MS-I/O: A Distributed Multi-Storage I/O System

Xiaohui Shen and Alok Choudhary*
Core Technology Dept, Personal Communication Sector, Motorola Inc
1000 Technology Way, Libertyville, IL 60048, axs095@email.mot.com

Abstract

More and more parallel applications are running in a distributed environment to take advantage of easily available and inexpensive commodity resources. For data intensive applications, employing multiple distributed storage resources has many advantages. In this paper, we present a Multi-Storage I/O System (MS-I/O) *that can not only effectively manage various distributed storage resources in the system, but also provide novel high performance storage access schemes. MS-I/O employs many state-of-the-art I/O optimizations such as collective I/O, asynchronous I/O etc. and a number of new techniques such as data location, data replication, subfile, superfile and data access history. In addition, many MS-I/O optimization schemes can work simultaneously within a single data access session, greatly improving the performance.*

Although I/O optimization techniques can help improve performance, it also complicates I/O system. In addition, most optimization techniques have their limitations: they are not working for all situations and simply putting them together may not work well. Therefore, selecting accurate optimization policies requires expert knowledge which is not suitable for end users who may have little knowledge of I/O optimizations. So the task of I/O optimization decision should be left to the I/O system itself, that is, automatic from user's point of view. We present a User Access Pattern *data structure which is associated with each dataset that can help MS-I/O easily make accurate I/O optimization decisions.*

1 Introduction

Data intensive applications have presented challenging problems to computational scientists. A major problem is the I/O performance when many datasets are stored and frequently accessed. Over the years, scientists have developed techniques for I/O optimizations such as collective I/O [14, 38, 23, 10], prefetching [15, 13], data sieving [38, 10], caching [8] and so on. But the performance problem still persists because data intensive applications are generating more data and the usage of these datasets is more frequent than ever. In addition, the advantages of distributed systems have changed the traditional parallel computing environment toward employing multiple distributed compute resources over the network [2, 18, 17, 16]. As far as I/O system is concerned, employing multiple inexpensive and easily available *storage resources* in I/O system is a natural step toward a distributed environment for data intensive computing. Designing such an I/O system is a challenging task. The I/O system should provide first, an efficient way to manage various local and remote resources in the system; Second, an easy-to-use interface that observes user's programming practice; and finally and the most importantly, high performance. User's requirements may vary greatly from application to application which demands the I/O system be flexible enough to handle all the situations. Incorporating I/O optimization schemes already been developed and finding new opportunities are important tasks for such I/O systems.

Another important issue is how to make an accurate decision among various I/O optimization candidates. It is not that simple to achieve performance gain by turning on all the optimization schemes and simply putting them together. For example, if the application's compute part is insignificant, aggressive prefetching may not improve the performance since there is little chance to overlap I/O and computation. Another example is that if user's access pattern conforms to the data's layout on storage, collective I/O may incur extra internal communication overhead, so it is better not to use collective I/O. In addition, we believe making an I/O optimization decision should not be the task of the application developers because first of all, user's focus is the application itself, she should not be diverted to the performance problems of her applications; second, making decisions among various optimization candidates requires expert knowledge, and therefore, it is not a trivial task. Unfortunately, the existing I/O systems do require users with such kind of expertise. For instance, the MPI-IO [38, 37] provides more than 30 I/O functions, how to choose a suitable function among them is not an easy task. Therefore, the decision of I/O optimizations should be left to the I/O system itself. On the other hand, the user has the best knowledge of how her dataset will be stored and accessed, so user's involvement in decision making is very important. The key

*Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, choudhar@ece.nwu.edu

here is that user's involvement should be high-level, concerning no underlying optimization techniques and specific I/O functions.

Embarking on the ambitious goals stated above, we make the following contributions in this paper:

- Present a *Multi-Storage I/O System* (MS-I/O) architecture in a multiple storage resource environment.
- Present MS-I/O API and its meta-data management mechanisms.
- Present a *User Access Pattern* structure to help MS-I/O automatically make accurate I/O optimization decisions.

The remainder of the paper is organized as follows. In Section 2 we introduce the system architecture of our Multi-Storage I/O system. We first present the reasons for incorporating multiple storage resources in I/O system, followed by describing each component of MS-I/O architecture. In Section 3 we introduce the database tables which is one of important parts in MS-I/O and user programming interface. In Section 4 we present the *user access pattern* structure and various optimization candidates. How these optimizations are selected according to user's access pattern are also presented. The performance numbers are presented in Section 5. We first introduce our experimental environment, followed by experiment results for a number of experiments. In Section 6, the related work is presented. Finally we conclude the paper in Section 7.

