Deep scientific computing requires deep data

W. T. C. Kramer
A. Shoshani
D. A. Agarwal
B. R. Draney
G. Jin
G. F. Butler
J. A. Hules

Increasingly, scientific advances require the fusion of large amounts of complex data with extraordinary amounts of computational power. The problems of deep science demand deep computing and deep storage resources. In addition to teraflop-range computing engines with their own local storage, facilities must provide large data repositories of the order of 10–100 petabytes, and networking to allow the movement of multi-terabyte files in a timely and secure manner. This paper examines such problems and identifies associated challenges. The paper discusses some of the storage systems and data management methods that are needed for computing facilities to address the challenges and describes some ongoing improvements.

Introduction

Deep scientific computing has evolved to the integration of simulation, theory development, and experimental analysis as equally important components. The integration of these components is facilitating the investigation of heretofore intractable problems in many scientific domains. Often in the past, only two of the components were present: Computations were used to analyze theoretical ideas and to assist experimentalists with data analysis. Today, however, beyond each component informing the others, the techniques in each domain are being closely interleaved so that science investigations increasingly rely on simulations, observational data analyses, and theoretical hypotheses virtually simultaneously in order to make progress.

High-performance computing is now being integrated directly into some experiments, analyzing data while the experiment is in progress, to allow real-time adaptation and refinement of the experiment and to allow the insertion of human intuition into the process, thus making it very dynamic. When computational models operate in concert with experiments, each can be refined and corrected on the basis of the interplay of the two. The integration of computing with the other investigative methods is improving research productivity and opening new avenues of exploration.

In many cases, investigations have been limited by the computational power and data storage available, and these constraints, rather than the scale of the question being studied, have determined the resolution of a simulation or the complexity of an analysis. As available computational power, memory, and storage capacity increase, investigations can be expanded at a natural scale rather than being constrained by resources. But deep scientific computing can still be constrained by an inadequate capability to cope with massive datasets. In order to handle massive amounts of data, attention must be paid to the management of temporary and long-term storage, at the computing facility and elsewhere, and to networking capabilities to move the data between facilities.

An important aspect of the challenge of deep computing is the fact that today and in the foreseeable future no computational system can hold all needed data using on-line, local disk storage. As discussed later, for many applications, each step of a simulation produces gigabytes (GB) to terabytes (TB) of data. A deep computing system is used by multiple applications and for many time steps, so any delay in being able to move and access the data means under-utilizing the computational resource. Thus, a key subsystem in every facility involved in deep computing is a large data archive or repository that holds hundreds of terabytes to petabytes (PB) of storage. These archives

0018-8646/04/$5.00 © 2004 IBM

are composed of a hierarchy of storage methods ranging from primary parallel disk storage to secondary robotic tape storage to possibly tertiary shelf-based tape storage. The efficient management of data on such systems is essential to making the computational systems effective.

While large-scale computational and storage systems have been in place for decades, it is only in the past 20 years that networking began to change the way in which computing is performed. Initially, this was done via remote log-in access over connections that were relatively slow compared to the computing power and storage of the time. Since the mid-1990s, networking capabilities have evolved to the point that they have significantly changed the way in which large-scale resources are used. Indeed, the explosion of raw Internet bandwidth is enabling people to envision new paradigms of computing. One such new paradigm is *Grid computing* with the Open Grid Service Architecture [1]. Flexible access to computing and storage systems is now being implemented as a part of the Grid. This paper does not deal specifically with Grid issues, but concentrates on the underlying functions and methods required to enable distributed systems to reach their full potential.

Network capabilities have seen manyfold fundamental improvements in hardware, such as the change from copper-based networking to optical-fiber-based networking. Although these hardware improvements are expected to continue into the future, the performance of the networking protocols that were designed to operate on significantly lower-speed networks has not grown with the network capacity. End host paths from memory to the network also have often not kept pace with the improvements in the network capabilities. These lags have caused serious limitations in the end-to-end efficiency and utilization of applications running on the network. End-to-end networking technology must now keep pace, or it will not be able to match the exponentially increasing computational power of new systems and the dramatic increases in storage capacity. Middleware associated with the Grid introduces even more demands on the underlying data and network infrastructure. Furthermore, the protection of intellectual and physical assets in a networked environment is critical.

Because of the intense on-demand needs of many applications, a new requirement is emerging—the widespread deployment of high-performance network connections within and across shared networks. This requirement is different from the principles that led to the scalable Internet, which has grown rapidly over the last 10 to 15 years. The use of these network connections by multiple scientific fields will entail new concepts of fairness and new modes of network operation, monitoring, and management. The need for new solutions is heightened by the rapid development of Grids, which implicitly assume that adequate networks capable of quantifiable high performance will be available on demand for priority tasks.

In short, the use and movement of deep data adds another level of complexity to high-performance computing. This paper discusses several of the challenges posed by the need to handle massive amounts of scientific data at the very high end, and describes some possible approaches for doing so. It also examines the interplay among the three elements that make up deep computing: computation, storage, and networking. Unless these three are balanced, high-end computing will be less effective in addressing future needs.

The rest of this paper is organized as follows: The first section discusses examples of the applications that drive deep-data science in order to identify the capabilities and services needed to support them. The second section deals with methods of providing and managing associated large data repositories. The third section discusses networking problems that prevent deep data from flowing efficiently through the network and presents some methods of recognizing and resolving these problems. These sections include the following themes:

- Deep science applications must now integrate simulation with data analysis. In many ways this integration is inhibited by limitations in storing, transferring, and manipulating the data required.
- Very large, scalable, high-performance archives, combining both disk and tape storage, are required to support this deep science. These systems must respond to large amounts of data—both many files and some very large files.
- High-performance shared file systems are critical to large systems. The approach here separates the project into three levels—storage systems, interconnect fabric, and global file systems. All three levels must perform well, as well as scale, in order to provide applications with the performance they need.
- New network protocols are necessary as the data flows are beginning to exceed the capability of yesterday's protocols. A number of elements can be tuned and improved in the interim, but long-term growth requires major adjustments.
- Data management methods are key to being able to organize and find the relevant information in an acceptable time. Six methods are discussed that can be built into the applications and eventually into the underlying storage and networking infrastructure.
- Security approaches are needed that allow openness and service while providing protection for systems. The security methods must understand not just the application levels but also the underlying functions of storage and transfer systems.

- Finally, monitoring and control capabilities are necessary to keep pace with the system improvements. This is key, as the application developers for deep computing must be able to drill through virtualization layers in order to understand how to achieve the needed performance.

Applications that drive deep data science

Ideally, users would like all resources to be virtualized and not to have to deal with storage or transfer components and issues. However, in deep computing, the virtualization scheme breaks down because of the sheer magnitude of the problems and data. Virtualization implementations are typically targeted to more general cases in magnitude and intensity. Hence, for deep computing, the user and application often have to know much more about the implementation and details of the features and functions of a system than they would like. The following examples demonstrate that it is no longer possible to consider an application as limited by computation, networking, or storage. All are needed simultaneously. The main reason for using distributed computing resources is that the data size and/or the computational resource requirements of the task at hand are too large for a single system. This motivates the sharing of distributed components for data, storage, computing, and network resources, but the data storage and computational resources are not necessarily sited together. In this section, we describe several phases of the scientific exploration process that illustrate such requirements in order to identify the capabilities and services needed to support them, emphasizing end-to-end performance from the user's point of view.

Data production phase

Many scientific projects involve large simulations of physical phenomena that are either impossible or too expensive to set up experimentally. For example, it is too expensive to set up combustion or fusion experiments to investigate the potential benefit of a new design or to discover design errors. Instead, detailed simulations, usually involving high granularity of the underlying mesh structures, are used to screen candidate designs. Similarly, simulations are used in climate modeling because it is impossible to recreate climate phenomena accurately in a laboratory. In high-energy physics, simulations are conducted before the actual multi-billion-dollar experiments in order to design the hardware and software systems necessary to process the data from the experiment.

The above examples are typical of simulations that produce multi-terabyte datasets from long-running parallel computations. Providing such simulations with adequate computing resources may involve a single site with a large computing facility or an aggregation of multiple computing resources at multiple sites. There must be disk storage resources large enough to hold the simulation data and fast enough to keep pace with its generation so that the computing resources are used effectively. Furthermore, the data must be moved to deep archives as it is generated in order to free up the disk storage as rapidly as possible.

An example of generating a large volume of data during the simulation phase is a colliding black hole simulation [2] performed at the National Energy Research Scientific Computing Facility (NERSC). The collision of two black holes and the resulting gravitational waves were simulated. Since the gravitational wave signal that can be detected by interferometers in the field is so faint as to be very close to the level of noise in these devices, the simulated wave patterns are important tools for data interpretation. The code used performs a direct evolution of Einstein's general relativity equations, which are a system of coupled nonlinear elliptic hyperbolic equations that contain millions of terms if fully expanded. Consequently, the computational and storage resource requirements just to carry out the most basic simulations are enormous. These simulations had been limited by both the memory and the CPU performance of today's supercomputers.

One of the simulations, depicted in **Figure 1**, used 1.5 TB of RAM and more than 2 TB of disk storage space per run on the NERSC IBM SP* system. Runs typically consumed 64 of the large-memory nodes of the SP (containing a total of 1,024 processors) for approximately 48 wall-clock hours at a stretch. In the space of three months, these simulations consumed 400,000 CPU hours, simulating one full orbit before coalescence. Not only was this simulation very intensive in memory, on-line I/O, and CPU requirements, but it had extreme networking needs as well. In addition to the challenge of moving so much data off the computational engine to a storage archive, the entire application was designed to be interactively monitored and steered using advanced visualization tools. For example, at the 2002 Supercomputing Network Bandwidth Challenge competition (SC2002), this application used almost 17 gigabits per second of data bandwidth for the full application to be visualized in real time across systems at seven sites in four different countries [3]. Now that the concept has been successfully demonstrated, future efforts to expand the time scale of the simulation for a more complete understanding are expected to require 5 TB of RAM, 10 TB of on-line disk storage per run, and more than six million CPU hours.

Another example of an intensive data production phase is in the area of climate modeling. A recent data production run completed the first 1,000-year control simulation of the present climate [5] with the new Community Climate System Model (CCSM2) [6] developed at the National Center for Atmospheric

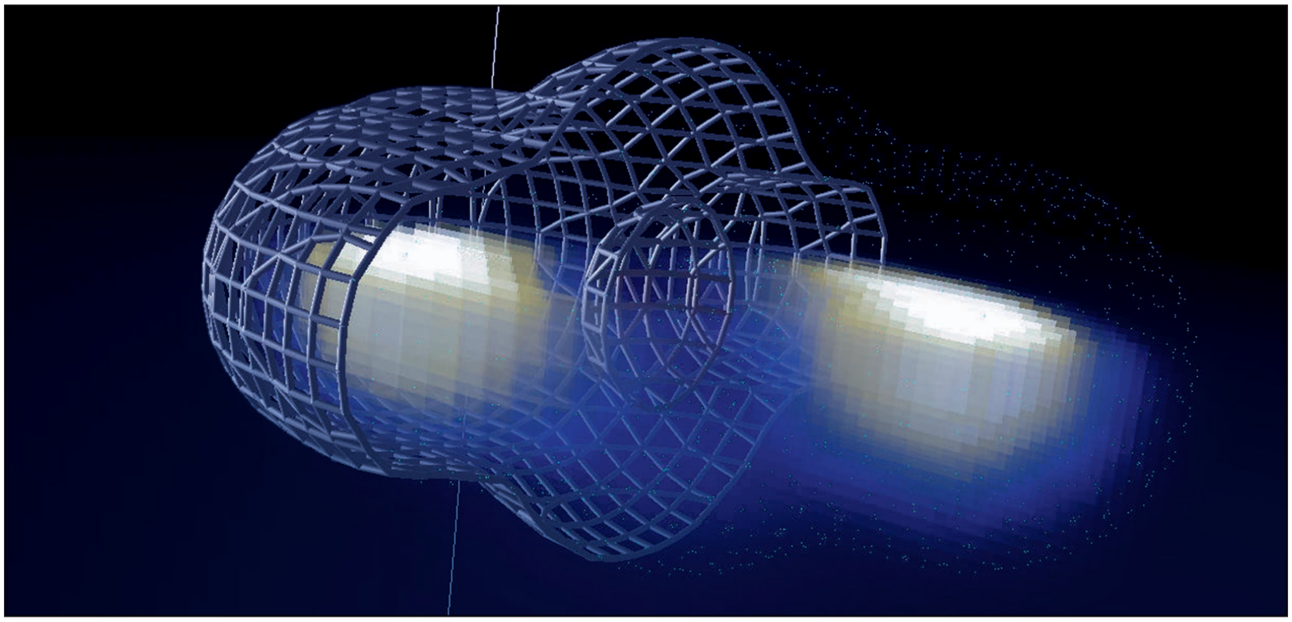

Figure 1

Simulation of binary black hole inspiral based on computations done at NERSC. The simulation required 400,000 hours of CPU time on the 1024 POWER3* microprocessors of the NERSC IBM SP system, 1.5 TB of RAM, and more than 2 TB of disk storage. From [4], with permission; © 2003 IEEE.

