2000

A Java graphical user interface for large-scale scientific computations in distributed systems

X Shen

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Wei-keng Liao

Alok Choudhary

A Singh

Recommended Citation

X. Shen, G. Thiruvathukal, W. Liao, A. Choudhary, A. Singh, A Java graphical user interface for large-scale scientific computations in distributed systems, In proceedings of the Fourth International Conference on High-Performance Computing in the Asia-Pacific Region-Volume 1, 2000.

A Java Graphical User Interface for Large-Scale Scientific Computations in Distributed Systems

X. Shen, G. Thiruvathukal[*], W. Liao, A. Choudhary, and A. Singh[*]
Center for Parallel and Distributed Computing
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
{xhshen,wkliao,choudhar}@ece.nwu.edu

Abstract

Large-scale scientific applications present great challenges to computational scientists in terms of obtaining high performance and in managing large datasets. These applications (most of which are simulations) may employ multiple techniques and resources in a heterogeneously distributed environment. To work effectively in such an environment is crucial for modern large-scale simulations. In this paper, we present an integrated Java graphical user interface (IJ-GUI) that provides a control platform for managing complex programs and their large datasets easily. As far as performance is concerned, we present and evaluate our initial implementation of two optimization schemes: data replica *and* data prediction. Data replica *can take advantage of 'temporal locality' by caching the remote datasets on local disks;* Data prediction, *on the other hand, provides prefetch hints based on datasets' past activities that are kept in databases. We first introduce the* data contiguity *concept in such an environment that guides* data prediction. *The relationship of the two approaches is discussed in closing.*

1 Introduction

Modern simulations generate huge amounts of data and employ multiple techniques to subsequently process the data. These processes include data analysis, visualization, post-processing, etc. As the size of the datasets for these applications is typically huge, hierarchical storage must be utilized as the data repository. Additionally, databases can be introduced to help users manage data at a high level and to obtain high performance. Therefore, there are multiple resources involved in a modern large-scale scientific environment. As technology trends are moving towards distributed systems, all these resources may be distributed geographically, and these resources may be heterogeneous (i.e. different architecture, operating system, etc.) Without designing an efficient integrated environment, users may be overwhelmed in this environment, where a number of difficult decisions need to be made, often involving manual intervention or explicit programming. Consider the important issue of performance. In a distributed environment, distance between the user and the data is a major consideration for two reasons. First, the data sets may be located elsewhere (i.e. at another site). Second, the data sets may be on tertiary storage. In either case, it benefits the user to have an integrated environment that makes it possible for her to make appropriate performance decisions. Furthermore, when the user considers performance, it is not only the performance of the application (i.e. the simulation) butthe performance of other processes as well: visualization, data analysis, etc.

A computational scientist thus needs to fully consider the following issues in one overall picture when she wants to perform a high-performance simulation or other large-scale application:

- **High performance for simulations** For data intensive applications, state-of-the-art I/O optimizations such as collective I/O, prefetch, prestage and so forth should be employed.
- **High performance for other processes** If the user only considers simulation itself, it may result in bad performance when she comes to visualization, post-processing and data analysis. The user should be careful to arrange the layouts of her datasets properly on storage systems.
- **Easy-to-use** Database techniques are employed to fulfill this purpose. The database can maintain meta-data information about the applications, datasets, storage systems and so on and it provides easy query capabilities.
- **Integrated graphical environment** If the user works

[*] School of Computing, Telecommunications, and Information Sciences, JHPC Laboratory, DePaul University, email: gkt@cs.depaul.edu, arti@jhpc.cs.depaul.edu

Proceedings of the Fourth International Conference/Exhibition on
High Performance Computing in Asia-Pacific Region
0-7695-0589-2/00 $10.00 © 2000 IEEE

on an uniform platform rather than deal with distributed resources manually and explicitly, high efficiency can be achieved. Java proves to be a powerful language for such a task.