2 System Architecture of MS-I/O

2.1 Advantages of Multiple Storage Resource Architecture

In a traditional computing environment, the compute resource is tightly coupled with local file systems, i.e. using local disks as data storage. As data intensive applications, especially large-scale scientific applications may have very large storage space requirement, people have to employ large storage system such as tertiary storage systems (HPSS[13, 20], UniTree[40]) and large database systems (Oracle). In addition, these large storage resources may no long tightly coupled with local compute resource: they could be distributed over wide area network. A major concern of accessing these storage resources is performance. I/O (remote I/O) evaluation and optimizations are more important than ever in such an environment. Both traditional I/O optimizations and new I/O techniques should be employed. We have built a run-time library (SRB-OL)[30] that provides a variety of state-of-the-art optimizations for tertiary storage access (HPSS). Although these optimizations can significantly improve the performance compared to naive approaches, further improvement is impeded by the physical nature of storage media. For example, the tape system such as HPSS requires a minimum of 20 to 40 seconds to be ready to move the data and data transfer rate is very slow compared to disks. The popular I/O optimizations such as collective I/O, data sieving and so on can not eliminate this overhead when the data resides on tapes. In general, the remote large storage archival systems suffer from large data access latency, while, on the other hand, the local fast storage systems suffer from limited storage capacity. So the users have to satisfy the capacity requirement at the cost of loss of performance requirement. We think this dilemma is rooted in the traditional *single storage resource architecture*. In this architecture, the application has only one storage resource available for storing user's data. The performance improvement would saturate even if many state-of-the-art optimizations are applied. We believe incorporating multiple storage resources in an I/O system is a promising solution to address the tough performance problems discussed above. In this *Multi-Storage Resource Architecture*, an application can be associated with multiple storage resources that could be heterogeneously distributed over networks. These storage resources could include local disks, local databases, remote disks, remote tape systems, remote databases and so on. In short, any kind of storage media can be added to this architecture. The advantages of employing multiple resources are three-fold.

- First of all, it increases the logical storage capacity of the system. The total available storage space can be significantly increased by adding as many storage resources in the system as possible.
- Second, *multi-storage resource system* provides a more flexible and reliable computing environment. For example, failure of one storage system may not impede the computation when other resources are available. It is very common that the remote large storage system is down for system failure or maintenance, so that the experiment can not go on in a single storage resource architecture. A *multi-storage resource system*, however, can help the user avoid relying on one storage resource for computing.
- Finally, as far as the performance is concerned, a *multi-storage resource system* can provide new opportunities for further performance improvement. With multiple storage resources coupled with the simulation, the application can speculatively store the datasets to the 'best' storage media favorable for the post-processing programs such as data analysis and visualization thereafter. For example, if the user wants to carry out visualization on a specific dataset just after it is generated,

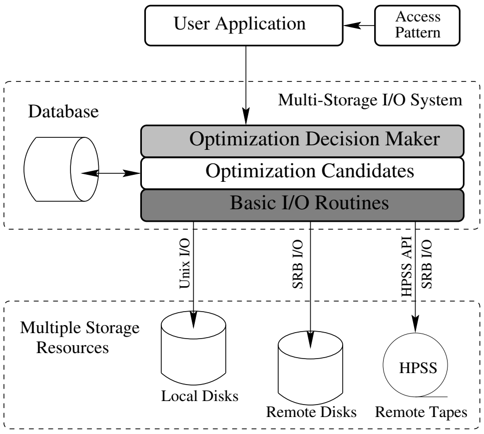

Figure 1. Architecture of MS-I/O.

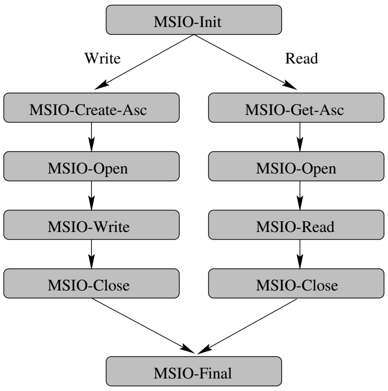

Figure 2. A typical MS-I/O flow.

she can suggest the system write this dataset locally (if it does not exceed local storage capacity.), where visualization tools are installed and all other recently unused datasets to the remote large storage system for permanent archival. So her visualization tools can directly read data from local disks rather than go all the way to the remote tapes as in a *single storage resource architecture*. In general, each generated dataset has its purpose of usage and the *multi-storage resource architecture* allows this purpose to be best implemented (in terms of performance) by storing it on a suitable storage medium, which is impossible in a *single storage resource architecture*. In other words, this architecture can combine the capacity advantage of remote large storage system and the performance advantage of local fast storage system.

2.2 MS-I/O Architecture

The main task to exploit multiple storage resources discussed previously is to build a *Multi-Storage I/O system* (MS-I/O). This I/O system should provide an easy-to-use and uniform user interface, as well as storage selection transparency. The design and implementation of our Multi-Storage I/O system observe these requirements. Figure 1 shows the architecture of Multi-Storage I/O system. On the top is the user application. At the bottom are various storage resources such as local disks, remote disks, remote tapes and so on. MS I/O, which sits between user application and storage resources, is a middleware that optimizes the data flow between user application and storage resources. MS-I/O consists of four components: Basic I/O routines, Optimization Candidates, Databases and Optimization Decision Maker. The functions of these components are described as follows.

- **(1) Basic I/O routines** It provides basic and native I/O interfaces to various storages. For example, the I/O interface to local disks is usually UNIX I/O routines, access to the remote disks is via Storage Resource Broker (SRB) [6, 32, 5]. For the remote tape systems such as HPSS, the interface could be HPSS API or SRB. These native interfaces to storage access are not optimized for parallel and distributed data access.

- **(2) Optimization Candidates** This layer provides many state-of-the-art optimization schemes including new techniques we developed recently. These optimizations include collective I/O, prefetching, subfile, superfile, asynchronous I/O, data location selection, data replication, data access history and so on. One or several schemes together can be applied to a data access. The decision is made by Optimization Decision Maker.

