Cluster Computing 6, 189–200, 2003

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

X. SHEN * and A. CHOUDHARY
Center for Parallel and Distributed Computing, Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, USA

C. MATARAZZO and P. SINHA
Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550, USA

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 requirements in order to satisfy storage capacity requirements in a conventional computing environment. Further performance improvement is impeded by the physical nature of these storage media even when state-of-the-art I/O optimizations are employed.

In this paper, we present a *distributed multi-storage resource architecture*, which can satisfy both performance and capacity requirements by employing multiple storage resources. Compared to a traditional *single storage resource architecture*, our architecture provides a more flexible and reliable computing environment. This architecture 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. It provides application users with high-performance storage access even when they do not have the availability of a single large local storage archive at their disposal. We also develop an Application Programming Interface (API) that provides transparent management and access to various storage resources in our computing environment. Since 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 performance data stored in databases. The experiments show that our *multi-storage resource architecture* is a promising platform for high performance distributed computing.

Keywords: multi-storage resource architecture, I/O performance prediction, data intensive computing

1. Introduction

Many large-scale scientific applications are data intensive: they generate huge amounts of data, presenting a major problem for computational scientists [28]. 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 rate that data is produced increases and the amount significantly exceeds that of disk capacity, large-scale scientific applications have to turn to other large storage resources. These storage resources include tertiary storage systems such as HPSS [12,18], UniTree [40] and large database systems such as Oracle. With the shift from typical computing centers to a more distributed computing enviornment, these additional large storage resources may no longer be locally available. These storage archives, due to nature and complexity are most likely available at a few large centers for remote access. For example, the tertiary storage system (HPSS) we use 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 be tightly coupled, rather, they may be geographically distributed across wide area networks. In such a distributed environment, the I/O problem becomes more serious: an I/O call in this case has become a remote I/O access [16] across networks, thus the I/O cost is many times higher than a local disk I/O cost (bear in mind that I/O access speed already lags far 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 environments. Both traditional I/O optimizations and new I/O techniques should be employed to address this problem. We have built a run-time library (SRB-OL) [26] that provides various state-of-the-art optimizations for tertiary storage access (HPSS). Although these optimizations can significantly improve performance compared to naive approaches, further improvement is impeded by the physical nature of storage media. For example, a tape system such as HPSS requires a minimum of 20 to 40 seconds to be ready to move the data and the data transfer rate is very slow compared to disks. I/O optimizations such as collective I/O, data sieving and 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 result in a significantly complex I/O system and force the user to specify more precise hints to the system, otherwise a 'false' prefetch

* Corresponding author.
E-mail: xhshen@ece.nwu.edu

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 medium available for storing the user's data. The performance improvement would saturate even if many state-of-the-art optimizations have been 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 and visualization. Thus performance here means an overall performance for all of these processes.

In this paper, a *multi-storage resource architecture*, a promising approach to solve this problem is presented. In our 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, it increases the logical storage capacity of the system. The total available storage capacity can be significantly increased by adding as many storage media to the system as possible.
- Second, a multi-storage resource system provides a more flexible and reliable computing environment. For example, failure of one storage component may not impede the computation because other storage options are available. Often the remote large storage system, especially one in production, is shutdown for system failure or maintenance, so that the experiment cannot be performed 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 performance is concerned, a *multi-storage resource architecture* provides new opportunities for further performance improvements for scientific simulations. With multiple storage resources coupled with the simulation, the application can speculatively store the datasets to the 'best' storage medium which is most favorable for the desired post-processing, such as data analysis and visualization. For example, if the user wants to carry out visualization on a specific dataset just after simulation generates it, she can suggest that the system dump this dataset locally (if it does not exceed local storage capacity), where visualization tools are installed and all other unused datasets are sent to the remote large storage system for permanent archival. So the user's visualization tools can directly read data from local disks rather than going all the way to the remote single storage resource. In general, each generated dataset has its purpose for usage and the *multi-storage resource architecture* allows this purpose information to be exploited for improved performance by storing it on a suitable storage medium. In short, this architecture can combine the capacity advantage of remote large storage systems and the performance advantage of local small storage system for best usage.

To make use of this *multi-storage resource architecture*, an effective user interface is required. This interface should be easy-to-use and not require that the user change her programming practices. In addition to the interface issues, this paper addresses I/O performance prediction. A large volume of information is available in the literature on performance prediction for computing resources [29], but few studies discuss I/O performance prediction. I/O usually dominates performance in data 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 given that other storage media can be easily added.

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

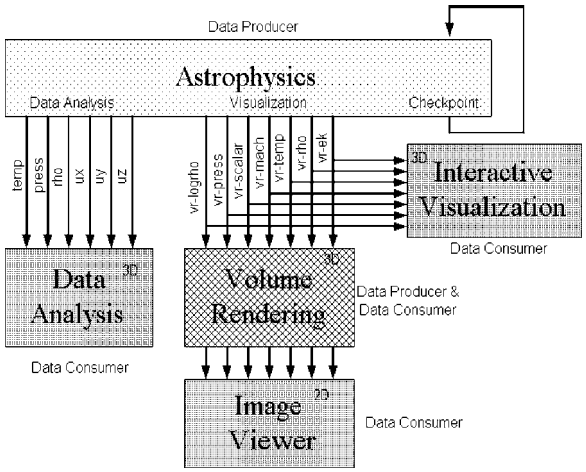
4. Establish an I/O performance prediction mechanism that can accurately predict I/O performance across diverse storage resources before the user actually carries out experiments.

