Dynamic Active Storage for High Performance I/O

Chao Chen and Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, TX, USA
Email: chao.chen@ttu.edu, yong.chen@ttu.edu

***Abstract*—Many high-end computing applications in critical areas of science and technology are becoming more and more data intensive. These applications transfer large amounts of data from storage nodes to compute nodes for processing, which is costly and bandwidth consuming. The data movement often dominates the applications' run time. Active storage provides a promising solution for these applications by moving appropriate computations from compute nodes to storage nodes. The prior research has achieved considerable progress and developed several active storage models. However, the existing studies have neglected the influence of data dependence on the performance of active storage systems. This study shows that the data dependence has a critical impact on active storage, and the ignorance of dependence can lead to waste of the precious bandwidth. To address this issue in active storage, this paper also presents a new Dynamic Active Storage (DAS) architecture that analyzes the bandwidth requirement of operations, determines the applicability for active storage requests, and optimizes data layout on servers to minimize the bandwidth requirement. Experimental tests have been conducted, and the results have confirmed that the proposed DAS architecture outperforms existing active storage systems. The DAS architecture reduces the data movement caused by data dependence and improves applications' performance over existing schemes. It holds a promise for high performance I/O system in high-end computing.**

***Keywords*-dynamic active storage; active storage; high end computing; data intensive computing; parallel I/O; parallel file systems**

I. Introduction

The data volumes of many critical high-end computing (HEC) applications in science and engineering, such as astrophysics, geographic systems, climate modeling and weather forecasting, medical image processing, and high-energy physics, have grown considerably in complexity and scale. For example, the data sizes of climate modeling, combustion, and astrophysics simulations range from 100TB to 10PB [25]. Most of these applications have data-intensive components. The storage and analysis of data remain a serious bottleneck for these data-intensive applications. Although there are many reasons, the most important one is that the bandwidth between the compute nodes and the storage nodes has not improved at the same rate as the storage capacity of these systems and data requirements of applications. The data explosion demands the rapid evolution of storage technologies.

Active storage provides a promising solution to addressing the limited I/O bandwidth issue for HEC applications with growing data demands [7, 19, 25, 32]. The essential idea of active storage is to move computation close to data and to reduce bandwidth requirement by offloading appropriate data-intensive operations to storage nodes. Each active storage node is responsible for processing its local data, and then sends the results back to compute nodes for further computation. It has been proven effective in reducing bandwidth requirement and improving the system performance [7, 19, 25, 32].

The desired applications' access pattern for active storage would be that there is no dependence among data from different storage nodes, which means each active storage node does not need to request dependent data from other storage nodes when processing each local data element. However, it is hard to meet such desired situation in practice. For instance, in the fields of *Geographic Information System* (GIS) and *Medical Image Processing*, many commonly used operations, such as *flow routing*, *flow accumulation* [27] and *median filter*, always require eight neighbor data items to process each data element (except data elements on boundary), as shown in Fig. 1. Due to the fact that a file is distributed and stored in a striped manner in parallel file systems, it is possible that the data elements required for processing are distributed on different storage nodes. In this case, the data intensive operation needs to request data from other storage nodes if the file is not distributed in an ideal manner. If active storage requests are carried out as normal, it can potentially cost more bandwidth consumption than normal processing and degrade the system performance because of excessive movement of dependent data. It is desired for active storage systems to identify whether an operation is suitable to be offloaded to the storage nodes for the best system performance. If there is no performance gain or even performance degradation than moving data to compute nodes for processing, the active storage system should not offload the request for the best system performance.

It is common that successive operations share the same data dependence patterns in many data-intensive applications. For instance, the *flow-accumulation* operation always follows the *flow-routing* operation. The *flow-routing* operation generates the same size intermediate terrain image with

0190-3918/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPP.2012.22

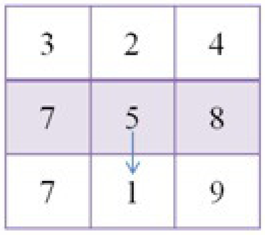


Figure 1. Example of Single Flow Direction Calculation in Flow Routing (depending on 8 neighbor data items).

each cell filled with directions, and the *flow-accumulation* operation consumes this intermediate image data to calculate how much water would flow through each cell. They both need to access 8-neighbor data elements to process the data in the central cell [2, 27]. In such situation, if the image data can be distributed in a reasonable method, the *flow-accumulation* operation would be able to avoid data movement caused by data dependence and fully leverage the power of active storage systems. There is a great need to have an active storage aware of data dependence and arranging data in a proper manner for minimizing data movement and bandwidth consumption.

In this paper, we propose a novel *dynamic active storage (DAS) architecture* to address these issues. The fundamental idea is to dynamically determine operations that are beneficial to be offloaded and processed on storage nodes by analyzing the data dependence and communication cost of operations. We have built a prototype for the proposed DAS architecture. A bandwidth prediction core is embedded in the prototype to make the decision for operations. To further improve the active storage systems, an *active storage oriented data distribution model* is integrated in the dynamic active storage systems to minimize the data movement. The experimental results have confirmed advantages of the dynamic active storage architecture over existing active storage systems.

The rest of paper is organized as follows. Section II briefly reviews related work in active storage, active disks, parallel file systems, and runtime systems. In section III, we present the proposed *dynamic active storage architecture* and discuss its design and implementation. Experimental and analytical results are presented in Section IV. Section V concludes this study and discusses future work.