Research (NCAR). This simulation produced a long-term, stable representation of the earth's climate. Few climate models in the world can achieve this combination of accuracy, consistency, and performance; previous simulations contained too much drift to allow a complete, uncorrected simulation of 1,000 years. Computationally, the full CCSM2 code is complex, consisting of five integrated models that are organized to execute concurrently within a single job. The components exchange data at various frequencies appropriate to the large-scale physical processes being simulated through a "flux coupler" component. Each simulated year requires 6 GB of data to feed the next step in the simulation, and many intermediate files are produced. The requirements for this ongoing effort increase by a factor of 2 or more every year.

Data post-processing phase

Post-processing involves running application programs to interpret simulated or observed data. While some applications, such as the black hole simulation, use very little input data in the data production phase, the post-processing phase requires access to entire datasets generated by simulation programs or experiments. Post-processing components must be capable of performing the computation at the sites where the data is located or moving the data and the computation to a common site. Depending on the amount of data to be moved, this phase may be very lengthy. However, in many applications it is possible to overlap the movement of the input data with the computation, if the interpretation programs do not require all of the data at once. The interpretation programs may generate datasets larger than the input datasets.

An example of the post-processing of experimental data is the work of the Nearby Supernova Factory (SNfactory) [7]. Discovering supernovae as soon as possible after they explode requires imaging the night sky repeatedly, returning to the same fields every few nights, and then quickly post-processing the data. The most powerful imager for this purpose is the charge-coupled device (CCD) camera built by the Jet Propulsion Laboratory. This camera delivers 100 MB of imaging data every 60 seconds, and an upgraded version of the camera will more than double this. The new images are computationally compared to images of the same field using digital image subtraction to find the light of any new supernovae. Because the amount of data is so large (50 GB per night per observatory, or 18.6 TB per year), the image archive even larger, and the computations so extensive, it is critical that the imaging data be transferred to a large computing center (in this case NERSC) as quickly as possible. The refined data is then analyzed and compared to theoretical simulations in order to select candidate stars to watch more closely. The candidate list is then distributed to observatories around the world. This time-centered processing has resulted in a dramatic increase in the rate of detection of Type Ia supernovae, which now averages more than eight per month. This project brought new understanding of the universe and its fate, concluding almost a century of debate—one of the key scientific discoveries in recent times. The project is now contributing to the design of future experiments, such as the Supernova/Acceleration Probe (SNAP), a satellite which is now being developed.

Another example of the need for post-processing is the U.S. Department of Energy (DOE) Coupled Climate Model Data Archive, which makes output data from DOE-supported climate models freely available to the climate-modeling community [8]. This is the largest single collection of publicly available climate model output. Results from the NCAR-coupled general circulation models, PCM (pulse-code modulation) and CCSM2, are currently available. The data in these archives has been post-processed from the original output data so that it can be stored and accessed in a database that makes the data more convenient and useful to climate researchers. The volume of the post-processed data includes various summaries of the data as well as an inverted representation of the data, organized as time series per variable. Consequently, the volume of the post-processed data exceeds the volume of the original simulated data.

Current network limitations affect users of this climate data collection in two ways. First, and foremost, individual researchers generally download subsets of this data to their own remote sites for inclusion in their own specialized analysis programs. Hence, slow networks can limit the amount of data analyzed in a practical way. Second, because of the volume of post-processed data, several days are often necessary to transfer the contents of an entire simulation from NCAR mass storage to the NERSC High Performance Storage System* (HPSS). This is a substantial fraction of the time required to generate the post-processed data. In the future, projects such as the Earth System Grid [9] offer the prospect of supporting efficient distributed access to the collection. In this vision, model data would reside on storage media at the supercomputing center that produced the data. Metadata catalogs and interpretation programs would provide a seamless interface to the database, hiding the distributed nature of the underlying files. For this concept to be practical, however, network speeds must be increased substantially over current rates.

Data extraction and analysis phase

This phase generally involves the exploration of selected subsets of the data in order to gain insights into the data and to reach and present new conclusions about the data. The storage of data away from the site where the analysis is being done forces the use of a distributed computing model. For example, consider the need to create a sequence of images of the temperature variation over some region of the world for a certain ten-year period. The simulation may contain data for the entire globe over hundreds of years for 20 to 30 different variables in addition to temperature. The problem here is to extract the subset of the data needed, perhaps from multiple archives, and move it to the visualization site. The main capabilities and services required in this case are computing and disk storage resources. But the amount of space needed is only for the selected subset, typically a small fraction of the original dataset. Applications filter and extract the desired data at the location where the data resides, and move only the filtered data to the client's site. Assembly of the filtered data requires invoking an assembly application program and handing it the filtered subsets of the data.

A large-scale data analysis effort involving hundreds to thousands of collaborators worldwide is typical in several high-energy and nuclear physics experiments. One example that recently started full production is the STAR detector (Solenoidal Tracker At RHIC) at Brookhaven National Laboratory. The data analysis and simulation studies require the extraction of subsets of the data to be used by 400–500 collaborators from about 50 institutions worldwide. The post-processing phase takes the raw data from the detector and reconstructs particle tracks, momenta, and other data about collisions. In the data extraction and analysis phase, physics results are derived by carrying out statistical analysis of large numbers of particle collisions that must be extracted from archived files.

The STAR detector produces more than 300 TB of data per year, and it is only one of the experiments at the Relativistic Heavy Ion Collider (RHIC). All told, the four experiments at RHIC produce between 1 and 1.5 PB of data per year, and newer experiments will be even more voluminous. In 2007, when the Large Hadron Collider (LHC) goes on line at CERN in Europe, just one of the experiments, ATLAS (A Toroidal LHS ApparatuS), is expected to produce 1.5 PB of raw data per year. ATLAS is the largest collaborative effort ever attempted in the physical sciences, with 2,000 physicists participating from more than 150 universities and laboratories in 34 countries. The direct interaction of so many widely dispersed collaborators is made possible by tools for efficiently accessing, organizing, and automatically managing massive datasets.

Another rapidly growing area of science that will require efficient data extraction and analysis tools is genomics and bioinformatics. While currently relatively small in data requirements compared with some of the other disciplines, bioinformatics has large and highly distributed data needs that are growing at exponential rates. A single assembly of the fish *Fugu rubripes* [10], done with the JAZZ Genome Assembler [11] created by the DOE Joint Genome Institute, generated 30 GB of data files and used 150 GB of working space—and this species has an unusually small genome for a vertebrate. The leading sequencing facilities are now able to sequence one or more organisms a day, and the rate of increase with new technology is such that more and more raw sequences are being produced. Research in comparative genomics will require the extraction of datasets from the genomes of many different species. Projections are that within five years, many sites will have 100 TB of genomic data stored in the form of assembled and annotated genomes. If the raw image data were completely saved in digital form, the data requirements could be as much as 1,000 times greater.

Dynamic data discovery process

In the above phases, we assume that all of the input data for a computational job is available prior to execution. However, there is a growing trend toward more adaptive simulations, in which the input data required by an analysis depends on the results of just-executed computations. We can refer to this method of exploration as *dynamic data discovery*. A good example of this process is data mining, in which the researchers initially may not know exactly what they are looking for, but they want to find and map correlations and see which correlations represent significant trends. Agile data access and management techniques are a necessity for this kind of research, and detailed pre-planning of the data transfers is not always possible.

Another instance of dynamic data discovery is the running of simulations of different resolutions or initial conditions simultaneously with mutual feedback between the simulations so that they can refine each other's results. For example, typical global climate models today cannot resolve very narrow current systems (including fronts and turbulent eddies) that play a crucial role in the transport of heat and salt in the global ocean, nor can they resolve important sea ice dynamics that occur in regions of complicated topography, such as the Canadian Archipelago. Feeding data from the global model into a higher-resolution regional model, then transferring the regional results back to the global model, could increase the precision and accuracy of climate simulations.

Still another important example of dynamic data discovery is computational steering of a simulation on the basis of analysis or visualization of the current results. With interactive visualization, the client may choose to "stir" the simulation parameters using visualization-based tools, or to zoom in to obtain higher-granularity data for a more limited space. In this case, it may be necessary to change the plan of execution on the basis of observations of partial results. Computational steering requires that a control channel to the executing service be open, that the execution process be interruptible, and that a new or modified plan can be submitted. As in the previous examples, there is no implication that logical subtasks have to be performed in a sequential fashion. On the contrary, all subtasks should be performed in parallel if possible.

Table 1 summarizes the current and projected storage requirements [12] for several DOE Office of Science scientific disciplines using NERSC. Each discipline has multiple simulations and analyses going on simultaneously.

The user's view: End-to-end performance and function

An important goal in building deep computing capabilities that address the needs of scientists is providing responsiveness to the user. From the user's point of view, performance is measured by the time between the initiation of an action and its completion. In the case of a data transfer, this may be the period of time from the point at which the user issues the command to transfer the data to the point at which the data is available for use on the target machine. In order to effectively do scientific processing, the entire data path—including the path through the machine, the storage, the archives, and the networks—must present as little delay as possible and a path of the highest bandwidth possible for operations to occur in a timely manner. In the simple case, data flows from the memory of a source system, through a network interface card, over the local network, through the network interface of the destination system, and into its memory (and perhaps to on-line storage). Even in this simple model, there are a number of potential bottlenecks.

The simple model presented above is rarely the reality. The data path is usually much more complex and involves many more components, including routers, storage systems, archives, and other networks. Consider the example of a user working on a large-scale system located at a different site. The system may support computation and/or experimental analysis for any of the previously discussed phases. The user has a desktop and also a small server system that has relatively modest data and computational capability. The large-scale system generally has the data archived in a mass storage system. Other storage resources may also be assigned temporarily to the job to run the computation. In the simplest case, the computing and storage resources are all in one system, and the internal switch fabric is used. More often, the mid- to long-term storage is provided by another system, connected by an Internet Protocol (IP) or Fibre Channel

Table 1 Storage requirements for selected scientific applications.

Scientific discipline	*Near term*	*Five years*	*More than five years*
Climate	Currently there are several data repositories, each of the order of 20 to 40 TB.	Simulations will produce about 1 TB of data per simulated year. There will be several data repositories, each from 1 to 5 PB.	More detailed and diverse simulations. There will be several data repositories of the order of 10 PB each.
High-energy physics	Between 0.5 and 1.2 PB per experiment per year with five to ten experiments. Need network rates of 1 Gb/s.	1 PB or more per experiment per year with five to ten experiments. Need network rates of 1,000 Gb/s.	Exabytes (1,000 PB) of data with wide-area networking more than 1,000 Gb/s.
Magnetic fusion	0.5 to 1 TB per year with networking (for real-time steering and analysis) of 33 Mb/s per experimental site (three sites planned).	100 TB of data with network rates at 200 Mb/s per experimental site.	Hundreds of TB.
Chemistry	Simulations produce 10–30-TB datasets.	Each 3D simulation will produce 30–100-TB datasets.	Large-scale molecular dynamics and multi-physics and soot simulations produce 0.2 to 1 PB per simulation.
Bioinformatics	1 TB.		1 PB.

[13] network. When the application must move the simulation data to the archive, the data is organized into a large number of files whose movement to the archive is reliable and verifiable. After files are moved to the archive, the temporary storage is released automatically (garbage collection) for other uses. File movement from temporary storage to the archive can start as soon as each file is generated, which requires monitoring and progress reporting of file movement. A long-lasting job that may take many hours cannot be expected to be restarted in case of partial failure, so checkpoint and restart capabilities must also be supported. Also, in contrast to the simple case, the data must now flow through routers in the network. A router buffers each packet as it arrives and then sends it to the next router along the path toward the destination. At each step or hop, the packet may be redirected, broken into smaller parts, rejected, delayed, or just lost. The paths traveled by packets are determined by the routing protocols and the current status of the network. Sometimes different packets from the same data stream take different paths, at different speeds, and with different numbers of hops. It is not possible for the user to determine whether any of this is occurring, but any step along the way may affect and degrade end-to-end performance. Another factor in the achieved performance over the network is the transport protocol: Many protocols include some form of flow and/or congestion control which can limit their sending rate.

