A Distributed Multi-Storage Resource Architecture and I/O Performance Prediction for Scientific Computing

Xiaohui Shen and Alok Choudhary
Center for Parallel and Distributed Computing
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
{xhshen,choudhar}@ece.nwu.edu

Abstract

I/O intensive applications have posed great challenges to computational scientists. A major problem of these applications is that users have to sacrifice performance requirement in order to satisfy storage capacity requirement in a conventional computing environment. Further performance improvement is impeded by the physical nature of these storage media even state-of-the-art I/O optimizations are employed.

In this paper, we present a distributed multi-storage resource architecture *that can satisfy both performance and capacity requirements by employing multiple storage resources. Compared to traditional* single storage resource architecture, *our architecture provides a more flexible and reliable computing environment. It can bring new opportunities for high performance computing as well as inherit state-of-the-art I/O optimization approaches that have already been developed. We also develop an Application Programming Interface (API) that provides transparent management and access to various storage resources in our computing environment. As I/O usually dominates the performance in I/O intensive applications, we establish an I/O performance prediction mechanism which consists of a performance database and a prediction algorithm to help users better evaluate and schedule their applications. A tool is also developed to help users automatically generate the performance database. The experiments show that our* multi-storage resource architecture *is a promising platform for high performance distributed computing.*

1 Introduction

Large scientific simulations, especially I/O intensive applications, are generating huge amounts of data that has become a major problem facing computational scientists[13]. In a traditional computing environment, the compute resource is tightly coupled with local file systems, i.e. using local disks as data storage. As the speed of increase in data size significantly exceeds that of increase in disk capacity, large-scale scientific applications have to turn to other large storage resources. These storage resources include tertiary storage systems such as HPSS[5, 7], UniTree[17] and large database systems such as Oracle. This technology change has shifted the traditional computing environment to a distributed environment because the large storage archival systems are not so popular as the compute resources, so it is very common that these large storage systems are not locally installed. For example, the available tertiary storage system (HPSS) for us is located at San Diego Supercomputer Center (SDSC), while our application is running at Argonne National Lab and Northwestern University. Therefore, the compute resources and storage resources may no longer tightly coupled, rather, they may be geographically distributed across wide area networks. In such a distributed environment, the I/O problem seems more serious: an I/O call in this case has become a remote I/O access[6] across networks, thus the I/O cost is many times worse than a local disk I/O cost (bear in mind that I/O access speed is already far lagged behind memory access even in a local computing environment.). In addition, the network may bring more issues such as reliability, quality of service, security and so on in a remote I/O access.

A major concern of I/O in a distributed environment is performance. I/O (remote I/O) evaluation and optimizations are more important and urgent 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)[10] that provides various 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 general I/O optimizations such as collective I/O, data sieving and

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

so on can not eliminate this overhead when data resides on tapes. Aggressive prefetch or prestage may partially solve this problem by overlapping I/O access and computation, but they may significantly complex I/O system and force the user to specify more precise hints to the system, otherwise a 'false' prefetch or prestage may actually hurt performance. 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 loss of performance. We think this dilemma is rooted in the traditional *single storage resource architecture*. In this architecture, the application has only one storage media available for storing the user's data. The performance improvement would saturate even if many state-of-the-art optimizations are applied. Note that a scientific simulation here not only concerns the application that generates data, it also includes 'post' processing of these datasets, such as data analysis, visualization etc. on datasets just generated. So the performance here means an overall performance for all these processings.

In this paper, we present a *multi-storage resource architecture* which is a promising approach to solve the problem discussed previously. In this 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 logic storage capacity of the system. The total available storage capacity can be significantly increased by adding as many storage media 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 especially in production is shutdown 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 and most importantly as far as the performance is concerned, a *multi-storage resource architecture* provides new opportunities for further performance improvement for scientific simulations. 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 such as data analysis and visualization thereafter. For example, if the user wants to carry out visualization on a specific dataset just after simulation generates it, she can suggest the system dump 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 short, this architecture can combine the capacity advantage of remote large storage system and the performance advantage of local small storage system for best usage.

To efficiently make use of this *multi-storage resource architecture*, an effective user interface is required. This interface should be easy-to-use, so the user does not need to change her programming practice and should be scalable, so other storage resources can be easily added. Another issue this paper addresses is I/O performance prediction. There is a lot of study in literature on performance prediction for computing resources[12], but few study is on I/O performance prediction. The I/O time usually dominates the performance in I/O intensive applications, so accurate I/O performance prediction can greatly help the user evaluate and schedule her applications. Embarking on these goals, our contributions in this paper are summarized as follows.