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 that includes several applications and some tools. In section 3 the system architecture of our multi-storage resource system is presented. We first describe a logical view of this architecture, followed by the physical environment of our experiments. In section 4 we present the I/O performance predictor. First we 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 for experiments performed in our new architecture. The experiment results are also compared to prediction results by our I/O performance predictor. In section 6, the related work is presented. We conclude the paper in section 7 and some future work is also discussed.

```
/* initialization */
N = max-num-of-iteration;
M = max-num-of-datasets;
for (j = 0, j < M; j ++) {
  freq(j) = I/O frequency of dataset(j);
}

/* main */
for (i = 0; i < N; i ++) {
  for (j = 0; j < M; j ++) {
    if ((i % freq(j)) == 0)
      read/write dataset(j);
  }
}
```

(a)



(b)

Figure 1. I/O model (a) and data flow of applications in our simulation environment (b).

```
/* initialization */
N = max-num-of-iteration;
f1 = frequency of dumpping data for analysis;
f2 = frequency of dumpping data for visualization;
f3 = frequency of dumpping data for checkpoint;

/* main */
for (i = 0; i < N; i ++) {
  if ((i %f1) == 0) write datasets for analysis;
                /* temp, press, rho, ux, uy, uz */
  if ((i %f2) == 0) write datasets for visualization;
                /* vr_ rho, vr_temp, vr_press, vr_em */
  if ((i %f3) == 0) write datasets for checkpoint;
                /* restart */
}
```

```
/* initialization */
N = max-num-of-iteration;
f1 = 1; /* frequency of read input dataset "indata" */;
f2 = 1; /* frequency of write image dataset "image" */;

/* main */
for (i = 0; i < N; i ++) {
  read input dataset "indata";
  generate image dataset "image";
  write image dataset "image";
}
```

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

2. I/O model and introduction to applications

Figure 1(a) shows a typical I/O model for scientific applications. N represents the 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 from or written to 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 may be very large and/or each dataset may be large, the total amount of data could be huge.

Figure 1(b) 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 [21,38] 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. This 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 a data flow point of view, Astro3D is a data producer; it generates three kinds of datasets: one for data analysis which include six variables (*press, temp, rho, ux, uy* and *uz*); one for visualization which includes seven variables (*vr-scalar, vr-press, vr-rho, vr-temp, vr-mach, vr-ek* and *vr-logrho*); and one for checkpoint which includes six variables (*restart-press, restart-temp, restart-rho, restart-ux, restart-uy* and *restart-uz*). The user can specify on the command line the dump frequency of each kind of dataset, total number of iterations and problem size (3-dimensional) of datasets (figures 1 and 2).

The second application is a data analysis program. This application is a data consumer in that it takes one of the datasets generated by Astro3D (*press, temp, rho, ux, uy* or *uz*) and calculates the difference between two consecutive timesteps. This will show how a dataset changes as a simu-

lation progresses. The algorithm applied is Maximum Square Error (MSE) between two consecutive timesteps. Other data analysis programs can easily be substituted if desired.

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 the datasets (3-dimensional) generated by Astro3D (*vr-scalar, vr-press, vr-rho, vr-temp, vr-mach, vr-ek* or *vr-logrho*) and then performs a parallel volume rendering algorithm to generate a 2-dimensional image dataset for each iteration. This image dataset is then dumped to a storage medium for later use (figure 2).

In addition, the two tools are an image viewer and an interactive visualization tool such as VTK. They are both data consumers. The image viewer reads two dimensional image datasets generated by Volren and the interactive visualization tool takes datasets directly from Astro3D figure 2.

In figure 1 we view this whole picture as a complete simulation environment. The user performs experiments with Astro3D first, and after the simulation has completed, 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. Thus an overall performance improvement can be achieved.

3. System architecture and experimental environment

3.1. Logic architecture of multi-storage resource system

In this section, we present our multi-storage resource architecture. A logical architecture of this environment is depicted in figure 3. This architecture can be logically described with five levels:

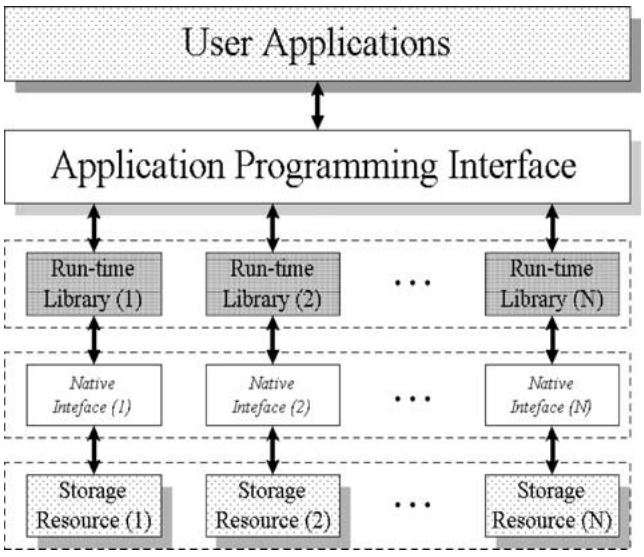


Figure 3. Logic architecture of multi-storage resource system.

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 components that actually hold the data.

