do not address in this paper the problems that arise when the structure of application-specific data is not decomposable into appropriately sized chunks. Additionally, when in-situ computations execute on a storage node, data elements must be present in the form specified by applications in order for in-situ computations to act without further communication. Figure 2 illustrates two system fragmentation policies. On the left, two clients access *Data A* and *Data B*, which are striped across multiple storage nodes. For the sake of example, assume they are each arrays from different simulation experiments and are contained in separate files. This data layout requires clients to access multiple nodes to fulfill a request. On the right, *Data A* and *Data B* represent application-defined data elements that are stored on individual nodes. As can be seen, In-situ computations accessing data elements within this system do not require additional data fetches.

Structured storage is the basis on which we build the rest of our system. It is crucial in enabling in-situ processing, which we expand upon in Section 3.2, and heavily influences our definition and use of ADTs, which are detailed in Section 3.3.

3.2 In-situ Processing

The idea of moving computations closer to the data they depend on is not new. Much of the previous research in this area can be seen in the Active Disks project [23]. However,

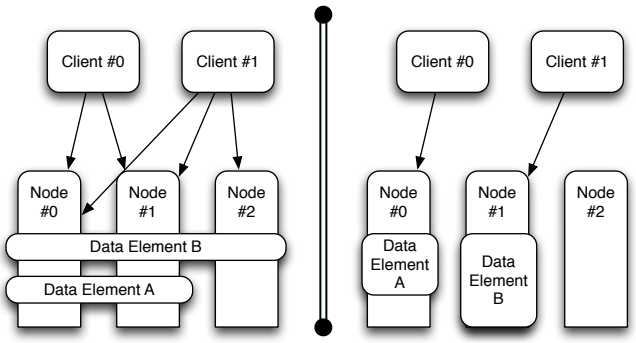

Figure 2: Two clients, each accessing a single piece of data in a parallel file system without (left) and with (right) data alignment. The reduction in communication connections is the result of data alignment

recent advances have spurred renewed interest in the topic of in-situ computations within storage systems. For example, storage systems using a design that makes use of object stores (OSDs) are moving towards an architecture that helps more easily express and address structured storage needs. Additionally, many systems make use of storage nodes with significant amounts of processing power near the disk. The Active Disks project demonstrated performance increases by moving computations from a host to an on-disk co-processor, eliminating the disk-to-host interconnect. However, this architecture is impractical for large, distributed computations that require modern execution environments, and enhanced network communication features, both of which are required to interact with application-level data structures that span multiple storage nodes.

While the arguments posited by Riedel generally hold true today, in that the aggregate power of many processors near data will be more powerful than moving the data to remote processors in certain circumstances, the limiting link is no longer the system bus, but rather inter-storage node communication links. In other words, the delineation that used to be placed between host CPU and disk processor is now manifest by the delineation between computing nodes and storage nodes. The limitation in the communication link is not just one of speed, but also of latency and capacity. It is in light of this that we view the appropriate computational unit to be the storage node (in the context of a modern parallel file system), and not an on-disk processor.

As we've stated, the logical way to avoid the current bottleneck of inter-storage node communication is to move the processing to the data. We envision this happening via developer code interacting with ADTs implemented within the storage system, the storage system identifying the storage nodes with data resident which the application needs access to, and then the ADTs providing access to the data on the storage nodes. When application code is executed within our system it will need to specify parameters such as the files being accessed, how its output is to be handled, and execution performance requirements, among others. We classify this latter requirement into Performance Management, which we discuss in Section 3.4.

An additional benefit is made possible by the combination of the structure of data in a parallel file system with our

data alignment requirement. Since the contents of files can be split over multiple storage nodes and the data on each storage node can be acted upon independently, it is now possible to realize intra-file parallelism, in addition to the inter-file parallelism leveraged by previous systems that combined storage with computation, without incurring significant communication overhead. Significant performance gains are possible in light of this new found level of parallelism, which was not possible when files were constrained to existing entirely on a single disk with a single processor, as in previous systems that attempted to facilitate in-situ computations.

3.3 Abstract Data Types

Drawing on the idea that structured storage facilitates aligned data access, and that storage system programming models bridge the gap between data layout and application-centric views of data, we draw an analogy to the concept of an abstract data type (ADT). Specifically, an ADT is a specification of data, and the interface through which operations are applied to the data. Storage system oriented ADTs thus use well-defined structured storage as the specification of data, and export an interface to clients specific to a particular abstract data type, and thus to a particular data structure. The power of an ADT inspired architecture in storage systems facilitates adaptive performance optimizations by adjusting ADT instances for particular applications, such as preventing data element fragmentation as defined in Section 3.1. Furthermore, because data structure and access methods are well-defined through a specification, ADT transformations can be defined that provide for maximum system flexibility.