(1) Present a *multi-storage resource architecture* that provides opportunities for further performance improvement and better data manipulation of users' applications.

(2) Design and implement an API that provides transparent access to diverse storage resources. Our design of this API is scalable since other storage media can be easily added.

(3) Provide run-time library (I/O optimization) for each kind of storage resource. Each storage resource has its own data access characteristics and we provide state-of-the-art I/O optimizations for each type of storage resource. So access to each kind of storage resource is optimized.

(4) Establish an I/O performance predictor that can accurately predict I/O performance across diverse storage

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

resources before the user actually carries out the experiment. The user can have better evaluation of her application via this performance predictor.

(5) Implement a tool called PTool to help the user automatically establish the I/O performance database that is used by performance predictor.

The remainder of the paper is organized as follows. In Section 2 we describe a data model that captures I/O characteristics of most I/O intensive applications. We also introduce our simulation environment which includes several applications and some tools. In Section 3 the system architecture of our multi-storage resource system is presented. We first describe a logic view of this architecture, followed by physical environment of our experiments. In Section 4 we present I/O performance predictor. We first introduce a basic performance model and a tool that can help automatically generate the performance database, then we describe our I/O performance prediction algorithm. In Section 5, we show performance numbers of experiments performed in our new architecture. The experiment results are also compared to prediction results by our I/O performance predictor. We conclude the paper in Section 6 and some future work is also presented.

2 I/O Model and Applications

Figure 1 (left) shows a typical I/O model for scientific applications. N represents maximum number of iterations of the application and M is the total number of datasets. The I/O frequency of $dataset(j)$ is given by $freq(j)$. In the main loop, $dataset(j)$ will be either read or written for every $freq(j)$ iterations. The computing part, which is not shown in the figure, may be interleaved with I/O operations. As the number of iterations could be very large and/or each dataset could be large, the total amount of data would be huge. Figure 1 (right) shows our simulation environment which includes several applications and tools. It is a representative of typical scientific simulation environments.

The main application is an astrophysics application called Astro3D or astro3d [1, 15] henceforth. Astro3D is a code for scalably parallel architectures to solve the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature. The code has been developed to study the highly turbulent convective envelopes of stars like the sun, but simple modifications make it suitable for a much wider class of problems in astrophysical fluid dynamics. The algorithm consists of two components: (a) a finite difference higher-order Godunov method for compressible hydrodynamics, and (b) a Crank-Nicholson method based on nonlinear multigrid method to treat the nonlinear thermal diffusion operator. These are combined together using a time splitting formulation to solve the full set of equations. From data flow's point of view, Astro3D is a data producer: it generates three kinds of datasets: one for later possible data analysis which include 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*). The user can specify in command line the dump frequency of each kind of datasets, total number of iterations and problem size (3 dimensional) of datasets (Figures 1, 2). 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 timesteps. This will show how dataset changes as simulation goes on. The algorithm applied is Maximum Square Error (MSE) between two consecutive timesteps. Other data analysis programs can easily substitute this program if needed. 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 perform parallel volume rendering algorithm and generate two dimensional image dataset for each iteration. This image dataset is then dumped to storage media for later usage. (Figure 2).

The two tools are an image viewer and an interactive visualization tool such as VTK. They are both data consumers. The image viewer read two dimensional image datasets generated by Volren and the interactive visualization tool takes datasets directly from Astro3D (Figure 2).

We view this whole picture (Figure 1) as a complete simulation environment. The user performs experiments of Astro3D first and later on, she may carry out one or more of the post processings (data analysis, volume rendering and interactive visualization) on the generated datasets. The performance improvement of one component should not impede the improvement of other components. Therefore, the overall performance improvement can be achieved.

3 System Architecture and Experimental Environment

3.1 Logical Architecture of Multi-Storage Resource System

In this section, we present our multi-storage resource architecture. A logic architecture of this environment is depicted in Figure 3. This architecture can be logically layered into five levels.