- **(3) Optimization Decision Maker (ODM)** The task of decision maker is to make decisions on which optimization candidate/candidates should be utilized for a data access. It is possible that multiple optimization schemes can be applied simultaneously to a single data access. The decision correctness is important since a wrong decision may actually hurt the performance. To make an accurate decision, ODM needs more information about user's usage of her datasets. These information could come from two sources in our MS-I/O: (1) user's access pattern (Section 4) which describes data's current and future usage, and (2) database which keeps the data's access history. Knowing the past, present and future of a data's access, MS-IO is able to make I/O optimization decision accurately and automatically. While in other I/O systems, the user has to deal with optimization manually. This not only adds extra burden to the user and even worse, user's decision may not be accurate due to lack of knowledge.

- **(4) Databases** Incorporating database in I/O systems has many advantages. First of all, it provides an easy way to use and manage multiple resources in the system. For example, large-scale scientific applications may generate huge number of data files, effectively managing file names and locations are a hard task. By keeping these information in database, the user can simply specify a dataset name which is convenient and natural to her, and let the I/O system searches the database to find file names, locations and paths etc. Second, as far as performance is concerned, database keeps user's data access history and this information can help make an optimization decision.

3 Database Tables and MS-I/O Routines

The main tables in MS-I/O database include run table, shared data attribute table, dataset table, execution table and performance table. Their functions are summarized as follows.

- **Run table** Records basic information of each run of the application. The information includes location (compute) of the experiment, problem size, I/O frequency, date and time when the experiment starts and so forth.
- **Dataset table** Keeps dataset names involved each run. It also has an attribute to store the encoded user access pattern for each dataset.
- **Shared data attribute table** Keeps shared data attributes for the datasets with the same association. These data attributes include data dimension, data size, I/O mode (read/write), data types and so on.
- **Execution table** Records I/O activities of the experiment. For each I/O activity, records the storage location of the I/O, file name, path, offset, optimization policy used and so on.
- **Performance table** Records the timings of the experiment, including total execution time, total I/O time, total compute time. As we are more interested in the I/O performance, the breakup of the total I/O time according to storage resources is also provided: they are local disk I/O time, remote disk I/O time and remote tape I/O time. So after experiment, the performance statics are automatically been collected. By inspecting the performance table, the user can easily find where the performance bottleneck is.

Figure 2 shows the basic MS-I/O flow. MS-I/O is easy to learn and use because first of all, the design and implementation of MS-I/O observes UNIX programming convention, the user does not need to change her programming practice; second, a MS-I/O routine may involve database access, but it is transparent to the user; finally, it presents users with a uniform interface although various I/O optimization schemes are employed in the system. While in other I/O systems such as MPI-IO, users have to explicitly choose different I/O routines to achieve high performance data access. The functions of MS-I/O routines and their relationship to databases are summarized as follows.

(1) MSIO-Init Initialize the global environment. It also fills out run table when the experiment starts.

(2) MSIO-Create-Association For those datasets sharing the same attributes such as data size, dimension, I/O mode(read/write) and data types etc, create an association id for them, insert a row into shared data attribute table. Meanwhile, records each associated dataset in dataset table.

(3) MSIO-Get-Association Get the shared data attribute information before a read operation.

(4) MSIO-Open Open a MS-I/O object. The optimization decisions are also made here. The decisions include which storage resource should be responsible for the data access, whether collective I/O, asynchronous I/O etc. should be used or not. If the open is for read, the execution table will be consulted to find the file name, path, location and so on.

(5) MSIO-Write and MSIO-Read Perform actual I/O operation to the selected storage resource. Each I/O operation is recorded in the execution table.

(6) MSIO-Close Close the opened MS-I/O object.

(7) MSIO-Final Terminate a MS-I/O session. Put the performance numbers in the performance table.

4 User Access Pattern for Optimization Decisions

4.1 User Access Pattern

One of major tasks of MS-I/O is to automatically select I/O strategies. On the other hand, user's involvement in decision making is crucial since only the user has the complete picture of how and when her data will be stored and accessed. But user's involvement should be high-level, that is, she only describes the features of her data usage, concerning no low-level details of optimizations.

We developed a data structure called *User Access Pattern* to help users provide data usage hint. The user access pattern is associated with each dataset and is passed to MS-I/O which then makes optimization decisions.

The key for MS-I/O to obtain a better picture of the usage of a dataset is that the user provides the information of her data usage as early as when the data is created. This can help MS-I/O place the data on a better storage and in a better way for future access. So the user access pattern includes access patterns for both write and read operations. Table 1 shows the MS-I/O access pattern and its influence on the optimization decisions.

Note that all the fields in Table 1 are natural to the user: it only concerns the usage of the data, so it does not require users any I/O optimization knowledge. One feature of this write access pattern is that it includes user's future usage information such as when the data will be used (*WhenAccess*), how frequently it will be used (*AccessFrequency*) and how large of it will be used (*FutureReadSize*). Providing this kind of information can greatly help MS-I/O better place the data favoring future usage (read)[1].

4.2 Optimization Decision Making

The optimization schemes employed by MS-I/O include some new approaches developed in MS-I/O such as data location, subfile[27], superfile[2] and so on, as well as powerful well-known optimizations such as collective I/O, asynchronous I/O etc. MS-I/O tries to combine multiple I/O optimizations in a single I/O session automatically. In this sub-section, we present how optimization decisions are made by MS-I/O according to user's access patterns.

