Multilevel Active Storage for Big Data Applications in High Performance Computing

Chao Chen
Department of Computer Science
Texas Tech University
Lubbock, TX, 79409
Email: chao.chen@ttu.edu

Michael Lang
Los Alamos National Laboratory
Los Alamos, NM, 87544
Email: mlang@lanl.gov

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, TX, 79409
Email: yong.chen@ttu.edu

***Abstract*—Given the growing importance of supporting data-intensive sciences and big data applications, an effective HPC I/O solution has become a key issue and has attracted intensive attention in recent years. Active storage has been shown effective in reducing data movement and network traffic as a potential new I/O solution. Existing prototypes and systems, however, are primarily designed for read-intensive applications. In addition, they generally assume that offloaded processing kernels have small computational demands, which makes this solution a poor fit for data-intensive operations that have significant computational demands, including write-intensive operations. In this research, we propose a new Multilevel Active Storage (MAS) solution. The new MAS design can support and handle both read- and write-intensive operations, as well as complex operations that have considerable computational demands. Experimental tests have been carried out and confirmed that the MAS approach is feasible and outperformed existing approaches. The new multilevel active storage design has a potential to deliver a high performance I/O solution for big data applications in HPC.**

***Keywords*—Big Data, active storage, high performance computing, data-intensive computing, parallel file systems**

I. INTRODUCTION

Many scientific applications tend to be more and more data intensive today due to the requirements of scientific discoveries. Sloan Digital Sky Survey (SDSS) and Arctic Systems Reanalysis (ASR) [9] are two of examples that respectively generates 100TB and 23.14TB data in each run. The data size of these applications will continue to increase in the near future. The large volume of data undoubtedly provides great opportunities for scientific discoveries. However, it also brings great challenges to the high performance computing community. Providing a high performance I/O system is one of the well-known challenges, and has attracted intense attention with several notable approaches proposed.

Among existing approaches, *Active Storage* has shown great promise in addressing the I/O bottleneck issue for scientific applications with growing data demands. However, existing active storage prototypes [6, 14] primarily focus on supporting read-intensive operations. As it is easy to identify the common operations, such as a lookup, among different data analysis applications; and the analysis kernel are normally predefined in a library called *Processing Kernels* [14] as a service of parallel file system. In contrast, the technique for write-intensive applications, which are certainly common in scientific areas, is not well studied. With the rapid increase of the output size, write performance of I/O systems becomes highly important. But, existing designs of active storage cannot carry out write operations effectively, because: 1). "processing kernel" design pattern is not suitable for write-intensive operations due to the difficulty of identifying common operations from different write-intensive applications; 2). write-intensive operations normally require more computations than read-intensive operations, and storage nodes can not afford enough resources for these operations. For example, Resolving dynamic equations for climate model is clearly more complex than the lookup operation. However, currently employed HPC systems show that limited computational resources(14% at Hrothgar Cluster at Texas Tech) can be leveraged for active IO service.

Motivated by the limitations discussed above, we propose a new framework, called Multilevel Active Storage(MAS). The active I/O requests are served in two levels. The first level is a storage level, which is exactly the same with the idea from existing works, and leverages compute resources of each storage node to conduct simple operations. The second level is at data nodes that are connected to the storage nodes through high-speed networks, serves for complex operations that require more computations with dedicated resources. The MAS also provides a mechanism to execute the user-implemented operations at the data nodes, which makes it more flexible than existing works. It is a significant extension to existing prototypes.

The rest of this paper is organized as follows. Section II and Section III present the design of the Multilevel Active Storage and its evaluations respectively. Section IV discusses the related work and compares with this research. Section V concludes this study.

II. MULTILEVEL ACTIVE STORAGE SYSTEM DESIGN

The Multilevel Active Storage (MAS) aims to extend the existing active storage prototypes to support both read-intensive and write-intensive operations highly effectively. In this section, we will introduce the design of the MAS.