II. Related Work

Extensive studies have focused on improving the performance of data-intensive HEC applications at various levels. This section discusses existing studies along three lines: disk-level improvements, file system level improvements, and runtime system improvements.

A. Disk-level Improvements for Data Intensive High-End Computing

Active disks or intelligent disks appeared in early 1990s [1, 3, 4, 8, 11, 12, 20, 23, 28] and were proposed to address I/O bottleneck issue. An active disk or intelligent disk integrated a processing unit inside the disk, and offloaded computations to embedded processing units. Both hardware architectures [3, 8, 11] and programming models [1, 20, 21] were studied to address the problem. *Keeton et al.* first introduced the intelligent disks concept for decision support systems by offloading computation to disks that has increasing processing and RAM capability [11]. *Acharya et al.* proposed a stream-based programming model for active disks [1]. *Riedel et al.* presented a detailed analysis of active disks for scan-intensive applications that search in database with large amounts of data [20]. These active disks or intelligent disks, however, are designed to explore the power of embedded processor, and have limited computation-offloading capability. It is easy to see that active storage provides a more powerful platform for the same purpose.

B. File System Improvements for Data Intensive High-End Computing

The idea of *active storage* is generated from active disks. It is proposed in the context of parallel file systems to serve the similar purpose with active/intelligent disks and has gained increasing attention [7, 13, 14, 16–19, 22, 24, 30–32]. *Sivathanu et al.* proposed an active storage prototype where servers receive the service command from clients by extending the *RPC* (Remote Procedure Call) diagram [22]. *Ma et al.* proposed an multi-view storage system architecture with virtual file system technology to provide a flexible active storage [13, 14]. *Wickremesinghe et al.* proposed a method to assign the computation to storage nodes dynamically according to available resources to achieve system load balance [30]. *Felix et al.* presented the first implementation of active storage on *Lustre* file system [7]. *Son et al.* proposed a more complete implementation of the active storage on *Parallel Virtual File System* (PVFS) [25].

The active storage system is generally more powerful than active disks or intelligent disks, and thus can offload more computations to reduce data movement. A key research issue for active storage is the communication between the compute nodes (clients) and storage nodes (servers). Most of existing studies have been concentrated on this issue, and have achieved notable progress with several active storage prototypes built.

All existing studies, however, have not considered the problem of data dependence among storage nodes, as discussed in Section I. Such problem is common in many data-intensive applications. The proposed dynamic active storage system in this paper aims to address this issue. The DAS system determines whether an operation is suitable to be offloaded on the fly. In addition, it employs a new data layout method for automatically distributing data in the way of avoiding data dependence among storage nodes.

C. *Runtime System Improvements for Data Intensive High-End Computing*

Current parallel programming models and runtime systems are primarily designed for computation-intensive applications. These programming models include Message Passing Interface (MPI) [29], Global Arrays [15], and Unified Parallel C [6]. These programming models and runtime systems focus on the memory abstractions and communication mechanism among processes. I/O is treated as a peripheral activity and often a separate phase in these programming models and runtime systems. High-end computing systems and high-throughput systems such as Condor [26] always moves data to computation and have been observed bottlenecks for data intensive applications. The recent MapReduce programming model and runtime system that move computation to data is an instant hit and have been proven effective for many data-intensive applications [5]. MATE [10] is another API based on MapReduce. Due to the programmer-managed reduction object, it has lower memory requirements at runtime, and likely to achieve better performance than MapReduce. Ex-MATE [9] provides an extension to MATE supporting disk-resident reduction objects and updating them efficiently. The MapReduce model, however, is typically layered on top of distributed file systems such as Google file system and Hadoop distributed file system. They are not designed for high performance computing semantics and analytics. MATE is designed under multi-core environment and ex-MATE is specially designed for *graph mining* applications. The proposed dynamic active storage system inherits the features of active storage and is designed specially for high performance computing systems. It is more effective than MapReduce in HPC environments.

III. Dynamic Active Storage Architecture

This section describes the proposed dynamic active storage (DAS) architecture. We first introduce the architecture and then describe the core components in detail.

A. *DAS Architecture*

In high-end computing systems, there are two ways to deploy storage and compute nodes in general. The first way is to deploy storage nodes separately from compute nodes. It can provide highly parallel I/O and is widely used in high-performance computing. The second approach is to collocate storage and compute operations on the same node, which is suitable for MapReduce/Hadoop distributed data-intensive processing model. In this paper, we focus on the first model as it is the mostly common architecture in HEC systems.

Fig. 2 illustrates the high-level view of the proposed dynamic active storage architecture. Applications interact

