High-Speed Distributed Data Handling for On-Line Instrumentation Systems[1]

William E. Johnston[2], William Greiman, Gary Hoo, Jason Lee, Brian Tierney, Craig Tull
Computing Sciences
Ernest Orlando Lawrence Berkeley National Laboratory
{WEJohnston, WHGreiman, GJHoo, Jason_Lee, BLTierney, CETull}@lbl.gov

Douglas Olson, Nuclear Science Division
Ernest Orlando Lawrence Berkeley National Laboratory
DLOlson@lbl.gov

Abstract

The advent (and promise) of shared, widely available, high-speed networks provides the potential for new approaches to the collection, organization, storage, and analysis of high-speed and high-volume data streams from high data-rate, on-line instruments. We have worked in this area for several years, have identified and addressed a variety of problems associated with this scenario, and have evolved an architecture, implementations, and a monitoring methodology that have been successful in addressing several different application areas.

In this paper we describe a distributed, wide area network-based architecture that deals with data streams that originate from on-line instruments. Such instruments and imaging systems are a staple of modern scientific, health care, and intelligence environments. Our work provides an approach for reliable, distributed real-time analysis, cataloguing, and archiving of the data streams through the integration and distributed management of a high-speed distributed cache, distributed high-performance applications, and tertiary storage systems.

1.0 Introduction

In this paper we discuss aspects of our work that relate to a project whose goal is to demonstrate a scalable approach to the problem of high-bandwidth data handling for analysis of high-energy and nuclear physics data. We also consider the application of the approach to having a source of data (20-40 megabytes/s) that is remote from the computational and storage facilities.

The high energy nuclear physics (HENP) accelerator detector data analysis problem consists of two parts - data collection and event reconstruction (phase 1) and physics analysis (phase 2).

In phase 1 of a physics experiment, a detector puts out a steady state data stream of 20-40 megabytes/s. Traditionally, this data is archived and a first level of processing is performed at the experiment site. The resulting second-level data is also archived and requested later for further analysis. The data thus is archived at the experiment site in “medium” sized tertiary storage systems.

---

1. The work described in this paper is supported by the U. S. Dept. of Energy, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division (http://www.er.doe.gov/production/octr/mics) under contract DE-AC03-76SF00098 with the University of California, and by DARPA, Information Systems Technology Office (http://www.ito.darpa.mil). Contact: wejohnston@lbl.gov, Lawrence Berkeley National Laboratory, mail stop: B50B-2239, Berkeley, CA, 94720, ph: 510-486-5014, fax: 510-486-6363, http://www-itg.lbl.gov). This is report no. LBNL-40697.

2. W. E. Johnston: mail address: 50B-2239, Lawrence Berkeley National Laboratory, Berkeley, CA 94720. Tel: +1-510-486-5014, fax: +1-510-486-6363, wejohnston@lbl.gov, http://www-itg.lbl.gov/~johnston.

We believe that the "medium" sized tertiary systems at experiment sites can be replaced by a distributed storage system consisting of a high-speed, high-capacity disk-based cache and very large tertiary systems at dedicated storage sites. For the various data sources and sinks, the cache provides:

- a standardized approach for high data-rate interfaces;
- an "impedance" matching function (e.g., between the coarse-grained nature of parallel tape drives in the tertiary storage system and the fine-grained access of hundreds of applications);
- flexible management of cache resources to support initial caching of data, processing, and interfacing to tertiary storage.

We propose the use of the LBNL-developed Distributed-Parallel Storage System (DPSS) as the cache for all stages of data manipulation. The DPSS provides a scalable, dynamically configurable, high-performance, and highly distributed storage system that is usually used as a (relatively long-term) cache of data. It is typically used to collect data from on-line instruments and then supply that data to analysis applications, or to supply data to high data-rate visualization applications as in the case of the MAGIC wide-area gigabit testbed where the DPSS was originally developed. (See [Lau94], [DPSS], and [MAGIC].) The system is also being used in satellite image processing systems and for distributed, on-line, high data-rate health care imaging systems.

We have two specific objectives for this project. One is to demonstrate the use of distributed computational systems to do the phase 1 data processing in real time. The real-time processing of this data potentially supports two capabilities. First, it can provide auxiliary information to assist in the organization of data as it is transferred to tertiary storage (the experiment is expected to generate about 1.7 terabytes/day), and second, it can provide feedback to the instrument operators about the functioning of the accelerator - detector system and the progress of the

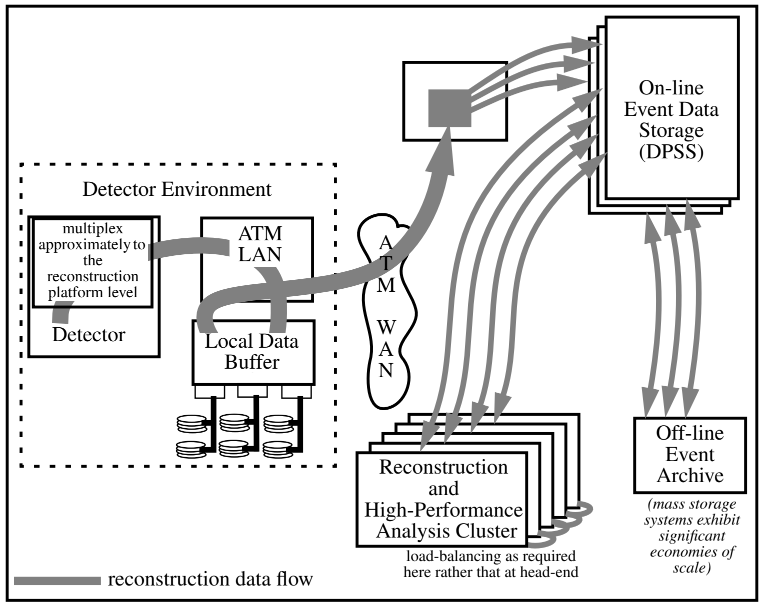

Figure 1a Distributed physics data handling, Phase 1data flows.

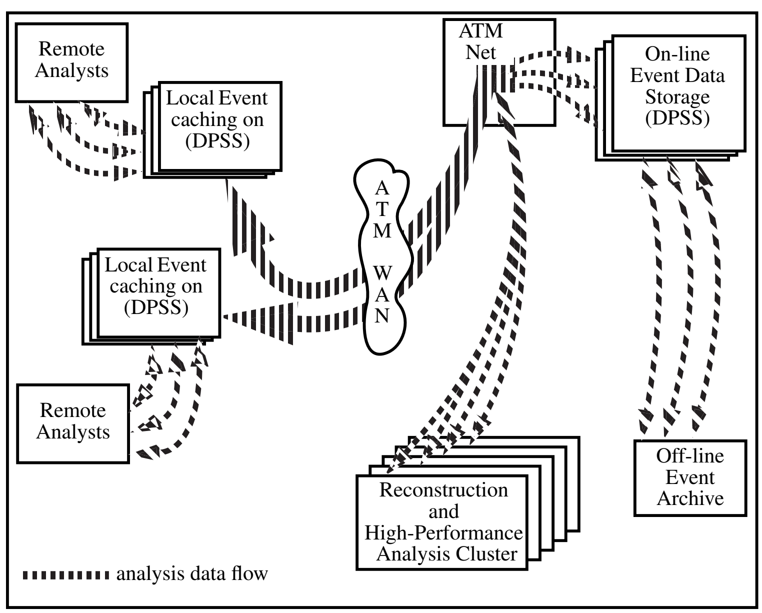

Figure 1b Distributed physics data handling, Phase 2 data flows.

experiment, so that changes and corrections may be made. This environment is illustrated in Figure 1a.

The second objective is to demonstrate an architecture that maximizes the efficiency of the phase 2 analysis of the data. This involves using a high-speed cache (the DPSS) as a large “window” on the tape-based data in the tertiary storage system in order to support the use of both local and remote computational resources. This is illustrated in Figure 1b.

The architectural issues include the organization of the cache, the various interfaces to the cache, the management of the movement of data to and from the tertiary storage systems, and management in a wide area network.

We describe an experiment that is designed to validate and demonstrate the approach, and some early results using an OC-12 (622 mbits/sec) ATM network to connect the components that implement the architecture. The STAR experiment at RHIC ([STAR1], [STAR2]) is used as the basis for a realistic example.[3]

2.0 The Overall Model

The high-speed data handling model is based on the idea of a standard interface to a large, application-oriented, distributed disk-based cache. Each data source deposits its data in the cache, and each data consumer takes data from the cache, usually writing the processed data back to the cache. In almost every case there is also a tertiary storage system manager that migrates data to and from the cache. (See Figure 2.)

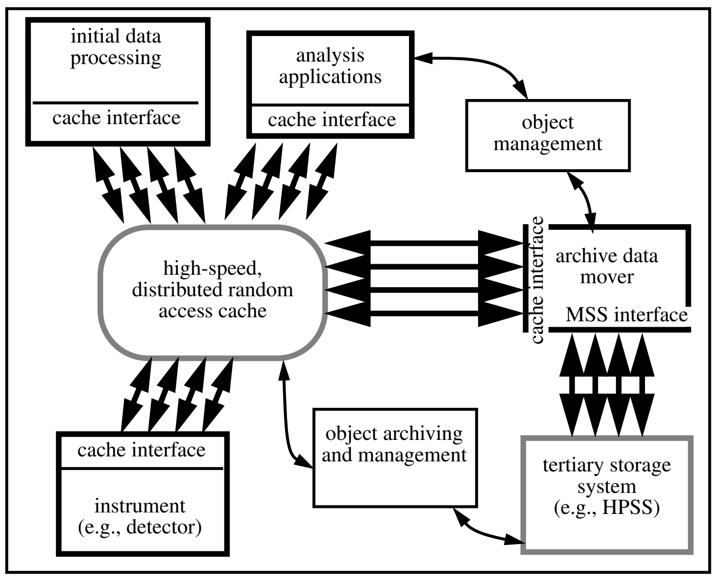

Figure 2 **The Data Handling Model**

Depending on the size of the cache relative to the objects of interest, the storage system management (object manager + archive data mover of Figure 2) may only involve moving partial objects to the cache; that is, the cache is a moving window for the object/data set. The cache - application interface can (and for this application, does) implement Unix disk I/O semantics: upon posting a read, the available data is returned; requests for data in the data set but not yet migrated to cache cause the application-level read to block until the data is migrated from tape to cache.