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

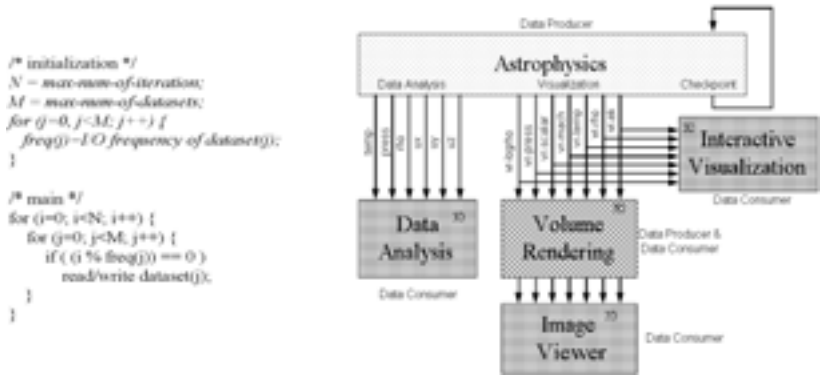

Figure 1. I/O model (left) and Data ow of applications in our Simulation Environment (right).

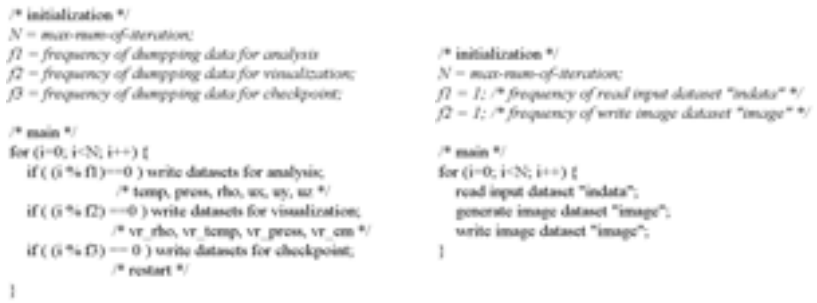

Figure 2. I/O model of Astro3D and Volren.

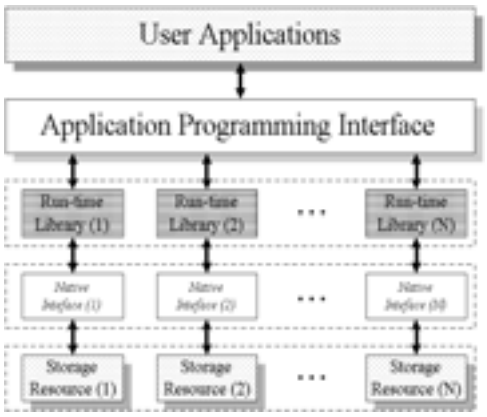

Figure 3. Logical Ar chitecture of multi-storage resource system.

- **Physical Storage Resources** At the bottom of this five-layered architecture are various storage resources. These resources could include local disks, local databases, remote disks, remote databases, remote tape systems and other storage systems. They the the actual holder of data.

- **Native Storage Interface** The layer residing above the storage resources is *native storage interface* . Each storage resource has its own access interface provided by the vendors of these storage systems. These interfaces to various storage systems are well established and developed by vendors. For example, the interface to a local database can be an embedded C API which is provided by database vendor. The interface to local disks is usually a filesystem. To access remote disk and tape systems, SRB is now a popular interface. Some tape systems such as HPSS also provide their own APIs for users. Unfortunately, the major concerns of these native interfaces are portability, ease-of-use and reliability etc. Few of them have fully considered performance issues in a parallel and distributed computing environment demanded by computational scientists. In addition, it is impossible for the application level users to change these interfaces directly to take care of performance issues. So this layer is performance-insensitive.

- **Run-time Library** One methodology to address performance problem of native storage interface is to build a run-time library that resides above it. The only concern of these libraries is performance. It captures

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

the characteristics of user's data access pattern and perform an optimized data access method to the native storage interface. For example, MPI-IO, which is built on top of local filesystems, takes advantage of collective I/O, data sieving, asynchronous I/O and so on to gain performance improvements. For remote disk and tape systems, our previous work SRB-OL [10] also employs other novel optimization approaches such as subfile, superfile etc as well as those found in MPI-I/O. This layer is performance-sensitive.

- **User API** On top of Run-time libraries is Application Programming Interface (API) layer. This API is used in user applications to provide transparent access to various storage resources and selection of appropriate I/O optimization strategies and storage types.

- **User Application** The top layer in our logical architecture is user applications. The user writes her program by using API and passes a high level hint in the API. This hint is high level since it is not concerned with low level details of storage resources and I/O optimization approaches. It only describes how the user's dataset will be partitioned and accessed by parallel processors, how her dataset will be used in the future, what kind of storage systems the user expects to put her datasets on etc.

We can also think the the bottom two layers, storage resources and native interfaces, as physical level. They are provided by commercial vendors and the user has on control inside them. The top layer can be viewed as application level. The middle two layers, user API and run-time libraries can be thought of as system level, which is provided by the system developers like us. The purpose of this layer is to mediate between physical level and application level to optimize the raw access from application level to physical level.