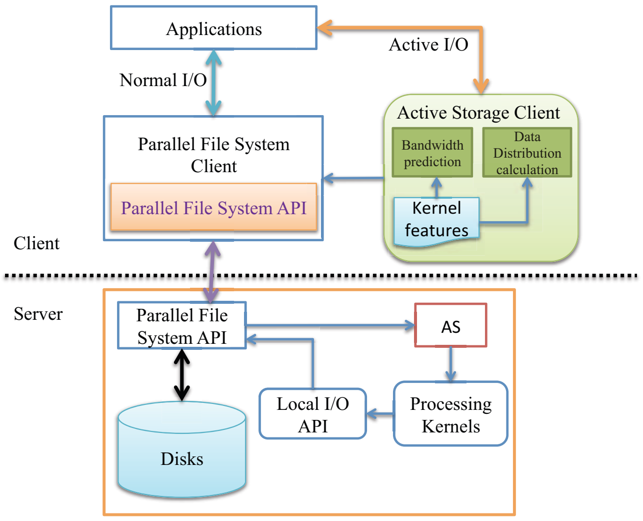


Figure 2. Dynamic Active Storage Architecture

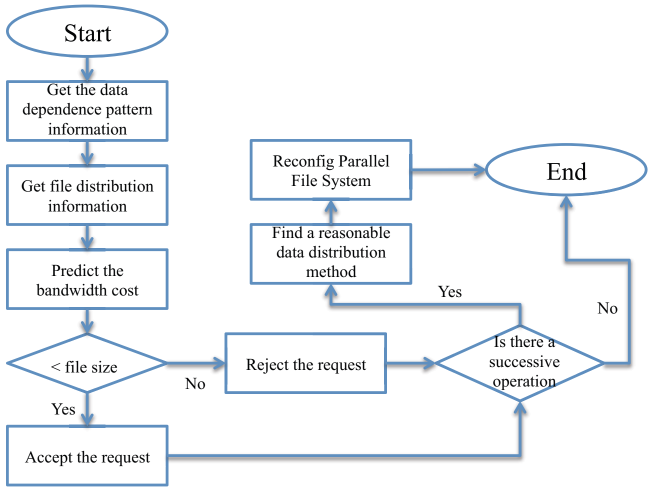
with the client of parallel file systems for normal I/O operations, and the *Active Storage Client* responds to active storage I/O requests. The *Processing Kernels* are designed as separate components and can run independently. They are invoked by the *AS* (Active Storage) component which is a helper process when an active storage I/O is served. The local I/O *API* is abstracted from parallel file systems. It provides a function that abstracts local strips as a file and reads local data for *Processing Kernels*.

The dynamic active storage system works as follows. First, applications pass active storage I/O requests to the *Active Storage Client*, and then the *Active Storage Client* begins to analyze the bandwidth cost of the operation according to the distribution information of the requested data and its data dependence pattern. If the operation requires more bandwidth than servicing it as a normal I/O operation, the request will be served as in normal instead of as an active storage request. If the request is served as an active storage request, it will be transferred to the storage nodes through improved parallel I/O *APIs*, similarly as done in [25]. Meantime, the dynamic active storage calculates an appropriate data distribution method with the consideration of data dependence and arranges the data to minimize data movement among storage servers [1]. A helper process is invoked to call the processing kernels to carry out offloaded computations. Fig. 3 demonstrates the workflow of our prototype.

B. *Kernel Features Description*

In order to predict the bandwidth cost for an operation, it is necessary to know its data dependence pattern, which describes which data elements will be required when processing an element. In our prototype, a component called *Kernel Features* is embedded in the active storage client

[1]Parallel file systems such as PVFS2 provide the required APIs.



Start
Get the data dependence pattern information
Get file distribution information
Predict the bandwidth cost
< file size
Yes
Accept the request
No
Reject the request
Is there a successive operation
Yes
Find a reasonable data distribution method
Reconfig Parallel File System
End
No

Figure 3. Workflow Chart of Dynamic Active Storage System

to identify data dependence patterns. The patterns can be implemented and represented as a plain text file or an XML file. Two sections including operator name and dependence relationship are used to describe a pattern. In general, a file can be abstracted as a *one-dimension* array of bytes in file systems. Thus the dependence can be described with offsets among mutually dependent data elements. If the dependent data is located before the processing data, the respective offset is negative; otherwise it is a positive value. The format of a record is as follows:

```
Name:operator
Dependence: offset0, offset1,
offset2 ...
```

For example, the *flow-routing* operation depends on 8-neighbor data elements. Assuming the width of a map/image is $imgWidth$, its record is as follows:

Name:*flow-routing*
Dependence: $-imgWidth + 1$, $-imgWidth$, $-imgWidth - 1$, -1, 1, $imgWidth - 1$, $imgWidth$, $imgWidth + 1$

C. Bandwidth Analysis and Prediction

In many data-intensive HEC applications, the most useful data dependence patterns are *4-neighbor* and *8-neighbor* patterns. Most operations in these applications would process the data elements through analyzing the *4- or 8-neighbor* data elements for each data element. *Flow-routing*, *flow-accumulation*, *surface slop analysis*, *digital evaluation model establishment*, *median filter*, and *2D Gaussian filter* are examples with such data dependence patterns. Fig. 1 illustrates the *flow-routing* operation on the central element. It compares the value of central element to every *8-neighbor* element and find out the element with the minimum value as the flow direction of the central element.