Native Storage Interface The layer residing above the storage resources is the *native storage interface*. Each storage resource has its own access interface provided by the vendors of the storage systems. These interfaces to various storage systems are well established and are developed by the vendors. For example, the interface to a local database can be an embedded C API which is provided by the database vendor. The interface to local disks is usually a filesystem. To access remote disk and tape systems, the storage resource broker, SRB, developed by San Diego Supercomputer Center [REF] is a popular interface. Some tape systems such as HPSS also provide their own APIs for users. A few of the major concerns with 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. In addition, it is impossible for the application level users to change these interfaces directly to take care of performance issues. Thus this layer is performance-insensitive.

Run-time Library One methodology to address performance problems of native storage interface is to build a run-time library that resides above it. The only concern of the library at this level is performance. It captures the characteristics of user's data access patterns and performs an optimized data access method to the native storage interface. For example, MPI-IO, which is built on top of local file systems, 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 [26] 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 the Run-time libraries is the Application Programming Interface (API) layer. This API is used in user applications to provide transparent access to various storage resources and select appropriate I/O optimization strategies and storage types.

User Application The top layer in our logical architecture is user applications. The user develops her applications by using our API and passes a high level hint to the API. This hint is high level since it does not concern the 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 the dataset will be used in the future, or the kind of storage systems the user expects to send the datasets.

We can also think of the bottom two layers, storage resources and native interfaces, as the *physical level*. They are provided by commercial vendors and the user has no control over them. The top layer can be viewed as the *application*

level. The two layers in the middle, user API and run-time libraries can be thought of as the *system level*, which is provided by system developers like us. The purpose of this layer is to mediate between the physical level and the application level, and to optimize the raw access from application level to physical level.

For example to clarify the use of this architecture, a user would write her application using our API when she needs to perform I/O. Our API would decide which storage resource should be responsible for the data access. Then the optimized I/O requests by the run-time library are actually performed on the selected storage resources. Note that selecting target storage resources is fine-grained: it can be as fine as a specific dataset rather than the whole run. This means that different datasets may be spread across 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 a single storage resource architecture.

3.2. Experimental environment

Figure 4 shows our overall experimental environment (including I/O performance predictor and tools). The compute resource on which the applications run is an IBM SP2, which is located at Argonne National Laboratory. Each node of the SP-2 is a 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 as follows:

Local Postgres Database This database is installed at Northwestern University.[1] This 'small' database is used to store the meta-data of our system. The meta-data describes information about applications and users running in the system, and information about each dataset and its characteristics. These characteristics include the storage resource type on which 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 the API layer can use this information to locate each dataset that the user is interested in, and also make an optimized I/O decision. This database also stores performance-related data (section 4), so the I/O performance predictor can consult this database to make performance predictions. The native interface to the Postgres database is an 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 the native interface, or other approaches to optimize meta-data access.

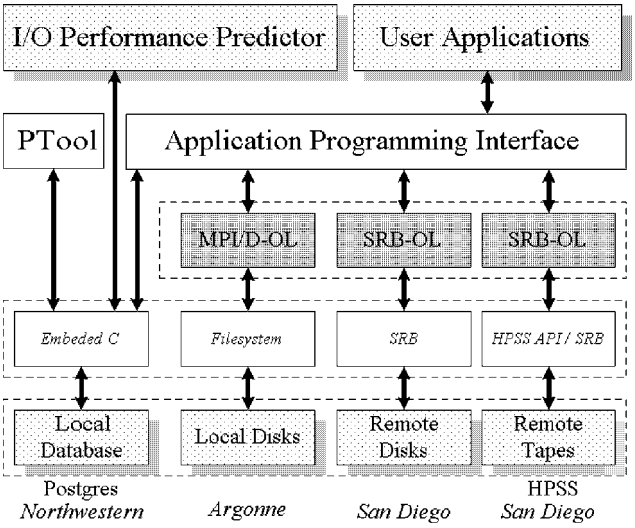


Figure 4. System architecture of multiple storage resource system and I/O performance predictor.

Local Disks Local disks are the most popular traditional storage resource for saving a user's data files. As scientific computing often generates huge amounts of data that may exceed the storage 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 the general UNIX file systems, PIOFS and so on. On top of this interface is the MPI-IO run-time library or D-OL that provides collective I/O, data sieving, asynchronous I/O and other optimization schemes. D-OL is a run-time library we ported to local disks from our SRB-OL [26] work. Compared to MPI-IO, D-OL performs slightly better for the write operation than MPI-IO but slightly worse for the read operation. This small performance difference does not matter and can be ignored. We use D-OL in our experiments in this paper only because it allows us to easily design a general performance prediction algorithm for all storage media.

Remote Disks The remote disks in our environment are located at the 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] [2,3,30] as the native interface to remote disks. The run-time optimization library, which is called SRB-OL, is presented in our previous work [26]. Besides optimization approaches that can be found in MPI-IO, SRB-OL also provides optimization schemes such as subfile, superfile and so on.

Remote Tapes The remote tape system we use in our environment is the High Performance Storage System (HPSS) [12]. Although HPSS can be configured as multiple hierarchies, we only use its tapes, i.e. only one level of a hierarchy, for simplicity. The remote tapes have very large storage spaces and we assume they 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 the native interface in our work. The SRB-OL run-time library is also applied to HPSS.