Data repositories for HPC systems

As mentioned above, any high-performance computing (HPC) facility supporting deep science must have multiple subsystems. There are one or more computing platforms, local data storage at the computing platforms, a data repository archive, visualization and other pre- and post-processing servers, local networking, and connections to one or more wide-area networks. Some facilities also have robotic tape storage. **Figure 2** shows the logical diagram of one such site—the flagship supercomputing facility of the DOE Office of Science, the NERSC Facility [14], located at the Lawrence Berkeley National Laboratory. The challenge at such facilities is to make efficient use of the available resources while performing the computations in an effective and timely manner. This challenge requires efficient individual storage components and software that can manage the combination of these components effectively. In the remainder of this section, we discuss the design of storage components and the software to effectively manage and stage for computation the data on the storage resources. Topics include storage systems, unified file systems, and managing large datasets in shared storage resources.

High Performance Storage System

The High Performance Storage System* (HPSS) [15] is one of several systems that serve to provide data storage and archive repositories. While not as common as some commercially oriented systems such as TSM** [16],

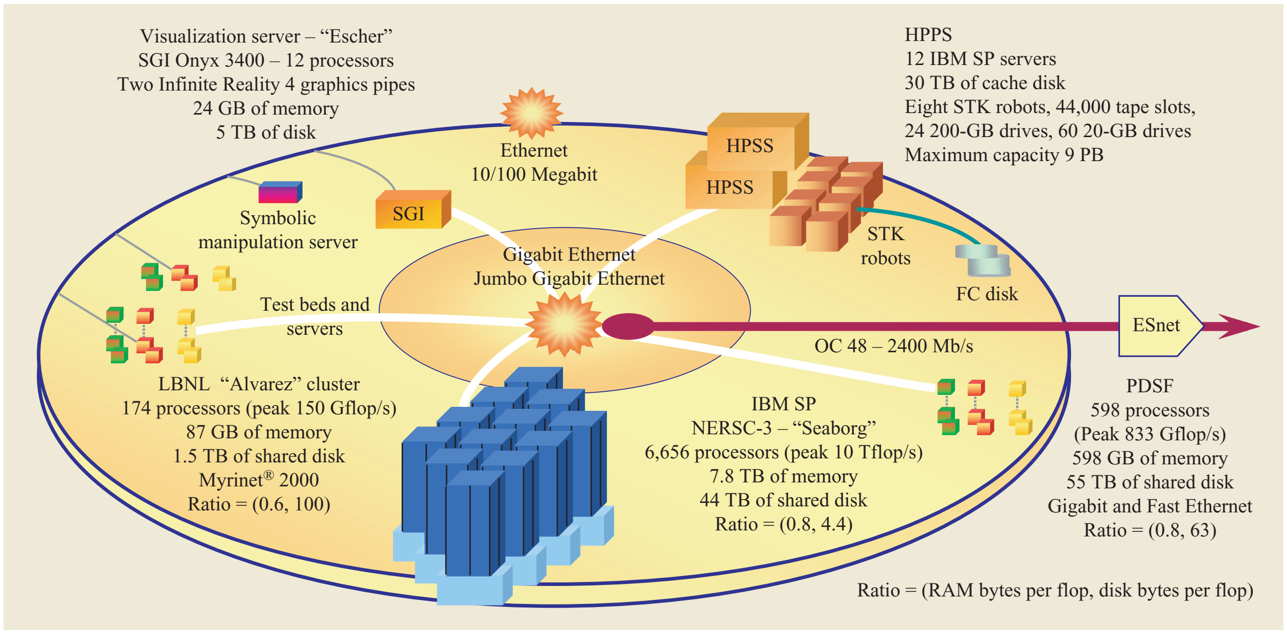

Figure 2

The NERSC system for deep science computing. The system contains one or more very large computational platforms (in this case a 10-Tflop/s IBM SP), a set of small computational systems (in this case several IA-32 clusters that range up to hundreds of CPUs), and a very large data archive repository (here HPSS, consisting of eight STK robots and 15 TB of Fibre Channel cache disk, with a maximum capacity of almost 9 petabytes). The networking consists of a major local-area network as well as one or more major wide-area connections.

SAMFS** [17], and VERITAS** [18], HPSS has been shown in the course of ten years of service to be effective, reliable, and highly scalable. It has replaced the Cray Data Migration Facility (DMF) [19] as arguably the most popular storage system used at supercomputing facilities. HPSS was developed as a collaborative effort involving IBM, six of the national research laboratories, the DOE (Department of Energy) Lawrence Livermore, Los Alamos, Sandia, Lawrence Berkeley, and Oak Ridge national laboratories, and the NASA (National Aeronautics and Space Administration) Langley Research Center. It has been in production service since 1996 at several sites, and now is used for large-scale data repositories at more than 25 different HPC organizations. While HPSS has many novel features, it is instructive to look at its design, evolution, and usage, since it is typical of systems that have to meet the requirements of deep computing sites.

HPSS design

HPSS is designed to move, store, and manage large amounts of data reliably between high-performance systems. The system provides very scalable performance that is close to the maximum of the underlying physical components and will track improvements in these components into the future. It must be parallel in all regards in order to achieve the high-performance goals required to move terabytes of data in a reasonable time period. It provides security and reliability and supports a wide range of hardware technology that can be upgraded independently. Thus, it is modular in design and function and treats the network as the primary mechanism for data movement. Rather than designing a system that was tied to the computational resource, the collaboration realized that HPSS had to be designed as a modular system, but also had to stand alone as a system itself. The HPSS architecture follows the IEEE Storage System Reference Model, Version 5 [20]. This model was developed from the experiences of several older archive storage systems such as the MSS system developed at NASA Ames Research Center, the Common File Storage System developed at Los Alamos National Laboratory, and others.

HPSS treats all files as bit streams. The internal format of a file is arbitrary and is defined by the application or originating system. HPSS stores all of the bits associated with a fileset on physical devices that are arranged in a hierarchy according to physical performance characteristics. There can be any number of hierarchies, which usually consist of different-speed disk and tape devices. A storage hierarchy is a strategy for moving data between different storage classes. Storage classes may consist of a single storage technology, such as a single type of tape, or multiple types of media. For example, one class may be very-high-speed RAIDs (Redundant Arrays of Inexpensive Disks) with parallel hardware interfaces, while another class is composed of slower, cheaper disks with more capacity. Most sites have one or more storage classes that use tape as the storage media. Often the tape is automatically managed with robot tape libraries such as those from StorageTek[1] or IBM.

A typical HPSS configuration is shown in **Figure 3**. Data flows over the network to a cache disk storage device. Then, following the site policy, one or more copies of the data move to lower storage classes, which presumably consist of cheaper but slower storage devices. Once data moves, it is deleted from the original storage class, making room for new data to move to the higher storage class. This process is called migration.

It is also possible to segregate types of files into different hierarchies. For example, very large files may be handled by placing them on very fast, very expensive disks, but then migrating them to tape media designed to hold large amounts of data. HPSS provides both serial and parallel access to data stored in its storage classes.

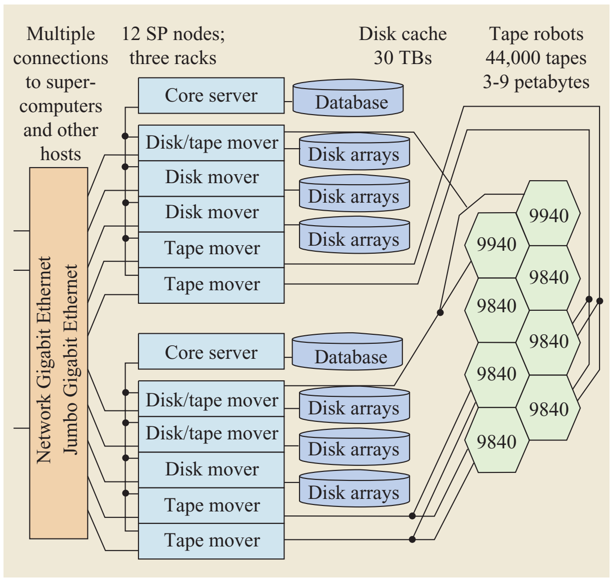

Figure 3

Typical of HPSS configurations for serving high-performance scientific centers, the NERSC HPSS system consists of a complex mesh of hardware and software, tied together with the HPSS software.

HPSS functions

In order to make HPSS work, a number of functions are implemented by servers:

- The *mover* manages the transfer of bits from one device to another. Devices may be tape drives, disk drives, network interfaces, or any other media. HPSS can support third-party transfers, in which the mover just manages the transfer and the data flows directly from source to destination.
- Devices and data paths are prepared for data movement by a set of *core servers*. One core server is the *name server*, which maps a human-readable file name to a system-generated bitfile ID. The name server also manages relationships among files, which may be grouped and joined together into directory hierarchies like a standard file system.
- Once the name of a file is mapped to a file ID, it must be mapped to the class of physical media on which it resides. The *storage server* does this. The storage server finds the actual physical device (say, a tape in a tape robot) and gets it mounted so that the transfer can begin.
- The *physical volume library* (PVL) manages all of the physical media in a system. It works with the *physical volume repository* (PVR) and other software and hardware to locate tapes and cause them to be mounted in the appropriate drives so that the mover can access them. The PVRs issue the tape library commands that manage the actual media in the system.
- The *migration manager* is responsible for migrating data from one medium to another. This includes moving data from disk to tape to free up disk space, and migrating data from one tape to another (often more dense) tape. Tape migration is done to consolidate tapes as files are deleted and to move data from old to new tape media, thus allowing automatic conversion.
- The *storage system manager* allows operators and administrators to manage the storage system. It provides a GUI (graphical user interface) which displays the status of servers, tasks, and resources. It passes commands to the *system manager* to perform operations on HPSS components, including allocation and configuration of resources, and initialization and termination of tasks.
- A *location server* provides a mechanism for clients to locate servers and gather information about HPSS systems.

HPSS provides multiple ways to interface with the system. The most basic are the File Transfer Protocol (FTP) and a high-performance parallel version (PFTP) created for HPSS. Another interface, HSI (Hierarchical Storage Interface), was developed to support more user-

[1] StorageTek Corporation, Louisville, CO.

friendly commands and graphic user interface interactions. HPSS also provides the ability to export HPSS data as network file systems (NFSs), distributed file systems (DFSs), and extended file systems (XFSs). HPSS supports the Data Migration Application Programming Interface (DMAPI) and is currently implementing standard grid interfaces.

Security was a major design goal for HPSS. Its components communicate in an authenticated and, when necessary, encrypted manner. It has auditing abilities beyond standard UNIX** and can enforce features such as access control lists.

The performance of HPSS has been demonstrated to be highly scalable, and the system is highly reliable, even while it handles 20 million or more files and petabytes of data. Transfer rates to and from HPSS run up to 80% of the underlying media rate, including Gigabit Ethernet.

Technologies needed for unified file systems

An emerging issue for many sites involved with deep computing is the waste and inefficiency surrounding the use of on-line disk storage. Currently, each high-performance system must have a large amount of local disk space to meet the needs of applications with large data sets. This file space is limited, and often applications and projects must live within quotas. When users access different machines, they typically make complete copies of the application and data needed. Thus, having separate storage on each machine is inefficient both in storage and in productivity. The main reason why it persists as the norm is that only local storage provides the I/O rates needed by deep applications. While file systems such as NFS and other distributed file systems are convenient, they lack the high performance and scalability required.

Fortunately, the confluence of several technologies is making it possible to address this problem more robustly. At the base technology level, new disk storage devices and Fibre Channel fabric switches make it possible to attach a single device to multiple systems. Network-attached storage (NAS) and storage-area networks (SANs) provide some fundamental building blocks, although not many operate in a truly cross-vendor manner yet. Finally, shared or cluster file systems provide a system-level interface to the underlying technology. Combinations of the new technologies are beginning to approach the performance rates of locally attached parallel file systems.