For most optimization candidates, the decision is straightforward. For example, to decide whether collective I/O should be used, MS-I/O needs only to check whether the data partition is consistent with the data's storage layout: if not, the collective I/O is applied. Table 2 shows the conditions to turn on an optimization candidate. The only complicated decision is from *data location*, which decides a storage resource responsible for storing a dataset, since the decision is influenced by quite a few factors such as $WriteSize$, $WhenAccess$ and $AccessFrequency$, which

Table 2. Decision Making Strategies.

Optimizations	Conditions
`Collective I/O`	$DataPartition <>$'B**'
`Data Location`	$L = S + W + F$
`Asynchronous I/O`	$ComputeTime =$'large'
`Subfile`	$WriteSize =$'huge' and $FutureReadSize =$'partial'
`Superfile`	$WriteSize =$'small' and $WriteSequence =$'yes'
`Data Replication`	Data Location is 'remote disks' or 'remote tapes', and current $AccessFreqency =$'frequent'
`Data Access History`	Depending on access history, an optimization may be applied

may suggest conflicting results. For example, a small $WriteSize$ may suggest a local disk; but if it will not be accessed frequently, a remote storage resource such as a remote disk or a remote tape would be more appropriate. To make a reasonable trade-off among conflicting access pattern fields, we developed a linear algorithm by properly assigning positive and negative numbers to access pattern fields. For example, the data location L can be given by

1. $L = S + W + F$;
2. $if L < 0$ then $L = 0$;
3. $if(L > 2)$ and $(L <> Infinite)$ then $L = 2$.

Where the values of S, W and F are as follows: $S(WriteSize)$ = 0 ($small$), 1 ($medium$), 2 ($large$) or 3 ($huge$); $W(WhenAccess)$ = -1 ($soon$), 1 ($long$), Infinite ($never$); $F(AccessFrequency)$ = -1 ($frequent$), 1 ($seldom$), Infinite ($never$). So the value of L can be 0 (local disks), 1 (remote disks), 2 (remote tapes) or Infinite (no storage). The trick of the above algorithm is that we assign negative values to S and F when they prefer a fast storage medium. For example, if a dataset is large ($S = 2$), but it will be accessed shortly ($W = -1$) and frequently ($F = -1$), then the combined result is $L = S + W + F = 2 - 1 = 0$. So a local disk is chosen to place the data. On the other hand, if a dataset is small ($S = 0$), but it will be seldom accessed ($F = 1$) and not used recently ($W = 1$), then the remote tape is selected ($L = S + W + F = 0 + 1 + 1 = 2$).

Another significant feature of MS-I/O is that many optimization candidates in MS-I/O can coexist in a single I/O session, so the performance improvement is more significant than using only one single optimization scheme found in many other I/O systems. For example, data location, collective I/O, asynchronous I/O and data replication can cowork in a single I/O access session.

[1]One question may arise for this user access pattern: how to define large, small for data size and frequent or seldom for data use frequency etc. We believe that there should be no absolute values to differentiate them. Just like in natural language, the user decides these values according to her own usage of data. Different users may specify different values even for the same application with the same problem size, but that is fine because the key of these hints is to make 'hot' data on a faster storage no matter what the real size and use frequency are.

[2]Due to the relative high overhead of creating/opening files in remote storage systems such as remote disks and remote tapes, they are not suitable for storing large amount of small files typically found in digital library systems. We created *Superfile* concept to handle this type of limitation. The basic idea is that when these small files are created, we transparently concatenate these small files into one large *superfile*. Later on, when the user accesses these small files, the first read call will bring the whole large *superfile* into main memory, so the subsequent read requests to other files can be served directly from main memory, eliminating multiple remote data accesses.

Table 1. User Access Pattern Data Structure for Read and Write Operation.

Field	R/W	Description	Values	Influence
`DataPartition`	R+W	How the data is partitioned among processors. A *BBB* means the data is partitioned in a (Block, Block, Block) way among processors for a three dimensional array.	BBB, B** BB, B* etc.	collective I/O
`WriteSize`	W	How large the dataset will be generated.	huge,large medium, small	data location, subfile,superfile
`WriteSequence`	W	Whether there are a sequence of data files (time steps) for this dataset will be generated.	yes/no	superfile, asyn I/O
`WhenAccess`	W	When this dataset will be used (read).	soon,long, never	data location, data duplication
`AccessFrequency`	R+W	How often this dataset will be accessed.	frequent, seldom,never	data location data duplication
`ComputeTime`	R+W	Whether the compute time is a significant part.	large, small	asyn I/O
`FutureReadSize`	W	How large of the dataset will be accessed.	whole/partial	subfile
`FutureReadSequence`	W	Will a sequence of data files will be accessed.	yes/no	asyn I/O superfile
`ReadSize`	R	How large of the dataset will be accessed.	whole/partial	subfile
`ReadSequence`	R	Will a sequence of data files will be accessed.	yes/no	asyn I/O superfile

5 Performance Evaluation

In this section, we present performance numbers. We first introduce our experimental environment and applications, followed by experiment results.

