# System Issues in Implementing High Speed Distributed Parallel Storage Systems

***Brian Tierney (bltierney@lbl.gov),***
***Bill Johnston[1] (wejohnston@lbl.gov),***
***Hanan Herzog, Gary Hoo, Guojun Jin, Jason Lee***

*Imaging Technologies Group*
*Lawrence Berkeley Laboratory[2]*
*Berkeley, CA 94720*

## Abstract

In this paper we describe several aspects of implementing a high speed network-based distributed application. We describe the design and implementation of a distributed parallel storage system that uses high speed ATM networks as a key element of the architecture. The architecture provides what amounts to a collection of network-based disk block servers, and an associated name server that provides some file system functionality. The implementation approach is that of user level software that runs on UNIX workstations. Both the architecture and the implementation are intended to provide for easy and economical scalability in both performance and capacity. We describe the software architecture, the implementation and operating system overhead issues, and our experiences with this approach in an IP-over-ATM WAN. Although most of the paper is specific to a distributed parallel data server, we believe many of the issues we encountered are generally applicable to any high speed network-based application.

## 1.0 Introduction

In recent years many technological advances have made distributed multimedia servers a reality, and people now desire to put "on-line" large amounts of information, including images, videos, and hypermedia databases. Increasingly, there are applications that demand high-bandwidth access to this data, either in single user streams (e.g., large image browsing, uncompressible scientific and medical video, and multiple coordinated multimedia streams) or, more commonly, in aggregate for multiple users. Here we describe a network distributed data server, called the Image Server System (ISS). The ISS is being used to supply data to a terrain visualization application that requires 300-400 Mbits/s of data to provide a realistic visualization. Both the ISS and the application have been developed in the context of the MAGIC Gigabit testbed ([5] and [11]).

---

1. Correspondence should be directed to Bill Johnston, Lawrence Berkeley Laboratory, Bldg. 50B - 2239, Berkeley, CA, 94720. Tel: 510-486-5014, fax: 510-486-6363).

2. This work is jointly supported by ARPA - CSTO, and by the U. S. Dept. of Energy, Energy Research Division, Office of Scientific Computing, under contract DE-AC03-76SF00098 with the University of California. This document is LBL report LBL-35775.

To address a point frequently raised, compression is not practical in the case of terrain visualization because the cost of decompression is prohibitive in the absence of suitable hardware implementations.

To comment briefly on the relevant technologies, current disk technology delivers about 4 Mbytes/s (32 Mbits/s), a rate that has improved at about 7% each year since 1980 [10], and there is reason to believe that it will be some time before a single disk is capable of delivering streams at the rates needed for the application mentioned. While RAID [10] and other parallel disk array technologies can deliver higher throughput, they are still relatively expensive, and do not scale well (at least economically), especially in the scenario of multiple network-distributed users (where we assume that the sources of data, as well as the multiple users, will be widely distributed). Wide area Asynchronous Transfer Mode (ATM) networks are being built on a SONET infrastructure, which has the characteristic that bandwidth aggregates upward (contrary to our current network hierarchy, where the slowest networks are at the top of the hierarchy). This characteristic makes it possible to use the network to aggregate many low-speed sources into a single high-speed stream.

The approach described here differs in many ways from RAID, and should not be confused with it. RAID uses a particular data layout and redundancy strategy to secure reliable data storage and parallel disk operation. Our approach, while using parallel disks and servers, imposes no particular layout strategy (in fact, this is deliberately left to the application domain), and is implemented entirely in software (though the data redundancy idea of RAID might be applied across servers).

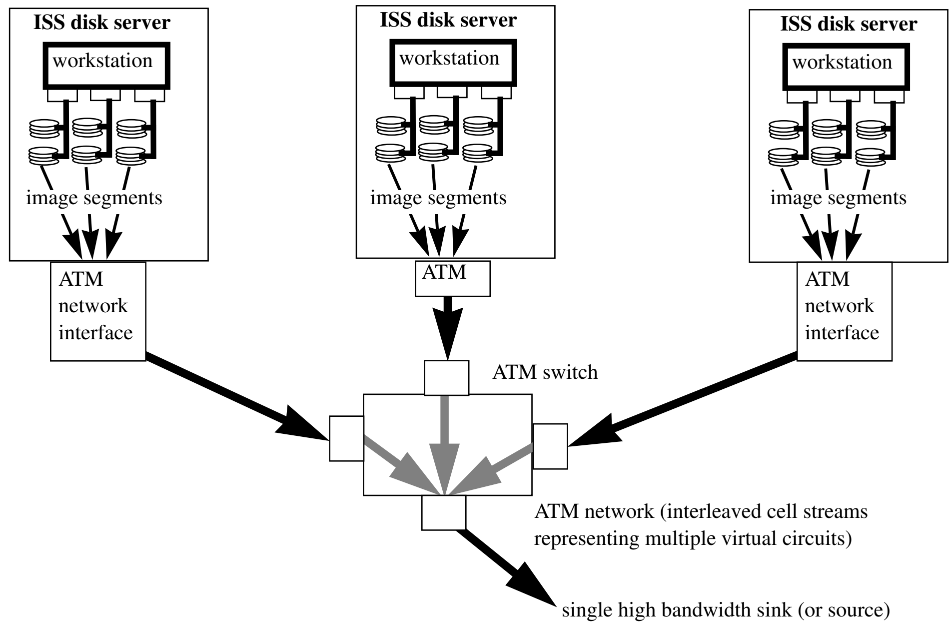
At the present state of development and experience, the system we describe is used primarily as a large, fast "cache" for image or multimedia data. In our approach, reliability with respect to data corruption is provided by the usual OS and disk controller-level mechanisms, and reliability of the overall system is a function of user-level strategies of data replication. The data of interest (tens to hundreds of GBytes) is typically loaded from archival tertiary storage, or written into the system from live video sources. (In the latter case, the data is also archived to bulk storage in real-time.)