Generally, the cache is large compared to the available disks of a typical computing environment, and very large compared to any single disk (e.g., hundreds of gigabytes).

This general model has been used in several data-intensive computing applications. For example, a real-time digital library system (see Figure 3 and [DIGLIB]) collects data from a remote medical imaging system, and automatically processes, catalogues, and archives each data unit together with the derived data and metadata, with the result being a Web-based object representing each data set.

3. RHIC is a high energy physics accelerator and STAR is a detector (instrument) at RHIC.

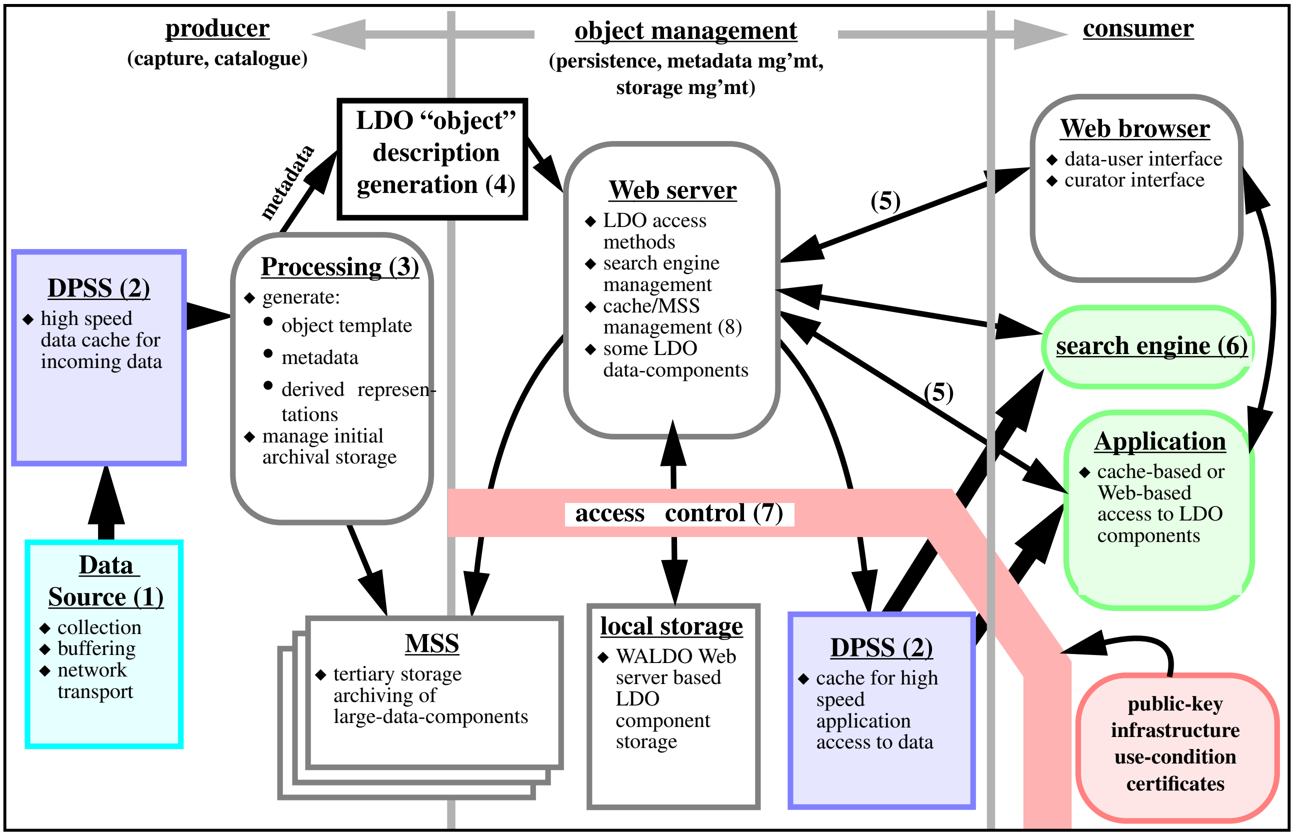

Figure 3 Distributed Large-Data-Object Overall Architecture and Data Flow

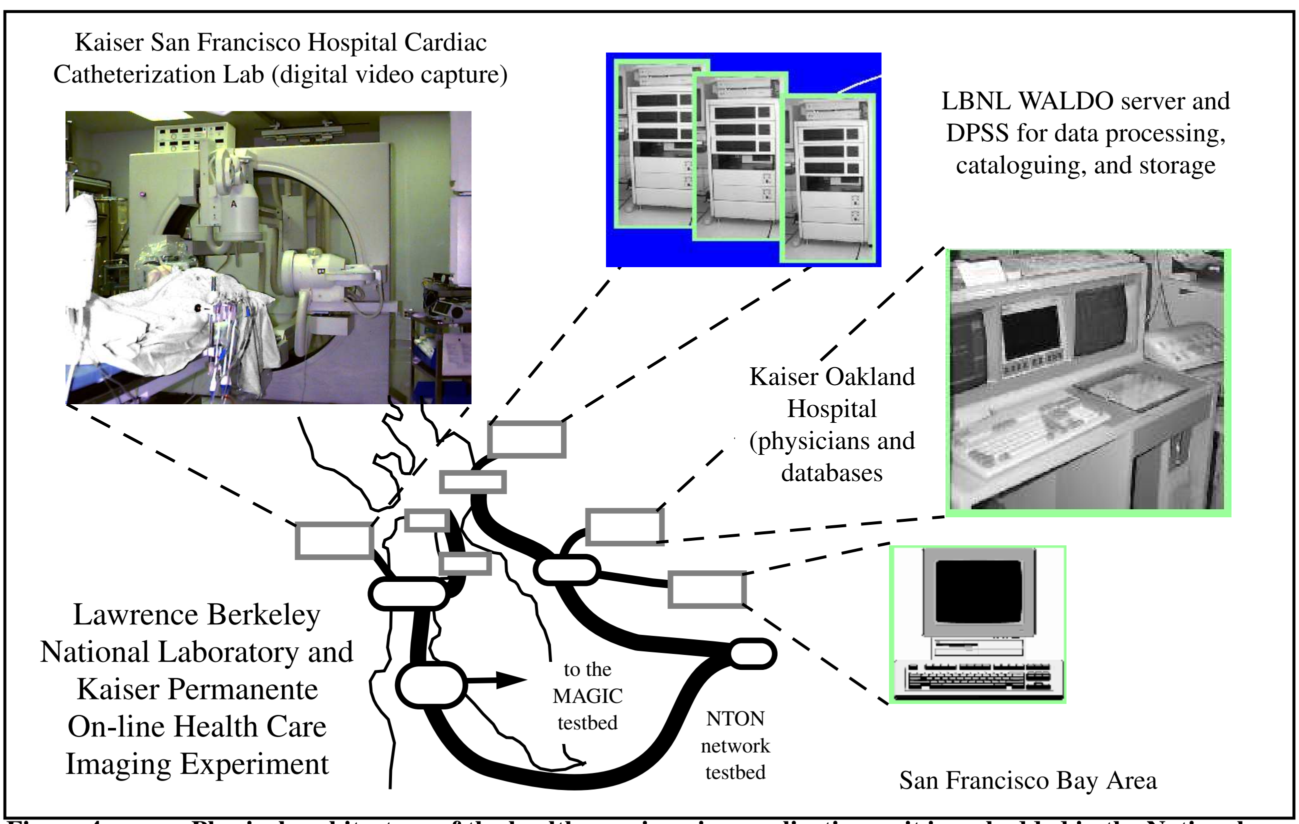

Figure 4 Physical architecture of the health care imaging application as it is embedded in the National Transparent Optical Network testbed.

This automatic system operates 10 hours/day, 5-6 days/week with data rates of about 30 Mbits/sec during the data collection phase (about 20 minutes/hour). (See Figure 4.)

3.0 Prototype Architecture for HENP Distributed Analysis

The STAR analysis framework (STAF - see [Tull97]) is being used to provide a realistic application environment in which to validate and refine the data-handling architecture and implementation.

Generally speaking, STAF (Figure 5) manages self-describing data structures on behalf of analysis modules. Data is requested through a standard interface that supports several communications models, including the DPSS cache. The data is converted to machine-specific format and placed into memory data structures, whence it is accessed by the analysis modules.