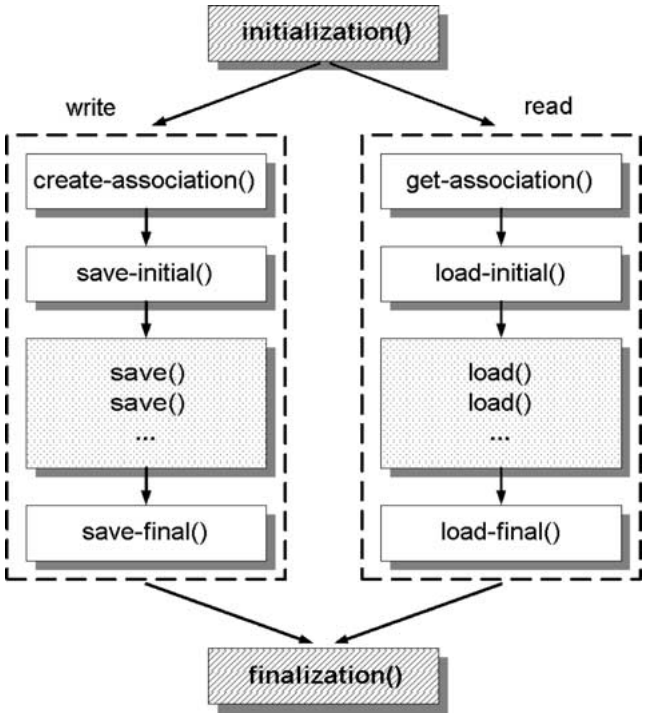


Figure 5. A typical I/O flow. The execution starts with the initialization() routine. The left side of the figure shows how the write operation progresses and right side shows how the read operation progresses. The execution flow is ended by the finalization() routine.

In summary, we have identified four storage resources in our system: one 'small' local database serves as a meta-data repository; 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 of an application. For example, if the user is going to carry out visualization on a 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, such as local disks, which are 'close' to the visualization tools, and all the other unused datasets are sent to other slow storage media but with larger capacities for permanent archiving Later on, the user can directly access this dataset from the local disk which is the fastest storage medium in our environment.

Figure 5 shows the I/O flow and main functions in our API [27]. In order for the user to specify her hints, each dataset is associated with a 'location' attribute. The user can explicitly specify it as one of the following values:

- LOCALDISK suggests dataset be placed on local disks;
- REMOTEDISK suggests dataset be placed on remote disks;
- REMOTETAPE suggests dataset be placed on remote tapes;
- AUTO/DEFAULT leaves it to the system to decide. Default is remote tapes;
- DISABLE suggests this dataset not be dumped because it will not be used for this run.

The system is most effective if the user understands how her datasets are going to be accessed in the future, so she can easily specify these hints.

Before we show our performance numbers in this environment, we will present an I/O performance predictor, which 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 if the user can estimate the I/O cost before she actually carries out her experiments, enabling her to make better arrangements of her experiments. Therefore, I/O performance prediction is a very important planning tool for the scientist. 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 $T_{\text{read/write}}(s)$, time to close file $T_{\text{fileclose}}$ and time to close the communication connection $T_{\text{connclose}}$:

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

where s is the size of a single data transfer. For the local filesystem, there is no communication setup, so $T_{\text{conn}} = 0$ and $T_{\text{connclose}} = 0$. For other distributed resources, the communication setup is a constant for each storage resource. In addition, file open and file close are also constants. The file seek time is also a constant for disk systems due to its random access mechanism. $T_{\text{read/write}}(s)$ 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 the database to obtain these numbers to perform prediction algorithms. The main task is to time $T_{\text{read/write}}(s)$ 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 stores them in the database directly. Therefore, the user can easily set up her basic performance prediction database in a single run. Figures 6, 7 and 8 show read/write time for various data sizes and table 1 shows the file open, file close times, etc. We recognize that the system load effects our timing measurements and may impact the accuracy of our predictions.

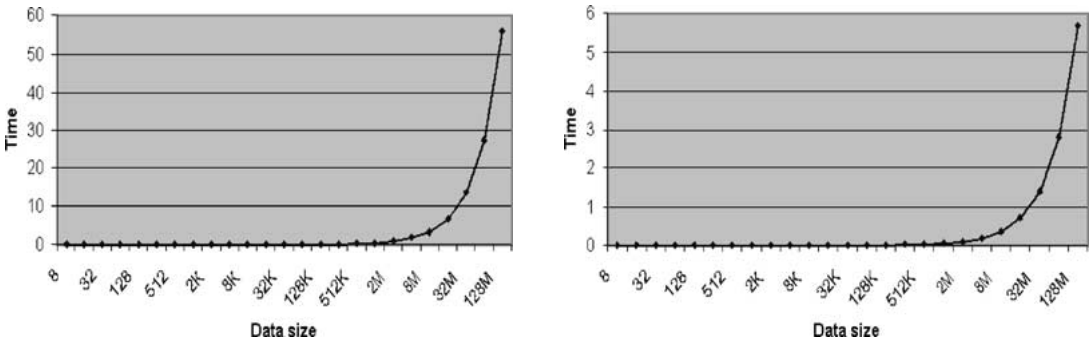


Figure 6. Read (a) and write (b) time on local disks.