A. MAS Architecture

Figure 1 illustrates the system architecture of the proposed MAS. In our design, MAS consists of three layers: 1) the top layer contains compute nodes for running application codes; 2) data nodes at the second layer are used by the MAS to serve

978-1-4799-1293-3/13/$31.00 ©2013 IEEE

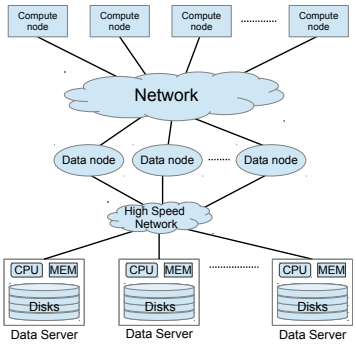

Fig. 1. System Architecture of MAS. The system has three layers connected with different network and serves as different functions

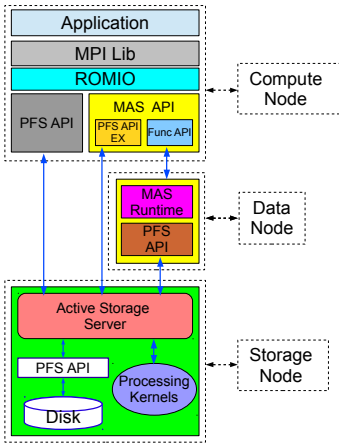

Fig. 2. Software Stack of MAS

active I/O requests; and 3) storage nodes at the bottom layer primarily serve normal I/O requests.

Similar to existing active storage prototypes, the MAS also leverages the available resources at each storage node to serve active I/O requests requiring minor computations. Because this level is closest to the data, it has the minimum data movement in the system. Simple operations that require little computations, such as lookups, maximum and minimum value calculations, could be offloaded and benefit from active I/O service of this level.

A number of data nodes are introduced as the second layer to serve user-defined complex operations. Data nodes provide more computation resources for active I/O requests. The operations requiring significant computations can be offloaded to this layer. A run-time system is designed to manage the data nodes. It provides users an interface to offload and execute arbitrary kernels on the data nodes. A subset of data nodes are designed as master nodes that manage the rest of data nodes. Compute nodes need to communicate with the master nodes to determine which data node the operation should be offloaded to. The master nodes work simultaneously to avoid the possible communication bottleneck.

MAS is a middle-ware between the MPI-IO library and parallel file systems. It contains two fundamental components: *MAS API* and *MAS Runtime*. The *MAS API* encapsulates *PFS_API_EX* and *MAS_Func_API* for applications. The *PFS_API_EX* is designed to extend parallel file system APIs. Applications can use these APIs to access the storage level active I/O services. The *MAS_Func_API* is designed for users to access data node level active storage services. It enables transferring I/O requests to MAS runtime on data nodes. User-defined operations will be serviced at this level.

B. MAS API

MAS API is an extension to the MPI-IO library. This design allows applications to access the MAS services through an enhanced MPI-IO library with minimal code changes. Similar to the work in [14], the enhanced MPI-IO API is defined as *MPI_File_Op_Ex()* (the *Op* could be *read* or *write*). The new APIs derive all of the arguments used in traditional MPI-IO APIs, and introduce additional two arguments to indicate the operations and the active storage level, in which the operation should be served. Different from the design in [14], MAS uses operation names to identify the offloaded operations, and an additional argument, *op_type*, to indicate the active I/O service level. If *op_type* is set to 0, the active I/O will be served at the storage node, and MAS API will call the related API defined in *PFS_API_EX* to transfer the I/O request to the storage node directly. Otherwise, if it is set to 1, the active I/O will be served at data nodes and the MAS API will call the API defined in *MAS_Func_API* to transfer the request to data nodes. The details of the API design are shown in Table I. It compares the APIs of work [14], the traditional MPI-IO, and the MAS.