- **Optimizations in the integrated environment** Given the fact that the speed and reliability of networks are not met with the user's requirements, aggressive optimizations in a distributed environment are required to hide the network latency and reduce the probability of network failures.

A computational scientist would be overwhelmed if she has to consider all these issues which are beyond her expertise. Although works addressing the above issues have been done separately in literature, few of them have considered them in an overall framework. Brown et al. [3] build a meta-data system on top of HPSS using DB2 from IBM. The SRB [2] and MCAT [12] provide a uniform interface for connecting to heterogeneous data resources over a network. Three I/O-intensive applications from the Scalable I/O Initiative Application Suite are studied in [8]; however, all of these works only address one aspect of the set of issues presented in the previous section. Our previous work [6] is a first step toward considering multiple factors in an overall picture. We have designed an active meta-data management system that take advantage of state-of-the-art I/O optimizations as well as maintaining ease-of-use features by employing relational database techniques. In this paper, we present further considerations about integrated environment and its optimizations in distributed systems based on our previous work. We make the following contributions:

- We present the design of an integrated graphical environment in Java (IJ-GUI) for large-scale scientific applications in distributed systems. In our unified framework, users work only on their local machines and our IJ-GUI hides all the details of distributed resources. Users can launch the parallel application, carry out data analysis and visualization, query databases and browse the tables in an uniform interface.

- We present a *data replica* optimization scheme that takes advantage of local disks as cache to significantly improve visualization performance in our unified graphical environment.

- We introduce the concept of *data contiguity* which can be used to characterize usr's behavior when she carries out visualization.

- We present a *data prediction* scheme that can guide the choosing of datasets for prefetching or pretransferring (from remote sites to the local disks)

- We propose an automatic code generator component (ACG) to help users utilize the meta-data management system when they are developing new applications.

The remainder of the paper is organized as follows. In Section 2 we introduce an astrophysics application and a parallel volume rendering application that we use in our work. A shorthand notation is introduced for convenience. In Section 3 we give an overview of our design of meta-data systems (MDMS). We present our design of the integrated Java graphical environment (IJ-GUI) for scientific simulations in Section 4. The functions and the inner-mechanisms that IJ-GUI provides are presented. In Section 5 we first introduce the concept of *data contiguity* that is helpful for optimizations. Then two optimization approaches, *data replica* and *data prediction*, are described and evaluated. Finally we conclude our paper in Section 6.

2 Introduction to Applications

Our first application (Astro-3D [1]) is a code for scalable parallel architectures to solve the equations of compressible hydrodynamics for a gas in which the thermal conductivity changes as a function of temperature. 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. From a computer scientist's point of view, the application is just generating a sequence of data and dumping them on storages. Later on, the user may visualize the datasets in which she may be interested. In Astro-3D example, it generates six datasets such as temperature, pressure, etc. for each iteration in one run. Each of these datasets is written in a single file. Therefore, a data file is uniquely identified by dataset name, run id and the iteration number. We make the following notations to express a data file that is generated by concatenating the dataset name, run id and iteration number: *dataset-runid-iteration*. For example, if the temperature is dumped at the first iteration in the fifth run, it is notated as *temperature-5-1*; if the pressure is dumped at the second iteration in the sixth run, it is expressed as *pressure-6-2* and so on. The user can specify the number of iterations and dataset sizes as command-line arguments for each run.

Our second application is a parallel volume rendering code (called volren henceforth). It generates a 2D image by projection given a 3D input file. From a computer scientist's point of view, again, volren just reads a 3D input file and creates an 2D image file for each iteration per run. For example, volren may generate four image files (*image-5-1, image-5-2, image-5-3, image-5-4*) given four input data files at the fifth run.