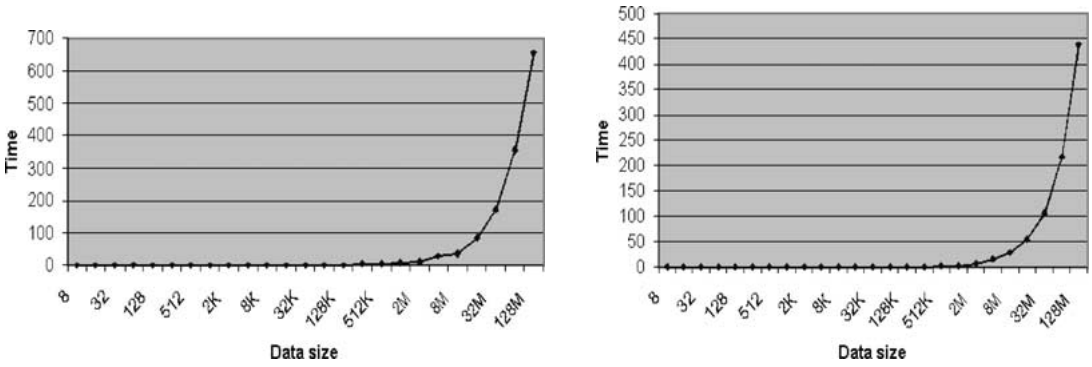


Figure 7. Read (a) and write (b) time on remote disks.

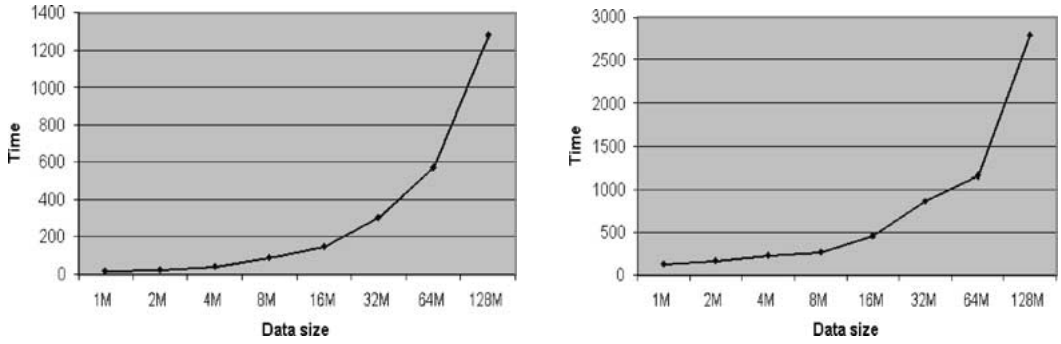


Figure 8. Read (a) and write (b) time on remote tapes (HPSS).

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

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 at a 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 affil-

iated with for each dataset and other high level hints. This access pattern is interpreted by our Application Programming Interface (API) into a data structure understandable by a 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 the total number of iterations. The predictor then calculates the number of 'native' I/O calls (through the 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 the physical level. Then by searching the performance database, the predictor can calculate the overall estimated I/O time for each dataset access. The following equation gives the prediction algorithm:

$$T_{\text{prediction}} = \sum_{j=1}^{M} \big(N/freq(j) + 1\big) n(j) t_j(s), \qquad (2)$$

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 the number of 'native' I/O calls required by $dataset(j)$ given an access pattern and I/O optimization approach. $t_j(s)$ is the unit I/O time searched from the 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 during each iteration. So $n(1) = n(2) = 1$. By consulting the performance database and according to equation (1), $t(1) = 0.12$ and $t(2) = 8.47$. So the total time is

$$\begin{aligned} T_{\text{prediction}} &= \sum_{j=1}^{M} \big(N/freq(j) + 1\big) n(j) t_j(s) \\ &= (120/6 + 1) \cdot 0.25 \\ &\quad + (120/6 + 1) \cdot 8.47 \\ &= 2.59 + 177.98 = 180.57(s). \end{aligned} \qquad (3)$$

Our experiment shows that the actual time is about 197.40 (figure 9) which is consistent with our prediction. One example of using this performance prediction is that the user can choose a 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 to be as small as possible. Our performance predictor can provide a lower bound for this parameter that might be very helpful for the user in choosing a suitable maximum run time.

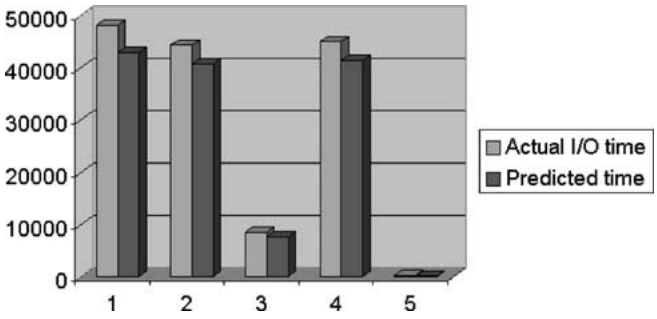


Figure 9. I/O time for Astro3D: (1) write all datasets to remote tapes; (2) write *temp* to remote disks and all other datasets to remote tapes; (3) write only *temp* to remote disks and *press* to remote disks; (4) write *vr-temp* to local disks and all the other datasets to remote tapes; (5) write only *vr-temp* to local disks and *vr-press* to remote disks.

5. Experiments

In this section, we present various new opportunities that our multi-storage resource architecture can provide for high performance computing.