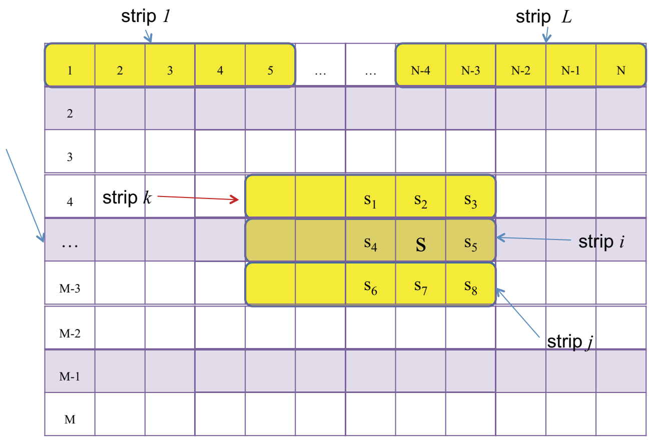


Figure 4. Data Organization of a File

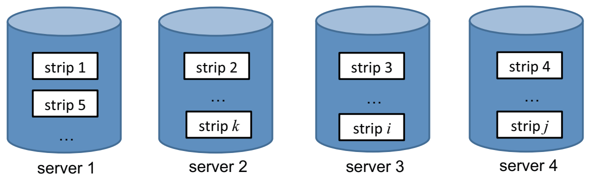


Figure 5. Data Distribution on Storage Servers

Files are stored in a striped manner in parallel file systems. A round-robin strategy is usually the default method for stripping in most parallel file systems. Fig. 4 shows how a file (e.g. a map or an image) is organized in a logical file view and divided into strips. Fig. 5 shows a possible distribution of strips among four storage nodes and how data is organized physically. As shown in Fig. 4, when operations with data dependence, such as *flow-routing*, operate on data elements S, $S_1 \sim S_8$ are needed, which means that $strip\ j$ and $strip\ k$ are needed and transferred from other two storage nodes. Due to the data dependence, such operation would require considerable data movement and bandwidth consumption, which is not desired. It is necessary to analyze the bandwidth requirement before carrying out offloaded requests. We introduce a model for predicting the bandwidth requirement through analyzing the strips distribution among storage nodes and adopt it in the DAS for identifying desired active storage requests dynamically.

The data dependence patterns of operations are obtained from kernel features' description file as discussed in the previous section. The data distribution information and strip size can be obtained from parallel file systems. With these information, our model first calculates the location of data elements and strips.

Given the data element size E (bytes) and the number of storage nodes D (numbered from $0 \sim D - 1$), for i-th ($i = 0, 1, 2, 3 \ldots$) data element in a file, its strip number and location (in the round-robin fashion) can be calculated as:

$$strip(i) = \frac{i \cdot E}{strip\_size} \quad (1)$$

$$location(i) = strip(i) \ mod \ D \quad (2)$$

where, the $strip\_size$ indicates how many bytes each strip consists of. For example, the default strip size is $64KB$ in PVFS2. The strip number of its dependent data and the location of them can be derived as follows:

$$strip(d_n) = \frac{(i + offset_n) \cdot E}{strip\_size} \quad (3)$$

$$location(d_n) = strip(d_n) \ mod \ D \quad (n = 0, 1, 2...) \quad (4)$$

The bandwidth cost for processing each element is estimated as follows:

$$bwcost = E * \sum_{j=1}^{n} a_j \quad (5)$$

$a_j = 1$ if $location(d_j) \neq location(i)$, or $a_j = 0$ if $location(d_j) = location(i)$. Thus, the total data movement would be $\sum_{j=1}^{n} a_j$ times of data size.

However, from the above equations, we can find that if $\frac{offset_n \cdot E}{strip\_size} \ mod \ D = 0 (n = \{0, 1, 2 \ldots\})$, all the dependent data elements would be located on the same storage node, and it will not require extra data movement if the operation is offloaded. Otherwise, the offloaded operation would need to request dependent data from other storage nodes when processing each data element. To better explain this critical problem, we illustrate it with an example. Assuming that when processing a data element, the operation depends on two data elements, and the distance between them is $stride$ elements, as shown in Fig. 6. When processing element l, elements m and n are required. The relationship among the locations of these three elements is as follows:

$$m = l - stride \qquad (m \geq 0) \quad (6)$$

$$n = l + stride \qquad (n \leq filesize) \quad (7)$$

The strips where elements l, m and n are located can be calculated by the following equations:

$$strip_l = \frac{l \cdot E}{strip\_size} \quad (8)$$

$$strip_m = \frac{m \cdot E}{strip\_size} \quad (9)$$

$$strip_n = \frac{n \cdot E}{strip\_size} \quad (10)$$

The storage nodes where elements l, m and n are located

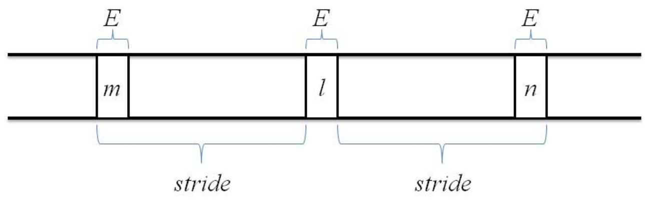


Figure 6. Data Dependence

can be calculated with the following equations:

$$L_l = \frac{l \cdot E}{strip\_size} \ mod \ D \quad (11)$$

$$L_m = \frac{(l - stride) \cdot E}{strip\_size} \ mod \ D \quad (12)$$

$$L_n = \frac{(l + stride) \cdot E}{strip\_size} \ mod \ D \quad (13)$$

From Eq. 11, Eq. 12 and Eq. 13, we can observe that if $\frac{stride \cdot E}{strip\_size} \ mod \ D = 0$, then $L_l = L_m = L_n$, which means that these three elements reside on the same storage server. In other words, the dependent data is located on the same storage server. In this case, offloading operations would not require extra data movement among storage servers. Otherwise, if $\frac{stride \cdot E}{strip\_size} \ mod \ D \neq 0$, it means that dependent data and strips need to be transferred from/to other storage servers. The total bandwidth cost can be estimated from Eq. 5. These equations are simple but effective, and are used to guide the DAS for an effective active storage system.