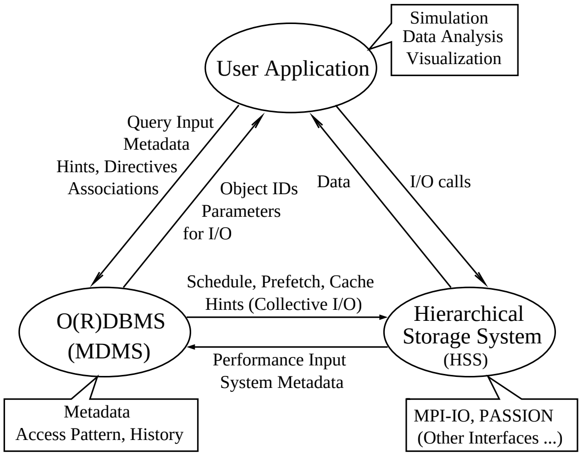

Figure 1. Three-tiered architecture.

3 Design of Meta-data Management System (MDMS)

Figure 1 shows a novel architecture we proposed in [6]. The three-tiered architecture contains three key components: **(1)** parallel application, **(2)** meta-data management system (MDMS), and **(3)** hierarchical storage system (HSS). These three components can co-exist in the same site or can be fully-distributed across distant sites. The MDMS is an active part of the system: it is built around an ORDBMS [17, 16] and it mediates between the user program and the HSS. The user program can send query requests to MDMS to obtain information about data structures that will be accessed. Then, the user can use this information in accessing the HSS in an optimal manner, taking advantage of powerful I/O optimizations like collective I/O [18, 5, 11], prefetching [10], prestaging [7], and so on. The user program can send *access pattern hints* to the MDMS and let the MDMS to decide the best I/O strategy considering the storage layout of the data in question. These access pattern hints span a wide spectrum that contains inter-processors I/O access patterns, information about whether the access type is read-only, write-only, or read/write, information about the size (in bytes) of average I/O requests, and so on. The MDMS design consists of design of database tables and the high-level MDMS user API. The database tables show what meta-data should be maintained and the MDMS user API shows how these meta-data will be used for I/O optimizations. The most distinguished feature of our design is that by comparing the user specified storage pattern and access pattern, our MDMS automatically chooses best I/O optimizations. The design details can be found in [15].

4 Design of Java Graphical User Interface

4.1 Architecture of Integrated Java GUI

As it is distributed in nature, our programming environment involves multiple resources across distant sites. Consider our current working environment: we are working on local HP or SUN workstations, the visualization tools are installed on a Linux machine, our database (POSTGRESQL) is located on another machine and our parallel applications run on a 16 node IBM SP2. Although these machines are within our department, they could be distributed across a wide area and connected via the Internet.

When a user starts to work in such a distributed system, she needs to go through the following procedures:

(1) log on to SP2 and submit the parallel application.

(2) When the application is finished, she needs to log on to the database host and use native SQL language to inspect the database to find datasets she would be interested in for visualization.

(3) When the user is interested in a particular dataset, she would transfer the data file explicitly, for example using ftp, from SP2 where data are located to the visualization host (DATA) where visualization tools reside.

(4) Log on to the visualization host (DATA) and start the visualization process.

(5) Repeat steps 2-4 as long as there exist datasets to be visualized.

Obviously, these steps might be very time-consuming and inconvenient for the users. To overcome this problem (which is due to the distributed nature of the environment), an integrated Java graphical user interface (IJ-GUI) is implemented and integrated to our application development environment. The goal of the IJ-GUI is to provide users with an integrated graphical environment that hides all the details of interaction among multiple distributed resources (including storage hierarchies).