Storage system based ADTs also provide a convenient abstraction level for expressions of application policy. An ADT tailored for stream-based multimedia applications may require ADT functions (e.g. "play") to have significantly different performance requirements than other functions of the same ADT. Additionally, separate instances of the same ADT serving distinct applications may have parameterized policies which are fully defined by the application interacting with the ADT instance. Supporting rich abstractions through ADTs will require general treatments at all levels of the system.

3.4 Performance Management

Distributed file systems rely on the ability of storage nodes to service requests related to the data managed by a node. In contemporary distributed file systems, storage nodes require relatively small amounts of computational ability, and usually operate on dedicated servers. As a result, most processing which executes on storage nodes does so on behalf of applications remotely accessing the data managed by a node. In the general case, organization-level policies may dictate the importance of distributed applications, such as the time-share received, or QoS guarantees. However, in current parallel file systems, storage nodes service requests independent of system-wide policies, offering only best-effort service. Enabling in-situ computations on a storage node presents additional challenges as computations compete for shared resources, and affect the ability of a storage node to satisfy the behavioral constraints of applications.

Managing the performance of distributed, data-intensive applications is a problem that requires end-to-end solutions. Multiple applications with diverse performance requirements share many resources in HEC environments. For example, data must be transmitted over shared network connections, multiple applications indirectly share the resources of storage nodes, and collocated computations directly make use of host resources. Leveraging our research in end-to-end performance management of distributed applications, including managing the performance of the network, disk I/O scheduling, and CPU scheduling [7, 22, 15], enables us to make performance guarantees for both applications accessing data and in-situ computations.

3.5 Scalability, Availability, and Durability

Distributed storage systems offer many types of services and guarantees. These range from reliability and recovery, to scalability and snapshots. We have considerable background knowledge in implementing these services from our previous work with Ceph[28]. Furthermore, distributed systems supporting high-profile, long-running computations, such as those used in the scientific community, must supply users with safety guarantees that users have come to expect. The semantics of these features and guarantees are made in today's storage systems without application-specific knowledge. While this may simplify algorithms and implementations, incorporating application-specific knowledge into the system is required to correctly support the co-existence of application-specific semantics (i.e. structured storage), and the semantics of conflicting system features, such as replication and fragmentation.

4. CURRENT PROTOTYPE

This paper has illustrated the complex design space for a scalable HEC environment designed for data-intensive computations. While we have presented many design requirements, including specific design details, our current prototype system is only in its preliminary stages, and is not necessarily representative of future prototypes. The system is a Python-based framework for exploring storage ADTs in the context of a distributed object-based storage system. The system allows a developer to create customized programming models on top of an object-based storage abstraction. Programming models are created by defining a client-side interface, and any number of remote-procedure calls which execute on object-storage devices in support of client-side operations. A unified object-addressing scheme is used by programming models to express a mapping between the data view at client-side, and physical storage within the storage cluster. The system contains no performance management, security, or policy representation. Using the system we have examined how traditional byte streams, and a distributed skip list data structure map onto the architecture, and have found it natural to reason about distributed data structures implemented in our system because of the unified view of the implementation (i.e. data type implementations cross-cut system layers, from client-side to the OSD layer). As we discover new abstractions on which to build ADTs that take into account our list of system requirements, additional prototypes will be designed. We anticipate that the next iteration of our system will be implemented within the Ceph object-based storage system [28].

A simple example of how a client application would interact with the skiplist ADT can be seen if we consider

the simple act of inserting a sequence of random integers. The application would instantiate a skiplist object, insert each element one-by-one using an "insert()" call. Items can be retrieved via "retrieve()" and deleted via "remove()" calls. Within the storage system the elements are kept in distributed skiplists which grow and contract as data is inserted and removed. The performance characteristics of a single skip list are maintained in the larger, distributed skiplist and an "iterate()" function allows for all data in the structure to be accessed, in key order.

While this example is very simple it should help form a mental image of how application code interacts with our system. It's simple to imagine that the skip list could be replaced with a B-* tree and the application code would not have to change. Functions in the storage system could be optimized for certain workloads, insert-heavy versus read-heavy for example, and data structures could even morph on demand, an array could decide to change itself into a tree structure once it has grown to a certain point, dictated by a preset policy, or if an application requested the storage system to make the conversion.