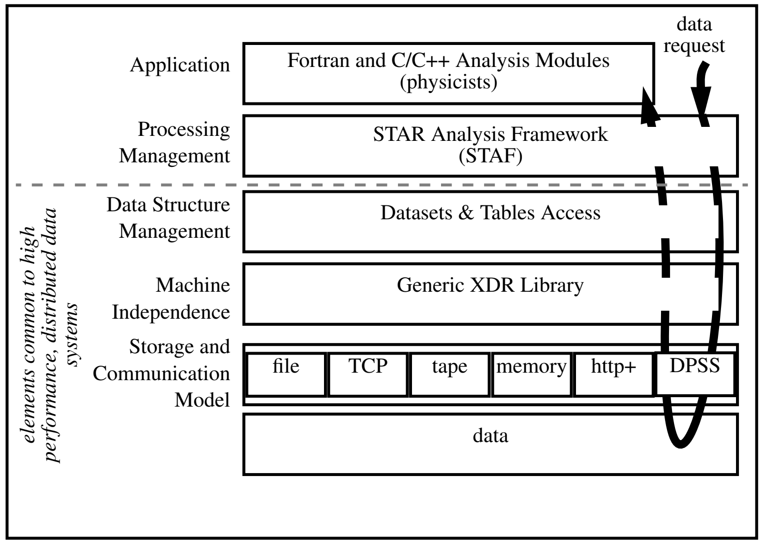

Figure 5 **STAF: The STAR analysis framework.**

The prototype architecture for HENP data analysis is illustrated in Figure 5. The analysis phase is a second level of processing of the detector data, and typical data volumes are 1700 gigabytes/day, with a processing requirement of six KSpecInt92/ Mbyte/sec. [Olson]

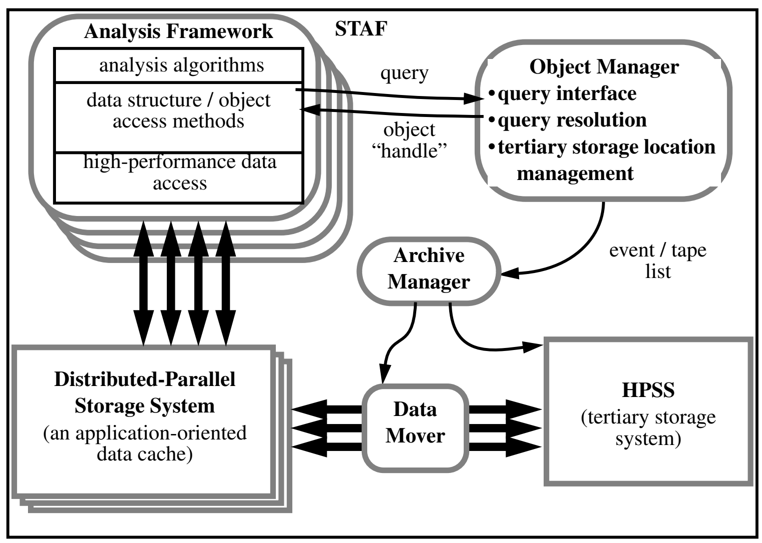

Figure 6 **Architecture**

The analysis framework generates queries that produce a list of objects of interest. This list of objects, then, has to be retrieved from tertiary storage, and loaded into the cache for processing. The loading process involves parallel transfers from the tertiary storage system to the cache. When an object (or partial object) has been loaded into the cache, the object manager is notified, and in turn it notifies the analysis code. Multiple instances of the analysis code (operating under the control of a work flow manager) running simultaneously on many different systems then read data from the cache into memory, and processing commences. In a typical configuration (e.g., Figure 1b) the analysis systems may be widely distributed, and they all consume data from the cache, and return results to the cache.

4.0 The Cache Architecture

The Distributed-Parallel Storage System (DPSS) serves several roles in high-performance, data-intensive computing environments. This application-oriented cache provides a standard interface for high-speed data access, and provides the functionality of a single very large, random access, block-oriented I/O device (i.e., a “virtual disk”). It provides high capacity (we anticipate a terabyte for the full-blown version of the experiment described here) and serves to isolate the application from the tertiary storage system. Many large data sets can be logically present in the cache by virtue of the block index maps being loaded even if the data is not yet available. In this way processing can begin as soon as the first data has been migrated from tertiary storage.

Generally speaking, the DPSS can serve as an application cache for any number of high-speed data sources (instruments, multiple mass storage systems, etc.). The naming issue (e.g., resolving independent name space conflicts) is handled elsewhere. For example, in the on-line health care imaging system mentioned above, the name space issue is addressed by having all of the data represented by Web-based objects which are managed by the LBNL Wide Area Large Data Object management architecture (WALDO) [DIGLIB]. At the minimum WALDO provides globally unique naming, and serves as a mechanism for collecting different sources of information about the data. In our model, the Web object system also provides a uniform user (or application) frontend for managing the data components (e.g., migration to and from different mass storage systems) and it manages use-conditions (PKI access control - see [Johnston97]).

The DPSS provides several important and unique capabilities for the distributed architecture. The system provides application-specific interfaces to an extremely large space of logical blocks (16-byte indices); the DPSS may be dynamically configured by aggregating workstations and disks from all over the network (this is routinely done in the MAGIC testbed [MAGIC], and will in the future be mediated by the agent-based management system); it offers the ability to build large, high-performance storage systems from the inexpensive commodity components; and it offers the ability to increase performance by increasing the number of parallel operating DPSS servers. A cache management policy interface operates on a per-data set basis to provide block aging and replacement.

The high performance of the DPSS is obtained through parallel operation of independent, network-based components. Flexible resource management - dynamically adding and deleting storage elements, partitioning the available storage, etc. - is provided by design, as are high availability and strongly bound security contexts. The scalable nature of the system is provided by many of the same design features that provide flexible resource management, which has the capability to aggregate dispersed and independently owned storage resources into a single cache.

When data sets are identified by the object manager and are requested from tertiary storage, the logical-to-physical block maps become immediately available. The data mover operates asynchronously, and if an application “read” requests a block that has not yet been loaded, then the application is notified (e.g., the read operation blocks). At this point the application can wait or request information on available blocks in order to continue processing. (STAF reads megabyte-sized data units, but processes these units independently.)

For the STAF application, file I/O semantics are provided in the DPSS access interface, and reads do not complete until data is available.

The internal architecture of the DPSS is illustrated in Figure 7

.

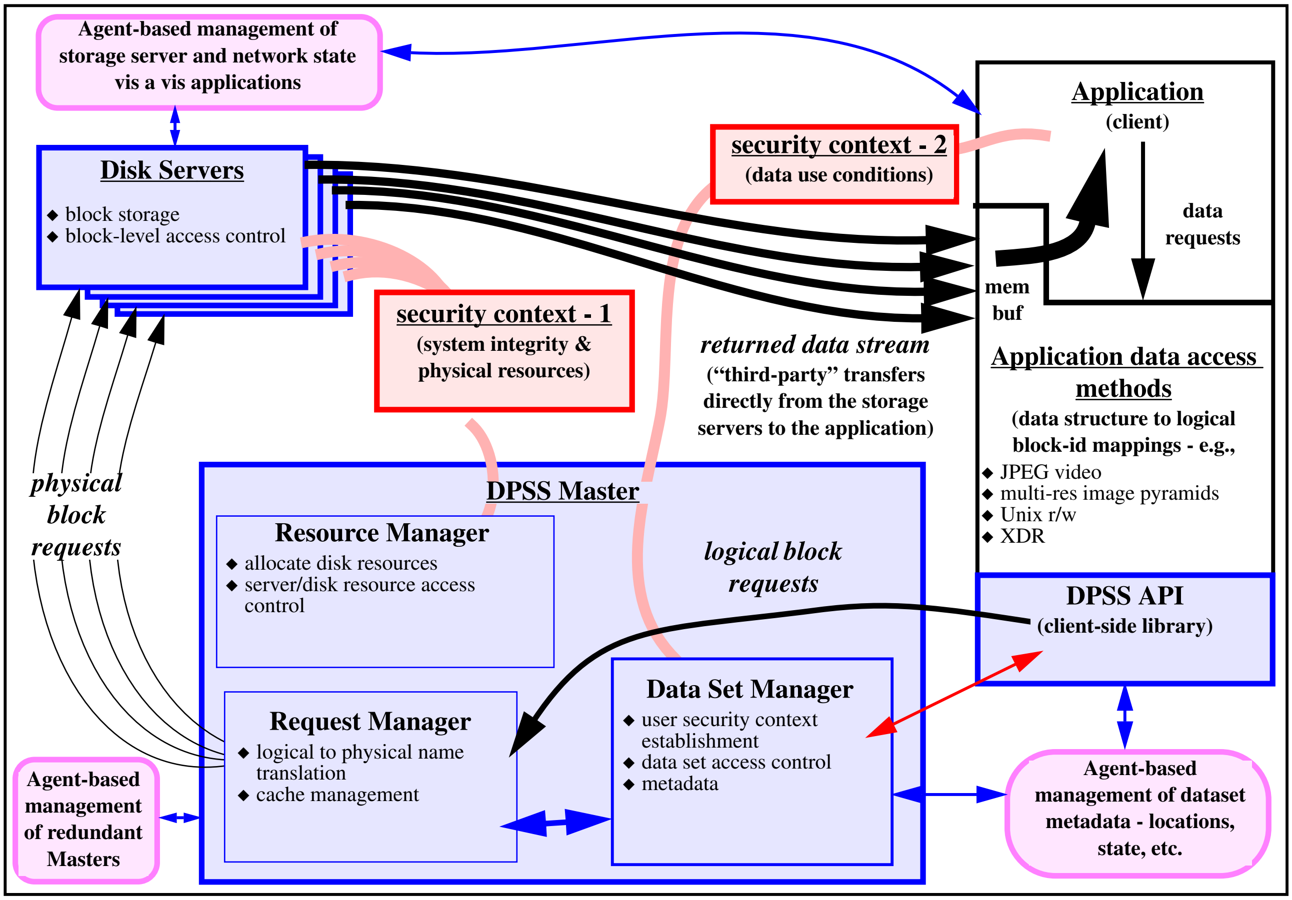

Figure 7 Distributed-Parallel Storage System Architecture

Typical DPSS implementations consist of several low-cost workstations, each with several SCSI controllers, and several disks on each controller. A three-server DPSS can thus provide transparent parallel access to 20-30 disks. The data layout on the disks is completely up to the application, and the usual strategy for sequential reading applications is to write the data "round-robin" (stripe across servers), otherwise the block locations are randomized when they are written. (Our experience has shown that random placement of blocks provides nearly optimal parallelism for a wide range of read patterns if the number of independent disks is large.)