Java has been chosen because it is an enabling access language and operating environment that supports all platforms of interest, including IBM AIX, Linux, Windows NT, Solaris, and others. Transparency is made possible by the many platform-independent abstractions of Java, including process management (the built-in Runtime and Process classes), multithreading (a language feature), networking and streams (built-in classes), GUI components (the Abstract Windowing Toolkit), and database access (Java Database Connectivity, or JDBC). Java has proven to be flexible and delivers good performance in all of these dimensions without being in any way on the critical path of

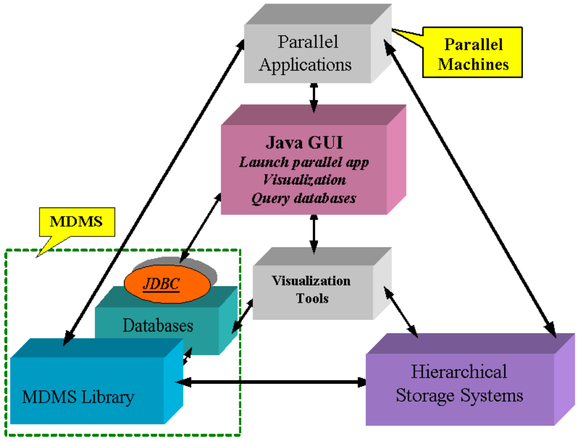

Figure 2. Java GUI in overall system.

performance in our environment. In this environment, the users need to work only with IJ-GUI locally, rather than go to different sites to submit parallel applications or to do file transfers explicitly. Figure 2 shows how IJ-GUI is related to other parts of our system. It actively interacts with three major parts of our system: (1) parallel machines to launch parallel applications. (2) databases through JDBC to help users query meta-data from databases. (3) visualization tools to carry out visualization process.

4.2 Main functions of IJ-GUI

The main functions that IJ-GUI provides are described as follows:

- **Registering new applications** To start a new application, the user needs to create a new suite of tables for the new application. By IJ-GUI, the user needs only to specify attributes of run table that she would be interested in, and all the other tables will be created automatically with run table.

- **Running applications remotely** The applications are usually running somewhere on parallel machines such as SP2, which are specified by the user when she registers a new application. Therefore, remote shell command is used in IJ-GUI to launch the job on remote parallel machines. The user can specify command line arguments in the small text fields as well. Defaults are provided and the user can change them as needed. The running results will be returned in the large text area.

- **Data Analysis and Visualization** Users can carry out data analysis and visualization through our IJ-GUI. Data Analysis may come in a variety of flavors, it is quite application specific. For some applications, data analysis may simply calculate the maximum, minimum or average value of a given dataset, for some others, it may be plugged into the application and calculate the difference between two datasets and decide whether the dataset should be dumped now or later. Our systems current method of data analysis is to calculate the maximum, minimum and means of each dataset generated. From the IJ-GUIs point of view, it is no different than just submitting a remote job. Visualization is becoming an important approach in large-scale scientific simulation to inspect the inside nature of datasets. It is often a little bit more complicated than data analysis: first of all, the users' interest in a particular data set may be very arbitrary. Our approach is to list all the candidate datasets by searching the database by user-specified characteristics such as maximum, minimum, means, iteration numbers, pattern, mode and so on. Then the candidates are presented in radio box for user to choose easily. Second, the datasets that are created by parallel machines, are located either at parallel machines or stored in hierarchical storage systems. But our visualization tools are installed in some other places. Therefore, inside IJ-GUI, we transparently copy the data from the remote parallel machine or hierarchical storage systems to the visualization host and then start the visualization process. The user does not need to check the database for interesting datasets or do data transfer explicitly. The only things the user has to do are to checkmark the radio box for interesting datasets, select a visualization tool (vtk, xv etc.), and then click the Visualization button to start the process of visualization. Our current visualization tools include Visualization Toolkit (VTK), Java 3D, XV etc.

- **Table browsing and searching** Advanced users may want to search the database to find datasets of particular interest. So the table browsing and searching functions are provided in our IJ-GUI. The user can just move the mouse and pick a table to browse and search the data without logging on to a database host and typing native SQL script.