File system technologies

Without reliable, scalable, high-performance shared file systems, it will be impossible to deploy the center-wide file systems needed at deep computing sites to support activities such as interactive steering and timely visualization. There are currently two major approaches for sharing storage between systems: network-attached storage (NAS) and storage area networks (SANs).

Network-attached storage is a general term for storage that is accessible to client systems over general-purpose IP networks from the local storage of network-attached servers. Since data transfers between storage servers and clients are performed over a network, such file systems are limited by network bandwidth, network protocol overhead, the number of copy operations associated with network transfers, and the scalability of each server. The file system performance is often constrained by the bandwidth of the underlying IP network.

Storage area networks provide a high-performance network fabric oriented toward block storage transfer protocols and allow direct physical data transfers between hosts and storage devices, as though the storage devices were local to each host. Currently, SANs are implemented using Fibre Channel (FC) protocol-based high-performance networks employing a switched any-to-any fabric. Emerging alternative SAN protocols, such as iSCSI and SRP (SCSI RDMA Protocol), are enabling the use of alternative fabric technologies, such as Gigabit Ethernet and Infiniband, as SAN fabrics. Regardless of the specific underlying fabric and protocol, SANs allow hosts connected to the fabric to directly access and share the same physical storage devices. This permits high-performance, high-bandwidth, and low-latency access to shared storage. A shared-disk file system will be able to take full advantage of the capabilities provided by the SAN technology.

Sharing file systems among multiple computational and storage systems is a very difficult problem because of the need to maintain file system coherency through the synchronization of file system metadata operations and coordination of data access and modification. Maintaining coherency becomes very challenging when multiple independent systems are accessing the same file system and physical devices.

Shared file systems that directly access data on shared physical devices through a SAN are commonly categorized as being either symmetric or asymmetric. Asymmetric shared file systems allow systems to share and directly access data, but not metadata. In such shared file systems, the metadata is maintained by a centralized server that provides synchronization services for all clients accessing the file system. Symmetric shared file systems share and directly access both data and metadata. Coherency of data and metadata is maintained through global locks which are maintained either by lock servers or through distributed lock management performed directly between participating systems.

Shared file systems are not common, because implementing them is inherently difficult. Implementations of asymmetric unified file systems are the more common of the two types, because centralized metadata servers are easier to implement than distributed metadata management. However, symmetric shared file systems promise better scalability without the bottleneck and single-point-of-failure problems inherent in the asymmetric types.

Storage technologies

A new category of storage systems called *utility storage* is being introduced by new storage vendors such as 3PARdata** [21], Panasas ActiveScale Storage Cluster** [22], and YottaYotta** [23], who promise to deliver higher levels of scalability, connectivity, and performance. These storage systems use parallel computing and clustering technology to provide increased connectivity and performance scalability, as well as redundancy for higher reliability.

The storage provided by utility storage can scale from a few dozen to a few hundred terabytes, or even in the petabyte range, with a transfer rate as fast as a few thousand MB/s and the ability to sustain beyond 100,000 I/O operations per second (IOPS).

SAN fabric technologies

Today, most shared file systems either are very slow or use a proprietary interconnect from a single vendor. In the latter case, the file system is limited to the products of that vendor, or more often to a subset of the vendor's products. For a deep computing facility, that is not sufficient. Recently, several viable interconnect fabrics have started to emerge that may allow the introduction of high-performance, heterogeneous storage. While this is a positive sign, it also means that in most cases a site will be dealing with multiple interconnect fabrics that are bridged together. Such environments will require that the fabric bridges operate efficiently and without introducing large latencies in bridged communications.

Fibre Channel technology has recently undergone an upgrade from 1-Gb/s to 2-Gb/s bandwidth, with 4-Gb/s and 10-Gb/s bandwidths on the near-term horizon. These increases will allow substantially improved storage performance. Fibre Channel is also showing substantially improved interoperability between equipment from different vendors in multiple-vendor SANs.

Using Ethernet as a SAN fabric is now becoming possible because of the iSCSI standard [24]. The iSCSI protocol is a block storage transport protocol. The protocol allows the standard SCSI packets to be enveloped in Ethernet packets and transported over standard IP infrastructure, thus allowing SANs to be deployed on IP networks. This is very attractive, since it allows SAN connectivity at a lower cost than can be achieved with Fibre Channel, although also with lower performance. The iSCSI protocol will allow large numbers of inexpensive systems to be connected to the SAN and use the shared file system through commodity components.

The emerging Infiniband (IB) interconnect [25] shows promise for use in a SAN as a transport for storage traffic. Infiniband offers performance (both bandwidth and latency) beyond that of either Ethernet or Fibre Channel, with even higher bandwidths planned. The current 4× Infiniband supports 10-Gb/s bandwidth, while 12× Infiniband with 30-Gb/s bandwidth is poised for release. However, beyond demonstrating the ability to meet the fundamental expectations, advanced storage transfer protocols (e.g., SRP) and methodologies for Infiniband technology have to be developed and proven, as do fabric bridges between Infiniband, Fibre Channel, and Ethernet SANs.

Unified file system architectures

Several promising new unified file system architectures have been designed for high-performance cluster environments, typically with some kind of high-speed interconnect for the messaging traffic. Many of the new architectures perform storage transfers between client nodes and storage nodes over the high-speed interconnect.

Currently there are several major projects at supercomputing centers that are addressing the issues of unified file systems. One is the Global Unified Parallel File System (GUPFS) project at NERSC [26] (**Figure 4**) and another is the ASCI Pathforward Scalable Global Secure File System (SGSFS) project [27] at Lawrence Livermore National Laboratory.

Storage virtualization has been widely used as a way to provide higher capacity or better performance. Virtualization can also be implemented at the file system level. File system virtualization allows multiple file systems to be aggregated into one single large virtual file system to deliver higher performance than a single NFS or NAS server could provide. A few examples of federated file systems include the IBM General Parallel File System [28], Lustre [29], the Hewlett-Packard DiFFS [30], the Maximum Throughput InfinARRAY [31], and the Ibrix SAN-based file system [32].

Figure 5 shows a potential architecture for the federated file systems using the high-speed interconnect for the storage traffic.

R&D challenges for unified file systems

The technologies needed for unified file systems at deep computing sites face a number of research and development challenges regarding scalability, performance, and interoperability.

One of the major storage issues is the degree of efficiency with which the storage can handle I/O requests from multiple clients to individual shared devices (each a logical unit number, or LUN). The scalability of single-

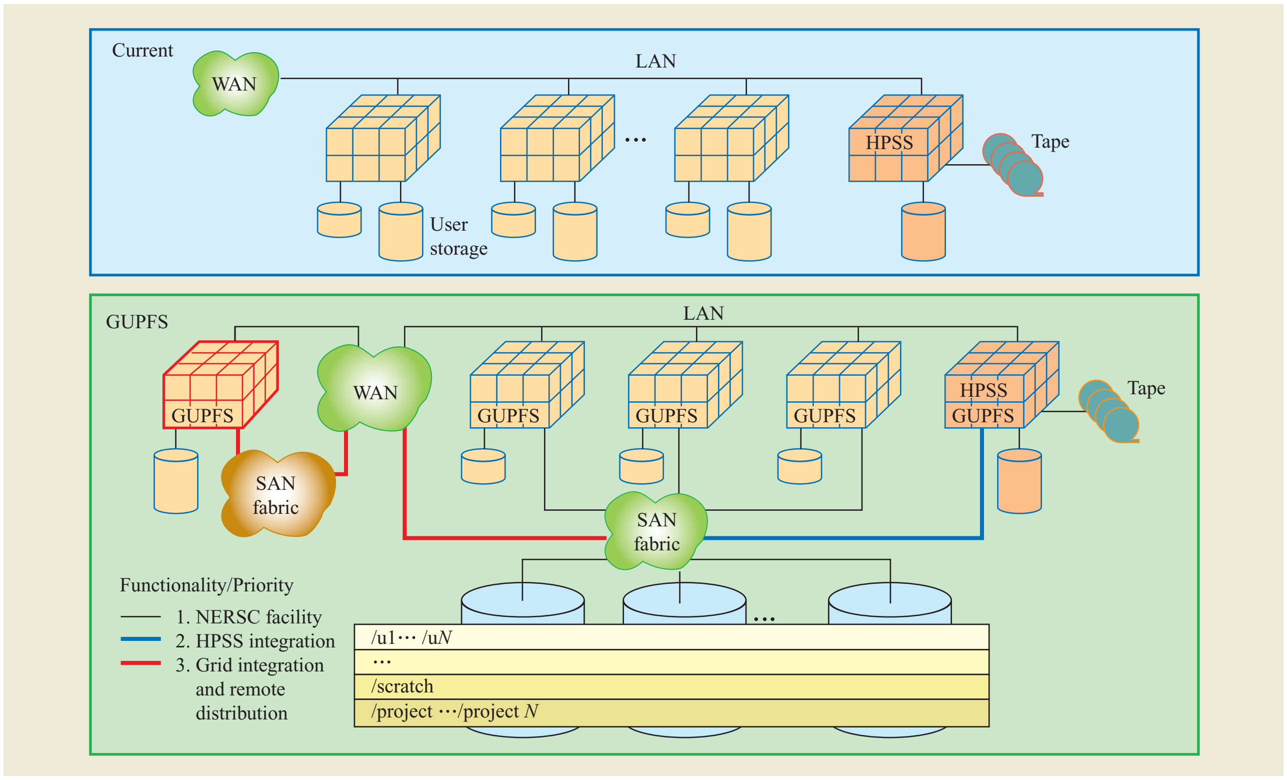

Figure 4

The Global Unified Parallel File System (GUPFS) being developed at NERSC. Currently, each major system in a facility must have large amounts of local disk space in order to achieve the performance levels needed for deep science. In the next four to five years, emerging technology should enable high-performance parallel file systems that will allow a large amount of persistent storage to be shared while maintaining high-performance I/O transfer rates.

device access has been a common problem with many storage systems, both in the number of initiators (clients) allowed on a single device and in the performance of shared access. On most storage systems, the maximum number of client initiators allowed on a single LUN has been less than 256; deep computing sites will require storage devices to support thousands of simultaneous accesses. The envisioned GUPFS implementation, for example, is a shared-disk file system with tens of thousands of client systems. That system must provide high performance of individual components and interoperability of components in a highly heterogeneous, multi-vendor, multiple-fabric environment. The ability of the file systems to operate in such a mixed environment is very important to the ultimate success of a useful global unified file system for deep computing.

In addition to simply supporting very large numbers of simultaneous accesses, storage devices must be able to efficiently recognize and manage access patterns. Current storage devices are unable to do so, and simultaneous sequential accesses by even a few tens of clients appear as random access patterns, resulting in inefficient cache management. New scalable cache management strategies will be required.

To adequately support thousands of clients, storage devices will have to be able to deliver tens to hundreds to thousands of GB/s of sustained bandwidth, employing multiple SAN interfaces. To operate effectively in the multiple-SAN/interconnect-fabric environment expected at deep computing sites, the storage devices will have to support multiple types of fabric interfaces simultaneously (e.g., Fibre Channel, Infiniband, and Ethernet interfaces) through building block modules.

In order to support tens of GB/s sustained I/O rates for very large numbers of clients, SAN fabric switches with very large numbers of ports must be fielded in order to minimize the ports needed for the fabric mesh. This in turn requires fabric switches with much higher (tens of GB/s) interswitch link capabilities to facilitate link aggregation to the high-performance storage devices.

Although storage and fabric vendors are beginning to recognize and address these issues, it will take significant research and development to support the needed functionalities.

Managing large datasets in shared storage resources

In the previous sections we introduced archiving technologies and methods of sharing on-line disk space between computing servers. These systems provide the basic components we need to begin to address the application phases mentioned in the introductory sections of this paper, but they are not sufficient. What is still needed is the software to manage the movement of the data between the storage media and to the computing servers.

Consider an analysis application, such as the analysis of transaction data of a grocery store for the purpose of decision-making. If the amount of data is small, the analysis program loads all of the data into memory and performs the analysis in memory. If the amount of data does not fit into memory, it is brought piecewise from a disk cache. If the level of computation is high enough that it is not slowed down by access from disk, there is no loss in efficiency. Various optimization techniques can be used, such as methods to reorganize the data according to the access patterns, or using indexes to minimize the access time or disk I/O. Another technique is to speed up access from disks by providing file systems that use parallel striping methods, such as General Parallel File System (GPFS) or Parallel Virtual File System (PVFS) [33]. Such techniques have been the subject of many studies in the domain of data management research. Our purpose in this section is not to cover the above techniques, but rather to discuss the problems that arise if the amount of data is so large that it does not fit on disk, but must be stored in archival storage. For example, the amount of scientific data generated by simulations or collected from large-scale experiments has reached levels that cannot be stored in the researcher's workstation or even in a local computer center. Access to data is becoming the primary bottleneck in such data-intensive applications. This is a new class of problems, because access from archival tertiary storage is relatively slow, and other optimization methods must be applied. What can be done to manage data stored in tertiary storage systems more efficiently?