An example to work under this architecture is that the user writes her application using our API when she needs to perform I/O. Our API then decides which storage resource should be responsible for the datasets of these I/O requests. Then the optimized I/O requests by run-time library are actually performed to the selected storage resources. Note that selecting target storage resources is fine-grain: it can be as fine as a specific dataset rather that the whole run. This means that different datasets may be spread to different storage media even within a single run. We will demonstrate in the subsequent sections that this architecture is more flexible and scalable for high performance distributed computing than in a single storage resource architecture.

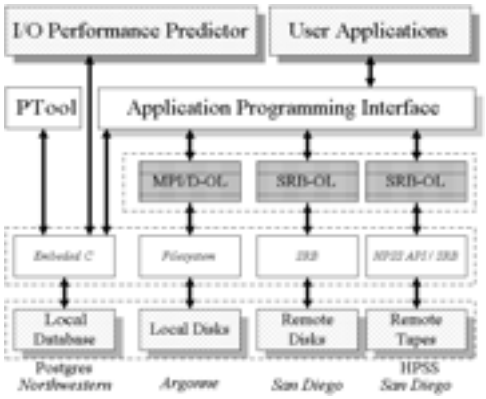

Figure 4. System Architecture of multiple storage resource system and I/O performance predictor.

3.2 Experimental Environment

Figure 4 shows our overall experimental environment (including I/O performance predictor and tool). The compute resource on which applications carry out is IBM SP2 which is located at Argonne National Laboratory. Each node of the SP-2 is RS/6000 Model 390 processor with 256 megabytes memory and has an I/O sub-system containing four 9 gigabytes SSA disks attached to it. The storage resources involved in this environment are:

- **Local Postgres Database** This database is installed at Northwestern University[1]. This 'small' database is used to store meta-data of our system. These meta-data describe information about what applications and users are involved in the system, information about each dataset and its characteristics. These characteristics include storage resource that each dataset is stored or to be stored, file path and name of each dataset, how each dataset is partitioned among processors, how it is stored (storage pattern) on storage systems and so on. The other layers such as API layer can use these information to locate each dataset that the user is interested in and also make an optimized I/O decision. This database also store performance-related data (Section 4), so that I/O performance predictor can consult this database to make performance prediction. The native interface to Postgres database is Embedded C API provided by the database vendor. As meta-data access is inexpensive, there is no need to provide a run-time library on top of native interface or other approaches to optimize meta-data access.

[1]Northwestern University is very close to Argonne National Lab and data exchange between Postgres database at Northwestern and the application at Argonne is small, so we treat it as a local database.

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

- **Local Disks** Local disks are the most popular traditional storage resource for saving user's data files. As scientific computing often generates huge amounts of data that exceeds the capacity of local disks, local disks are only suitable for storing small data files. On the other hand, local disk access is much faster than remote disk and tape access. Our multi-storage resource system allows us to take this opportunity for novel optimizations. The native interfaces to local disks is general UNIX filesystem, PIOFS and so on . On top of this interface is MPI-IO run-time library or D-OL that provides collective I/O, data sieving and asynchronous I/O etc optimization schemes. D-OL is a run-time library we ported to local disks from our SRB-OL [10]. Compared with MPI-IO, D-OL performs slightly better for write than MPI-IO but slightly worse for read. This small performance difference do not matter and can be ignored. The user can choose what kind of library she wants. We use D-OL in our experiments in this paper only because it allows us relatively easier to design a general performance prediction algorithm for all storage media.

- **Remote Disks** The remote disks in our environment is located at San Diego Supercomputer Center (SDSC). Compared to local disks, remote disks have both larger storage capacity and data access latency. We use SDSC's Storage Resource Broker (SRB) [2] [3, 14, 2] as native interface to remote disks. The run-time optimization library which is called SRB-OL is developed in our previous work [10]. Besides optimization approaches that can be found in MPI-IO, SRB-OL also provides some novel optimization schemes such as subfile, superfile and so on.

- **Remote Tapes** The remote tape system we use in our environment is High Performance Storage System (HPSS) [5]. Although HPSS can be configured as multiple hierachies, we only use its tapes, i.e. only one level of hierachy for simplicity. 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. The native interface to HPSS could be SRB or HPSS internal API. As we are not allowed to use HPSS's internal API at present[3], we also use SRB as native interface in our work. The SRB-OL run-time library is also applied to HPSS.