- **Automatic Code Generator** Our IJ-GUI relieves users great burden of working in a distributed system with multiple resources. For an application that has already been developed, the user would find it very easy to run her application with any parameters she wants: she can easily carry out data analysis and visualization, search the database and browse the tables. For a new application to be developed, however, although our high-level MDMS API is easy to learn and use, the user may need to make some efforts to deal with data structure, memory allocations and argument selects for the MDMS functions. Although these tasks may be considered routine, we want to reduce them to almost zero by designing an Automatic Code Generator (ACG) for MDMS API. The idea is that given a specific MDMS function and other high-level information such as the access pattern of a dataset, ACG will automatically generate a code segment that includes variable declarations, memory allocations, variable assignments and identifications of as many of the arguments of that API as possible. The most significant feature of ACG is that it does not just works like a MACRO which is substituted for real codes: it may also consult databases for advanced information if necessary. For example, to generate a code segment for set-run-table(), which is to insert one row into the run table to record this run with user-specified attributes, our ACG would first search the database and return these attributes, then, it uses these attributes to fill out a predefined data structure as an argument in function set-run-table(). Without consulting the database, the user has to deal with these attributes by hand. Our ACG is integrated within our IJ-GUI as part of its functions. The user can simply copy the code segment generated and paste them in her own program.

Our current IJ-GUI is implemented as a standalone system, we are also embedding it into the web environments, so the user can work in our integrated environment through a web browser.

5 Optimizations in IJ-GUI

One significant feature of our IJ-GUI is that it also provides performance-oriented optimizations. When we look back the main functions that IJ-GUI provides in section 4, we would find that running new application and data analysis do not incur a performance problem because IJ-GUI only launches a remote job; registering new applications, table browsing and searching only negotiate with database for exchanging meta-data, namely, the size of data is small. Therefore, the only performance problem the user may encounter is in the process of visualization. The size of data sets for visualization could be very large since it is typical for modern simulations to generate huge datasets nowadays. These datasets are usually stored on tertiary storage systems such as HPSS [7]. Without saying, the performance to access tertiary storage systems is considered poor. Moreover, technology trend favors distributing resources across distant sites rather than centering on one spot, i.e., our visualization tools could be installed far away from where datasets are stored. In addition, the network failures may significantly reduce the data access reliability. In this section, we present two optimization schemes, both of them aggressively take advantage of meta-data kept in databases.

5.1 Data Replica

The idea of data duplication is that if we can keep a copy of data on local visualization host, therefore, when the user asks for this data later, without going to remote tertiary storage systems, the local copy can be served. In other word, we take advantage of ‘temporal locality’. The idea is very much like introducing a cache between processor and main memory, but the difference is that there is no data consistency problem in this case because the datasets here are read-only. Based on these characteristics, employing data duplication approach seems very attractive because there is no data consistency problem as encountered in processor-memory cache which significantly complicates the system and compromise the performance achieved.

As in processor-memory cache, the data replica should also address capacity problem: the local host has limited space for data duplications. This means when the allocated space for data replica is exceeded, some replica will be chosen evicted. The LRU (Least-Recently-Used) approach applies very well here: we associate an attribute, reference counter, to each cached data replica, whenever this dataset is accessed by the user, the reference counter will increment by one. Therefore, when the local host space is exceeded, the datasets with small reference numbers will be evicted until sufficient space is obtained.

5.2 Data Contiguity

We introduce the *data contiguity* concept in this subsection which is helpful for guiding our next optimization approach. We define two datasets are *contiguous* if their iteration number is contiguous with same dataset name and run id (called iteration-contiguous datasets), or if two datasets are associated (as described in Section 3) and with same run id (called association-contiguous datasets), or if the run id is contiguous with the same dataset name and iteration number (called run-contiguous). For instance, *temperature-5-1* and *temperature-5-2* are iteration-contiguous, *temperature-5-1* and *pressure-5-1* are association-contiguous since temperature and pressure are associated according to Section 3, and *temperature-5-1* and *temperature-6-1* are run-contiguous.