Suppose one has 1 TB in tertiary storage that one wishes to analyze, using a supercomputer that can analyze the data in parallel. Getting the data to the supercomputer requires downloading the data from tape to disk. Assuming that there is available a parallel disk system with 100 disks and that each disk can read data at 10 MB/s, data can be streamed to the supercomputer at a rate of 1 GB/s. Thus, the data can be read by the

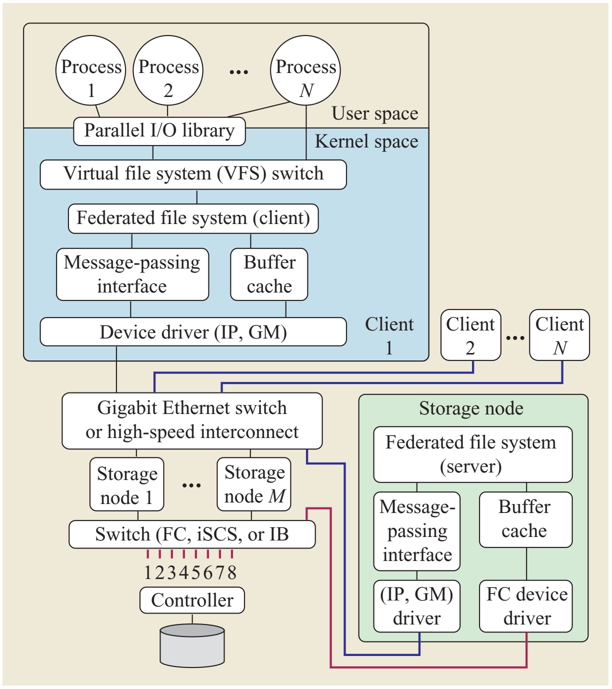

Figure 5

A potential architecture for integrating a large global file system with federated components.

supercomputer in 1,000 seconds, which is reasonable for analyzing this quantity of data. However, in order to achieve this rate from tertiary storage, 100 tape drives, each reading at 10 MB/s, would have to be dedicated to this task for 1,000 seconds. This is not a very practical solution, especially considering datasets in the petabyte range. Also, with the tapes available today, each tape may hold 50–200 GB, and multiple tapes may have to be loaded, adding latency as a result of mounting and dismounting tapes. In this scenario the robotic tape system is the bottleneck. What techniques can be applied? We now discuss several.

1. Overlap data processing with staging data to disk. In many applications, it is possible to start processing subsets of the data. For example, one can start analyzing transaction data piecewise, such as transactions for a day at a time. Assuming that data is organized in large files, one can devise software that will stream data to the analysis program and concurrently continue to stage files from tape. Streaming data to the analysis programs is also an effective way of sharing disk cache with many users concurrently. Even if each user needs most of the space of the disk cache, it is not necessary to move all of the data to disk for exclusive use and schedule users to operate sequentially. Instead, each user gets part of the disk cache, and the data is streamed to all of the applications concurrently. To support this streaming model, it is important that a user release files already processed as soon as possible, so that quota space can be reused. This requires systems that support the concept of *pinning and releasing* files. Pinning a file guarantees that the file will stay in the shared disk cache for a period of time. If the file is released within this period of time, the space can be reclaimed and reused. Otherwise, at the end of that time, the file will be released by the system.

2. Observe incremental results on the streaming data with increasing accuracy.
The streaming model discussed above also provides opportunities to end analysis or simulation tasks early, thus saving a tremendous amount of computing and storage resources. Often, the analysis produces statistical results, such as histograms. These can be obtained incrementally as the data streams through the analysis programs. This permits the client to observe the data statistics as they are generated with increasing accuracy. When a sufficient level of accuracy is achieved, the client can stop the analysis. Given that proper sampling of the data is performed (e.g., applying random sampling of files from tapes), it is sufficient to read only 5–10% of the data in most applications to achieve the desired accuracy. This idea has been exploited in the database community for online query processing, but it is even more important for the analysis of massive amounts of data streaming out of robotic tape systems. A similar idea can be applied for the processing of large simulations. Some scientific simulations such as astrophysics or climate modeling may take many days of computation and produce several terabytes of data each. It is important to identify at an early stage whether a simulation is progressing correctly. Again, statistical analysis and visualization techniques can be used to observe the progress of the simulation.

3. Share "hot" data that is brought to disk.
Another opportunity for using disk caches effectively to reduce reading data from tape is to share files among users. This is the case when a community of users share disk caches to analyze the same very large dataset. Often, a user will access the same files repeatedly, or different users interested in the same aspect of the data will request files that another user previously staged to disk from tape. Such files are referred to as *hot* files. In order to share hot files, one must have software to track file usage and "smart" predictive algorithms to determine which files to evict when space is needed.

4. Minimize tape mounts.
One of the most costly aspects of dealing with robotic tape systems is the time it takes to mount a tape. This is typically in the range of 20–40 seconds with current technology. Another latency problem is searching for a file on a tape, which can also take 20–40 seconds depending on the tape size and the search speed. Avoiding these delays results in a large performance gain. To achieve this benefit, a scheduling system can be used that has the flexibility to stage files in tape-optimized order over many clients. Such a scheduling system must also ensure that no one job is postponed unfairly. The scheduling system has to have information on file location on tapes (i.e., which tapes, and location on the tape). The HPSS system mentioned above is designed with some minimal optimizations for submitted file requests. However, it does not have a global view of the files needed for an entire job. If entire jobs are provided to a front-end system for many users, that system may be able to perform global optimization ahead of time by grouping file access requests in a tape-optimized order.

5. Use indexing techniques for finding desired subsets of the data.
Another important technique for reducing the amount of data that is read from tape systems is having information on the content of the files, so that only the relevant files are accessed for a given analysis. This problem is so severe in some application domains, such as high-energy physics, that subsets of the data are pre-extracted by performing full scans of the very large datasets and selecting the subsets of interest. Many such subsets are generated, which only adds to the replication of the data and the use of more storage. Indexing techniques can eliminate the need to scan the data many times to generate desired subsets. For example, in high-energy physics, the objects stored in files are chunks of data, 1–10 MB each, called *events*, representing high-energy collisions of particles. It is possible to obtain a set of properties for each event, such as the energy of the event and the types of particles produced by that event. There are billions of such events stored in thousands of files. An index of the event properties can identify the files of the desired events for the analysis, and then only those files will be accessed.

6. Automate garbage collection.
As mentioned above, one of the biggest problems with shared storage is waste caused by moving files into the storage spaces but never removing them. It is impossible to track which file is really valuable, and ineffective to force users to "clean up." Another approach is to have shared spaces managed by letting space be used dynamically. The main concept is that a file is brought to the shared disk cache on a temporary basis with some

minimal *lifetime* guarantee. As mentioned above, the concept of pinning can be used, with a lifetime associated with the pin. The lifetime should be long enough for a client to perform the intended task, but it is then expected that the client will release the file in order to obtain more space. If the client is irresponsible and does not act within the lifetime, the file can be removed by the storage system manager when the lifetime expires. This methodology works only if each valuable file is stored in an archive and is brought to the shared disk cache for processing or analysis only. The ability to pin and release a file within a lifetime is the key to supporting automatic garbage collection.

Storage Resource Managers (SRMs): A software layer to support the dynamic management of storage

The concepts described above have been developed at the Lawrence Berkeley National Laboratory and other Department of Energy laboratories and are implemented in SRMs [34, 35], which are middleware software modules whose function is to provide dynamic space allocation and file management on shared storage resources. The term *storage resource* refers to any storage system that can be shared by multiple clients. We use the term *client* here to refer to a user or a software program that runs on behalf of a user.

SRMs dynamically manage what should reside on the storage resource at any one time. They complement Compute Resource Managers and Network Resource Managers in providing storage reservation and dynamic storage availability information for planning the execution of high-performance jobs. SRMs can be thought of as managing two types of resources: spaces and files. When managing space, SRMs negotiate space allocation with the requesting client, and/or assign default space quotas. When managing files, SRMs allocate space for files, invoke file transfer services to move files into the space, pin files for a certain lifetime, release files upon the client's request, and use file replacement policies to optimize the use of the shared space. SRMs can be designed to provide effective file sharing for read-only files by monitoring the activity of shared files and making dynamic decisions on which files to keep until space is needed. In addition, SRMs perform automatic garbage collection on unused files by removing files whose lifetime has expired when space is needed.

A *Disk Resource Manager* (DRM) manages a shared disk cache. This disk cache can be a single disk, a collection of disks, or a RAID system. The disk cache is made available to the client through some operating system that provides a file system view of the disk cache, with the usual capability to create directories and open, read, write, and close files. The function of a DRM is to manage this cache using a policy that can be set by the administrator of the disk cache. The policy may restrict the number of simultaneous requests by users, or may give preferential access to clients on the basis of their assigned priority.

A *Tape Resource Manager* (TRM) is a middleware layer in front of a robotic tape system. Such tape systems are accessible to a client through fairly sophisticated Mass Storage Systems (MSSs) such as HPSS or Unitree. TRM systems usually have some disk cache that is used to stage files temporarily before transferring them to clients. MSSs typically provide a client with a file system view and a directory structure, but do not allow dynamic opening, reading, writing, and closing of files. Instead, they provide some way to transfer files to the client space, using transfer protocols such as FTP and variants of FTP. The function of the TRM is to accept requests for file transfers from clients, queue such requests in case the MSS is busy or temporarily down, and apply a policy regarding the use of the MSS resource. As in the case of a DRM, the policy may restrict the number of simultaneous requests by users, or may give preferential access to clients on the basis of their assigned priority.

A *Hierarchical Resource Manager* (HRM) is a TRM that has a staging disk cache for its own use. It can use the disk cache for pre-staging files for clients and for sharing files between clients. This functionality can be very useful, since a request from a client may be for multiple files. Even if the client can process only one file at a time, the HRM can use its cache to pre-stage the next files. Furthermore, the transfer of large files on a shared network may be sufficiently slow that while a file is being transferred, another can be staged from tape. Because robotic tape systems are mechanical, they have a latency of mounting a tape and searching for the location of a file. Planning the pre-staging in a tape-optimized order can help eliminate this latency. Another advantage of using a staging disk in an HRM is that it can be used for file sharing. Given that multiple clients can make requests for multiple files to an HRM, the HRM can choose to leave a file in cache longer so that it can be shared with other clients on the basis of use history or anticipated requests.

SRMs can support three possible types of files in a shared storage resource, depending on their expected usage and function. These file types are necessary for the smooth functioning of an SRM and automatic reclaiming of unused space.

1. *Permanent files:* This type of file is stored in a location that is intended to be permanent, usually on a tape archive. Typically, it is the location where files are stored immediately after they are created. A permanent file is a physical file that can be created and removed only by the owner of the file. This is consistent with the usual concept of files owned by users. A permanent file

can be thought of as having a pin with an indefinite lifetime.
2. *Volatile files:* Volatile files are created dynamically to satisfy user requests to process the files. The files are not owned by the user, but rather by the SRM. A volatile file should stay in the cache if the demand for it is high; otherwise, it can be removed when space is needed. A volatile file can be thought of as a temporary file, except that it has a lifetime associated with it. The lifetime guarantees that the file will be in the cache as long as necessary, but after the lifetime expires it can be removed from the cache by the SRM. The client can also terminate the lifetime of a pinned file. This early release of a file permits the SRM to reuse the space as soon as it is needed.
3. *Durable files:* A durable file is like a volatile file in that it has a lifetime associated with it, but is also like a permanent file, because when the lifetime expires, the file is not automatically eligible for removal. Instead, the SRM advises the client or an administrator that the lifetime has expired, and they can then take the necessary corrective action. The durable file type was introduced to provide temporary space for files generated by large simulations, which should be transferred as soon as possible to an archive. This is a reasonable approach for sharing temporary space yet protecting important files. Like volatile files, durable files can be released by the user.