In sum, we have identified four storage resources in our system. One 'small' local database serves as meta-data depository, the other three resources: local disks, remote disks and remote tapes are repositories for actual data files. In general, the larger the storage capacity, the more expensive the data access cost. Based on characteristics of these distributed multi-storage resources, a unique distributed computing paradigm is identified: different datasets can be speculatively dumped to different storage resources for different purposes even within a single run. For example, if the user is going to carry out visualization on dataset *vr-temp* shortly after it is generated by Astro3D application, she can provide this information when she performs the experiment. Then this dataset will be written to a fast storage resource, local disks for example, which is 'close' to visualization tools, and all the other unwanted datasets to other slow storage media with large capacity for permanent archival. A specific dataset could be small enough to fit the local fast storage media. Late on, the user can directly access this dataset from local disks which is the fastest storage media in our environment. This speculation of dumping interested datasets to fast storage media can significantly improve the overall performance that is impossible for a system with a single storage type.

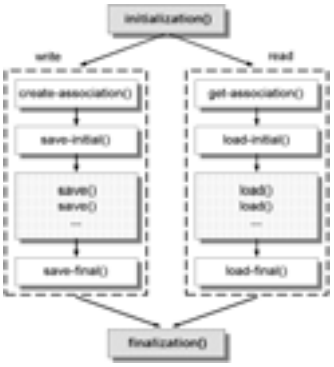

Figure 5. A typical I/O ow .

To effectively and efficiently make use of this *multi-storage resource architecture*, We implemented an *Application Programming Interface* (API). This API is easy to use because it observes UNIX programming practice. It provides an unique interface to transparently access various heterogeneous storage systems in our computing environment. In addition, the user provides high-level hints and our API transparently consults the database and decides storage type and I/O optimization approach internally according to user's hints and meta-data information kept in database. Other storage media can also be easily added. Figure 5 shows the I/O flow and main functions in our API [11]. In order for the user to specify hints, each dataset is associated with a 'location'. The user can explicitly specify it as one of the following values:

- LOCALDISK suggest dataset be placed on local disks;

[2] SRB is a client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated datasets. Using SRB has some advantages.

[3] HPSS's internal API is in general reserved for system administration.

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

- REMOTEDISK suggest dataset be placed on remote disks;
- REMOTETAPE suggest dataset be placed on remote tapes;
- AUTO/DEFAULT leave to system to decide. Default is remote tapes;
- DISABLE suggest this dataset not be dumped because it may not be used this run.

The user should be clear how her datasets are going to be used in the future, so it is easy for her to specify this hint.

Before we show our performance numbers in this environment, we will present an I/O performance predictor that gives a quantitative view of the performance of various storage resources used in our experiments.

4 I/O Performance Predictor

4.1 Performance Model

As I/O cost is significant for many large-scale scientific applications, it is very useful that the user can estimate the I/O cost before she actually carries out her experiments, so that she can have better arrangements of her experiments. Therefore, I/O performance prediction is very important. In our multi-storage resource system, an I/O call may initiate a remote data access across networks, so in general, the cost of a single I/O call in such an environment $T(s)$ can be broken down into time for communication setup T_{conn}, time for file open T_{open}, time for file seek T_{seek}, time to transfer data $T_{read/write}(s)$, time to close file $T_{fileclose}$ and time to close communication connection $T_{connclose}$.

$$T(s) = T_{conn} + T_{open} + T_{seek} + T_{read/write}(s) + T_{fileclose} + T_{connclose} \quad (1)$$

Where s is the size of a single data transfer. For the local filesystem, there is no communication setup, so $T_{conn} = 0$ and $T_{connclose} = 0$. For other distributed resources, the communication setup is a constant for each storage resource. In addition, file open, file close are also constants. The file seek time is also a constant for disk systems due to random access mechanism. $T_{read/write}(s)$ is a function of data size s. Therefore, the basis of our I/O performance prediction is to construct a performance database that maintains all the components in Equation 1 for each storage resource, so the performance predictor can search database to obtain these numbers to perform prediction algorithm. The main task is to time $T_{read/write}(s)$ for various data sizes on different storage media. To efficiently obtain these numbers, we built a tool called PTool that can automatically generate all these numbers. This program automatically measures read/write time of various data sizes and store them in database directly. Therefore, the user can easily set up her basic performance prediction database in a single run. Figure 6 and 7 show read/write time for various data sizes and Table 1 shows the file open, file close times etc.