5.0 Performance Measurement

The combination of generalized, autonomous management of distributed components (e.g., via agents - see [Where]) and accurate monitoring of *all* aspects of the environment in which data moves have turned out to be critical aspects of the debugging, evaluation, and management of widely distributed, high data-rate applications.

We have developed an approach for analysis of the operation of distributed applications in high-speed wide-area networks that is designed to identify all of the issues that affect performance, and to isolate the problems arising from individual hardware and software components.

Our methodology involves recording every event of potential significance together with precision tim-stamps, and then correlating events on the basis of the logged information. This allows us to construct a comprehensive view of the overall operation, under realistic operating conditions, revealing the behavior of all the elements of the application-to-application communications path in order to determine exactly what is happening within complex distributed systems. In particular, we have instrumented our distributed storage system and its client applications to do timestamping and logging at every critical point. We have also modified some of the standard Unix network and operating system monitoring tools to log events of interest using a common format. This monitoring functionality is designed to facilitate performance tuning, distributed application performance research, the characterization of distributed algorithms, and the management of functioning systems (by providing the input that allows adaptation to changes in operating conditions). The approach allows us to measure network performance in a manner that is a much better "real-world" test than, e.g., ttcp, et al, and allows us to accurately measure the dynamic throughput and latency characteristics of our distributed application code -- "top-to-bottom" and "end-to-end".

Detailed monitoring is also a practical tool for problem analysis, as has been demonstrated in the analysis of a TCP over ATM problem that was uncovered while developing the monitoring methodology in the ARPA-funded MAGIC gigabit testbed (a large-scale, high-speed, ATM network). See [Tierney1].

The high-level motivation for this work is two-fold.

First, as developers of high-speed network-based distributed services, we often observe unexpectedly low network throughput and/or high latency. The reason for the poor performance is frequently not obvious. The bottlenecks can be (and have been) in any of the components: the applications, the operating systems, the device drivers, the network adapters on either the sending or receiving host (or both), the network switches and routers, and so on. It is difficult to track down performance problem because of the complex interaction between the many distributed system components, and the fact that problems in one place may be most apparent somewhere else.

Second, we want to develop approaches to building predictable, high-speed components that can be used as building blocks for high-performance applications, rather than having to "tune" the applications top-to-bottom as is all too common today.

5.1 An Example: The DPSS Timing Facility

To illustrate the approach we describe how it is used in the development and debugging of our Distributed-Parallel Storage System.

When applications request data blocks from the DPSS, the request lists take the following path (see Figure 8). A request (a list of data blocks) goes from the application to the name server ("START"), where the logical block names are translated to physical addresses (`server:disk:disk_offset`), then the individual requests are forwarded to the appropriate disk servers. At the disk servers, the data is read from disk into local cache, and then sent independently to the application (which has connections to all the relevant servers). Timestamps are gathered before and after each major DPSS function, such as name translation, disk read, and network send, etc. All timestamps are logged by the DPSS servers so that the precise timing of each step is noted for each data block. The DPSS specific timestamps are sent with the data block back to the requesting application, where logging can be performed using the DPSS client library. Other events and timestamps are logged using, e.g., the Unix `syslog` facility.

Timestamp consistency is critical for this approach, and is provided by GPS-based Network Time Protocol (as described below), which allows us to make precise throughput and latency measurements throughout the widely distributed DPSS system and underlying network. Using this approach, we can

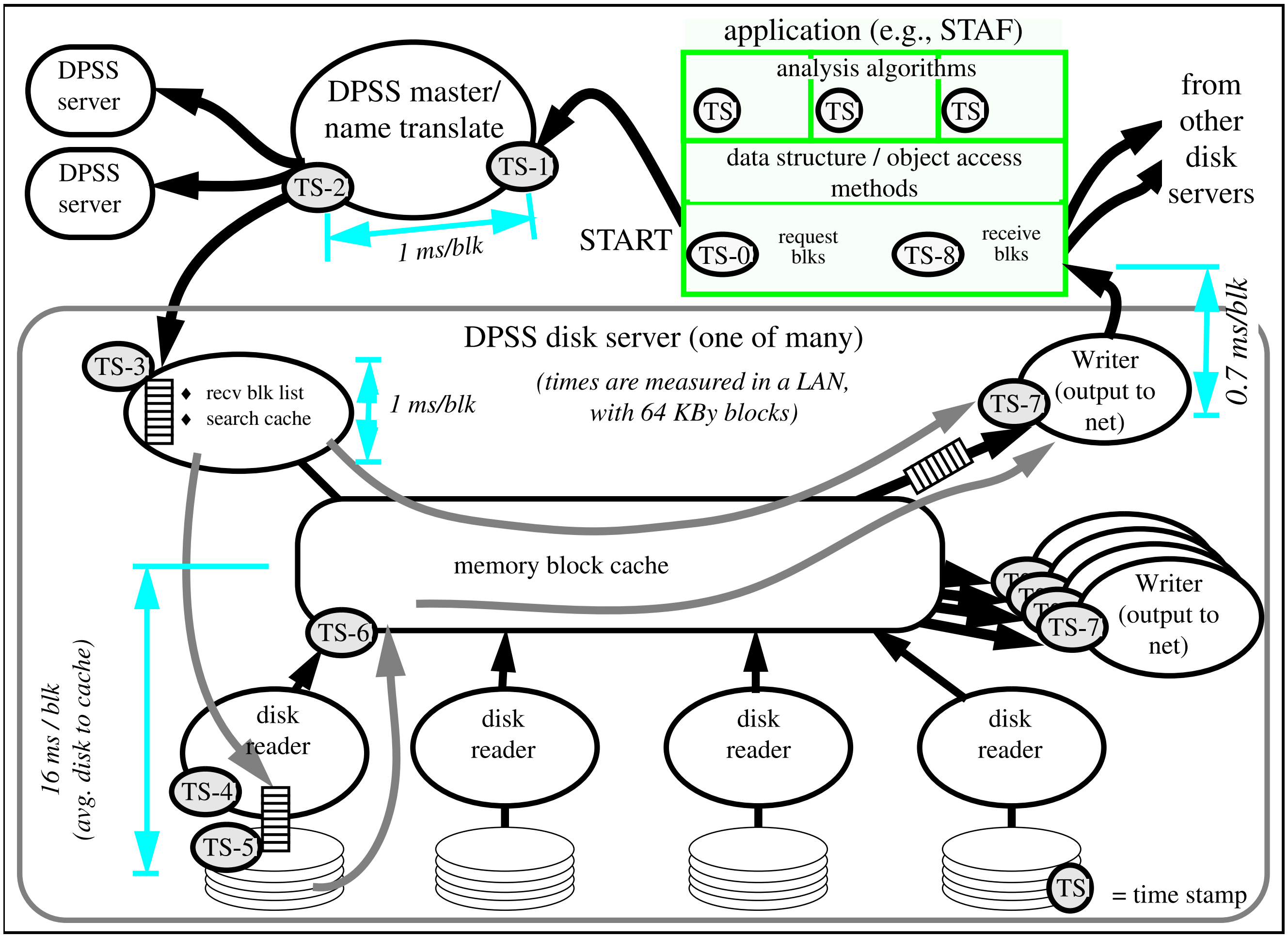

Figure 8 **Monitoring for a DPSS application.**

pinpoint delays, and their impact on the overall system, to within narrowly-specified steps in the data path.

OS and Network Layer Monitoring

To complement the monitoring at the application level and in the DPSS, we also monitor various operating system and network conditions. For example, we currently collect and log the following types of information on every platform involved in the distributed system:

- TCP retransmits
- CPU usage (user and system)
- CPU interrupts
- AAL 5 information
- ATM switch buffer overflows
- ATM hosts adapter buffer overflow

Common logging format

To process the several gigabytes of log files that can be generated from this type of logging, all events are logged using a common format (see [Tierney2]):

```
keyword; hostname; Unix_date_seconds; nano-sec; data; data; data;......;
```

The "`keyword`" is a unique identifier describing what is being logged. By convention, the first part of the keyword is a reference to the program that is doing the logging (e.g., `DPSS_SERV_IN`, `VMSTAT_SYS_CALLS`, `NETSTAT_RETRANSSEGS`, `TV_REQ_TILE`). Each log entry also contains both the host-name of the system on which the event occurred and a timestamp.

The end of every logging record can contain any number of "data" elements. These are used to record any information about the logged event that may later prove useful. For example, the `NETSTAT_RETRANSSEGS` event records one data element, the number of TCP retransmits since the previous event; the `DPSS_START_WRITE` event records the logical block name, the data set ID, a "user session" ID, and an internal DPSS block counter; etc.

Log File Analysis Tools

Some of the log records are "associated" by virtue of being collected and carried in the data block request message as it works its way through the system, and back to the client (this is, of course, specific to the DPSS). Other records are logged locally or remotely, which allows a quite general application of these techniques.