5.1 Experimental Environment

The compute resource is an IBM SP-2 located at Center for Parallel and Distributed Computing (CPDC) of Northwestern University. Each node has 128 MB memory and 2 GB disk. The internal communication bandwidth is 24MB/sec. The storage resources (Figure 1) include:

- **Postgres Database** This database of MS-I/O is installed on a Linux machine at CPDC of Northwestern University.
- **Local Disks** Local disks here refer to the disk storage associated with SP-2. Local disks are the most popular traditional storage resource for saving user's data files. The basic I/O interface to local disks is UNIX I/O routines.
- **Remote Disks** The remote disks in our environment is located at San Diego Supercomputer Center (SDSC). We use SDSC's Storage Resource Broker (SRB) [6, 32, 5] as native interface to remote disks. Compared to local disks, remote disks have both larger storage capacity and data access latency.
- **Remote Tapes** The remote tape system we use in our environment is High Performance Storage System (HPSS) [13]. The remote tapes have very large storage space and we assume it can hold any size of data. But the cost to access tape-resident data is extremely expensive. So it is suitable to store large, permanent and infrequently used data. The native interface to HPSS could be SRB or HPSS internal API, we also use SRB as native interface in this work.

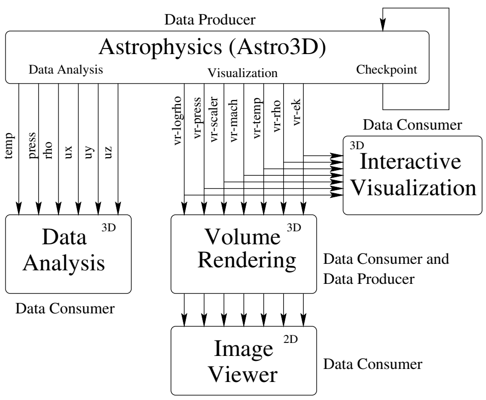

Figure 3. Data Flow of Applications.

5.2 Applications

Figure 3 shows our applications and tools which include a data producer and several data consumers. It is a representative of many scientific simulation environment.

The first application is an astrophysics application called Astro3D or astro3d [25, 39] henceforth[3]. From I/O's point of view, Astro3D is a data producer: it generates three kinds of datasets: one for later possible data analysis which in-

[3]Astro3D is a code for scalable parallel architectures to solve the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature.

clude six datasets (*press, temp, rho, ux, uy* and *uz*); one for visualization which includes seven datasets (*vr-scalar, vr-press, vr-rho, vr-temp, vr-mach, vr-ek* and *vr-logrho*); one for checkpoint which includes six datasets (*restart-press, restart-temp, restart-rho, restart-ux, restart-uy* and *restart-uz*). These datasets may be used by subsequent post-processing programs. The user can specify in command line the dump frequency of each kind of datasets, total number of iterations (time steps) and problem size (3 dimensional).

The second application is a data analysis program. This application is a data consumer in that it takes one of datasets generated by Astro3D (*press, temp, rho, ux, uy* or *uz*) and calculates the difference between two consecutive time steps. This will show how dataset changes as simulation goes on. The algorithm applied is Maximum Square Error (MSE) between two consecutive time steps.

The third application is a parallel volume rendering code (called Volren henceforth). It generates a 2D image by projection given a 3D input file. This application is both a data consumer and data producer. It takes one of datasets (3 dimensional) generated by Astro3D (*vr-scalar, vr-press, vr-rho, vr-temp, vr-mach, vr-ek* or *vr-logrho*) and then performs parallel volume rendering algorithm and generates a two dimensional image data file for each iteration. This image dataset is then dumped to storage for later usage. As the output (writing 2D image files) is insignificant compared to input (reading 3D dataset), we focus on the input part of I/O, that is, we only view Volren as a data consumer in this paper.

5.3 Experiments

- **(1) Data Analysis** Suppose the user conducts an experiment on Astro3d, and then choose a generated dataset to carry out data analysis. In the first case, the user does not provide any access pattern information: this corresponds to the naive I/O systems without any optimizations. So all the datasets are stored on remote tapes. When the user carries out data analysis, the data is read from remote tapes without any optimizations. The total I/O time is show in Figure 4(1). Then, the user realizes that her data is partitioned in (Block, Block, Block), so she adds this information to read access pattern: *DataPartition = 'BBB'* and run data analysis program again. As data is partitioned in (Block, Block, Block), collective I/O is applied by MS-I/O. The performance improvement is significant (Figure 4(2)). Further, if the user knows that one of her datasets will be used soon and its size is large (problem size 128^3), so she can pass these hints as *WriteSize = 'large'* and *WhenAccess = 'soon'* when it is generated by Astro3d. This write access pattern will suggest the data be placed on remote disks rather than remote tapes. So when the user carries out data analysis, the I/O time is show in Figure 4(3). Again, when collective I/O is also applied by adding *DataPartition = 'BBB'*, the performance is improved dramatically (Figure 4(4)). Finally, if the user knows that she will frequently perform data analysis on the dataset, so she further adds *AccessFrequency = 'frequent'* to the access pattern, then the dataset will be placed on the local disks. The I/O time of data analysis this time would be Figure 4(5) and Figure 4(6) if *DataPartition = 'BBB'* is also provided. We can see that the more access pattern information the user provides, the more performance improvement can be achieved.