For the file types described above, the semantics of file pinning and releasing and the lifetime of a file provide the basis for dynamically managing the content and space allocation of a storage resource. These concepts are also useful in supporting the data streaming model described in the previous section. This is achieved by staging files into a disk space and letting the applications access the staged files concurrently with the staging of additional files. As soon as files are released by the application, the space can be used to stage additional files.

Several versions of prototype SRMs have been developed and used in test cases as part of the Particle Physics Data Grid (PPDG) [36] and Earth System Grid (ESG) [9] projects. A prototype of an HRM was also developed at the Fermi National Accelerator Laboratory to interface with their Enstore mass storage system [37]. In addition, efforts are now underway to coordinate the SRM functionality across several projects, including the development of an HRM at the Thomas Jefferson National Accelerator Facility to interface with their JASMine mass storage system [38] and the European Data Grid to interface with their CASTOR mass storage system [39]. The emerging concepts and interfaces seem to nicely complement other Grid middleware services being developed by various Grid projects, such as providing efficient and reliable file transfer, replica catalogs, and allocation of computing resources.

Networking for HPC systems

The techniques described up to this point have dealt mostly with the data movement, storage, and archiving requirements of deep computing within a facility. However, in many of the scientific cases listed in the introduction, the data is not at the same location as the computing servers, or the results have to be moved to a different location. In these situations, the wide-area network connecting the sites becomes involved, and efficient transfer of the data across this network is required.

When high-performance computing applications utilize resources spread across a wide-area network, they usually require particular bandwidth, latency, reliability, security, and privacy guarantees. The end-to-end network performance of an application is a product of the application behavior, the machine capabilities, the network path, the network protocol, and the competing traffic. If the expected performance is not achieved, it is often difficult to ascertain the limiting factor without significant end-to-end monitoring and diagnostic capabilities.

Data is broken into packets by the network protocol before it is sent on the network. Most traffic on the network uses the Transmission Control Protocol (TCP). TCP provides reliable point-to-point transfer of data by retransmitting lost packets. It also incorporates flow and congestion control algorithms that allow it to adjust its sending rate to the network capabilities and cross traffic so that it can share the bandwidth fairly with other traffic. Over the years, many improvements to TCP have been implemented. Some of the well-known modifications include fast retransmit, fast recovery, selective acknowledgments (SACK), slow start, and delayed acknowledgments. These modifications were introduced to improve the behavior and fairness of TCP. Changes to the algorithms in the router have included explicit congestion notification (ECN) and random early detection (RED) among others. These changes in the routers are designed to improve fair allocation of resources to the various data streams flowing through the router and to provide feedback to the data streams regarding congestion.

The component that is most limited in its ability to transmit data through the network is usually referred to as the bottleneck. The bottleneck determines the maximum bandwidth that can be achieved on the path. The amount of bandwidth an application effectively utilizes in the network is usually less than the bottleneck bandwidth and is determined by many factors. In this section we examine several of these factors, including end host issues, routers and gateways, and transport protocols. We also discuss the importance of security, monitoring, and prediction.

End host issues

Although most supercomputer and cluster architectures use special high-speed interconnects such as Myrinet [40], Quadrix [41], or the IBM SP switch within the machine or cluster, one or more front-end hosts or nodes provide the only external access to the machine. These hosts are often the limiting factor. As an example of the issues for a node in such a system, we examine a typical PC cluster, for which the host hardware performance factors are the memory bandwidth and the memory and I/O interconnect bandwidth.

It is tempting to assume that the host throughput to and from the network is simply the slower of the I/O bus speed or the memory bus speed, but the reality is more complex. The real throughput that can be achieved in transferring data from the user memory on the machine to the network interface card is defined by adding 1) the time to copy data from user memory to the kernel memory across the memory bus and 2) the time to copy from the kernel memory to the network interface card. Typically it takes two memory bus cycles to copy data from the user memory to the kernel memory, and one I/O bus cycle to copy from the kernel memory to the network interface card. The number of memory bus cycles required to transfer a word from the kernel memory to the I/O bus is determined by dividing the memory bus speed by the I/O bus speed. Thus, the typical throughput between the user memory and the network interface of a typical PC is defined as

$$throughput = \frac{memory\ bandwidth}{2 + \dfrac{memory\ bus\ clock}{I/O\ bus\ clock}}. \qquad (1)$$

The typical hardware platform is an x86-based processor equipped with 32-bit/33-MHz and/or 64-bit/66–133-MHz peripheral component interconnect (PCI) buses as the I/O subsystem. The 32-bit/33-MHz PCI I/O subsystem provides a maximum bandwidth of 132 MB/s, which is equivalent to 1 Gb/s. The Hewlett-Packard 64-bit/133-MHz PCI-X subsystem provides 8.5-Gb/s I/O bandwidth. The memory bandwidth provided by these x86 motherboards, equipped with PCI subsystem and using 400-MHz double data rate (DDR) memory modules, varies from 650 to about 2,500 MB/s. Given the memory bus speed and the I/O bus speed, we can calculate the expected throughput to the network. For example, a motherboard with the latest VIA[2] PCI chipset (VT600, 32-bit/33-MHz PCI, DDR400 memory and AMD[3] 2700+ CPU) provides 850-MB/s memory bandwidth. The maximum expected throughput to the network is 485 Mb/s (60.7 MB/s = 850 MB/s ÷ 14). Clearly, a system equipped with a 32-bit/33-MHz PCI bus will not achieve gigabit throughput to the network if the data is coming from and going to the user space. Consider a high-end (server) x86 motherboard with 64-bit/133-MHz PCI-X bus with 200- to about 266-MHz DDR memory modules. In this system, the maximum memory bandwidth is around 1,100–2,500 MB/s. Thus, the expected maximum network throughput it can achieve is 2,500 MB/s × 1/4 = 625 MB/s, or 5,000 Mb/s, by Equation (1). If we also include DMA and memory controller overheads, the real throughput to the network will be slightly lower [42]. The latest Network Interface Card (NIC) speed available for these PCI-X systems is 10 Gb/s. Therefore, the current x86 systems with the fastest available memory and PCI bus are not able to provide enough I/O power to drive a 10-Gb NIC at full speed. Today's best systems are at least a factor of 2 below the required speed.

Although mechanisms such as OS bypass can reduce the number of copies required between sections of memory, the copy from memory to the I/O bus cannot be avoided. The computer industry is aware that the I/O bus is the bottleneck in current and future hardware systems, and fast I/O standards are proposed for a next-generation I/O bus. In the short term, PCI-X 2.0 and PCI-X 3.0 are both designed to increase the PCI bus clock rate and provide more I/O bandwidth. In the long term, the Intel** PCI-Express (formerly called 3GI/O, or third-generation I/O) is expected to provide even higher I/O bandwidth (64-Gb/s one-way total bandwidth with 32 lanes). This should help to improve the overall system I/O performance. However, this new-generation I/O subsystem will still be the bottleneck for network I/O, because this I/O standard will require time to mature before its potential is realized in practice, and network bandwidth will continue to increase during that time. The network interface card speed has on average doubled every year in the past several years. If this trend continues, the network interface cards will reach 1 Tb/s before PCI-X 3.0 and PCI-Express can be widely deployed.

Routers and gateways

There are three major classes of switching technology in use in the world today: internal computational switches (the IBM Colony and Federation switches, Myrinet, Quadrix, the Hewlett-Packard Superdome, etc.), local-area networks (Gigabit Ethernet and, to a lesser degree, Fibre Channel), and wide-area networks [most often Packet over SONET (Synchronous Optical NETwork), ATM]. For deep computing, these classes of switches must be effectively integrated, and end-to-end requirements should have a minimal performance impact.

Internal computational switches and shared disk are designed for very low latency and high bandwidth. Computational switches are based on very large frame

[2] StorageTek Corporation, Louisville, CO.
[3] VIA Technologies, Inc., Hsin-Tien, Taipei, Taiwan.

(packet) sizes, typically 64 KB, and suffer performance degradation at lower sizes. Local-area networks based on Gigabit Ethernet can run up to 9-KB jumbo frames reliably for on-site mass storage and backups. The 9-KB frame size increases performance compared with standard Ethernet by reducing overhead and CPU load; but compared with 64-KB frames, it has lower performance and higher latency than computational switches. Wide-area networks are typically restricted to 1,500-byte frames because of the need to support millions of simultaneous connections, even though the main protocols, SONET and ATM, will pass frames up to 9 KB.

This mismatch in maximum transfer unit (MTU) sizes, and, more significantly, the performance degradation that is associated with running small MTUs, make it difficult to effectively integrate a massively parallel computer system into any networked environment. The two major approaches used to date have been to add an external interface to each node or to turn one or more computational nodes into gateway routers and have external traffic flow across the internal switch to these gateways.

Both of these approaches have major weaknesses. Deep computing will require thousands to tens of thousands of nodes [43]. Adding an external gigabit interface to every computing node for external connectivity to data storage systems and other computational resources is not practical, nor will it allow single-stream performance to increase above current levels, even though this has the best chance for meeting large aggregate bandwidth requirements. Using a computing node that has a connection to the internal switch fabric as well as multiple bonded Gigabit Ethernet interfaces or 10-Gb/s Ethernet as a router is logistically easier and potentially has better single-stream performance, but aggregate performance will suffer, especially coupled with traffic comprising MTUs of different sizes on the computational switch. Also, a computing node that must run packets through its IP stack to divide the traffic into packets, generate packet headers, and perform flow and congestion control will never be able to keep up with the fastest switches and routers that just store and forward with all decisions in application-specific integrated circuits (ASICs).

One possible solution would be a Layer-7 router with a computational switch interface that could do the bridging, MTU repackaging, and load balancing. However, the chance of someone building this is very small. A more workable solution would be to modify a computing node to act like a high-performance router. Most of the hardware components for modifying a computing node already exist or would be simple to create. A computational switch, such as a Federation SMA3 adapter, and a 10-Gigabit Ethernet card, both with sufficient field-programmable gate array (FPGA) space to offload the IP protocol, would limit the duties of computing nodes to setting up remote direct memory access (RDMA) between the two interfaces. Frame fragmentation and coalescing are often done through TCP proxies, but should be programmable in a reasonable amount of FPGA space, thus reducing the adverse impact of widely disparate MTUs.

Transport protocols

The TCP congestion control algorithm aims to fully utilize the network path yet be fair to other traffic. There are two types of losses in the network: random and congestion. If network traffic arrives at a router or network interface and there is not enough capacity left to buffer the packet, the packet is discarded (congestion loss). In TCP, the receiver acknowledges data as it is received. When a sender receives three duplicate acknowledgments, it assumes that data has been lost, cuts its sending rate in half, and retransmits the data just above the duplicate acknowledgment. It then increases the sending rate by one each round-trip time until the next loss, and the cycle repeats. This algorithm is called additive increase, multiplicative decrease (AIMD). TCP also includes a slow-start algorithm, which is used at the beginning of a TCP connection to double the sending rate each round-trip time until the first loss is detected. TCP uses a congestion window to track the sending rate that is allowed. If the buffer size for sockets were not selected appropriately for the network connection, the congestion window might be artificially limited or the receiver might be overrun. Ideally the buffer size is continually tuned to the optimal size. The Net100 project has created a work-around daemon to perform this dynamic tuning [44].

Although TCP is relatively robust, in high-speed wide-area networks there are several issues. One primary problem is that the product of the bandwidth and the delay of the path is very large. To fully utilize the available bandwidth in such a path requires that the amount of data in flight at any point in time be equal to the bandwidth delay product. For example, in order to fill a one-gigabit path which has a 100-millisecond round-trip time and a packet size (MTU) of 1,500 bytes, a TCP stream would have to have of the order of 8,000 packets in flight continuously—the equivalent of 12 megabytes. The TCP protocol is designed with the premise that the random loss rate in network components is insignificant compared to the loss rate due to congestion. Thus, the TCP sending rate is a function of the packet loss rate (assumed congestion) in the network. The packet sending rate drops dramatically as a response to a congestion event (packet loss); then the sending rate increases slowly until the next congestion event is encountered. In the high-bandwidth delay product example above, an error rate of 2×10^{-8} (one packet every 555 round trips) or less must be maintained in order to allow TCP to fully utilize the available network capacity [45]. In high-speed networks, this required error rate is less than the random loss rate of the components without any congestion being present. Thus, TCP by its very design is prevented from utilizing the available bandwidth on high-bandwidth delay product paths.