5. RELATED WORK

Our research draws inspiration from several different areas, including work in structured storage, in-situ processing, parallel file systems, and security.

5.1 Structured Storage

Structured storage in an idea that was been around for several years [19]. In Boxwood, the authors recognized that building complex data structures on top of simple storage devices was difficult. Their means of addressing this impediment was implementing structures within the storage infrastructure. Like Boxwood we implement structure within our storage structure, but we take the principle of structuring data further than Boxwood did. In addition to creating structures within storage that applications can leverage as they see fit, our system also aligns data along several boundaries (application dictated boundaries such as the start of arrays or vectors and file system based boundaries such as the beginning of a block or its equivalent) and provides access to application data elements as they reside within the storage system, by which we mean in-situ computations can access elements on storage nodes without having to read the file from the beginning in order to determine where a specific data element begins in the stored data.

Structured storage is typically thought of as series of fixed size records, as in Boxwood and Vesta [19, 10]. Our system generalizes the idea of structured storage to accommodate arbitrary application data, while adhering to application and file system specific requirements related to fragmentation.

The NetCDF and HDF5 file formats, while not being examples of structured storage in the sense that a process on a storage node would not necessarily know where a given data element started in the byte-stream, highlight a potential benefit of structured storage. They each contain a header which describes the layout of data for the rest of the file. By embedding a header they allow different applications to process the data contained within them in its entirety or on a selective basis. This is more of a portability concern than it is a concern for processing data in-situ but the principle is relative to structured storage; propagating a dataset's NetCDF or HDF5 structure down into storage would allow for processing in storage as well as data portability.

The existence of the ADIOS [18] system is implicit recognition that adding a layer which reformats I/O for the specific storage layer being written to can yield significant performance improvements. This insight is echoed by testing done at Argonne National Laboratory [24] which found that significant performance gains could be achieved if file system specific information was used to align application data. These facts were the impetus for our decision to include file system parameters in the alignment process of application data.

Another paper on structured storage[13] makes the argument that modern data sets requires something in-between a file system, unstructured and rudimentary, and a database, too much overhead and too many unused features. The authors contend that a storage system should never hide expressive power from applications. We take this ethos, combined with the performance related findings from Argonne mentioned in the previous paragraph, a step further and state that applications should be actively given the ability to express themselves to a greater degree than has been possible to-date in terms of affecting data placement within storage systems. We believe that it is through this increased expression that functionality can be expanded and performance increased, as seen at Argonne.

Stonebraker [27] makes a case for a purpose built, simplified data base for the scientific community that leverages simple structures and allows for user-defined application code. This proposed system could easily be implemented within our work; a fact which we perceive as a portent that we are on the right track.

5.2 Moving Computations to Data

Active Disks[23], and related work[16, 4], leveraged the processing capability of disk controllers and the relative abundance of disk drives attached to hosts to execute computations at the data. Our system makes use of this method of achieving better parallel performance; however our system, making note of recent trends in server hardware, deviates from the original Active Disks paper in a few ways. Active Disks made the assumption that an entire file was resident on a given disk. Since HEC systems make use of distributed file systems, which fragment files in order to realize throughput parallelism, we can not make this assumption. Instead, we use the information provided by the application, which is the source of the data, to access portions of a file at data element boundaries and thereby enable access to application data without needing the entire file to be present on a given storage node. Another characteristic of the Active Disk model is that the specified processing was performed on file access. We find this aspect of Active Disks to be too limiting and do not impose it upon our system.

Google's MapReduce [11] is a very successful parallel computing system which leverages application specified information to partition input data as that data is read from a storage system. This parallelism allows MapReduce to automatically distribute the processing of large data sets and achieves impressive throughput rates on dedicated compute clusters. While this model of parallelization works well for a specific class of computations, there are aspects of

MapReduce while preclude it from being a general model for data-intensive computing. Namely, the intentionally limited programming model, which insulates programmers from needing to understand parallel programming but also limits the expressiveness that is available to the programmer.

Dryad [14] is a system that allows an application developer to use a dataflow graph to specify stages of execution and communications channels between these phases. At runtime, Dryad uses this information to execute phases on as many of the available hosts as the derived graph allows. This is another example of leveraging application specified information, in this case computation inter-relatedness, to realize parallel performance gains. The ability to programmatically express multiple stages of processing and how those phases are related is important if systems are to address workloads that go beyond simple, "embarrassingly parallel" processing of individual elements from within a file. Accordingly, our system will have a similar method for expressing the order of and relationship between phases of computation.