Tools to analyze log files include perl scripts[4] to extract information from log files, correlate and organize events based on time stamps, process name, and system identity, and write data files in a format suitable for using gnuplot[5] to present the results for analysis. These tools were used to generate the analytical graphs below.

In our previous work, the operation history re-assembly and analysis has been after-the-fact, which is useful for diagnostic and design change evaluation, but less so for monitoring running systems. Our current approach is to have agent-based monitoring and management components as an integral part of the distributed system, and these agents collect and analyze `netlogger` data in real time. (See [Where].)

Use of NTP

The DPSS name server, DPSS disk server, and application are typically on different physical hosts scattered over the network. To be able to perform meaningful analysis of a network-based system using timestamps, the clocks of all systems involved must be synchronized. In the MAGIC testbed, for example, all systems run the `xntpd` program [Mills], which synchronizes the clocks of each host. The MAGIC backbone segments are used to distribute GPS time via NTP, allowing the clocks of all hosts to synchronize within about 250 microseconds of each other. It has been our experience, at least so far in the wide-area, that almost all application-significant events can be accurately characterized by timestamps that are accurate to 0.5 ms, so the 250 microsecond accuracy is just right.

5.2 Analysis of Performance Experiments

Experiments have been performed to examine the detailed interaction between a DPSS, whose disk servers are distributed over both ATM LANs and a wide-area ATM network, and several applications. Our initial monitoring experiments focused on high-performance, highly distributed applications such as the TerraVision <=> DPSS combination, and these experiments (see [Tierney1]) illuminate the methodology, so we consider the analysis of one of these experiments that also provides a nice example of how data algorithms reflect in the measurements.

4. For more information see: http://www.metronet.com/perlinfo/perl5.html
5. For more information see: http://www.cs.dartmouth.edu/gnuplot_info.html

One of the block request semantics supported by the DPSS, and used by TerraVision, is that any time a new list of data block requests comes in, all pending requests are discarded (on a per-user and per-data set basis). This is one method by which applications can both pre-fetch data, and even be speculative about what data they will need in the near future. Pre-fetching is critical for maximum data performance because the DPSS is fairly heavily pipelined, and that pipeline must be kept full to deliver the maximum data rate. With a pre-fetch request, even if the DPSS cannot send all the requested data to the application, it is possible that the data was at least read from disk into the DPSS memory cache where it will remain available for faster retrieval (for a short time). This approach ensures that the data pipeline stays full and that disk server resources are never idle.

To date, the single most useful analysis technique for the `netlogger` data has been to construct and examine the data paths through the various system components, correlated in time with related events. These paths ("lifelines") are characterized by the time of the actions, events, or operations in all of the system components. So, for example, the DPSS actions, together with the TCP retransmit events, are

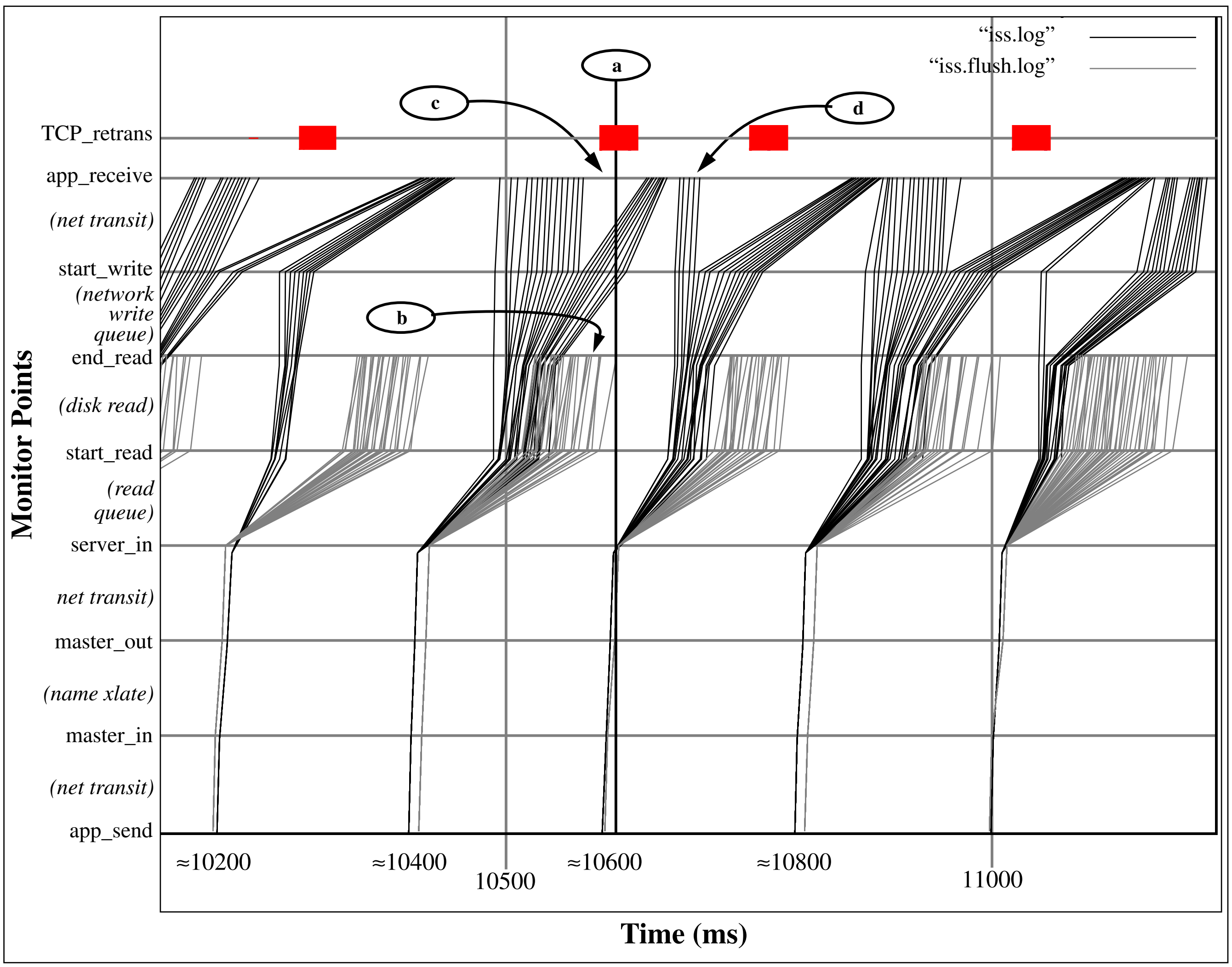

Figure 9 **One Server Test (ATM LAN, one SS-20 as server, tv_sim on DEC 3000/600)**

plotted vertically and the corresponding times horizontally in Figure 9. Each "single" line corresponds, in the case of a DPSS application, to the life of a data request, from application request of a data block, to receipt by the application. These lifelines have characteristic shapes and relationships that represent how the various algorithms in the system operates in the face of different environmental situations (e.g., network congestion).

Referring to Figure 9 and Figure 10, TerraVision sends a list of data requests every 200 ms, as shown by the nearly vertical lines starting at the *app_send* monitor points. The initial single lifelines fan out at the *server_in* monitor point as the requests resolve themselves in time and are placed in the disk read queues. So, each block request is first represented individually in the disk server read queue.

The traffic rates into and out of the name translation server (“master”) are very low, and while latencies in this service would have a significant impact on overall performance, this has not yet been an observable problem. (This is indicated in the figures showing block trace data where the difference between *app_send* and *server_in* monitor points represent the total latency to get the data requests out of the application, through the network, translated, and received by the server(s).)

The average read time is a figure of merit for the disks (seek + latency + read). Individual disk reads are a significant component of the overall performance, and most DPSSs have many disks in order to parallelize this operation.

As an indication of the sensitivity of this type of monitoring, consider the disk read signature in Figure 10. When two lifelines cross in the area between *start_read* and *end_read*, this indicates that a read from one disk was faster than a read from another disk. (This phenomenon is clearly illustrated for the server represented by the crossing solid lines in Figure 10 at A.) This faster read might be from disks with faster seek and read times (which was not the case in the experiment represented in Figure 9, as all participating systems used identical disks) or it might be due to two blocks being adjacent on disk so that no seek is required. However, given the two distinct groupings around 10ms and 20 ms, this probably characterizes the two different types of disks that were on the disk server used in this experiment.

In the experiment represented in Figure 9, the fact that most requests appear to be flushed at the *end_read* point (which actually indicates that the block is in the server send queue) rather than at *start_read* point (in the read queue) indicates that the network and/or application host is more of a bottleneck than the disks in the test configurations. If the disks were the bottleneck, then there would be more lines ending at *start_read*, because that is where the requests would be stalled when the next request arrives, and therefore where the “lifeline” is terminated by a flush.

Also in Figure 9, notice that many lifelines terminate at *end_read.* Any individual data request that is not satisfied by the disk server before the next request list arrives is flushed (discarded) from all the queues, but the data is retained in the memory cache if it was read from disk. The lifelines that started at 10,400 ms and were terminated at b, did so because the TCP write delay (probably caused by the retransmit event a little beyond 10,600ms, the time of which was not precisely obtained) at c “trapped” those blocks in the TCP write buffer. Block requests that were in the DPSS write queue when the next request list arrived (at 10,600 ms) were flushed from the queue. However, some of these blocks were re-requested in the 10,600 ms list, and these re-requests are satisfied very quickly because the data is in the disk server memory cache. This is seen in the nearly vertical lifelines at d.