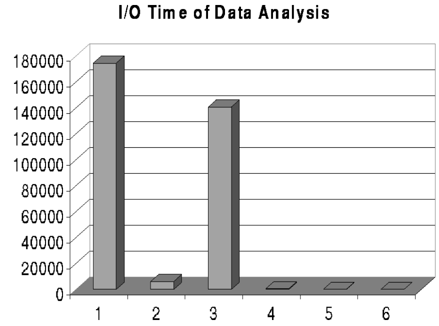

Figure 4. I/O Time of Data Analysis for Different Access Patterns. (1) No Access Pattern (Remote Tape); (2) DataPartition = 'BBB' (Remote Tape + Collective I/O); (3) WriteSize='large' + WhenAccess='soon' (Remote Disk); (4) WriteSize='large' + WhenAccess='soon' + DataPartition='BBB' (Remote Disk + Collective I/O); (5) WriteSize='large' + WhenAccess='soon' + AccessFrequency = 'frequent' (Local Disk); (6) WriteSize='large' + WhenAccess='soon' + AccessFrequency = 'frequent' + DataPartition = 'BBB' (Local Disk + Collective I/O).

- **(2) Volume Rendering** Similar to the last example, our next example is the volume rendering application. The data partition is $B**$ which conforms the data's layout on storage, so there is no benefit for collective I/O. But the computation is an important part of performance in this example, so the optimization policies include data location and asynchronous I/O. Figure 5 shows the performance numbers. Note that the most significant performance improvement by asynchronous I/O occurs between (3) and (4) on remote disks since the computation time and I/O time are closer than others, so there is more chance to overlap them. In other cases, either I/O dominates (1,2)

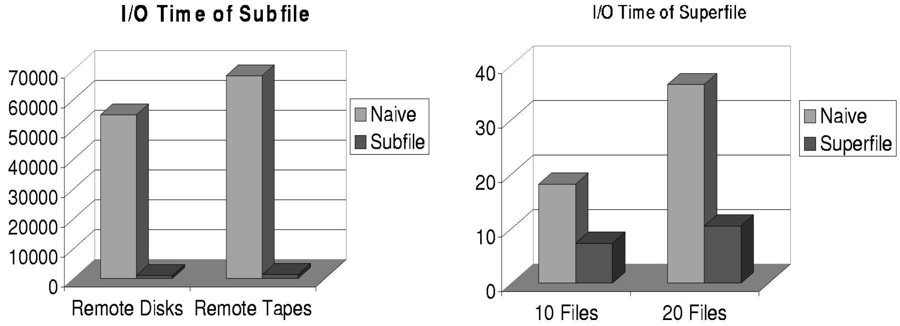

Figure 6. I/O Time for Subfile(left) and Superfile(right).

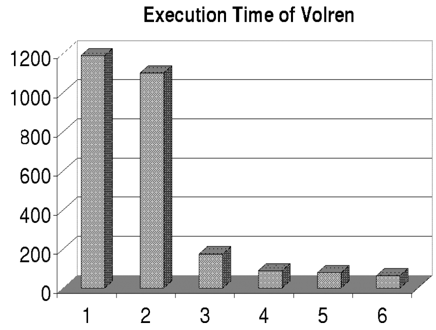

Figure 5. Execution time of Volren (1) No Access Pattern (Remote Tape); (2) ComputeTime = 'large' (Remote Tape + Asynchronous I/O); (3) WriteSize = 'large' + WhenAccess = 'soon' (Remote Disk); (4) WriteSize = 'large' + WhenAccess = 'soon' + ComputeTime = 'large' (Remote Disk + Asynchronous I/O); (5) WriteSize = 'large' + WhenAccess = 'soon' + AccessFrequency = 'frequent' (Local Disk); (6) WriteSize = 'large' + WhenAccess = 'soon' + AccessFrequency = 'frequent' + ComputeTime = 'large' (Local Disk + Asynchronous I/O).

(remote tapes) or computation dominates (5,6) (local disks) the performance, asynchronous I/O contributes less percentage of performance improvement.

- **(3) Subfile** If the user is going to create huge data files (1024^3) and will only access a portion of it (sub-cube 512), the user can provides access pattern as *WriteSize = 'huge'* and *FutureReadSize = 'partial'* when the data is created, subfile optimization is used (subfile chunk size 256^3). Figure 6 shows the results when the data is located at remote disks and remote tapes respectively.

- **(4) Superfile** If the user generates a sequence of small files (512) and will access it in a sequence too, she can provide access pattern as *WriteSize = 'small'* and *WriteSequence = 'yes'* and *FutureReadSequence = 'yes'*, then superfile technique is applied. The performance improvement is show in Figure 6 for 10 and 20 small files respectively.

- **(5) Data Replication** The user's access pattern may change. For example, when a medium-sized dataset is created, it is first placed on remote disks because the user does not provide further information. The problem is that when the user starts to access it frequently, the dataset has already been placed on remote disks: other optimizations can hardly help in this case. By making a local disk copy, Data replication of MS-I/O can solve this problem. The user's task is just providing the access pattern information such as *AccessFrequency = 'frequent'*. MS-I/O will automatically duplicate the dataset on local disks when the data is first access and the subsequent access requests can be serviced directly from local disks (Figure 7 (left)).

- **(6) Data Access History** The same situation as last example and in addition, the user is even not clear whether a remote dataset will be frequently accessed, our MS-I/O system can detect from the database that whether the dataset is being accessed frequently (say three times in half an hour) and it will automatically create a replica on local disks. The subsequent accesses will benefit from local disks even the user does not provide any new access pattern information (Figure 7 (right)).