TABLE I. A COMPARISON BETWEEN MPI_FILE_READ_EX() AND MPI_FILE_READ()

```
/* API design in previous work [14] */
MPI_File_read_ex(MPI_File fh, void *buf, int count, MPI_datatype, int operation, MPI_Status * status);

/* API design in this work */
MPI_File_read_ex(MPI_File fh, void *buf, int count, MPI_datatype, char * operation, bool op_type, MPI_Status * status);
MPI_File_write_ex(MPI_File fh, void *buf, int count, MPI_datatype, char * operation, bool op_type, MPI_Status * status);

/* API design in traditional MPI-IO */
MPI_File_read(MPI_File mpi_fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status);
```

C. MAS Runtime

The MAS runtime is supported on data nodes to help executing the user-defined operations. Making the system intelligently identify and offload the related operations is an ideal solution. However, it is very challenging to design these systems. MAS leverages additional tools to address this issue. For instance, the programmer can acquire the necessary information by using external tools, such as IOSIG+ [2] and Darshan [1]. IOSIG+ and Darshan can aid in tracing and analyzing the activities of the I/O operations in an application, which allows the programmer to gain insight to determine which operations need to be offloaded.

The MAS only requires programmers to define and implement the offloaded operations as functions in a separate source file. A new command option, -MAS, is added to the *mpicc* script. When the programmer using *mpicc* to compile the applications, they can use -MAS option to tell the *mpicc* compiler

which file contains the definition of offloaded operation. Then, *mpicc* would automatically compile the designated file into a lib file. At the same time, it will also generate a configuration file containing parameter information. An example is given in PART I and PART II of Table II. The user defines the offloaded function prototype as in Part 1. MAS does not require the function prototype to have a fixed number of parameters. However, given a function, the number of parameters is fixed. Then, the mpicc compiler will generate a configuration file according to the function prototype by listing all of its parameters one by one, as shown in PART II. This configuration file will be further used to generate the code framework executed on the data nodes, as shown in PART III.

When the MAS runtime receives a request, it will first execute a script to generate a piece of code to encapsulate the user-defined operation with the main function, as shown in PART III of Table II, and then execute another script to compile the newly generated file and link the lib file compiled by the *mpicc* in the previous step to generate another executable file. Finally, it will fork a process to invoke the offloaded operation. All of this work is hidden from programmers. Users only need to implement and run a code as they do in conventional platforms. MAS will compile, offload, and execute the defined and implemented function automatically.

TABLE II. EXAMPLE OF AUTOMATICALLY GENERATED CODE

PART I: /* Function definition */
```
void func(MPI_File fh, int var1, char *var2, int var3, int var4, ..., int *outbuf);
```

PART II: /* Generated configuration file */
```
func     — function name
MPI_File    — file handle 4        — number of arguments
int      — the type of first argument
char *   — the type of second argument
int      — the type of third argument
int      — the type of fourth argument
.....
int *    – output buf for results returned to compute processes
```

PART III: /* the framework of generated code */
```
#include <stdlib.h>
...
int main(argc, argv) {
    ...
    MPI_File fh_data;
    memcopy = (fh_data, argv[1], ...);
    int var1 = atoi(argv[2]);
    char *var2;
    memcopy(var2, argv[3], ...)      .....
    int *buf = argv[n];

    func(fh_data, var2, var3, var3, var4, ..., buf);
    return 0;
}
```

D. Communication between compute nodes and data nodes

The MAS runtime executes the offloaded operations by compiling and forking it from separate executable files. Normally, a function will have input parameters as initialization data for that function. Therefore, how to transfer the initialization data to the offloaded operations executed on data nodes becomes a challenge for the MAS framework.

The MAS addresses this challenge through three steps. First, the MAS requires the programmer to define the arguments in the function definitions that implement the offloaded operations. All of the initialization data will be transferred to the operation through function arguments. Second, the *buf* and *MPI_Datatype* arguments in the extended MPI-IO API are utilized to transfer the data to the data nodes. The programmer needs to use MPI_Datatype to encapsulate all the arguments used in the function with the same order that appears in the function definition. The initialization data will then be transferred to the data nodes through the *buf* argument. Third, when the MAS runtime receives an I/O request, it will also receive the initialization data. With the help of the configuration file generated during the compilation, the MAS can de-encapsulate the initialization data into separate arguments. At the same time, it will use *argc* and *argv* arguments to initialize the arguments in function definition. As a result, when it executes the operation, it will pass the initialization data to the executed process through *argc* and *argv* arguments.