D. Improved Data Distribution

Based on the data dependence and bandwidth requirement analysis, we know that whether an operation is appropriate to be offloaded to storage servers has a close relationship with how the data is distributed among storage servers. If a reasonable data distribution method is adopted, and mutually dependent data is located on the same storage node, the dynamic active storage system can reduce the bandwidth cost considerably. An example of data distribution for reducing the data movement is demonstrated in Fig. 7.

In Fig. 7, strips S_1, S_4 and S_6 are located on the same storage node to reduce the data movement caused by dependence. Nevertheless, it is unavoidable that S, S_2 as well as S_7 would be located on another storage node. These two storage nodes are neighbors as shown in Fig. 8. Thus, strips l, m and n need to be transferred from $storage \ p$ to $storage \ q$ when processing element S. Reversely, strips i, j, k need to be transferred from $storage \ q$ to $storage \ p$, when processing element S_4. The DAS employs a replication strategy to avoid these data transfers, as shown in Fig. 9, an improved data distribution for Fig. 8. Two extra copies of strips are stored in successive storage nodes. Therefore, no additional data transfer is required when each storage node processes its local data elements. A potential concern of this strategy is the storage capacity consumption. In this

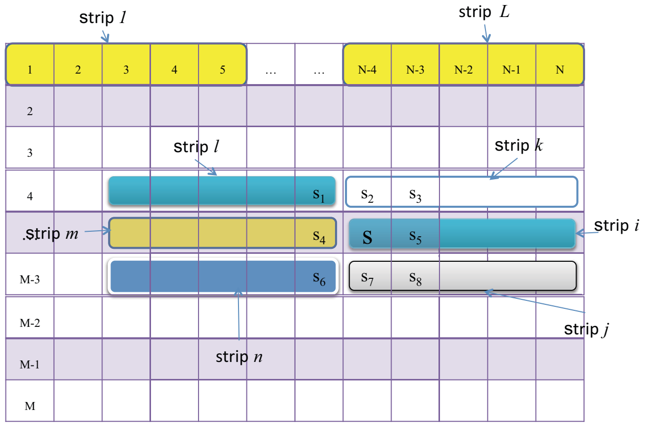


Figure 7. Improved Strip Distribution in DAS

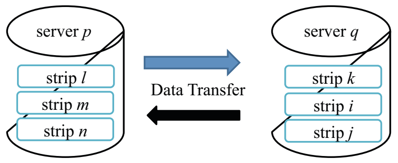


Figure 8. Data Distribution

example, it requires twice of extra storage space than the normal case. Although storage capacity grows drastically and is usually not a bottleneck any more, it could be a limited resource for data-intensive applications. To mitigate this issue, the DAS manages to store r successive strips in the same storage node. Meanwhile, the $1st$ strip is also stored in a previous storage node as a replicated copy, and the rth strip is stored in the next storage node similarly. Thus, the location of strips in Fig. 4 can be calculated as following:

$$L_l = \frac{l \cdot E}{r \cdot strip\_size} \; mod \; D \tag{14}$$

$$L_m = \frac{(l - stride) \cdot E}{r \cdot strip\_size} \; mod \; D \tag{15}$$

$$L_n = \frac{(l + stride) \cdot E}{r \cdot strip\_size} \; mod \; D \tag{16}$$

The storage capacity consumption is reduced to $\frac{2}{r}$ compared to a straightforward method. Based on the enhanced data distribution method, the criteria of offloading operations

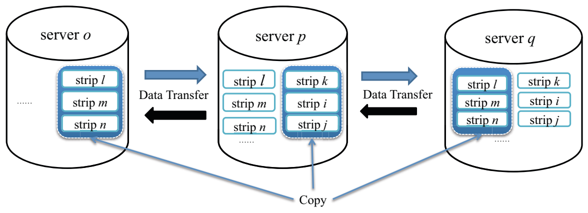


Figure 9. Improved Data Distribution

becomes as:

$$\frac{stride \cdot E}{r \cdot strip\_size} \; mod \; D = 0 \tag{17}$$

IV. Experimental Results and Analysis

We have carried out extensive experimental tests to verify the potential of the proposed dynamic active storage. This section presents the experimental settings and experimental results.

A. Experimental Settings

This subsection describes our experimental platform, the evaluated schemes, and the benchmarks we have used in our experiments.

1) Evaluation Schemes: To demonstrate the effectiveness of our proposed dynamic active storage system, we compared three schemes:

- Traditional Storage (TS). In this scheme, the servers are responsible for normal I/O operations. The analysis kernels are executed on the clients.
- Normal Active Storage (NAS). In this scheme, the data is distributed with normal round-robin pattern. The kernels are employed and executed at the server side, with each node processing its local data. When dependent data needed, it has to acquire them from neighbor server nodes, which is required by current active storage systems.
- Dynamic Active Storage (DAS). In this scheme, we implement the proposed dynamic active storage with the enhanced data distribution method.

2) Benchmarks: To evaluate the effectiveness of our approach, three data analysis kernels were used in the tests as shown in Table. I. These kernels were chosen because they are widely used in the areas of *GIS*, *Medical Image Processing* as well as *Data Mining*. For example, *flow-routing* and *flow-accumulation* are two basic operations used in *Terrain Analysis*. *2D Gaussian Filter* are usually used in *Medical Image Processing* to smooth the images. These operations are representative operations in data-intensive applications.