5.3 Data Prediction

The data replica can improve performance by taking advantage of ‘temporary locality’. But when a dataset is first asked by the user, or it is asked again when it is evicted from local disk due to limited space, the application will suffer poor data access latency as discussed previously. The idea to solve this problem is to make predictions about which datasets are going to be accessed later at a proper time. For instance, when the user read dataset *temperature-5-1* and it would most likely that the user would be also interested in *temperature-5-2*. Thus, an aggressive optimization approach is to prefetch the *temperature-5-2*. Here, prefetch means read datasets from storage systems or/and transfer data from remote sites to the local machine.

The questions must be answered here are: what data sets are candidates for prediction and when datasets should be prefetched.

The answer to the second question is straightforward: the prefetching could happen either when the user is examining the current dataset or when the user is searching or browsing the meta-data from databases. Thus, the prefetching can be overlapped by the user's current activity.

The first question, however, is not obvious, since the user's behavior could be very application-specific. But some general characterization of user's behavior can be identified by our *data contiguity* concept as follows. Users are likely to

- **Inspect iteration-contiguous datasets** These datasets of the same run are likely to be accessed shortly. For instance, when the user finishes examining temperature-5-1, she may be subsequently interested in the same dataset of next iteration: temperature-5-2, so she can find the difference of how temperature changes between two contiguous iterations.

- **Inspect association-contiguous datasets** As we have discussed in the previous sections, our MDMS library allows users to group different datasets in one association for manipulation. These associated datasets are closely related. For example, the associated temperature and pressure of Astro-3D application may influence each other. When the user carry out visualization, if she finds changes on temperature, she may then want to inspect how these changes influence the changes on pressure. Obviously, when the user is examining temperature, we can aggressively prefetch pressure datasets, thus when the user begins to inspect pressure, the data are already on the local disks.

- **Inspect run-contiguous datasets** The user may be interested in the same datasets(dataset name) on two contiguous runs. This is because the user may adjust parameters, re-run the application and examine the same datasets to see what changes.

In sum, *contiguous datasets* are most likely to be visualized by the users. Data contiguity can greatly help us make correct predictions. The prefetch based on data contiguity is very much similar to the processor-memory cache that employs block size to take advantage of spatial locality. In the next subsection, we present our initial implementation.

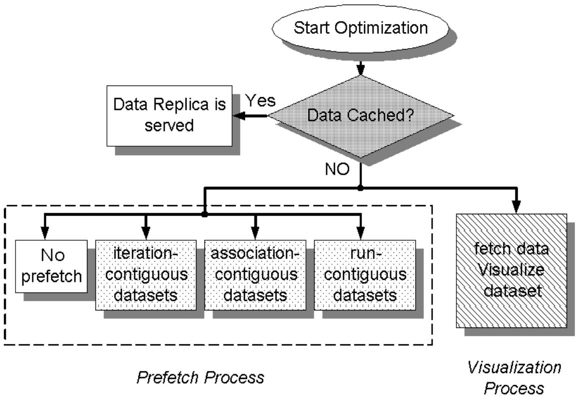

Figure 3. Optimizations in IJ-GUI.

5.4 Implementations

We create two performance-related tables in the database to help IJ-GUI perform aggressive optimizations for visualization. One table is called data-access-trace, whose attributes (fields) include application name, dataset name, iteration number, date and time, run id etc. Whenever the user carries out visualization on a dataset, a row is inserted to record users' behavior on datasets. Another table is called data-replica, whose attributes (fields) include, application name, dataset name, iteration number, local directory and reference counter. It keeps information about what datasets are currently cached on the local disk and how many times they are accessed by the user. Figure 3 shows a flow of how and what kind of optimizations are performed through our initial implementation. The optimization procedure is summarized as follows.

(1) Whenever the user asks for a dataset for visualization, our optimized IJ-GUI will consult the data-replica table to see if there is an cached copy on local disks. If yes, the local data replica will be served.