III. EVALUATION

A. Experiment Setup

Experimental evaluations were performed on the DISCFarm cluster at the Texas Tech University. DISCFarm contains 16 nodes with a total of 32 processors and 128 cores. These nodes are connected fully via 1Gigabits Ethernet Ports. Parallel Virtual File System (PVFS2) is deployed. In our default set for the MAS, there are 2 PVFS2 storage nodes, 4 data nodes, and 8 compute nodes. Three metrics including the performance (execution time), bandwidth, and scalability were evaluated in this study.

We evaluated three operations, including data assimilation, summation, and lookup. Data assimilation is a widely used operation in climate sciences. The climate scientists use it to generate reliable climate data sets. Several assimilation algorithms are commonly used, and the Ensemble Kalman Filter (EnKF) [8] is one of notable algorithms. In this study, the EnKf was used for evaluations. It consists of 6 matrix multiplications, 1 matrix addition, and 1 matrix substitution. The summation (sum) and lookup operations are simpler as compared to the data assimilation operation. Both of them are widely used in climate data analysis applications and many others. A sum operation calculates the total value of all specified data elements, and the lookup operation searches for and returns all elements that meet given criteria from a large volume of data sets.

The performance of MAS is compared against normal active storage and traditional storage. The configuration for each of them is listed as following:

- Multilevel Active Storage (MAS): This scheme is an implementation of the proposed multilevel active storage prototype. Data-intensive operations can be offloaded to the storage nodes of parallel file systems or dedicated data nodes. The default set is initially configured. As the performance potential of storage level nodes (normal active storage solution) has been extensively evaluated in previous works, we mainly focus on the evaluation of the data nodes level.
- Active Storage (AS): Operations are offloaded to the storage nodes. There are no data nodes. Twelve nodes of the DISCFarm cluster were configured as compute nodes and two nodes were configured as PVFS2 storage nodes, unless explicitly mentioned otherwise.

- Traditional Storage (TS): It has the same configuration with AS, except that analysis code is executed at compute nodes but not storage nodes.

B. Results and Analyses - Advantages and Limitations of Active Storage

The overall performance, I/O system bandwidth, and scalability were evaluated in our experiments. In this subsection and following subsections, we present the evaluation results and the related analysis. We first present the analysis of advantages and limitations of active storage in this subsection, and then compare the overall performance and the bandwidth of MAS, AS, and TS. At last we present the scalability analysis of the MAS.

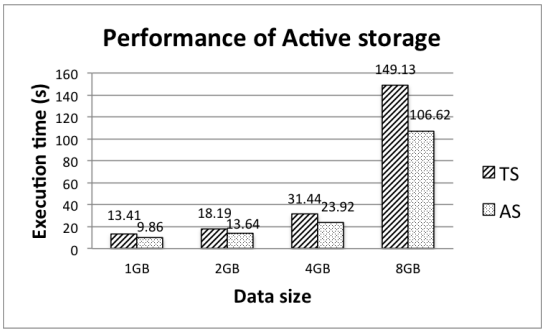

Fig. 3. Performance comparison of active storage and traditional storage (SUM operation, Observed CPU usage: 1.2%)

Figure 3 shows that for simple operations, SUM, Active storage outperforms the Traditional storage systems. It can complete the same-size tasks with less execution time. In our experiment, there are 12 storage nodes for TS to execute the SUM operation, while there are only 2 nodes for AS. This promising result mainly comes from the reduced data movement.

However, the storage nodes normally simultaneously serve for different I/O requests from different processes, which will occupy amount of resources, including CPU and memory, as observed in practice Hrothager cluster. In this section, we evaluate the impact of CPU usages. To achive this we increase the CPU usage using a simple program and then observe and compare the performance of AS against TS. Figure 4 plots the performance improvement of active storage under different CPU usages of storage nodes for LOOKUP operation. LOOKUP has least computations, and could be least affected by the CPU usages among the evaluated operations. However, Figure 4 shows that the active storage achieved better performance than the traditional storage when the CPU usage of storage nodes is under 42%. When the CPU usage increased to 63%, the improvement of active storage decreased drastically. It even had worse performance than the traditional storage when the CPU usage of storage nodes arise to more than 83%. These limitations of the normal active storage are due to the fact that its design only utilizes available computation resources on storage nodes. When the CPU usage on storage nodes is high or offloaded operations have significant computational demands, the normal active storage performs inadequately.