3) Experimental Platform: The platform we have used to perform experimental evaluation is the *Hrothgar* cluster at the *Texas Tech University*. The *Hrothgar* consists of 640 nodes, and each node is equipped with Intel(R) Xeon(R) 2.8GHz CPUs (12 cores per node) and 24GB memory. It has 600TB *Lustre* parallel file system storage. In the experimental tests, we were allocated 60 nodes. Part of these nodes were used to simulate the active storage nodes, and the rest was used as compute nodes. The ratio between active storage nodes and compute nodes can be configured, and the default ratio is 1 : 1. With this configuration, NAS, DAS and TS would have the same computation capability. Thus, it will show the influence of data dependence and data transfer clearly.

Table I
DESCRIPTION OF DATA ANALYSIS KERNELS

Name	Description
Flow-routing	Basic operation of terrain analysis application from *GIS*. It produces distinctive spatial and statistical patterns depending on the maximum number of downslope cells to which flow could be directed
Flow-accumulation	Another basic operation of terrain analysis application from *GIS*. It calculates accumulated flow as the accumulated weight of all cells flowing into each downslope cell in the output raster.
2D Gaussian Filter	Basic operation of signal and medical image processing. It takes the raw data as input and output the same size smoothed data

B. Experimental Results

This subsection presents the experimental results. We begin our analysis by focusing on the impact of our scheme on overall execution time, and then extend to the scalability analysis with increasing the number of nodes and data set size. At last, we evaluate and compare the bandwidth achieved of each scheme.

1) Performance Improvement: The first set of experiments we have carried out analyzes the influence of data dependence on the performance for active storage systems. In this experiment, 24 nodes were used and 12 of them were configured as storage nodes. The rest of nodes was configured as compute nodes. Fig. 10 reports the execution time of *flow-routing*, *flow-accumulation*, and *2D Gaussian Filter* tests with NAS and TS schemes respectively. The figure clearly shows that the performance of NAS is much lower than TS in these tests. The performance degradation of NAS is attributed to the following two reasons. First, because of the data dependence, each active storage server not only needs to conduct offloaded computations, but also serves the requests from other storage nodes (for the dependent data), which unavoidably increases the load of each active storage server. Second, the data dependence causes extensive data movement traffic among storage servers in the existing NAS scheme. Each strip was transferred multiple times among the storage nodes, because dependent data had to be transferred to local for processing. Clearly the current active storage schemes have drawbacks because of the ignorance of data dependence among active storage servers. The dynamic active storage architecture is proposed to address this issue.

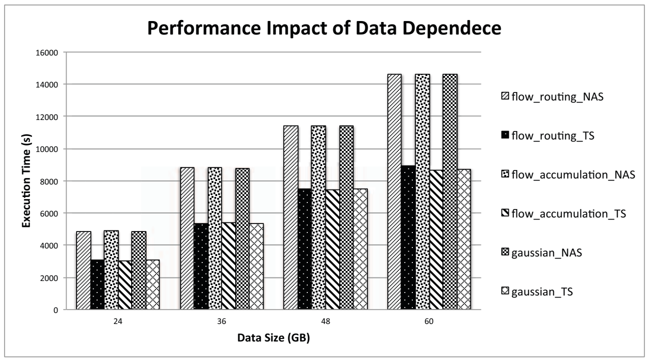


Figure 10. Comparison of Execution Time for NAS and TS Schemes.

Fig.11 reports and compares the execution time of all three schemes, NAS, TS, and the proposed DAS. These tests were conducted with $24GB$ data size. Similarly, 24 nodes with half configured as storage nodes and the other half as compute nodes. It could be observed that the proposed DAS achieves the best performance, with over 30% improvement compared to TS and 60% improvement compared to NAS. This performance improvement is the result of the awareness of data dependence in the proposed DAS architecture and the improved data distribution. The DAS scheme not only helps to avoid the excessive data transfer among storage nodes, but also reduces the workload of storage nodes for serving other nodes' requests. The proposed dynamic active storage system considers the data dependence and addresses the limitation of existing active storage systems well.

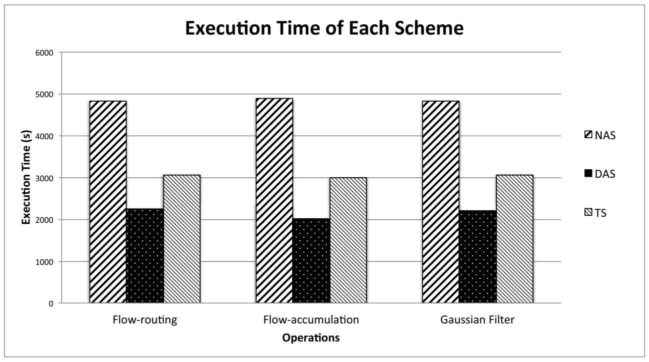


Figure 11. Comparison of Execution Time for NAS, DAS and TS Schemes.

2) Scalability Analysis: We have also carried out tests with varying the number of nodes and data sizes to study the performance and scalability of the proposed DAS architecture. The tests were conducted with 24, 36, 48, and 60 nodes, respectively, with data sizes ranging from 24GB to 60GB.