(2) If the dataset is not found on local disk, then IJ-GUI has to read it from HPSS or transfer it from remote sites. A new data replica entry will be inserted into data-replica table once the dataset has been transferred to local host. If the local space for data replica is exceeded, some data replica will be chosen as victims according to the rules discussed previously. After that, the user will start the process of visualization and stay some time on it. In the meanwhile, another thread or process is spawn and run simultaneously with the above procedure to make data prediction decisions. These decisions include no prefetching due to lack of information or what kind of datasets should be prefetched: iteration-contiguous, association-contiguous or run-contiguous datasets. Once a dataset is chosen then IJ-GUI prefetches it on background when the user is visualizing the current dataset on foreground.

5.5 Discussions

The two optimizations we have discussed so far may not have the same ‘weight’ when we design optimization schemes in our IJ-GUI: The data replica approach is always ‘safe’ because it has no data consistency problems as discussed previously and it only has a slight overhead of accessing databases in the worst case; the prefetching approach, however, may hurt performance if it is not handled properly: this could happen when a wrong predicted dataset evicts from local disks a useful dataset which would be used in the future. Therefore, employing prefetch scheme should be very careful. This requires precise prediction scheme and smart caching policy. P. Cao et al [4] have studied the relationship of prefetch and caching between disk and memory. We believe their works are also applied to our situation. In our future work, we would investigate such relationship in our context.

6 Conclusions

In this paper, we presented an integrated Java graphical user interface (IJ-GUI) to efficiently help users work on an environment that is characterized by distributed and heterogeneous natures. Our IJ-GUI provides users an unified interface to all the resources and platforms presented to large-scale scientific applications. In addition, two optimization approaches, *data replica* and *data prediction* have been integrated into our IJ-GUI to achieve high performance. The last approach is based on data contiguity characteristic of scientific datasets. All these works take advantage of Java’s powerful features such as platform independence, portability, process management, multithreading, networking and streams. In the future, we would investigate other optimizations in our environment, such as subfiling [13]. The relationship between prefetch and caching in our context will also be studied.

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. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In *Proc. CASCON’98 Conference,* Dec 1998, Toronto, Canada.

[3] P. Brown, R. Troy, D. Fisher, S. Louis, J. R. McGraw, and R. Musick. Meta-data sharing for balanced performance. In *Proc. the First IEEE Meta-data Conference,* Silver Spring, Maryland, 1996.

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

[5] 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.

[6] A. Choudhary, M. Kandemir, H. Nagesh, J. No, X. Shen, V. Taylor, S. More, and R. Thakur. Data management for large-scale scientific computations in high performance distributed systems, In *Proc. the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC’99),* August 3-6, 1999, Redondo Beach, California.

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

[8] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed. Input/output characteristics of scalable parallel applications. In *Proceedings of Supercomputing’95.*

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

[10] C. S. Ellis and D. Kotz. Prefetching in file systems for MIMD multiprocessors. In *Proc. the 1989 International Conference on Parallel Processing,* pages I:306–314, St. Charles, IL, August 1989. Pennsylvania State Univ. Press.

[11] 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.

[12] MCAT *http://www.npaci.edu/DICE/SRB/mcat.html*.

[13] 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.*

[14] R. Ramakrishnan. *Database Management Systems,* The McGraw-Hill Companies, Inc., 1998.

[15] 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. Submitted to International Conference on Supercomputing, May 2000, Santa Fe, New Mexico

[16] M. Stonebraker. *Object-Relational DBMSs : Tracking the Next Great Wave.* Morgan Kaufman Publishers, ISBN: 1558604529, 1998.

[17] M. Stonebraker and L. A. Rowe. The design of Postgres. In *Proc. the ACM SIGMOD’86 International Conference on Management of Data,* Washington, DC, USA, May 1986, pp. 340–355.

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

[19] 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. Fourth Annual Workshop on I/O in Parallel and Distributed Systems*, May 1996.

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