4.2 Performance Prediction Algorithm

Once the basic performance database is established, we can design the prediction algorithm. Remember in our multi-storage resource environment, the user's request for I/O is high-level in her application, i.e. the user specifies an access pattern that includes how data is partitioned among processors, what kind of storage resource the user wants to be affiliated with for each dataset and other high level hints. This access pattern is interpreted by our Application Programming Interface (API) to a data structure understandable by lower-layer run-time library that can perform I/O optimization for each storage resource. So for the performance predictor, the main input parameters are data access pattern, what storage resource is used and what kind of I/O optimization is used as well as I/O frequency for each dataset and total number of iterations. The predictor then calculate the number of 'native' I/O calls (through native interface) needed for the request and the data size (s) of each 'native' I/O unit according to how I/O will be performed at physical level. Then by searching performance database, the predictor can calculate the overall estimated I/O time for each dataset access. The following equation gives the prediction algorithm.

$$T_{prediction} = \sum_{j=1}^{M} (N/freq(j) + 1) \times n(j) \times t_j(s) \quad (2)$$

Where N, M and $freq(j)$ are total number of iterations, total number of datasets and I/O frequency of $dataset(j)$ respectively. $n(j)$ is number of 'native' I/O calls required by $dataset(j)$ given an access pattern and I/O optimization approach. $t_j(s)$ is unit I/O time searched from performance database according to unit data size s. The following example will show how the algorithm works. Suppose the user is going to generate only *vr-temp* (dataset(1)) and *vr-press*(dataset(2)) in Astro3D for every 6 iterations and the maximum iteration is 120. *Vr-temp* is written to local disks and *vr-press* is dumped to remote disks. Each dataset is 2M. So $M = 2$, $N = 120$ and $freq(1) = freq(2) = 6$. When collective I/O is applied, it allows the user to issue one single write for one dataset each iteration. So $n(1) = n(2) = 1$. By consult the performance database and according to Equation 1, $t(1) = 0.1234$ and $t(2) = 8.47$. So the total time is:

$$T_{prediction} = \sum_{j=1}^{M} N/freq(j) + 1) \times n(j) \times t_j(s)$$

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

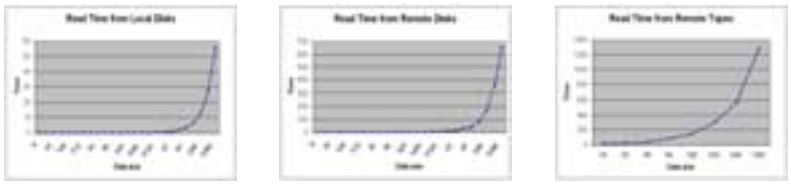

Figure 6. Read time for local, remote disks and remote tapes.

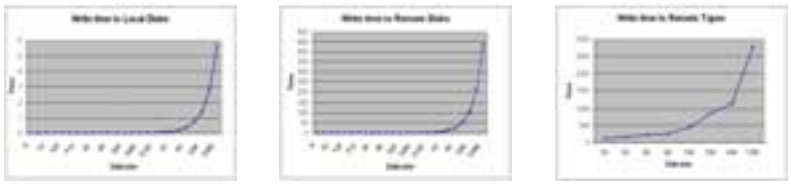

Figure 7. Write time for local, remote disks and remote tapes.

Table 2. Run-time parameter set of Astro3D

Item	Size	Datatype
Problem Size	128^3	-
Max num of iterations	120	-
Data Analysis Freq	6	Float
Data Visualization Freq	6	Unsigned Char
Checkpointing Freq	6	Float

$$= (120/6+1) \times 0.2534 + (120/6+1) \times 8.47$$
$$= 2.59 + 177.98 = 180.57(s) \quad (3)$$

Our experiment shows that the actual time is about 197.40 (Figure 8) which is consistent with our prediction. One example of using this performance prediction is that the user can choose better maximum run time parameter for her job. Our application is running on Argonne's SP2 which allows the user to specify a maximum run time for her job. The larger the maximum run time, the lower priority for scheduling. As the competition for job scheduling is keen, the user always wants to specify the maximum run time as small as possible. Our performance predictor can provide a lower bound for this parameter that might be great helpful for the user to choose a suitable maximum run time.

5 Experiments

In this section, we present various opportunities that our multi-storage resource architecture can take advantage of for high performance computing. These opportunities are impossible in a single storage resource architecture.