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 large 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 a tape system as a storage repository which is usually thought of as unlimited in space. The total I/O time is shown in figure 9(1) if we write all this data to tapes. Note that this time has already been optimized by collective I/O. Without collective I/O, it would be many times slower. Then suppose the user wants to perform data analysis (MSE) on dataset *temp*, then the total I/O time is shown in figure 10(a). 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 that the system place *temp* on a 'closer' storage

Table 2
Run-time parameter set of Astro3D.

Item	Size	Data type
Problem size	$128 \times 128 \times 128$	–
Max num of iterations	120	–
Data analysis freq	6	Float
Data visualization freq	6	Unsigned Char
Checkpointing freq	6	Float

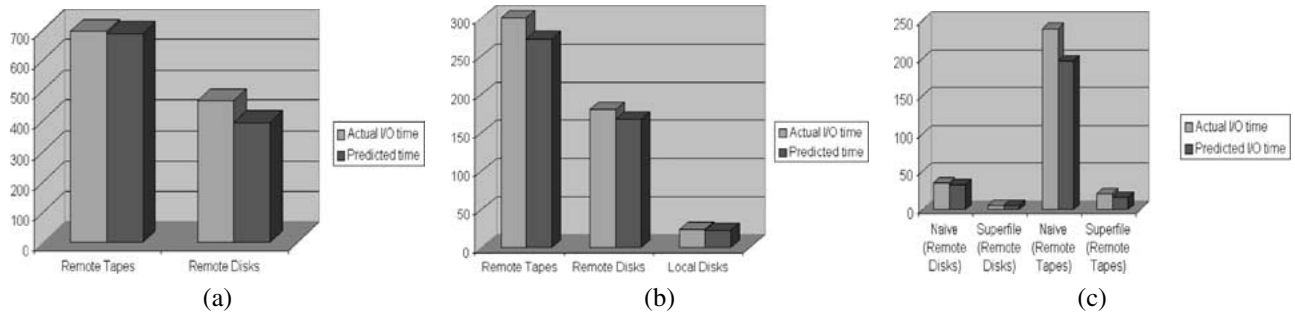


Figure 10. I/O time for data analysis (a), visualization (b) and superfile (c).

medium (such as remote disks in this example). This subset of the total datasets generated by Astro3D could be small enough to be held on a faster medium for fast data access and our multi-storage resource system can provide a platform for such an opportunity. Although the total time of generating all this data is not saved much by placing the *temp* on remote disks (figure 9(2)), when measuring the entire investigation including running the simulation and performing the data analysis, she can save significant time because the remote disk access will be faster than retrieving the data from tape (figure 10).

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 the other datasets by providing a 'location' hint such as DISABLE. So the total I/O time of Astro3D can be decreased significantly (figure 9(3)).

In our second example, the user performs parallel volume rendering on *vr-temp* or interactively visualizes the data using VTK, she can suggest that the system dump *vr-temp* to local disks. *Vr-temp* consists of unsigned characters, its size is small enough to 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 sent 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 9(5)). When the user reads *vr-temp* by parallel volume rendering or interactive visualization tools (VTK), the total read time is 10 times faster than from tapes. If user also reads *vr-press*, she can also save time by reading data from remote disks (figure 10).

Our next example is to demonstrate a novel optimization approach called *superfile* to efficiently access large numbers 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 written to one large *superfile* when they are created. Later on, when the user reads this 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 [26] in the naive approach. Figure 10(c) shows that the performance improvement is significant.

In our final example, suppose that the remote tape system is down for maintenance, recovery or other problems which could often happen. We can still satisfy large storage space requirements 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 figures 9 and 10. Our prediction is quite close to the actual I/O time.[4] Our performance predictor is integrated with our IJ-GUI [27], a Java graphical environment that can help the user submit her job, carry out visualization, perform data analysis and so on. Figure 11 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. Related work

The related work can be divided into four groups.

One is parallel file systems, including IBM Vesta [10] and PIOFS [11], Intel Paragon [24], HP Exemplar [5], Galley [23], PPFS [19], PIOUS [22] 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 improvements. The storage of these systems usually includes only 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 [35–37], PASSION [8,33,34], PANDA [7,25] and others [4,27,39]. These systems provide high level structured interfaces on top of low level native parallel file systems [20] and try to match the applications' data structure which is usually a 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.