We first present the case when the dataset size was increased. In this set of experiments, we tested all three schemes to compare their performance with the same system configuration as in previous experiments. The data size was increased from $24GB$ to $60GB$. Fig. 12 shows the execution time of operations with each scheme respectively. It can be observed that DAS has the lowest increase of execution time when the data size was increased with $12GB$. The detailed analysis shows that the execution time of DAS increased only 15% on average, while the execution time increased over 30% with NAS and TS schemes. The reason is that, when data accesses become intensive for a given system resource, the NAS scheme or TS scheme causes intensive data movement traffic either among storage nodes or between compute nodes and storage nodes, whereas the DAS addresses this issue.

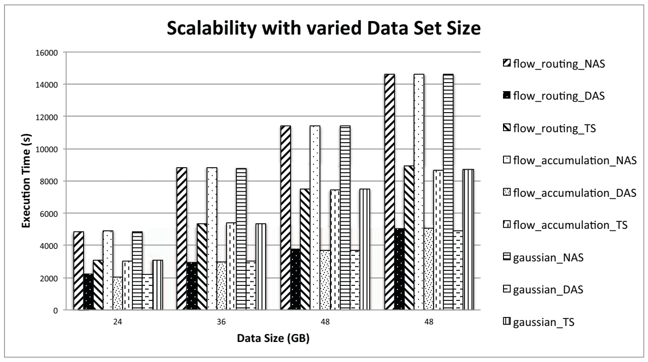


Figure 12. Comparison of Execution Time of NAS, TS, and DAS Schemes as Data Size Increased

In addition to varying data set sizes, we have also measured and compared the performance of DAS and TS with varying the number of nodes. Fig.13 shows the execution time of DAS and TS schemes with the number of nodes ranging from 24 to 60. The data size was fixed at 60GB. As observed from the results, both DAS and TS schemes are scalable with a large number of nodes. They share a similar scalability trend with the execution time reduced by about 15%, when the number of nodes was increased with 12 nodes. The DAS scheme is scalable as verified when the system size increased.

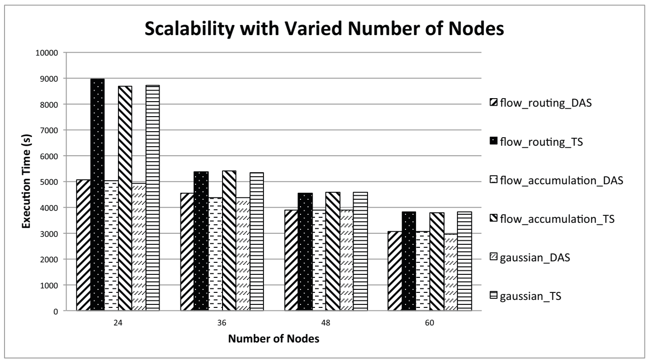


Figure 13. Comparison of Execution Time when the Number of Nodes Increased

3) Bandwidth Analysis: Fig. 14 plots the sustained bandwidth comparison among three schemes, NAS, DAS, and TS. This set of tests was conducted with the *flow-routing* operation under the same configuration as in the second set of tests. We observe that the bandwidth improvement with the DAS scheme is much higher than that of NAS and TS. The analysis shows that the DAS scheme improved the sustained bandwidth by nearly one fold when compared to the TS scheme. The experimental tests confirmed that the proposed dynamic active storage considerably improved the sustained bandwidth. It reduces excessive data movement traffic and improves the execution time of data-intensive HEC applications.

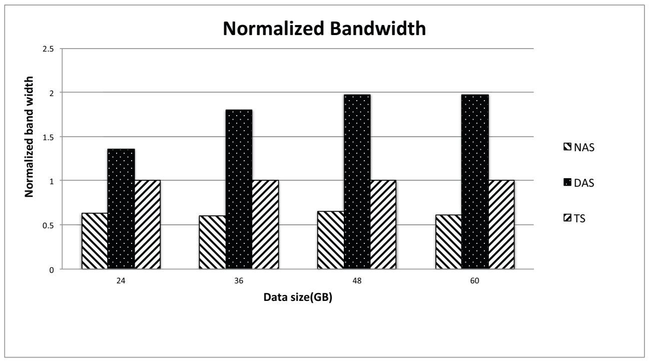


Figure 14. Normalized Sustained Bandwidth Improvement

V. Conclusion and Future Work

Poor I/O performance has been a bottleneck in many parallel computing systems and data-intensive high-end/high-performance computing applications. Active storage provides a promising solution for these applications by moving appropriate computations from compute nodes to storage nodes. Prior research, however, mainly focused on how to offload operations to the storage nodes, and ignored the importance of data dependence in operations. The ignorance of the data dependence can cause costly data movement and degrades the system performance.

This paper proposes a *Dynamic Active Storage (DAS)* architecture considering the data dependence issue in active storage systems. The contribution of this study is three-fold. First, we demonstrate that the data dependence in operations offloaded to storage servers have a clear impact on the system performance. Second, we present a new DAS system to address this issue. The DAS has a bandwidth requirement analysis component embedded and makes the offloading decision dynamically on the fly. An improved data distribution method is adopted in the DAS to minimize the data movement. Third, we have built a prototype and evaluated the proposed DAS with representative processing kernels. The results show that the proposed DAS scheme outperforms existing active storage architectures.

As many high-end computing applications tend to be highly data intensive than ever before, a high performance I/O system is critical and can significantly improve the productivity of applications. This study advances the active storage systems, and also calls for further research on dynamic, access-aware, and intelligent storage solutions to meet the growing data demands.