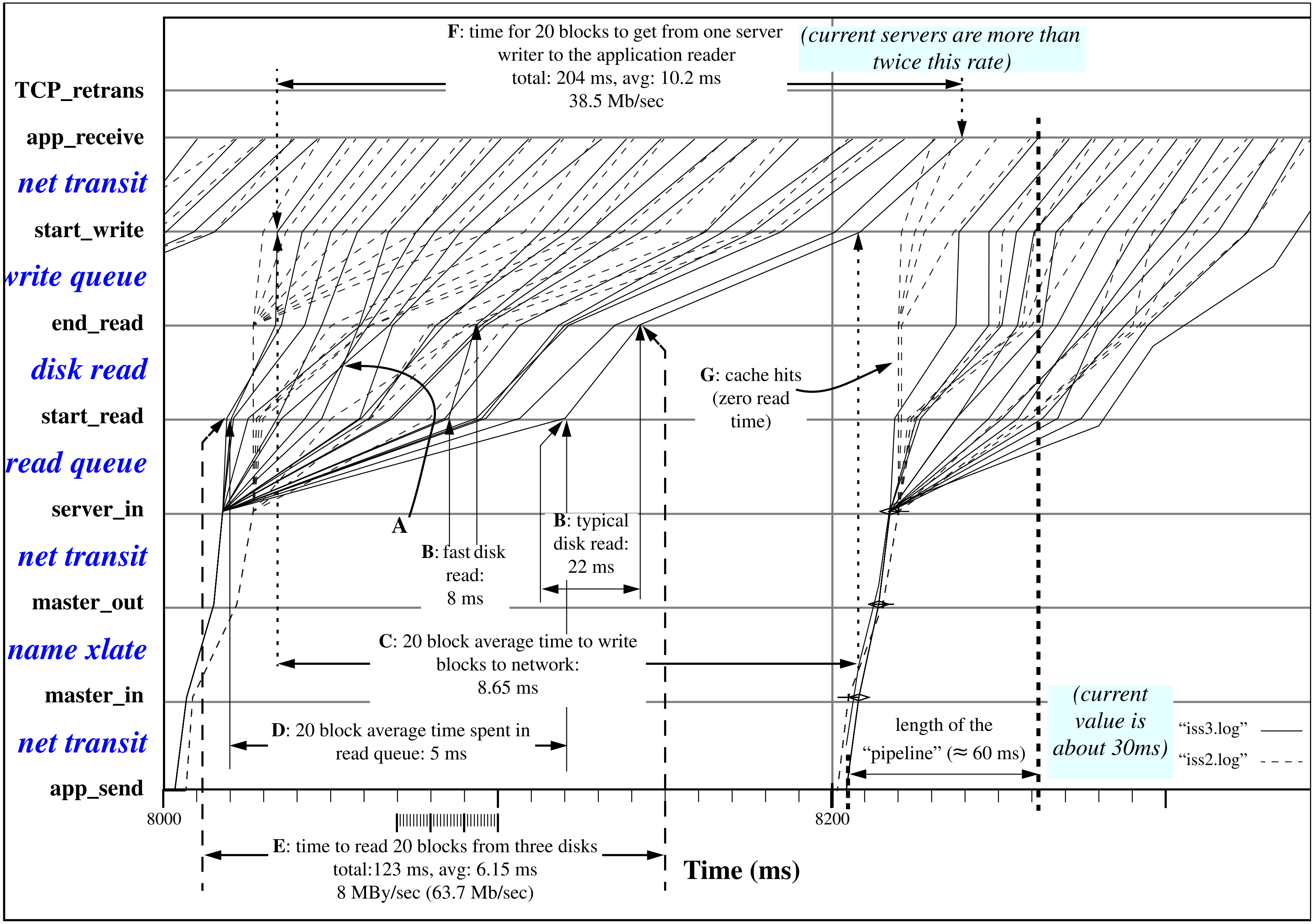

Figure 10 Detail From a Two Server, LAN, Experiment

Using these lifeline graphs it is also possible to get fairly detailed information on individual operations within the disk servers. Such detailed performance analysis is illustrated in Figure 10, for example, and shows us:

- at “B” two different characteristic disk reads (one with an 8 ms read time and one with a 22 ms read time);
- at “C” the time to cache a block and enter it into the network write queue is about 8.6 ms;
- at “D” the time to parse the incoming request list and see if the block is in the memory cache is about 5 ms;
- at “E” the overall (four disks operating in parallel) server data read rate is about 8 MB/sec;
- at “F” the actual throughput for this server while dealing with real data requests is about 39 Mb/s (the throughput is receiver limited in this case, and the “unconstrained” throughput is about three times that number);
- at “G”, there are two cache hits (blocks found in memory) as a result of previously requested, but not sent, data being requested.

Figure 11 illustrates “correct” operation of multiple servers. This LAN-based, two-server experiment

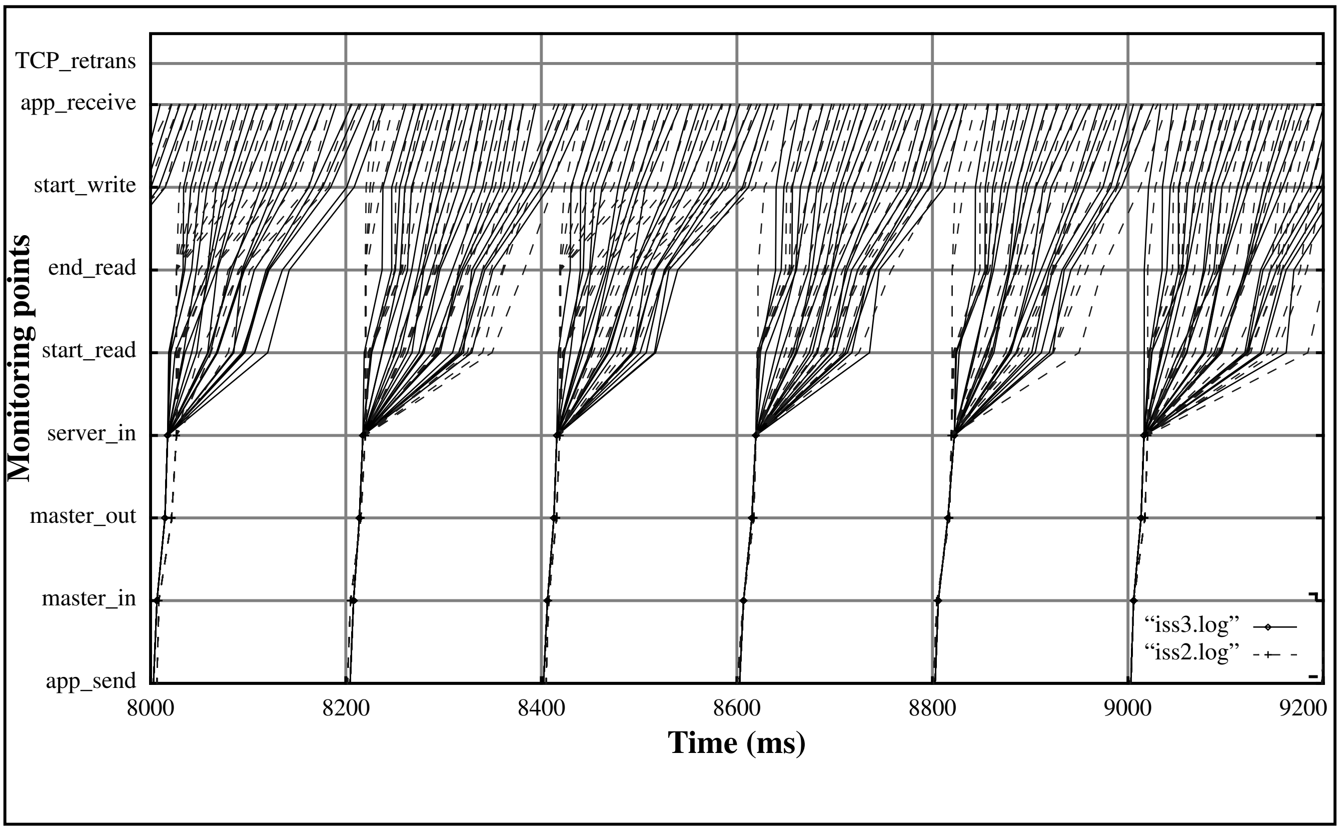

Figure 11 **Two Server Test (ATM LAN, two SS-20s as servers, tv_sim on DEC 3000/600)**

shows the interaction of lifelines for blocks from different servers, and a case where the independent servers are behaving almost “perfectly”: there are very regular block delivery patterns that alternate almost one-for-one between servers. Every lifeline termination at `app_receive` line indicates 64 kBy of data delivered to the application. Note that in many cases the two DPSS servers delivered data almost simultaneously, and so the lines overlap.

6.0 STAF Experiment

The physics data handling experiment is divided into several parts. In the first part we are trying to model the analysis (phase 2) environment. In a production configuration, it is expected that several hundred processors will operate in parallel, each on a single experiment "event" (1 - 10 MBy of data). The data archive query interface will select a set of tapes and records on the tertiary storage system. The current model has the storage system interface being 3-5 high-speed tape drives operating in parallel (each of which delivers about 9 MBy/s). This data is written to a DPSS from which STAF accesses the data.