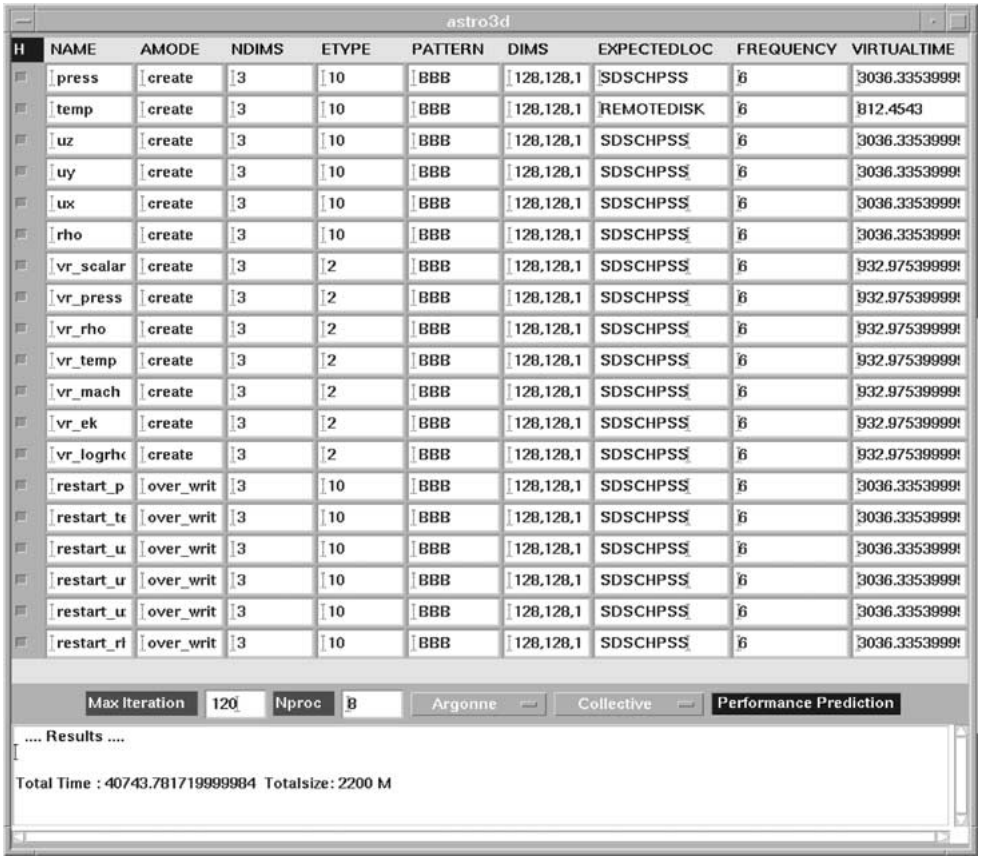


Figure 11. A prediction result of Astro3D. In this example, dataset *temp* is written to remote disks and all the other datasets are dumped to remote tapes. The I/O optimization approach is collective I/O and maximum iteration number is 120. The prediction result is commensurate with the actual I/O cost in figure 9(2).

The third group is distributed file systems such as NFS [32], xFS [1], and Coda [9]. 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 [13–15,17] 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* [6] has proposed some design principles of storage systems and data management for large data collections. For the large amounts of small files typically found in digital library systems, SRB's container [31] concept can significantly reduce the number of tape storage accesses, but it still suffers from multiple remote data accesses in a distributed environment.

7. Conclusions and future directions

In this paper, we have presented a *multi-storage resource architecture* for scientific simulations, which 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 without suffering their disadvantages. This architecture 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 is very close to the actual I/O time.

We would like to add more storage media to our system. Our current post-processing programs and tools are all installed locally. In the future, they could also be distributed. Until now, we required the user to explicitly specify the storage type hint. 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.

Acknowledgements

This research was supported by the 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 Reagan 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.

Notes

1. Northwestern University is very close to Argonne National Lab and the data exchange between Postgres database at Northwestern and the application at Argonne is small, so we treat it as a local database.
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. At SDSC, the HPSS internal API is, in general, reserved for system administration.
4. For remote systems, because the network is involved, there is fluctuation in performance numbers possibly due to different network traffic conditions. The numbers shown here are the most common cases.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli and R. Wang, Serverless network file systems, in: *Proc. of the 15th ACM Symposium on Operating Systems Principles* (1995) pp. 109–126.

[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. of CASCON'96 Conference* (1996).

[3] C. Baru, R. Moore, A. Rajasekar and M. Wan, The SDSC storage resource broker, in: *Proc. of CASCON'98 Conference*, Toronto, Canada (December 1998).

[4] R. Bennett, K. Bryant, A. Sussman, R. Das and J. Saltz Jovian, A framework for optimizing parallel I/O, in: *Proc. of the 1994 Scalable Parallel Libraries Conference* (1994).

[5] R. Bordawekar, S. Landherr, D. Capps and M. Davis, Experimental evaluation of the Hewlett-Packard exemplar file system, Proc. ACM SIGMETRICS Performance Evaluation Review 25(3) (1997) 21–28.

[6] 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.

[7] Y. Cho, M. Winslett, J. Lee, Y. Chen, S. Kuo and K. Motukuri, Collective I/O on a SGI CRAY Origin 2000: Strategy and performance, in: *Proc. of the 1998 International Conference on Parallel and Distributed Processing Technique and Applications* (1998) pp. 485–492.

[8] 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 (September 1994).

[9] Coda File System, http://www.coda.cs.cmu.edu.

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

[11] 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) (January 1995) 222–248.

[12] R.A. Coyne, H. Hulen and R. Watson, The high performance storage system, in: *Proc. of the Conference on Supercomputing*, Portland, OR (1993).

[13] I. Foster and C. Kesselman, Globus: A metacomputing infrastructure Toolkit, International Journal of Supercomputer Applications (1997) 115–128.

[14] I. Foster and C. Kesselman, The Globus project: A status report, in: *Proc. of IPPS/SPDP'98 Heterogeneous Computing Workshop* (1998) pp. 4–18.

[15] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann, Los Altos, CA, 1998).

[16] I. Foster, D. Kohr Jr., R. Krishnaiyer and J. Mogill, Remote I/O: Fast access to distant storage, in: *5th Workshop on I/O in Parallel and Distributed Systems* (1997).

[17] Global Grid Forum, http://www.gridforum.org.