Several approaches have been developed recently to address high-bandwidth delay product paths [46]. The solution most commonly used today is parallel streams. Parallel streams have the feature that a random loss affects one stream and that stream cuts its congestion window in half, causing the reduction in the combined congestion windows of the streams to be $1/(2 \times N)$, where N is the number of independent streams being used for the transfer (instead of the one half that a single stream would have experienced). If we look at fairness, parallel streams operate as N streams rather than one stream and thus get N times their share. This is primarily an issue when the link is congested. Another issue with parallel streams is that their use requires software to split the data across the streams and reassemble it at the destination. This introduces additional overhead within the hosts because of the context switches and assembly operations required.

One recently proposed variant of TCP is the FAST algorithm, which would use queuing delay rather than loss as a measure of congestion [47]. Another proposal, referred to as HighSpeed TCP, would modify the AIMD algorithm constants to make the back-off less than one half on loss and the linear increase faster when the congestion window is large [18].

Several mechanisms have been created to allow applications requiring high bandwidth to have priority access to the bandwidth. The most aggressive of these is to create a dedicated path for the traffic by reserving a dedicated link/circuit/channel. The best-known mechanisms for this include virtual circuits and RSVP (Research ReSerVation Protocol). Another mechanism is to mark the traffic as priority; then each router in the path expedites the traffic. The standard method for this is to use a bandwidth broker to arbitrate the access to the bandwidth on a pairwise basis. If the intervening routers agree to the priority path, the packets are marked as priority as they enter the network and each router in the path forward; the packets receive preference over all other traffic.

An alternative approach that has been proposed is the use of priority ratings on traffic or reserving virtual circuits. Although prioritizing traffic has received a lot of attention, it has generally been impractical. It is likely that traffic requiring guaranteed bandwidth will have to reserve virtual circuits through the network. This does not, however, remove the need to find a transport protocol that can effectively make use of the dedicated bandwidth.

Security

Deep computing also implies *deep security*. Protection of assets—computationally expensive or irreplaceable data as well as multi-million-dollar machines—is critical. In the business world and government, where the data is considered sensitive, the potential losses due to a security incident, with recovery time measured in weeks if not months, may far exceed the costs of reproducing the hardware and data.

Therefore, very common bottlenecks for distributed scientific applications are currently the security devices at the interface between a local network and a wider-area network. Firewalls and active intrusion detection systems (IDSs) that inspect packets and make decisions on whether or not to forward them will never be able to keep up with the high-end "store and forward" switches and routers. Conversely, relying only on passive devices for security may avoid performance impact but introduce more risk of delaying a response to an intrusion.

Fortunately, some workable solutions are currently available, and they lead to a path that may scale security to levels needed for deep computing. Scaling firewall performance under certain situations is as easy as adding more firewalls. While this approach does not solve the single-stream throughput requirements of deep computing, it does point to an interesting solution, tightly coupled security and routing policies.

Two routers and a diverse set of firewalls and IDSs acting as one system have the potential to scale both security and throughput. This scenario is shown in **Figure 6**. The simplest approaches could be having all Microsoft** Windows**-related protocols go through a dedicated Windows-specific firewall and UNIX traffic through a UNIX/Linux**-specific firewall based on static transport layer (Layer 4/TCP header information) routing policy. Performance-critical data streams could be routed to a third clear channel. More complex solutions contain stateful and application firewalls, FTP proxies, and IDSs that adjust local routing tables to shunt identified legitimate streams to clear channels, thus simultaneously improving throughput and reducing firewall loads.

A complementary technique of shunting suspected bad traffic through more aggressive scrutiny by extremely tight firewalls and active IDSs may also increase security. Looking at every FTP-data packet for every known HTTP (HyperText Transfer Protocol) payload is obviously not a good use of an IDS, but looking very closely at HTTP packets that show indications of malicious activity makes sense.

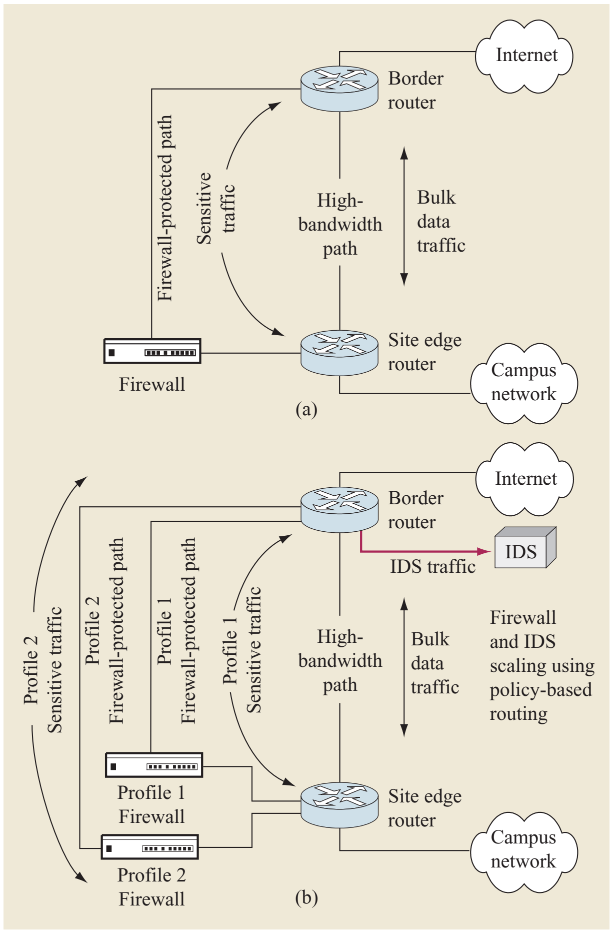

Figure 6

(a) Logical view of a typical network security implementation that allows high-bandwidth data transfers where there are two paths, one monitored for control and sensitive traffic and the other not monitored for the high-bandwidth bulk traffic. (b) Logical view of a more complex network security implementation that allows monitoring of all traffic, both the sensitive and the high-bandwidth data transfers.

Passive IDSs tend to rely on port mirroring or hardware taps, and analyze all traffic across a link. This has a 2× scaling issue for full-duplex links. Fortunately, not all packets contain the same level of information from a security point of view. For TCP, a synchronize (SYN) packet asking for a socket to be created conveys far more security information than an acknowledgment (ACK) packet for data transferred. Therefore, it is possible to have a passive IDS accomplish more real work with a much lower load by feeding it only the data that it wants to examine.

One implementation of this combined approach is the Bro system [48], developed by Vern Paxson. It was used at the SC2002 conference to successfully monitor the SCinet

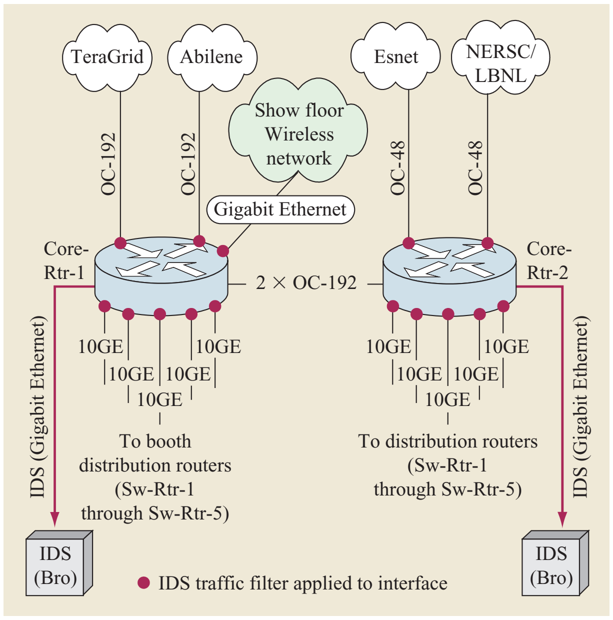
TeraGrid
Abilene
Esnet
NERSC/ LBNL
Show floor Wireless network
OC-192
OC-192
OC-48
OC-48
Gigabit Ethernet
Core-Rtr-1
Core-Rtr-2
2 × OC-192
10GE
To booth distribution routers (Sw-Rtr-1 through Sw-Rtr-5)
To distribution routers (Sw-Rtr-1 through Sw-Rtr-5)
IDS (Gigabit Ethernet)
IDS (Bro)
● IDS traffic filter applied to interface

Figure 7

Network diagram of the SC2002 SCinet, showing the network locations for the Bro IDS system as it monitored a total peak bandwidth of 93 Gb/s.

experimental network, which consisted of two OC-192 links, seven 10-Gigabit Ethernet links, and three Gigabit Ethernet links at the same time, with a single Gigabit Ethernet IDS and static Layer-4 filter-based port mirroring (**Figure 7**).

Not only can higher performance be achieved, but improved security can be gained by having IDSs dynamically update the filter rules in routers and/or on hosts to better watch suspect traffic. Having a complete copy of an incident as soon as it is detected is a possibility with dynamic filter-based port mirroring and is very useful to security analysts. The bandwidth requirements of deep computing make performing this task next to impossible any other way.

Monitoring and prediction

As illustrated by the cases discussed in the beginning of this paper, deep computing applications are large and consume significant storage, network, and computing resources before they complete. Also, many of these computations take a long time to complete. Typically, a project has only a limited allocation for use of a large-scale computing facility, and the project cannot afford to waste any of this allocation. Thus, it is important for the end user to be able to monitor the progress of the computing job so the user can take appropriate action if something goes wrong. In addition, if the user has access to more than one computing facility or resource, the user or scheduler needs accurate information to decide where to submit the job. Factors such as where the data and executable are currently located must be considered in this decision, and ideally the user needs predictions of future availability, since the job is not yet running.

Large-scale computing facilities generally provide information about their current loading. In order to make a decision about where to run a computing job, the user needs information about all of the components involved, including the network. Once the job is scheduled, a monitoring capability is needed to allow the user to recognize whether, for example, a job is blocked from being started because it is waiting for the disks to become available or because performance across the network is significantly worse than expected. Users can recognize these problems and take corrective action only if they have monitoring data that reports on all phases of the progress. This includes reporting across virtualization levels.

Traditionally, large computing facilities have needed to concentrate only on performance within their facilities. As many of the application examples show, there is now increasing dependence on the network as an integral part of the computing job. Diagnosing problems with a network path is difficult for the application, since it has control of only the two endpoints and none of the components in between. Problems in a distributed system might be caused by anything, including the application itself. A wide-area network path usually traverses several different administrative domains, and even the network administrators do not usually have access to all of the routers in the path.

One of the first problems of diagnosing what is wrong with the network is determining what the behavior should have been. Another difficulty is knowing what other traffic was present in the network. Network monitoring is still a research topic, but there are several monitoring tools that can help to indicate the location of a problem, including NetLogger [49], Self-Configuring Network Monitor (SCNM) [50], and others.

Concluding remarks

Increasingly, scientific advances require the fusion of large amounts of complex data with extraordinary amounts of storage, network, and computational power. The problems of deep science demand not only deep computing but also deep storage resources and deep networking capabilities. In addition to teraflop-range computing engines, facilities must provide large data repositories of the order of 10–100 petabytes, and networking to allow the movement of multi-terabyte files in a timely manner. This will be possible only if the technology for using and moving the data is greatly improved.

This paper has presented some examples of deep science application phases that have different data usage characteristics. It then discussed the current technology available to support these phases for large-scale applications. Each application phase challenges the technology in different ways. We have indicated, for each technology area, how the challenges can be overcome and some emerging work that is pointing the way to full-scale solutions. Finally, we have discussed areas that remain open for new solutions.

Only with the creation of improved technology will the scientific challenges of the twenty-first century be met. Very large, scalable, high-performance archives and shared file systems are just the underlying technologies that are necessary. Both of these will rely on layers of technology that must work together to provide high performance not just for many independent data flows, but also for very large, single data flows. New protocols are necessary, since the data flows are beginning to exceed the capability of yesterday's protocols. End host hardware improvements are also needed to take advantage of the network capabilities. Data management methods are key to being able to organize and locate the relevant information in an acceptable time. These methods will have to be built into the applications and eventually the underlying storage and networking infrastructure. In order to not become the throttling bottleneck, new security approaches will be needed that allow openness and service while providing protection. Finally, monitoring and eventually control capabilities will be needed to keep pace with the system improvements. Work is proceeding in all of these areas, and the authors are hopeful that technology will keep pace with the imagination of those of us who are doing deep computing.