Performance for the STAF application accessing data in the cache was measured using a DPSS configuration of three disk servers with two disks per server. STAF requests that data be loaded into memory-resident objects that are defined by serialized (XDR-encoded) records. Deserializing, in turn, causes data requests to be made through the DPSS file semantics interface which collects blocks from the DPSS servers, buffers them, and provides access to the buffer. The throughput rates are measured as data is delivered to the STAF physics analysis modules, a path that includes translating the data to the appropriate machine format and structuring it in memory (both of which are very fast operations). With a single instance of STAF running on a Sun E4000 system with an OC-12 (622 Mbit/s) ATM interface, a data rate of 22 megabytes/s is achieved for reading data, and 30 megabytes/s for writing data.

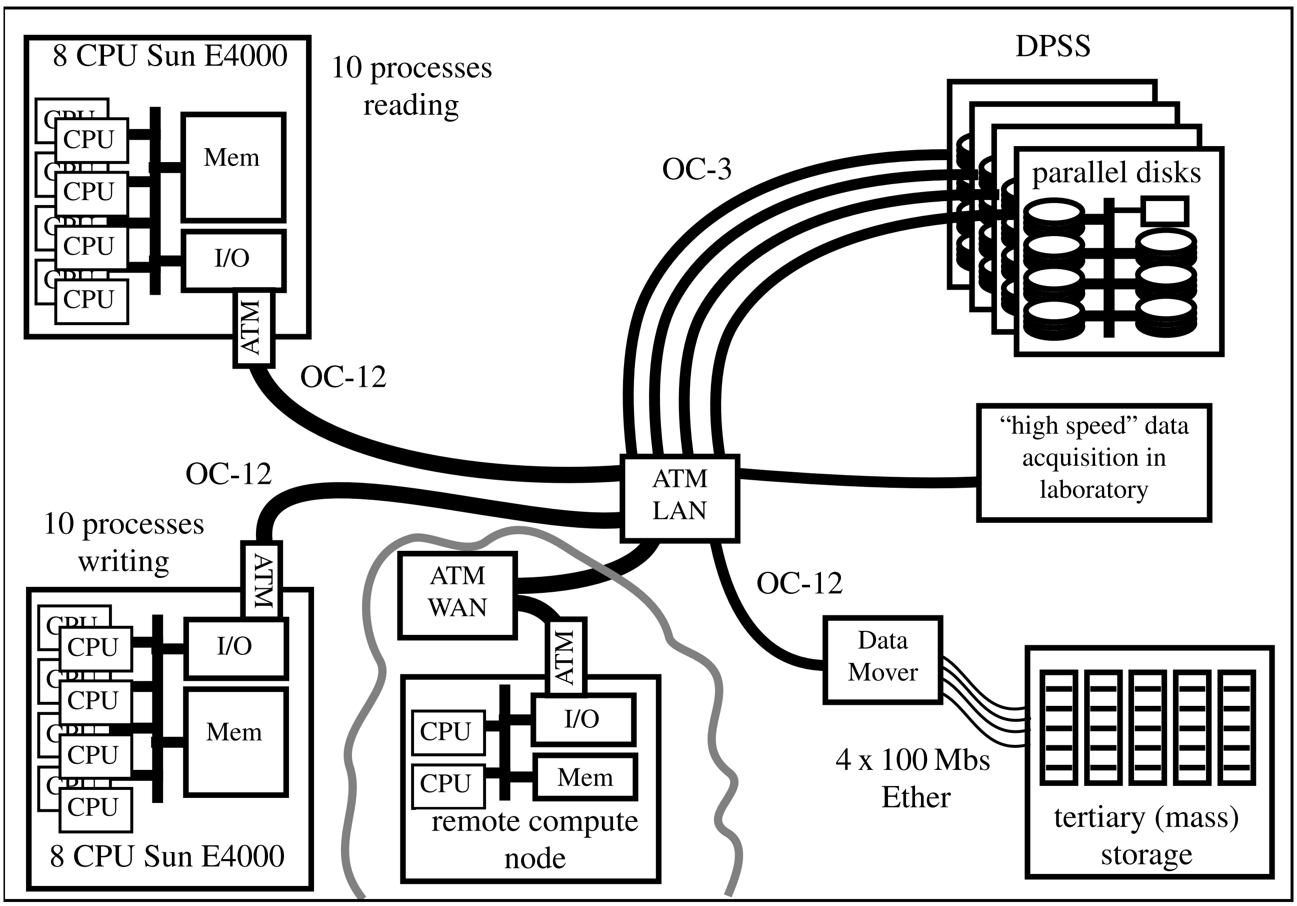

Figure 12 **Experiment setup.**

Using the experiment setup of Figure 12, and running 10 instances of the application simultaneously, results in the same aggregate throughput. This is illustrated in Figure 13 where each horizontal set of

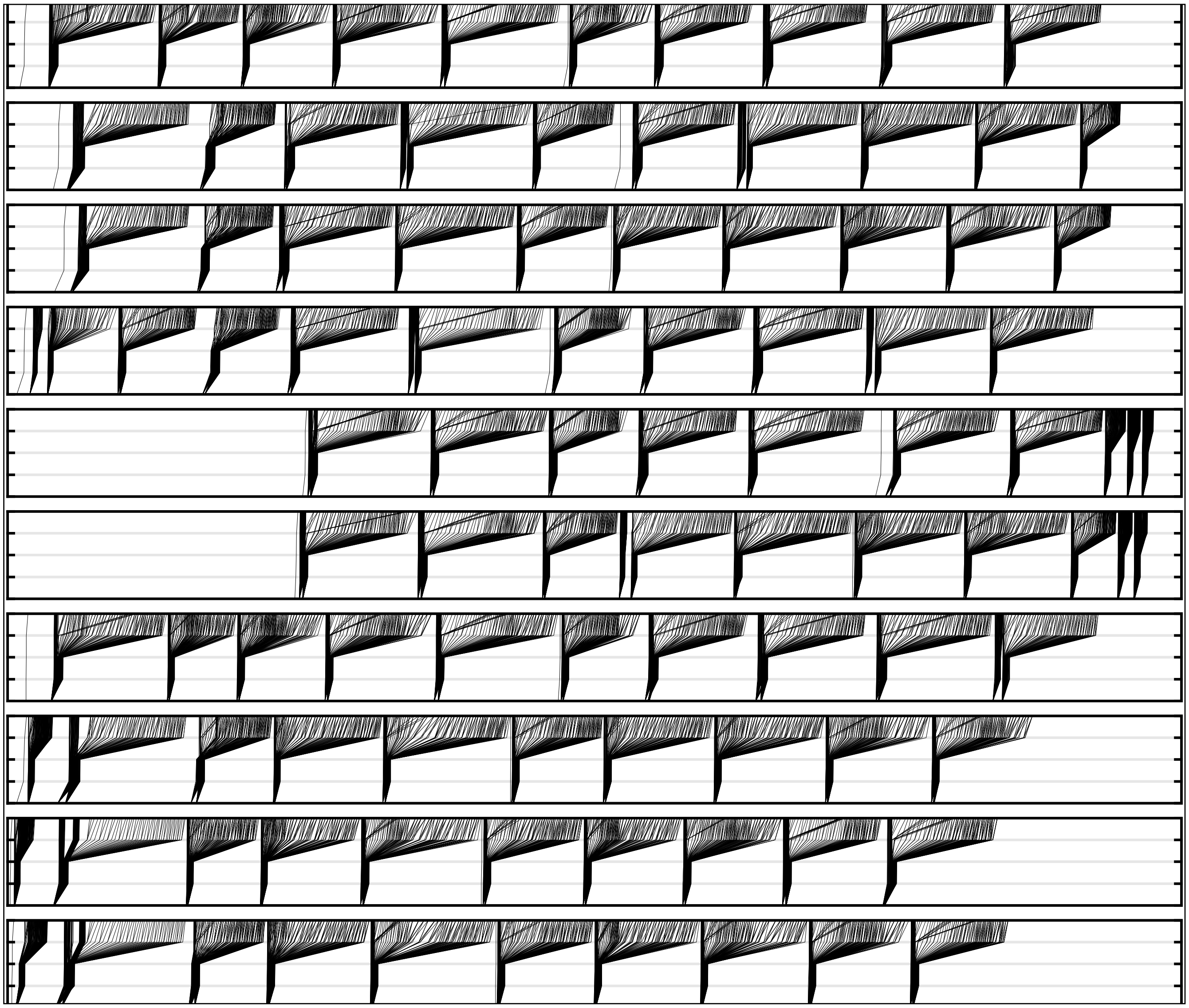

Figure 13 **Three DPSS servers delivering 10 gigabytes of data to 10 parallel applications.**

lifeline traces represents the data delivered to the STAF application (each request is for 10 megabytes of data, so each horizontal set represents a gigabyte of data delivered to the application). Moving vertically in the graph shows data being delivered to the individual instances of STAF. The traces are, of course, time correlated using the techniques described above, so that the whole graph represents the behavior of the overall system of ten applications operating in parallel accessing the same three disk servers to get data. Ignoring the application start-up and termination anomalies (which have reasonable explanations that are unrelated to the current discussion) the delivery of data to the applications is relatively smooth and fair. The sum of all lifelines in the graph show 10 gigabytes of data being delivered over a period of about 500 seconds, or about 20 megabytes/sec aggregate data delivery.

7.0 An Experiment in High-Speed, Wide Area Distributed Data Handling With Many Clients

Reviewing the framework described in Section 3.0, we see that the first-level data from the instrument system flows from the instrument, through the network, to a receiving cache (See Figure 1a). The DPSS is used for this cache, the servers of which may be located at one site, or (as in the MAGIC testbed) distributed across many sites. The first level of processing can be done directly out of the cache. The first-level data is also moved from cache to tertiary storage, and the results of this processing can be used to optimize data placement on tertiary storage. (As noted above, the write rates into the DPSS should be sufficient for this approach.)