Acknowledgment

The authors acknowledge the High Performance Computing Center (HPCC) at Texas Tech University at Lubbock for providing HPC resources that have contributed to the research results reported within this paper. URL: http://www.hpcc.ttu.edu. This research was supported in part by the Texas Tech University 2011 Growing Graduate Program.

References

[1] A. Acharya and J. Saltz. Active Disks : Programming Model , Algorithms and Evaluation. In *Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, 1998.

[2] J. F. O. Callaghan and M. D. M. The Extraction of Drainage Networks from Digital Elevation Data. *Computer Vision, Graphics and Image Processing*, 8:323–344, 1984.

[3] S. Chiu, W.-k. Liao, and A. Choudhary. Design and Evaluation of Distributed Smart Disk Architecture for I/O-Intensive Workloads. In *Proceedings of International Conference on Computational Science*, 2003.

[4] G. Chockler and D. Malkhi. Active Disk Paxos with infinitely many processes. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[5] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.

[6] T. El-Ghazawi and L. Smith. UPC: Unified Parallel C. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, 2006.

[7] E. J. Felix, K. Fox, K. Regimbal, and J. Nieplocha. Active Storage Processing in a Parallel File System. In *6th LCI International Conference on Linux Clusters: The HPC Revolution*, Chapel Hill, North Carolina, 2005.

[8] M. Franklin, R. Chamberlain, M. Henrichs, B. Shands, and J. White. An Architecture for Fast Processing of Large Unstructured Data Sets. In *Proceedings of the IEEE International Conference on Computer Design*, 2004.

[9] W. Jiang and G. Agrawal. Ex-mate: Data intensive computing with large reduction objects and its application to graph. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 475–484, 2011.

[10] W. Jiang, V. Ravi, and G. Agrawal. A map-reduce system with an alternate api for multi-core environments. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 84 –93, may 2010.

[11] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Record*, 27(3):42–52, 1998.

[12] H. Lim, V. Kapoor, C. Wighe, and D. Du, H.-C. Active Disk File System: a Distributed, Scalable File System. In *18th IEEE Symposium on Mass Storage Systems and Technologies (MSS '01.)*, pages 101–116, San Diego, CA, USA, 2001. IEEE Computer Society.

[13] X. Ma and A. L. N. Reddy. MVSS : Multi-View Storage System. In *21st International Conference on Distributed Computing Systems*, pages 31 – 38, Mesa, AZ , USA, 2001.

[14] X. Ma and A. N. Reddy. MVSS : An Active Storage Architecture. *IEEE Transactions on Parallel and Distributed Systems*, 14(10):993–1005, 2003.

[15] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: a Portable "Shared-memory" Programming Model for Distributed Memory Computers. In *SC'94*, pages 340–349, 1994.

[16] J. Piernas and J. Nieplocha. Efficient Management of Complex Striped Files in Active Storage. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, 2008.

[17] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of Active Storage Strategies for the Lustre Parallel File System. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007.

[18] L. Qin and D. Feng. Active Storage Framework for Object-based Storage Device. In *20th International Conference on Advanced Information Networking and Applications*, pages 97–101, 2006.

[19] B. Rich and D. Thain. DataLab: Transactional Data-Parallel Computing on an Active Storage Cloud*. In *IEEE/ACM High Performance Distributed Computing*, 2008.

[20] E. Riedel, G. Gibson, and C. Faloutsos. Active Storage For Large-Scale Data Mining and Multimedia. In *In Proceedings of the 24rd International Conference on Very Large Data Bases(VLDB '98)*, New York, NY, USA, 1998.

[21] E. Riedel, G. A. Gibson, and D. Nagle. Active Disks for Large-Scale Data Processing. *IEEE Computer*, 2001.

[22] M. Sivathanu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Evolving RPC for Active Storage. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, volume 37, New York, NY, USA, 2002. ACM.

[23] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *FAST*, 2003.

[24] C. W. Smullen, S. Rohinton, S. Gurumurthi, P. Ranganathan, and M. Uysal. Active Storage Revisited :

The Case for Power and Performance Benefits for Unstructured Data Processing Applications. *Proceedings of the 5th conference on Computing frontiers*, 2008.

[25] S. W. Son, S. Lang, P. Carns, R. Ross, and R. Thakur. Enabling Active Storage on Parallel I / O Software Stacks. In *26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[26] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[27] L. Toma, R. Wickremesinghe, L. Arge, J. S. Chase, and J. S. Vitter. Flow computation on massive grids. In *Proceedings of the 9th ACM international symposium on Advances in geographic information systems*, 2001.

[28] M. Uysal, A. Acharya, and J. Saltz. Evaluation of Active Disks for Decision Support Databases. In *Proceedings of 6th International Symposium on High-performance Computer Architecture*, pages 337–348, 2000.

[29] E. L. W. D. Gropp and R. Thakur. *Using MPI-2*. MIT Press, 1999.

[30] R. Wickremesinghe, J. S. Chase, and J. S. Vitter. Distributed Computing with Load-Managed Active Storage. In *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing(HPDC-11)*, 2002.

[31] Y. Xie, D. Fengt, and D. D. E. Long. Design and Evaluation of Oasis : An Active Storage Framework based on TIO OSD Standard. In *27th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2011.

[32] Y. Zhang and D. Feng. An Active Storage System for High Performance Computing. *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, pages 644–651, 2008.