Our base application is Astro3D, a data producer. It generates a number of datasets for different purposes. A typical run-time parameter set is shown in Table 2. This set of parameters will generate a total of about 2.2G data. As the user may carry out many such scale experiments with different parameters, the total amount of data size generated could be huge. Therefore, In a single storage resource environment, the user has to choose tape systems as storage depository which are usually thought of as unlimited in space. The total I/O time is shown in Figure 8 if write all these data to tapes. Note that this time has already been optimized by collective I/O. Without collective I/O, it will be many times worse. Then suppose the user wants to perform data analysis(MSE) on dataset *temp*, then the total I/O time is shown in Figure 9 (left). This is our base experiment for comparison, which is typical for a single storage resource system. We can see that the I/O cost is very expensive even if state-of-the-art I/O optimizations such as collective I/O is performed.

In our multi-storage resource architecture, on the other hand, if the user knows that she is going to carry out data analysis on dataset *temp* shortly after it is generated, she can suggest the system to place *temp* on a 'closer' storage medium such as remote disks in this example. A subset of total datasets generated by Astro3D could be small enough to be held on a faster media for fast data access and our multi-storage resource system can just provide a platform for this opportunity. Although the total time of generating all these data is not saved much by placing the *temp* on remote disks (Figure 8), but when the user carries out data analysis, she can read data from remote disks which saves a lot of time compared to from tapes (Figure 9).

Another opportunity our multi-storage resource system can provide is that if the user knows that she is only interested in *temp* and possibly *press* and all the other datasets are not going to be used for a particular run, she can DISABLE dump of other datasets by providing 'location' hint as DISABLE. So the total I/O time of Astro3D can be decreased significantly (Figure 8).

Our second example is that if the user is going to perform

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

Table 1. Timings for file open, close etc

Location	Type	Conn	Fileopen	Fileseek	Fileclose	Connclose
Local Disk	read	0	0.20	-	0.001	0
Local Disk	write	0	0.21	-	0.001	0
Remote Disk	read	0.44	0.42	0.40	0.63	0.0002
Remote Disk	write	0.44	0.42	-	0.83	0.0002
Remote Tape	read	0.81	6.17	-	0.46	0.0002
Remote Tape	write	0.81	6.17	-	0.42	0.0002

parallel volume rendering on *vr-temp* or carry out interactive visualization by VTK, she can suggest the system to dump *vr-temp* to local disks. *Vr-temp* is consisted of unsigned characters, its size is small that could fit on local disks in our example. As *vr-temp* is closely related to *vr-press*, the user may also possibly examine *vr-press* as well, so *vr-press* is put to remote disks for possible faster usage than from tapes. All the other datasets which will not be used are dumped to tapes. In addition, if the user knows that all the other datasets will not be used at all, she can also DISABLE them. So the total write time saved is huge (Figure 8). When the user read *vr-temp* by parallel volume rendering or interactive visualization tools (VTK), the total read time is 10 times less than from tapes. If user also reads *vr-press*, she can also save time by read data from remote disks (Figure 9).

Our next example is to demonstrate a novel optimization approach called *superfile* to efficiently access large number of small files from remote systems. Suppose the images files generated by Volren are going to be stored on remote disks or tapes. When superfile is applied, these small files will be transparently write to one large *superfile* when they are created. Later on, when the user read these data, the first read will bring all the data into memory. Then the subsequent read can be satisfied by copying data directly from main memory. In this approach, there is only one remote I/O access with large data size compared to multiple remote I/O calls with small data sizes that would dominate the I/O performance [10] in the naive approach. Figure 9 (right) shows that the performance improvement is significant.

Our final example is that suppose the remote tape system is down for maintenance, recovery or other problems which could often happen, we can still satisfy large storage space requirement for simulations by aggregating all the space of remote disks, local disks and other storage resources in the future, i.e. the user does not have to stop her experiments. Therefore, this multi-storage resource system can provide a more reliable computing environment.

We also show the predicted I/O time for each performance number in Figure 8 and 9. Our prediction is quite close to the actual I/O time [4]. Our performance predictor

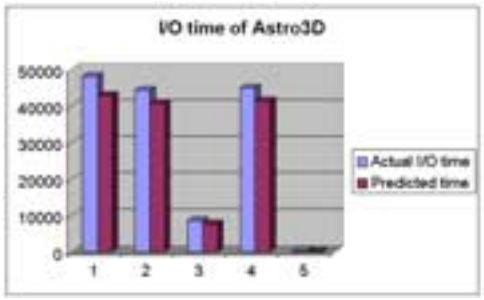

Figure 8. I/O time for Astro3D. 1: Write all datasets to remote tapes; 2. Write *temp* to remote disks and all other datasets to remote tapes; 3. Write only *temp* to remote disks and *press* to remote disks; 4. Write *vr-temp* to local disks and all the other datasets to remote tapes; 5. Write only *vr-temp* to local disks and *vr-press* to remote disks.

is integrated with our IJ-GUI[11], a Java graphical environment that can help the user submit her job, carry out visualization, perform data analysis and so on. Figure 10 shows a prediction result of Astro3D. It is very easy for the user to change parameters directly in the Java window to get other prediction results.