It is our contention that in the time frame of the next generation of physics experiments (2000-2005 AD), wide area networks will be easily capable of distributing the instrument output data stream anywhere in the US (and probably to Europe).

Distributing experimental data has two potential advantages. First, the first-level processing (which is easily parallelized) can be done using resources at the collaborators' sites (each experiment typically involves 5-10 major institutions). Second, large tertiary storage systems exhibit substantial economies of scale, and so using a large tertiary storage system at, say, a supercomputer center, should result in more economical storage, better access (because of much larger near-line systems - e.g., lots of tape robots) and better media management, especially in the long term, than can be obtained in local systems.

We are testing the wide-area aspects of the framework in the following experiment. A DPSS and a computing cluster are located at Lawrence Berkeley National Laboratory (LBNL). The NTON network testbed [NTONC] that connects Berkeley and Lawrence Livermore National Laboratory (LLNL) can be configured for a 2000 km, OC-12, path (by using the 16 OC-12 SONET paths that make up the 400 km underlying network). A high-speed workstation that has a collection of STAR events stored on its disks is located at LLNL and connected to NTON. This workstation will emit events at the same rate as the STAR detector, and this data will be cached on the DPSS at Berkeley. The computing cluster will process data out of the cache (doing "reconstruction") and those results will be written back to the cache. A storage manager will migrate data to tertiary storage (or a "null" system that has the same throughput characteristics, as there is little point in actually storing this synthetic data).

Except that the computing cluster will not have sufficient compute capacity to do all of the required processing at the operating data rates, this scenario - once it works as expected - should demonstrate the feasibility of wide area processing of this type of real-time data. The experiment is illustrated in Figure 14.

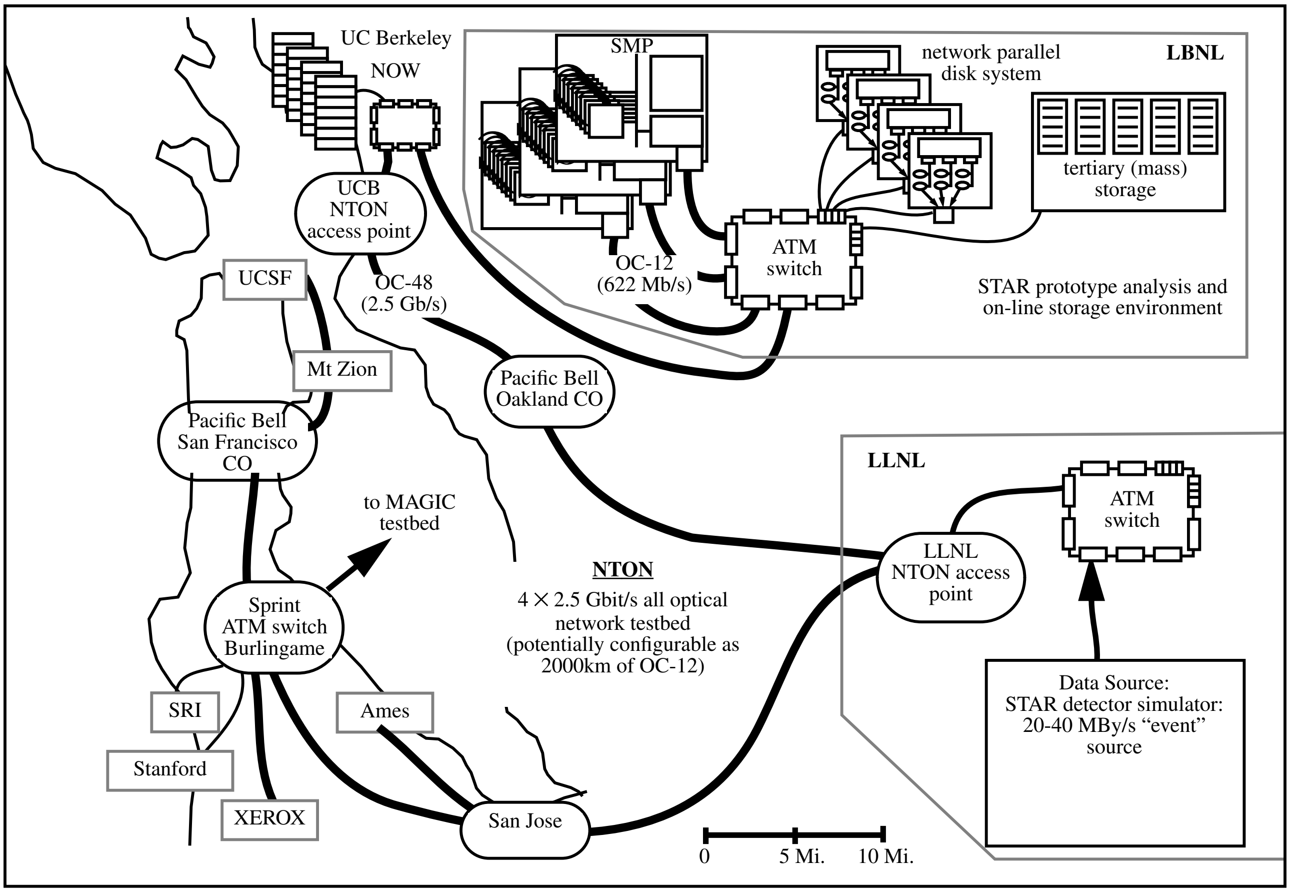

Figure 14 **The Distributed Processing Experiment**

8.0 Conclusions

The experiments described here are work-in-progress. The use of the DPSS as cache has demonstrated the required performance, functionality, and level of abstraction (the common interface concept), but a complete demonstration of scalability requires running hundreds of analysis processes. The wide area, high data-rate experiment configuration is nearly complete, and results are expected in the near future. We expect that this experiment will be successful, because several precursors have been carried out in the MAGIC testbed. However, experience has also shown that every significant increase in throughput and/or scale raises a new set of issues. More results will reported at http://www-itg.lbl.gov/DPSS as they are obtained.

9.0 References

DIGLIB "Real-Time Generation and Cataloguing of Large Data-Objects in Widely Distributed Environments", W.Johnston, Jin G., C. Larsen, J. Lee, G. Hoo, M. Thompson, B. Tierney, J. Terdiman. To be published in International Journal of Digital Libraries - Special Issue on "Digital Libraries in Medicine". Available at http://www-itg.lbl.gov/WALDO.

DPSS "The Distributed-Parallel Storage System (DPSS)". See http://www-itg.lbl.gov/DPSS.

Greiman97H Greiman, W., W. E. Johnston, C. McParland, D. Olson, B. Tierney, C. Tull, "High-Speed Distributed Data Handling for HENP". International Conference on Computing in

High Energy Physics, Berlin, Germany, April, 1997. Also available at http://www-itg.lbl.gov/STAR.

Johnston95V W. Johnston, and D. Agarwal, “The Virtual Laboratory: Using Networks to Enable Widely Distributed Collaboratory Science” A NSF Workshop Virtual Laboratory whitepaper. (See http://www-itg.lbl.gov/~johnston/Virtual.Labs.html)

Johnston97 “Security Architectures for Large-Scale Remote Collaboratory Environments: A Use-Condition Centered Approach to Authenticated Global Capabilities”. W. Johnston and C. Larsen, (draft at http://www-itg.lbl.gov/security/publications.html)

Lau94 “TerraVision: a Terrain Visualization System”. S. Lau, Y. Leclerc, Technical Note 540, SRI International, Menlo Park, CA, Mar. 1994. Also see: http://www.ai.sri.com/~magic/terravision.html.

MAGIC “The MAGIC Gigabit Network”, See: http://www.magic.net/

Mills “Simple Network Time Protocol (SNTP)”. D. Mills, RFC 1361, University of Delaware, August 1992.

NTONC “National Transparent Optical Network Consortium”. See http://www.ntonc.org. (NTONC is a program of collaborative research, deployment and demonstration of an all-optical open testbed communications network.)

Olson “HENP Data Analysis Requirements.” D. Olson, C. McParland, W. Johnston, and W. Greiman. http://www-rnc.lbl.gov/computing/ldrd_fy97/html/star_reqs.htm

STAR1 “Relativistic Nuclear Collisions Program”, H.G. Ritter. http://www-library.lbl.gov/docs/LBNL/397/64/Overviews/RNC.html

STAR2 “High Speed Distributed Data Handling for HENP”, W. Greiman, W. E. Johnston, C. McParland, D. Olson, B. Tierney, C. Tull. http://www-rnc.lbl.gov/computing/ldrd_fy97/henpdata.htm

Tull97 “The STAR Analysis Framework Component Software in a Real-World Physics Experiment”. C.Tull, W.Greiman, D.Olson, D.Prindle, H.Ward, International Conference on Computing in High Energy Physics, Berlin, Germany, April, 1997.

Tierney1 “Performance Analysis in High-Speed Wide Area ATM Networks: Top-to-bottom end-to-end Monitoring”, B. Tierney, W. Johnston, J. Lee, G. Hoo. IEEE Networking, May 1996. An updated version of this paper is available at http://www-itg.lbl.gov/DPSS/papers.html.

Tierney2 “NetLogger Toolkit: A Methodology and Set of Tools for Network and Distributed System Performance Analysis”, B. Tierney. Available at http://www-itg.lbl.gov/DPSS/logging.

Where “WHERE: Wide-area Helpers Enabling Reliable Environments” http://www-itg.lbl.gov/DPSS/agents/WHERE.html