6 Related Work

The related work can be divided into five groups.

One is parallel file systems, including IBM Vesta [11] and PIOFS [12], Intel Paragon [28], PPFS [21] and so on. These parallel file systems, either commercial or experimental, take advantage of parallel I/O techniques, caching, prefetching etc to achieve significant performance improvement. The storage of these systems usually includes only

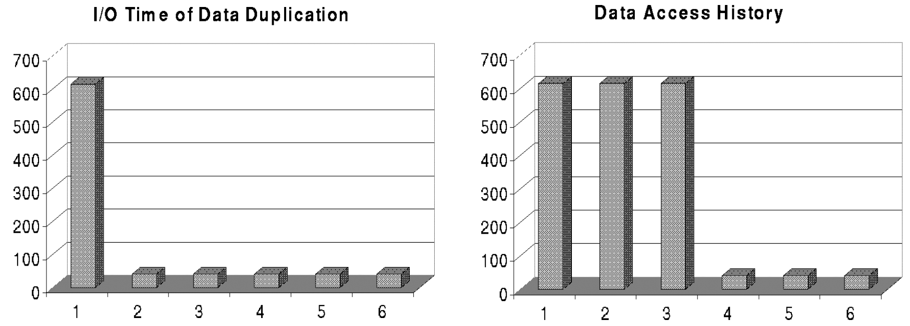

Figure 7. I/O Time for data replication(left) and data access history(right). Left: The user clearly knows that she is going to access a remote dataset frequently, so she specifies her access pattern information (*AccessFrequency = 'frequent'*). A local copy is made when the dataset is first accessed (1). Then the subsequent accesses will be served directly from local disks (2-6). Right: When the user is not clear whether the dataset will be frequently accessed, MS-I/O detects from the database that this dataset is accessed three times in half an hour (1-3) which means it is being frequently used, therefore a duplicated copy is made on local disks (3). Then the subsequent accesses can avoid remote data transfer (4-6).

secondary storage resources and they are tightly coupled with the compute nodes, so they do not scale well in capacity with the increase of applications' requirements. Therefore, the storage capacity required by large-scale data intensive applications could be a problem for these systems.

Another body of work includes run-time systems such as MPI-I/O [38, 36], PASSION [10, 34, 35], PANDA [29] and others [31, 7]. These systems provide high level structured interfaces on top of low level native parallel file systems [22] and try to match the applications' data structure which is usually multidimensional array. They also provide optimizations such as collective I/O and data sieving to solve the problems brought by native parallel file systems for many popular access patterns. Again, these systems do not help when application size increases.

The third group is distributed file systems such as NFS [33], xFS [4], and Coda [1]. These file systems provide easy access to distributed resources, but they are not designed for high performance parallel data access required by parallel applications.

The Grid [2, 18, 17] infrastructure will connect multiple regional and national computational grids, creating a universal source of pervasive and dependable computing power that supports dramatically new classes of applications. To address the data management problem of Grid, the *Data Grid* [9] has proposed some design principles of storage systems and data management for large data collections. Other data management systems can be found in [5, 31, 6, 32, 19, 26]. Generally, these systems use database's query capability to automatically keep track of huge amount of datasets generated by data intensive applications. However, how to effectively incorporate state-of-the-art I/O techniques with data management system is not well addressed in these systems.

The last group of work is the study on access patterns [24] in Pablo project [3]. Their work, however, is limited to only a small number of features about the data's usage, many other important access pattern information such as data's partition, access frequency etc are missing. In addition, the I/O optimizations of their work are limited to prefetching and caching in the secondary storage environment.

7 Conclusions and Future Directions

In this paper, we have presented a *Multi-Storage I/O System* which employs multiple distributed storage resources for high performance data intensive computing. MS-I/O is a scalable, multi-threaded, multi-storaged and multi-optimized I/O system. One of significant features of this I/O system is that the datasets generated by one application could be spread on different storage resources even within a single run of the application and this feature can bring new optimization opportunities which is impossible in the traditional single storage environment. A *User Access Pattern* structure has also been presented to help MS-I/O automatically and accurately choose I/O optimization strategies, so the user is released from hard task of I/O decisions.

From the experiments we can easily see that accessing data locally can bring more performance improvement than other optimization approaches. Unfortunately, local resources usually have limited storage space available for many users. Therefore, increasing local storage space to place more frequently used data locally is crucial for large-scale data intensive applications. Fortunately, the opportunity does exist since we have found that there are a lot of local storage resources that have not been fully utilized in a typical computing environment. We are in the progress of complementing our MS-I/O system by taking advantage of this opportunity.

Acknowledgments

This research was in part supported by Department of Energy under the Accelerated Strategic Computing Initiative (ASCI) Academic Strategic Alliance Program (ASAP) Level 2, under subcontract No W-7405-ENG-48 from Lawrence Livermore National Laboratories. We would like to thank Regan Moore and Mike Wan of SDSC for helping us with the usage of SRB. We thank Mike Gleicher and Tom Sherwin of SDSC for answering our HPSS questions.

References

[1] Coda file system. In *http://www.coda.cs.cmu.edu*.

[2] Global grid forum. In *http://www.gridforum.org/*.

[3] Pablo research group. In *http://www-pablo.cs.uiuc.edu/*.