6 Conclusions and Future Directions

In this paper, we have presented a *multi-storage resource architecture* for scientific simulations that provides a more flexible and reliable computing platform in a distributed environment. This architecture, compared to the traditional *single-storage resource architecture*, can combine the advantages of different classes of storage resources and avoid their disadvantages. It is also scalable since other storage resources can be easily added.

We also established an I/O performance prediction mechanism that can help the user better evaluate her applications. Our experiments have shown that the prediction

[4] For remote systems, as network is involved, there is fluctuation in performance numbers possibly due to different network traffic condition. The numbers shown here are the most common cases.

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE

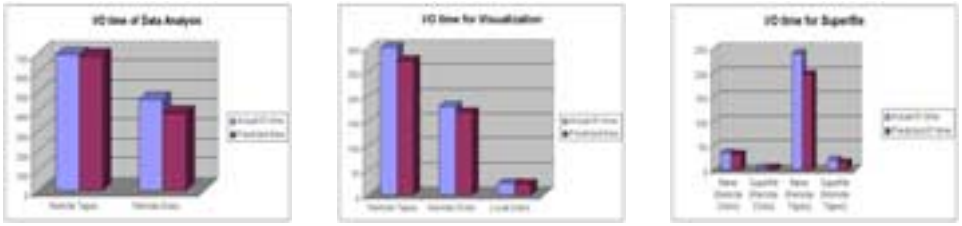

Figure 9. I/O time for Data Analysis, Visualization and Superle.

Figure 10. A Prediction Result of Astro3D.

is very close to the actual I/O time.

We would add more storage media in our system. Our current post-processing programs and tools are all installed locally. In the future, they could also possibly be distributed. So far, we require the user to explicitly specify storage hints. In the future, the user can also specify only a performance requirement for a particular run of her application and our system can automatically decide which storage resources should be used according to the capacity and performance of each storage resource.

References

[1] A. Malagoli, A. Dubey, and F. Cattaneo. A Portable and Efficient Parallel Code for Astrophysical Fluid Dynamics. *http://astro.uchicago.edu/Computing/On-Line/cfd95/camelse.html*.

[2] 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*, November 1996.

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

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

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

[6] I. Foster, D. Kohr, Jr., R. Krishnaiyer and J. Mogill. Remote I/O: Fast Access to Distant Storage. *Fifth Workshop on I/O in Parallel and Distributed Systems*, 1997.

[7] HPSS Worldwide Web Site. *http://www.sdsc.edu/hpss/*.

[8] D. Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proc. the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74. USENIX Association, Nov 1994.

[9] G. Memik, M. Kandemir, A. Choudhary, Valerie E. Taylor. APRIL: A Run-Time Library for Tape Resident Data. To appear in *the 17th IEEE Symposium on Mass Storage Systems*, March 2000.

[10] X. Shen, A. Choudhary and W. Liao. I/O Optimization and Evaluation for Tertiary Storage Systems. Submitted to *Internal Conference on Parallel Processing*, August 2000, Toronto Canada.

[11] 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. *International Conference on Supercomputing*, May 2000, Santa Fe, New Mexico.

[12] W. Smith, I. Foster and V. Taylor. Predicting Application Run Time Using Historical Information. *IPPS/SPDP '9 Workshop on Job Scheduling Strategies for Parallel Processing*, 1999.

[13] P. H. Smith and J. Van Rosendale. Data and Visualization Corridors. *Report on DVC Workshop Series*, 1998.

[14] SRB Version 1.1.4 Manual. *http://www.npaci.edu/DICE/SRB/OldReleases/SRB1-1-4/SRB1-1-4.htm*.

[15] R. Thakur, E. Lusk and W. Gropp. I/O Characterization of a Portable Astrophysics Application on the IBM SP and Intel Paragon. *MCS-P534-0895*, Mathematics and Computer Science Division, Argonne National Laboratory, 1995

[16] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. *Proc. the 7th Symposium on the Frontiers of Massively Parallel Computation*, February 1999.

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

9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)
1082-8907/00 $10.00 @ 2000 IEEE