Other work in the area of in-situ computations[26] points to the need for application code to be easily and dynamically transferred to the servers on which it will execute (storage nodes in our case). An issue raised in this paper, which we address below, is that new security questions are raised when user-written application code is executed inside of a distributed file system.

Previous work in the area of in-situ computations[30] makes it obvious that the application developer needs a means to succinctly express notions like ordering constraints and sub-elements of application structures, which will be addressed via appropriate programming models being developed. The Wickremesinghe paper also espouses the opinion, which we share, that ensuring collocated processing doesn't interfere with the storage node's data-serving responsibilities is a top priority.

Current systems that enable in-situ computations either execute on dedicated processing nodes[14, 31], which are disjoint from the storage nodes, or impose constrained programming models on the developer [30] for the sake of enabling related functionality, in this case static code analysis for the purpose of performance management. Neither of these seems appropriate for the system that we are developing as the former requires data movement, which we are striving to minimize, and the latter places a burden on the developer in order to deliver a benefit, storage node QoS, which we can provide via much simpler and less intrusive methods.

The work that comes the closest to the model of in-situ computations that we envision is the Active Storage in Lustre work being done at PNNL [20]. This system works at the storage node level, an OSD server in their case, and allows for application code to operate on data being stored in the parallel file system. The PNNL system requires that entire files be stored on a given node if the application code is to operate on the data. This requirement precludes the writing of large files if parallel file-systems are to be used, which is a huge limitation in HEC. The lack of application specific information about the data that makes up a file also precludes a class of optimizations that we seek to capitalize on with our work.

5.3 Parallel File Systems

Scientific data-intensive HEC environments rely heavily on distributed file systems for data storage. This dependency is caused by the throughput requirements of the computations being done within these environments. Examples of distributed file systems used in scientific HEC systems include: UCSC's Ceph, Clemson's PVFS, IBM's GPFS, Panasas' PanFS and Sun's Lustre [28, 8, 25, 29, 6]. These file systems fragment data without knowledge of application data structure. This behavior results in an application either needing to ask for specific byte ranges from the file system, which will likely cause communication with or among multiple storage nodes, or the entire file having to be read from the storage system, which is even slower. By implementing our system on top of existing parallel file systems we can leverage their ability to provide high performance access to storage while adding further performance gains, via data alignment, and increased functionality, in the form of allowing in-situ data computation.

Another limiting factor of parallel file systems being used in scientific HEC is the number of hosts attempting concurrent communication with the storage nodes. The large number of hosts in the HEC systems are overwhelming storage nodes ability to service network connections reliably and efficiently. This observation is confirmed by tests performed at Agronne National Laboratory[24]. In recognition of this issue, there are already systems in-use at Argonne which coalesce I/O operations for multiple hosts within a designated representative node, which then writes the aggregated data to the parallel file system. This system of I/O forwarding is indicative of the fact that the distributed storage system cannot handle the current number of clients that are attempting to access it at the same time without performance degradation. This raises serious questions about the ability of current systems to scale in light of the increasing demands of data-intensive scientific computing coupled with the prevailing tendency to grow computational clusters by adding more nodes.

5.4 QoS and Security

Allowing user-developed code to execute within a storage system introduces new security concerns[26]. For now we are planning on using a simple security system based on the Unix ACL model to allow applications to only access files, or portions of files, that the user which that the application is running as has access to. Further investigation of this topic is deferred to future work.

We will leverage QoS work done in our lab [21, 5] as a basis for developing a holistic QoS model.The QoS enforcement will also contain provisions for dealing with unruly application code. Denial-of-service attacks will be entirely prevented via virtualization of resources on the storage node and policy based options will exist to allow parameters designed to limit total resource use by an application.

6. FUTURE WORK

There are several topics where we are deferring to a later date. Scalability as it pertains to in-situ computations, failure recovery for application code and incorporating application support to existing cluster management infrastructure are all needed for this to be a complete system. We believe that this type of a system opens up itself to novel methods of indexing and searching, which we will investigate once we have a working prototype. A

large software company has indicated interest in partnering with us to research building a limited database, similar to Stonebraker's work, on top of our system once we have completed our preliminary research.

7. CONCLUSION

Current storage systems are not meeting the needs of the scientific HEC community. The rise of data-intensive computations is exposing issues caused by the separation of processing and storage. In order to alleviate these issues and enable further advances in data-intensive scientific research a new storage system needs to be created. We have made an argument that by employing structured storage, advances in performance management, well-constructed programming models and abstract data type-inspired interfaces along with existing parallel file systems it is possible to construct a system that has better data-serving performance, enables in-situ computations and can provide the requisite reliability and robustness while allowing the storage system to scale to larger sized deployments than is currently possible.