[4] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proc. of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 109–126, 1995.

[5] C. Baru, R. Frost, J. Lopez, R. Marciano, R. Moore, A. Rajasekar, and M. Wan. Meta-data design for a massive data analysis system. In *Proc. CASCON'96 Conference*, 1996.

[6] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *Proc. CASCON'98 Conference, Dec 1998, Toronto, Canada*, 1998.

[7] R. Bennett, K. Bryant, A. Sussman, R. Das, and J. S. Jovian. A framework for optimizing parallel i/o. In *Proc. of the 1994 Scalable Parallel Libraries Conference*, 1994.

[8] P. Cao, E. Felten, and K. Li. Application-controlled file caching policies. In *Proc. the 1994 Summer USENIX Technical Conference*, pages 171–182, 1994.

[9] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*.

[10] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. Passion: parallel and scalable software for input-output. In *NPAC Technical Report SCCS-636*, 1994.

[11] P. Corbett and D. Feitelson. The vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, August 1996.

[12] P. Corbett, D. Feitelson, J.-P. Prost, G. Almasi, S. J. Baylor, A. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. Herr, J. Kavaky, T. Morgan, and A. Zlotek. Parallel file systems for the ibm sp computers. *IBM Systems Journal*, 34(2):222–248, Janury 1995.

[13] R. A. Coyne, H. Hulen, and R. Watson. The high performance storage system. In *Proc. Supercomputing 93, Portland, OR*, 1993.

[14] J. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel i/o via a two-phase run-time access strategy. In *Proc. the 1993 IPPS Workshop on Input/Output in Parallel Computer Systems*, 1993.

[15] C. S. Ellis and D. Kotz. Prefetching in file systems for mimd multiprocessors. In *Proc. the 1989 International Conference on Parallel Processing*, pages 306–314, 1989.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. In *Intl J. Supercomputer Applications.*, pages 115–128, 1997.

[17] I. Foster and C. Kesselman. The globus project: A status report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[18] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[19] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project.

[20] Hpss worldwide web site. In *http://www.sdsc.edu/hpss/*.

[21] J. Huber, C. Elford, D. Reed, A. Chien, and D. Blumenthal. Ppfs: A high performance portable parallel file system. In *Proc. of the 9th ACM International Conference on Supercomputing*, pages 385–394, 1995.

[22] D. Kotz. Multiprocessor file system interfaces. In *Proc. the Second International Conference on Parallel and Distributed Information Systems*, pages 194–201, 1993.

[23] D. Kotz. Disk-directed i/o for mimd multiprocessors. In *Proc. the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74, 1994.

[24] T. Madhyastha and D. Reed. Exploiting global input/output access pattern classification. In *Proceedings of SC'97*, 1997.

[25] A. Malagoli, A. Dubey, and F. Cattaneo. A portable and efficient parallel code for astrophysical fluid dynamics. In *http://astro.uchicago.edu/Computing/On-Line/cfd95/camelse.html*.

[26] Mcat. In *http://www.npaci.edu/DICE/SRB/mcat.html*.

[27] G. Memik, M. Kandemir, A. Choudhary, and V. E. Taylor. April: A run-time library for tape resident data. In *the 17th IEEE Symposium on Mass Storage Systems*, 2000.

[28] B. Rullman. Paragon parallel file system. In *External Product Specification, Intel Supercomputer Systems Division*.

[29] K. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective i/o in panda. In *Proceedings of Supercomputing '95, San Diego, CA, December*, 1995.

[30] X. Shen, W. Liao, and A. Choudhary. Remote i/o optimization and evaluation for tertiary storage systems through storage resource broker. In *Proceedings of IASTED Applied Informatics, Innsbruck, Austria*, 2001.

[31] X. Shen, W. Liao, A. Choudhary, G. Memik, M. Kandemir, S. More, G. Thiruvathukal, and A. Singh. A novel application development environment for large-scale scientific computations. In *International Conference on Supercomputing, May 8-11, 2000, Santa Fe, New Mexico*, 2000.

[32] Srb version 1.1.4 manual. In *http://www.npaci.edu/DICE/SRB/OldReleases/SRB1-1-4/SRB1-1-4.htm*.

[33] H. Stern. *Managing NFS and NIS*. O'Reilly and Associates, 1991.

[34] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh. Passion runtime library for parallel i/o. In *Proc. of the Intel Supercomputer User's Group Conference*, 1995.

[35] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. Passion: Optimized i/o for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.

[36] R. Thakur, W. Gropp, and E. Lusk. A case for using mpi's derived datatypes to improve i/o performance. In *Proc. of SC98: High Performance Networking and Computing*, 1998.

[37] R. Thakur, W. Gropp, and E. Lusk. *On implementing MPI-IO portably and with high performance*. Preprint ANL/MCS-P732-1098, Mathematics and Computer Science Division, Argonne National Laboratory, 1998.

[38] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective i/o in romio. In *Proc. the 7th Symposium on the Frontiers of Massively Parallel Computation*, 1999.

[39] R. Thakur, E. Lusk, and W. Gropp. I/o characterization of a portable astrophysics application on the ibm sp and intel paragon. In *MCS-P534-0895, Mathematics and Computer Science Division, Argonne National Laboratory*, 1995.

[40] *UniTree User Guide, Release 2.0*. UniTree Software, Inc., 1998.