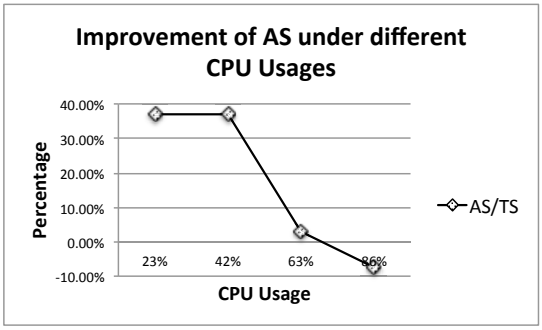

Fig. 4. Limitation of active storage (Lookup operation)

C. Results and Analyses - Performance of Multilevel Active Storage

The performance of the MAS is evaluated using a more complex operation, EnKF, in this subsection. As we have presented the performance of storage node level active storage, we focus on the data node level and the overall evaluation in the rest of this paper. EnKF is dominated by 6 matrix multiplications (the entire operation consists of 6 matrix multiplications, 1 matrix addition, and 1 matrix substitution). The computation requirements are substantially larger than SUM and LOOKUP operations. Figure 5 compares the execution time with the TS, AS, and MAS with different data sizes. From the results, we can find that the MAS achieved the best performance. It effectively reduced the execution time of the EnKF operation. Compared to the TS and AS, the average performance improvement of the MAS were 30% and 19% respectively, as shown in Figure 6.

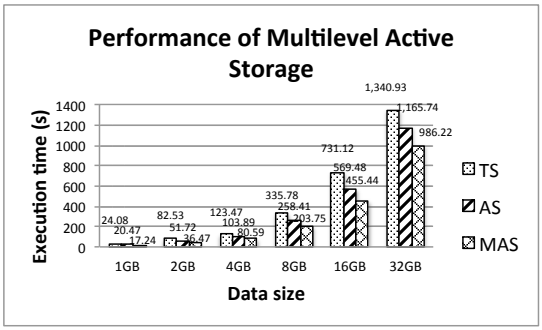

Fig. 5. Performance comparison of traditional storage, AS and MAS (EnKF operation, Observed CPU usage: 1.3%)

The results proves that the MAS can achieve better performance than the normal active storage for complex operations. These complex operations are certainly common in practical HPC systems, especially for write-intensive HPC applications. The value of a multilevel active storage system design resides in that it handles much more practical HPC workloads than prior studies and prototypes that handle light-computation kernels only such as SUM and LOOKUP operations.

Figure 7 reports the execution time of the EnKF operation under different CPU usages of storage nodes. These results show that, for the EnKF operation, the MAS improved the performance by 21% compared to the traditional storage. However, the normal active storage performed even worse than the traditional storage as given complex workloads and

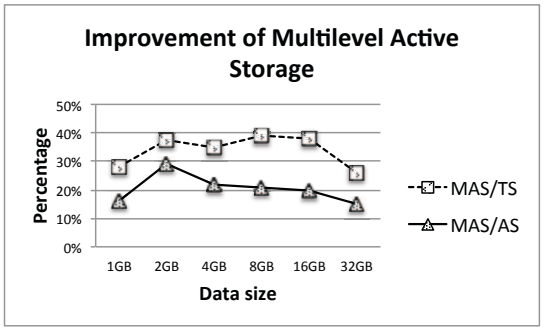

Fig. 6. Performance comparison of the TS, AS and MAS (EnKF operation, Observed CPU usage: 1.3%)

higher CPU usage on storage nodes (with 21%, 43%, 51%, and 83% CPU usages respectively). The benefits of the MAS came from two aspects: more computational and balanced resources for offloaded data-intensive operations than the normal active storage, and less data movement and network congestion than the traditional storage system.