8. REFERENCES

[1] Amazon simpleDB. http://aws.amazon.com/simpledb/.

[2] Apache couchDB. http://couchdb.apache.org/.

[3] Cassandra project. http://incubator.apache.org/cassandra/.

[4] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms and evaluation. *SIGPLAN Not.*, 33(11):81–91, 1998.

[5] D. O. Bigelow, S. Iyer, T. Kaldewey, R. C. Pineiro, A. Povzner, S. A. Brandt, R. A. Golding, T. M. Wong, and C. Maltzahn. End-to-end performance management for scalable distributed storage. In *PDSW '07: Proceedings of the 2nd international workshop on Petascale data storage*, pages 30–34, New York, NY, USA, 2007. ACM.

[6] P. J. Braam. The Lustre storage architecture. http://www.lustre.org/documentation.html, Aug. 2004. Cluster File Systems, Inc.

[7] S. A. Brandt, C. Maltzahn, A. Povzner, R. Pineiro, A. Shewmaker, and T. Kaldewey. An integrated model for performance management in a distributed system. In *4th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2008)*, Prague, Czeck Republic, July 2008.

[8] 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*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. t E. Gruber. Bigtable: A distributed storage system for structured data. In *osdi06*, Seattle, WA, November 2006.

[10] P. F. Corbett and D. G. Feitelson. The vesta parallel file system. *ACM Trans. Comput. Syst.*, 14(3):225–264, 1996.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 137–150, Berkeley, CA, USA, 2004. USENIX Association.

[12] I. C. for Information Technology Standards. SCSI Object-Based Storage Device Commands - 3 (OSD-3). Project Proposal for a new INCITS Standard T10/08-331r1, International Committee for Information Technology Standards, September 11 2008.

[13] R. Grimm, M. M. Swift, and H. M. Levy. Revisiting structured storage: A transactional record store. Technical report, University of Washington, http://pages.cs.wisc.edu/ swift/papers/tr00-04-01.pdf, 2000.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA, 2007. ACM.

[15] T. Kaldewey, A. Povzner, T. Wong, R. Golding, S. A. Brandt, and C. Maltzahn. Virtualizing disk performance. In *The IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*, St. Louis, Missouri, April 2008. Springer Journal of Real-Time Systems Award for Best Student Paper.

[16] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A case for intelligent disks (idisks). *SIGMOD Rec.*, 27(3):42–52, 1998.

[17] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yellick. Exascale computing study: Technology challenges in achieving exascale systems. Technical Report TR-2008-13, DARPA, September 28 2008.

[18] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *CLADE '08: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pages 15–24, New York, NY, USA, 2008. ACM.

[19] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou. Boxwood: abstractions as the foundation for storage infrastructure. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.

[20] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–10, New York, NY, USA, 2007. ACM.

[21] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn. Efficient guaranteed disk request scheduling with fahrrad. *SIGOPS Oper. Syst. Rev.*, 42(4):13–25, 2008.

[22] A. Povzner, T. Kaldewey, S. A. Brandt, R. Golding, T. Wong, and C. Maltzahn. Efficient guaranteed disk

request scheduling with fahrrad. In *Eurosys 2008*, Glasgow, Scottland, March 31 - April 4 2008.

[23] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 62–73, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[24] R. Ross. Meeting the needs of computational science at extreme scale. slides presented at University of California Santa Cruz, January 2009.

[25] F. Schmuck and R. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, page 19, Berkeley, CA, USA, 2002. USENIX Association.

[26] M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Evolving rpc for active storage. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 264–276, New York, NY, USA, 2002. ACM.

[27] M. Stonebraker, J. Becla, D. Dewitt, K.-T. Lim, D. Maier, O. Ratzesberger, and S. Zdonik. Requirements for science data bases and SciDB. In *CIDR Perspectives 2009*, January 2009.

[28] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: a scalable, high-performance distributed file system. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[29] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–17, Berkeley, CA, USA, 2008. USENIX Association.

[30] R. Wickremesinghe, J. Chase, and J. Vitter. Distributed computing with load-managed active storage. *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 13–23, 2002.

[31] P. Widener, M. Wolf, H. Abbasi, M. Barrick, J. Lofstead, J. Pullikottil G. Eisenhauer, A. Gavrilovsk, S. Klasky, R. Oldfield, P. G. Bridges, A. B. Maccabe, and K. Schwan. Structured streams: Data services for petascale science environments. Technical Report TR-CS-2007-17, University of New Mexico, November 2007.