[18] HPSS Worldwide Web Site, http://www.sdsc.edu/hpss.

[19] 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* (1995) pp. 385–394.

[20] D. Kotz, Multiprocessor file system interfaces, in: *Proc. of the 2nd International Conference on Parallel and Distributed Information Systems* (1993) pp. 194–201.

[21] 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.

[22] S. Moyer and V. Sunderam, PIOUS: A scalable parallel I/O system for distributed computing environment, in: *Proc. of the Scalable High-Performance Computing Conference* (1994) pp. 71–78.

[23] N. Nieuwejaar and D. Kotz, The Galley parallel file system, in: *Proc. of the 10th ACM International Conference on Supercomputing*, Philadelphia, PA (May 1996) pp. 374–381.

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

[25] K.E. Seamons, Y. Chen, P. Jones, J. Jozwiak and M. Winslett, Server-directed collective I/O in Panda, in: *Proc. of the Conference on Supercomputing*, San Diego, CA (December 1995).

[26] X. Shen, W. Liao and A. Choudhary, Remote I/O Optimization and Evaluation for Tertiary Storage Systems through Storage Resource Broker, in: *Proc. of IASTED Applied Informatics*, Innsbruck, Austria (2001).

[27] 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 2000).

[28] P.H. Smith and J. van Rosendale, Data and Visualization Corridors, Report on DVC Workshop Series (1998).

[29] W. Smith, I. Foster and V. Taylor, Predicting application run time using historical information, in: *IPPS/SPDP '9 Workshop on Job Scheduling Strategies for Parallel Processing* (1999).

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

[31] SRB Version 1.1.7 Manual, http://www.npaci.edu/DICE/SRB/OldReleases/SRB1-1-7/SRB.htm.

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

[33] 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).

[34] R. Thakur, A. Choudhary, R. Bordawekar, S. More and S. Kuditipudi, Passion: Optimized I/O for parallel applications, IEEE Computer 29(6) (1996) 70–78.

[35] 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).

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

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

[38] R. Thakur, E. Lusk and W. Gropp, I/O characterization of a portable astrophysics application on the IBM SP and Intel Paragon Preprint, MCS-P534-0895, Argonne National Laboratory, Mathematics and Computer Science Division (1995).

[39] S. Toledo and F.G. Gustavson, The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations, in: *Proc. of 4th Annual Workshop on I/O in Parallel and Distributed Systems* (1996).

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

Xiaohui Shen received his Ph.D. from Northwestern University, IL, in computer engineering, in January 2001. He is currently a Senior Software Engineer in Core Technology Department, Personal Communication Sector, Motorola Inc. His research interests include parallel and distributed computing, data management, I/O and storage systems and wireless Java. He received his M.S. and B.S. degrees from Department of Computer Science at Tsinghua University, Beijing. He has served as a program committee member for several international conferences and workshops.
E-mail: axs095@email.mot.com.

Alok Choudhary received his Ph.D. from the University of Illinois, Urbana-Champaign, in electrical and computer engineering, in 1989, M.S. from the University of Massachusetts, Amherst, in 1986 and B.E. (Hons.) from Birla Institute of Technology and Science, Pilani, India, in 1982. He is currently a Professor in the Electrical and Computer Engineering Department and the Kellogg School of Management, Marketing at Northwestern University, IL. From 1989 to 1996 he was a faculty member in the ECE Department at Syracuse University. He has worked in industry for computer consultants prior to 1984. His main research interests are in high-performance computing and communication systems and their applications in many domains information processing, data mining and scientific computing. In particular, his interests lie in the design and evaluation of architectures and software systems (from system software such as runtime systems, compilers and programming languages to applications), high-performance servers, and input-output. He has served as program chair and general chair for several conferences in parallel and high-performance computing areas. Choudhary received the National Science Foundation's Young Investigator Award in 1993 (1993–1999). He has also received an IEEE Engineering Foundation award, an IBM Faculty Development award and an Intel Research Council award.
E-mail: choudhar@ece.nwu.edu

Celeste Matarazzo is a computer scientist at Lawrence Livermore National Laboratory, currently working on the ASCI (Advanced Simulation and Computing) program. She has more than fifteen years of experience in software development. She is a research program manager and leader of the Data Science research group in the Center for Applied Scientific Computing. She also leads the ASCI Scientific Data Management (SDM) project. This project aims to provide intelligent assistance in managing terabytes of complex scientific data through the development of data models and tools and the integration of databases, storage, networks and other computing resources. Previous positions included developing software for climate modeling simulations, output devices, and defense applications. She has a Bachelor of Science degree in mathematics and computer science from Adelphi University.
E-mail: matarazzo1@LLNL.gov

Punita Sinha is a computer scientist at Lawrence Livermore National Laboratory. She has over 17 years of experience in software development. Her expertise is in large scale software development projects. She is currently working in the National Ignition Facility project at Livermore as part of the Integrated Computer Control Systems team for the world's largest laser system. She was previously working in the Scientific Data Management team of the ASCI (Accelerated Strategic Computing Initiative) Project at Livermore, evaluating academic research and commercial tools in data analysis and data management. Prior to LLNL she worked at IBM's Silicon Valley Lab developing commercial compilers and other application development tools. In addition to development work she has done business development for data management tools at IBM. She has an undergraduate degree in physics, chemistry and mathematics from Bangalore University, India; and a graduate degree in computer science from Boston University.