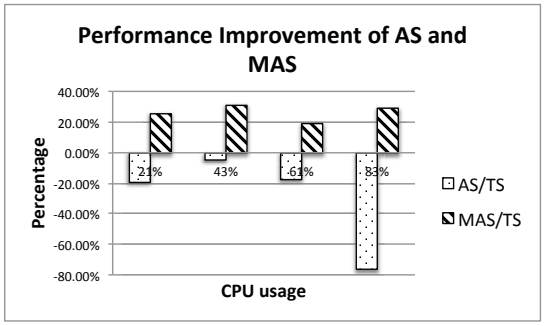

Fig. 7. Performance improvement comparison of AS and MAS under different CPU usages (EnKF operation)

D. Results and Analyses - Sustained Bandwidth Improvement

In this section, we evaluated the sustained I/O network bandwidth with the EnKF operation. As our interest is to compare the bandwidth usage and improvement of the multilevel active storage against the traditional active storage, the evaluation focused on these two schemes. The comparison results of these two schemes are plotted in Figure 8 with varying data sizes. The MAS outperformed the normal active storage in all cases, and the average I/O bandwidth improvement against the AS was around 1.3 times. This improvement further increased with more CPU resources of storage nodes occupied by other processes, which has been illustrated in Figure 7.

E. Results and Analyses - Scalability

In this section, we evaluated the scalability of the MAS. In the experiment, we fixed the ratio between data nodes and storage nodes as 2 : 1 but varied the number of data nodes and storage nodes, which means that the number data nodes increased with the number of storage nodes (with reduced number of compute nodes). Figure 9 reports the execution time of the EnKF operation. These results clearly show that the execution time was decreased steadily with increasing the number of data nodes, which confirmed the scalability of the

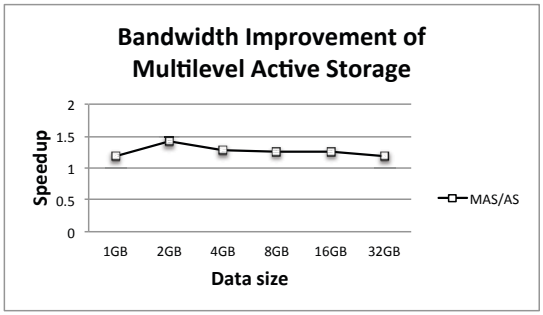

Fig. 8. Bandwidth improvement of the MAS (EnKF operation, CPU usage: 1.3%)

multilevel active storage (on a reasonable platform we evaluated). The trend suggests that the MAS can scale reasonably well and can achieve stable performance improvement given larger system sizes.

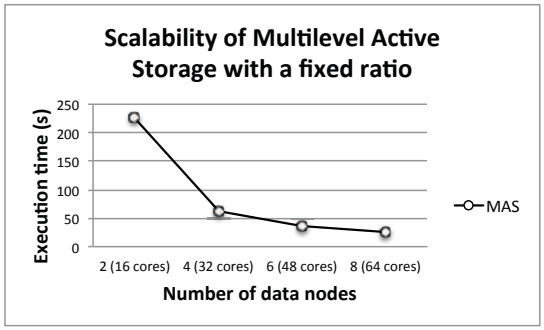

Fig. 9. Scalability of the MAS with a fixed ratio between data nodes and storage nodes (EnKF operation, ratio: 2 : 1)

IV. Related Work

High performance I/O and storage solutions have attracted intensive attention in recent years given the growing importance of supporting data-intensive/BigData applications. Many notable investigations were conducted and various approaches were proposed.

A. I/O Library Level Improvement

Collective I/O [13], data sieving [12, 13], and numerous enhancements [8, 36] are important I/O library level optimizations for a high performance I/O system. They can help to reduce the redundant data items and combine small size I/O requests into larger ones to reduce the communication cost. However, for large-scale or extreme-scale applications, only relying on collective I/O and data sieving can not deliver a high performance I/O solution. One important reason is that the data movement quickly limits the performance and the scalability becomes a critical concern. The proposed multilevel active storage in this study can address these issues efficiently. It reduces the data movement and bandwidth consumption and has a promising potential for high performance I/O.