Acknowledgments

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC 03-76SF00098.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Tivoli Systems, Sun Microsystems, Inc., VERITAS Software Corporation, The Open Group, 3PARdata, Inc., Panasas, Inc., YottaYotta, Inc., Hewlett-Packard Company, Myricom, Inc., Intel Corporation, Microsoft Corporation, or Linus Torvalds.

References

1. "Towards Globus Toolkit 3.0: Open Grid Services Architecture," *http://www.globus.org/ogsa/*.
2. M. Alcubierre, B. Bruegmann, P. Diener, M. Koppitz, D. Pollney, E. Seidel, and R. Takahashi, "Gauge Conditions for Long-Term Numerical Black Hole Evolutions Without Excision," *Phys. Rev. D* **67**, 084023 (2003).

3. "Lawrence Berkeley National Lab Leads International Team to Win High-Performance Computing Bandwidth Challenge," *http://www.lbl.gov/CS/Archive/othernews112502.html*.
4. J. Shalf and E. W. Bethel, "Cactus and Visapult: An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols," *IEEE Computer Graph. & Appl.* **23**, No. 2, 51–59 (2003).
5. "NERSC Helps Climate Scientists Complete First Ever 1,000-Year Run of Nation's Leading Climate Change Modeling Application," *http://www.lbl.gov/CS/Archive/headlines090402.html*.
6. Community Climate System Model, *http://www.ccsm.ucar.edu/*.
7. The Nearby Supernova Factory, *http://snfactory.lbl.gov/*.
8. U.S. DOE Coupled Climate Model Data Archive, *http://www.nersc.gov/projects/gcm_data/*.
9. "The Earth System Grid II: Turning Climate Model Datasets into Community Resources," *http://www.earthsystemgrid.org/*.
10. "Analysis of *Fugu rubripes* Genome Predicts Nearly 1,000 New Human Genes," *http://www.jgi.doe.gov/News/news_7_25_02.html*.
11. "Computational Analysis of Genomic Sequence Data," *http://www.nersc.gov/research/annrep01/sh_BER_06.html*.
12. "High Performance Networks for High Impact Science," Report of the August 13–15, 2002 Workshop on Networking, conducted by the DOE Office of Science, *http://doecollaboratory.pnl.gov/meetings/hpnpw/finalreport/high-impact_science.pdf*.
13. Fibre Channel Industry Association, *http://www.fibrechannel.org/*.
14. "NERSC Strategic Proposal FY2002–FY2006," *http://www.nersc.gov/aboutnersc/pubs/Strategic_Proposal_final.pdf*.
15. High Performance Storage System (HPSS); see *http://www4.clearlake.ibm.com/hpss/*.
16. IBM Tivoli® Storage Manager, *http://www-3.ibm.com/software/tivoli/*.
17. Storage Archive Manager File System, *http://www.sun.com/storage/software/data_mgmt/utilization/*.
18. VERITAS Software Corporation, *http://www.veritas.com/*.
19. Cray Inc. Data Migration Facility, *http://www.cray.com/products/software/dmf.html*.
20. "Reference Model for Open Storage Systems Interconnection—Mass Storage System Reference Model Version 5," developed by the IEEE Storage System Standards Working Group (Project 1244), Sept. 8, 1994, R. Garrison, Martin Marietta Corporation, Ed.; *http://www.arl.mil/IEEE/ssswg.html*.
21. 3PARdata, Inc., 4209 Technology Drive, Fremont, CA 94538; Phone: (510) 413-5999, *http://www.3pardata.com/*.
22. Panasas, Inc.; see *http://www.panasas.com/*.
23. YottaYotta Inc., 6020 - 104 Street, Edmonton, Alberta, T6H 5S4; Phone: (780) 989-6800; *http://www.yottayotta.com/*.
24. Julian Satran, Kalman Meth, Costa Sapuntzakis, Mallikarjun Chadalapaka, and Efri Zeidner, "iSCSI Internet Draft 19-January-03," *http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt*.
25. InfiniBand Trade Association, *http://www.infinibandta.org/home/*.
26. Gregory F. Butler, Rei Chi Lee, and Michael L. Welcome, "The Global Unified Parallel File System (GUPFS) Project: FY 2002 Activities and Results," Lawrence Berkeley National Laboratory Technical Report LBNL-52456 (2003), *http://www.nersc.gov/aboutnersc/pubs/GUPFS_02.pdf*.
27. Peter J. Braam and Rumi Zahir, "Lustre Technical Project Summary," Version 2, July 29, 2001, Cluster File Systems, Inc., Mountain View, CA, *http://www.lustre.org/docs/lustre-sow-dist.pdf*.
28. "General Parallel File System (GPFS) for AIX®," *http://www-1.ibm.com/servers/eserver/pseries/software/sp/gpfs.html*.
29. Lustre; see *http://www.lustre.org/*.
30. Christos Karamanolis, Mallik Mahalingam, Dan Muntz, and Zheng Zhang, "DiFFS: A Scalable Distributed File System," Hewlett-Packard Laboratories Technical Report HPL-2001-19 (2001), *http://www.hpl.hp.com/techreports/2001/HPL-2001-19.pdf*.
31. Maximum Throughput InfinARRAY, *http://www.max-t.com/html/products/infinarray.html*.
32. Ibrix, Inc., *http://www.ibrix.com/*.
33. The Parallel Virtual File System (PVFS), *http://www.parl.clemson.edu/pvfs/*.
34. Arie Shoshani, Alex Sim, and Junmin Gu, "Storage Resource Managers: Middleware Components for Grid Storage," presented at the Nineteenth IEEE Symposium on Mass Storage Systems (MSS02), 2002, *http://sdm.lbl.gov/~arie/papers/srm.mss02.pdf*.
35. Arie Shoshani, Alexander Sim, and Junmin Gu, "Storage Resource Managers: Essential Components for the Grid," *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds., Kluwer Academic Publishers, New York, 2003.
36. Particle Physics Data Grid, *http://www.ppdg.net/*.
37. Fermilab Mass Storage System, *http://hppc.fnal.gov/enstore/*.
38. JASMine—Jefferson Lab Asynchronous Storage Manager, *http://cc.jlab.org/scicomp/JASMine/*.
39. The CASTOR Project (CERN Advanced STORage Manager), *http://castor.web.cern.ch/castor/*.
40. Myricom®, Inc., *http://www.myri.com/*.
41. Quadrix Solutions, Inc., *http://www.quadrix.com/*.
42. G. Jin and B. Tierney, "System Capability Effects on Algorithms for Network Bandwidth Measurement," *Proceedings of the Internet Measurement Conference*, Miami, October 27–29, 2003; Lawrence Berkeley National Laboratory Report LBNL-48556, *http://dsd.lbl.gov/DIDC/papers/imc-2003.pdf*.
43. C. William McCurdy, Rick Stevens, Horst Simon, William Kramer, David Bailey, William Johnston, Charlie Catlett, Rusty Lusk, Thomas Morgan, Juan Meza, Michael Banda, James Leighton, and John Hules, "Creating Science-Driven Computer Architecture: A New Path to Scientific Leadership," Lawrence Berkeley National Laboratory Report LBNL/PUB-5483 (2002), *http://www.nersc.gov/news/ArchDevProposal.5.01.pdf*.
44. T. Dunigan, M. Mathis, and B. Tierney, "A TCP Tuning Daemon," *Proceedings of the IEEE/ACM SC2002 Conference*, November 2002, *http://www-didc.lbl.gov/papers/net100.sc02.final.pdf*.
45. S. Floyd, "High Speed TCP for Large Congestion Windows," Internet draft, 2003, *http://www.icir.org/floyd/papers/draft-floyd-tcp-highspeed-03.txt*.
46. B. Tierney, "TCP Tuning Guide," *http://www-didc.lbl.gov/TCP-tuning/TCP-tuning.html*.
47. C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST Kernel: Background Theory and Experimental Results," presented at the First International Workshop on Protocols for Fast Long-Distance Networks, February 3-4, 2003, CERN, Geneva, Switzerland, *http://netlab.caltech.edu/pub/papers/pfldnet.pdf*.
48. V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks* **31,** 2435–2463 (1999).
49. D. Gunter, B. Tierney, C. E. Tull, and V. Virmani, "On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service," presented at the Fourth International Workshop on Grid Computing (Grid2003), Phonix, AZ, November 17, 2003.

50. D. Agarwal, J. M. González, G. Jin, and B. Tierney, "An Infrastructure for Passive Network Monitoring of Application Data Streams," presented at the 2003 Passive and Active Measurement Workshop, San Diego, April 2003.

Received July 25, 2003; accepted for publication October 23, 2003

William T. C. Kramer National Energy Research Scientific Computing (NERSC) Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (WTKramer@lbl.gov). As NERSC Center General Manager and Division Deputy, Mr. Kramer is responsible for NERSC computational facilities and support. Prior to his Berkeley Laboratory appointment, Mr. Kramer was a member of the NASA Ames Research Center, where he was Chief of Advanced Air Transportation Technologies and was responsible for creating and implementing a research and development program for designing revolutionary new air traffic management systems. From 1988 to 1994, he was Branch Chief of NASA's Numerical Aerodynamic Simulation (NAS) Computational Services Branch, responsible for all aspects of operations and customer service for NASA's principal supercomputer center. He holds B.S. and M.S. degrees in computer science from Purdue University, an M.E. degree in electrical engineering from the University of Delaware, and, following his commitment to continuous education, he is a Ph.D. candidate in computer science at the University of California at Berkeley.

Arie Shoshani *Computational Research Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (AShoshani@lbl.gov).* Dr. Shoshani has been Group Leader of the Scientific Data Management Research and Development Group at the Lawrence Berkeley National Laboratory since 1978. The group's research activities focus on the development of algorithms and software for the organization, access, and manipulation of scientific databases. Dr. Shoshani's own technical work is mainly in the characterization of the unique requirements of scientific databases, query languages, modeling of statistical data, temporal data, sequence data, multidimensional data, data compression, and mapping techniques from extended entity-relationship schemas into relational schemas. He has been involved with several scientific projects, including the Human Genome Project, an epidemiological database for exposure to low-level radiation, the optimization of climate modeling data on tertiary storage, and the indexing and organization of high-energy physics data on tertiary storage. He received M.A. and Ph.D. degrees in computer sciences from Princeton University and a B.S. degree in control engineering from the Technion–Israel Institute of Technology.

Deborah A. Agarwal *Computational Research Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (DAAgarwal@lbl.gov).* Dr. Agarwal is Group Leader of the Collaboration Technologies Group and Department Head in the Distributed Systems Department at the Lawrence Berkeley National Laboratory. She has been a project leader responsible for the design and development of software to allow remote experimentation and collaboration between sites connected by a wide-area network. As an expert on reliable multicasting, she served as an advisor to the Preparatory Committee of the Comprehensive Test Ban Treaty Organization, and in recognition of her contribution was voted one of the Top 25 Women on the Web. Dr. Agarwal received a Ph.D. degree in electrical and computer engineering from the University of California at Santa Barbara.

Brent R. Draney *National Energy Research Scientific Computing (NERSC) Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (BRDraney@lbl.gov).* Mr. Draney is a member of the NERSC Networking and Computer Security Group. He received a B.S. degree from the University of Utah and has carried out graduate course work in statistics.

Guojun Jin *Computational Research Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (G_Jin@lbl.gov).* Mr. Jin received a B.S. degree in computer technique from Beijing Industrial University and an M.S. degree in computer science from San Francisco State University. His research and development interests include gigabit network-based distributed applications for scientific imaging and the use of video for dynamic object analysis and laboratory control; acquisition, storage, processing, analysis, editing, and display of a variety of scientific image data (MRI, video, etc.); algorithms on concurrent programming (thread, SIMD, and MIMD) for speeding up image processing, analysis, and visualization of three-dimensional images; and the development of high-speed network interface and system software.

Gregory F. Butler *National Energy Research Scientific Computing (NERSC) Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (GFButler@lbl.gov).* Mr. Butler holds a B.A. degree in astronomy from Northwestern University. He has more than twenty years of experience in high-performance scientific computing at federal installations, including system administration, software maintenance, and upgrades of high-performance production scientific computing systems. He also has more than ten years of experience in operating system development on multiple high-performance scientific computing platforms and operation systems.

John A. Hules *Information Technologies and Services Division, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California 94720 (JAHules@lbl.gov).* Mr. Hules received a B.A. degree in philosophy from Borromeo College of Ohio. He is currently a science writer and technical editor for the Computing Sciences organization at the Lawrence Berkeley National Laboratory. He previously worked in corporate and marketing communications for environmental and civil engineering firms.