B. File System Level Improvement

Active disk and active storage are two approaches that improve the I/O performance at the file system level and

closely related to this study. Active disk [3, 11] aims to reduce the data movement between the disk and memory by offloading appropriate computations to the embedded CPU on the disks. However, these studies are designed to explore the power of embedded processors, and the approaches they explored have limited computation-offloading capability. Meanwhile, Active storage [6, 10, 14] explores the computation resources of storage nodes where parallel file systems reside on. Felix et. al. [6] presented the first real implementation of active storage on the Lustre file system; Woo et. al. [14] proposed a more complete implementation on the Parallel Virtual File System (PVFS). These studies have shown the promise of active storage approaches. Blue Gene Active Storage [7] is a recent system that is developed from active storage prototypes. However, all these systems were designed with an assumption of small computation kernels, and primarily designed for read-intensive applications, whereas write-intensive applications with considerable computational demands are not well handled. This study advances the state-of-the-art of the active storage and employs a multilevel design to address issues of current active storage systems.

C. Runtime Level Improvement

Current HPC system runtime and programming models, such as Message Passing Interface, Global Arrays, and Unified Parallel C [5], are primarily designed for computation-intensive applications. They mainly focus on the memory abstraction and provides a rich set of methods to access distributed memories. These models and runtime always move data to computation, which has been criticized as a less-efficient approach for data-intensive applications. The recent MapReduce [4] programming model and runtime system that move computations to data is an instant hit and have been proven effective for many data-intensive applications. 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. The proposed MAS inherits the features of active storage, and is designed specially for high performance computing systems and applications.

V. Conclusion

Undoubtedly, scientific discoveries and innovations can benefit considerably from large-volume collected instrument data and simulation data. Domain scientists can gain insights and understand the phenomenon behind the data more effectively, but based on an assumption that the HPC system can deliver an effective I/O solution. In prior research, numerous approaches, such as collective I/O, data sieving, and active storage/disks, were studied. In this research, we extend prior active storage prototypes and propose a new Multilevel Active Storage. The MAS shares a similar idea with normal active storage, but it provides a more comprehensive framework that can serve both read- and write-intensive operations well. With dedicated data nodes, it handles complex operations that require considerable computational demands too. In addition, the MAS supports the execution of both predefined kernels and user-defined operations, which is more flexible. The initial results have shown that the MAS has a promising potential for data-intensive/BigData applications. In the near future, we plan to continue investigating active storage, programming model, and runtime solutions, to better support big data applications in HPC.

VI. Acknowledgment

This research is sponsored in part by the National Science Foundation under the grant CNS-1162488. 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 in this paper. URL: http://www.hpcc.ttu.edu.

References

[1] Darshan Project. http://www.mcs.anl.gov/research/projects/darshan/.

[2] IOSIG Project. http://www.cs.iit.edu/~scs/iosig.

[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] J. Dean and S. Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51:107–113, 2008.

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

[6] 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.

[7] B. G. Fitch, A. Rayshubskiy, M. P. T.J. Chris Ward, B. Metzler, H. J. Schick, B. Krill, P. Morjan, and R. S. Germain. Blue Gene Active Storage. In *HEC FSIO R&D Workshop '10*, 2010.

[8] P. L. Houtekamer, H. L. Mitchell, and L. Mitchell. Data Assimilation Using an Ensemble Kalman Filter Technique, 1998.

[9] NOAA. Overview of Current Atmospheric Reanalysis. http://reanalyses.org/atmosphere/overview-current-reanalyses, 2012.

[10] 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.

[11] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *2nd USENIX Conference on File and Storage Technologies*, pages 73–88, 2003.

[12] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. Passion: Optimized I/O for Parallel Applications. *Computer*, 29(6):70–78, June 1996.

[13] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In *Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, FRONTIERS '99, pages 182–, Washington, DC, USA, 1999. IEEE Computer Society.

[14] S. Woo, S. Samuel, L. Philip, C. Robert, and R. Rajeev. Enabling Active Storage on Parallel I/O Software Stacks. In *26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.