The Image Server System is an example of distributed parallel data storage technology. It is a multimedia file server that is distributed across a wide area network to supply data to applications located anywhere in the network. This system is not a general purpose, distributed file system in that the data units ("files") are assumed to be large and relatively static. The approach allows data organization to be determined by the user as a function of data type and access patterns. For most applications, the goal of data organization is that data should be declustered across both disks and servers (that is, dispersed in such a way that as many system elements as possible can operate simultaneously to satisfy a given request). This declustering allows a large collection of disks to seek in parallel, and allows all servers to send the resulting data to the application in parallel, enabling the ISS to perform as a high-speed image server. Architecturally, the ISS is a distributed, parallel mass storage system that uses UNIX workstation technology to provide a low-cost, scalable implementation. The data transport is via TCP/IP or RTP/IP[13]. The scalability arises from the high degree of independence among the servers: both performance and capacity may be increased, in essentially linear fashion, by adding servers. (Ultimately this is limited by the parallelism inherent in the data.) The general idea is illustrated in Figure 1 (Data Streams Aggregated by ATM Switches).

In our prototypes, the typical ISS consists of several (four - five) UNIX workstations (e.g. Sun SPARCStation, DEC Alpha, SGI Indigo, etc.), each with several (four - six) fast-SCSI disks on multiple (two - three) SCSI host adaptors. Each workstation is also equipped with an ATM network interface. An ISS configuration such as this can deliver an aggregated data stream to an application at about 400 Mbits/s (50 Mbytes/s) using these relatively low-cost, "off the shelf" components by exploiting the parallelism provided by approximately five hosts, twenty disks, ten SCSI host adaptors, and five network interfaces.

## 2.0 Related Work

In some respects, the ISS resembles the Zebra network file system, developed by John H. Hartman and John K. Ousterhout at the University of California, Berkeley [4]. Both the ISS and Zebra can separate their file access and management activities across several hosts on a network. Both try to maintain the availability of the system as a whole by building in some redundancy, allowing for the possibility that a disk or host might



**Parallel Data and Server Architecture Approach to the Image Server System**

**Figure 1 Data Streams Aggregated by ATM Switches**

be unavailable at a critical time. The goal of both is to increase data throughput despite the current limits on both disk and host throughput.

However, the ISS and the Zebra network file system differ in the fundamental nature of the tasks they perform. Zebra is intended to provide traditional file system functionality, ensuring the consistency and correctness of a file system whose contents are changing from moment to moment. The ISS, on the other hand, tries to provide extremely high-speed, high-throughput access to a relatively static set of data. It is optimized to retrieve data, requiring only minimum overhead to verify data correctness and no overhead to compensate for corrupted data.

There are other research groups working on solving problems related to distributed storage and fast multimedia data retrieval. For example, Ghandeharizadeh, Ramos, et al., at USC are working on declustering methods for multimedia data [3], and Rowe, et al., at UCB are working on a continuous media player based on the MPEG standard [12].

## 3.0 Applications

There are several target applications for the initial implementation of the ISS. These applications fall into two categories: image servers and multimedia / video file servers.

### 3.1 Image Server

The initial use of the ISS is to provide data to a terrain visualization application in the MAGIC testbed[3]. This application, known as TerraVision [5], allows a user to navigate through and over a high resolution landscape represented by digital aerial images and elevation models. TerraVision is of interest to the U.S. Army because of its ability to let a commander "see" the battlefield. TerraVision is very different from a typical "flight simulator"-like program in that it uses large quantities of high resolution aerial imagery for the visualization instead of simulated terrain. TerraVision requires large amounts of data, transferred at both bursty and steady rates. The ISS is used to supply image data at hundreds of Mbits/s rates to TerraVision. We are not using any data compression for this application because the bandwidth requirements for TerraVision are such that real-time decompression is not possible without using special purpose hardware.

In the case of a large-image browsing application like TerraVision, the strategy for using the ISS is straightforward: the image is tiled (broken into smaller, equal-sized pieces), and the tiles are scattered across the disks and hosts of the ISS. The order of images delivered to the application is determined by the application predicting a "path" through the image (landscape), and requesting the tiles needed to supply a view along the path. The actual delivery order is a function of how quickly a given server can read the tiles from disk and send them over the network. Tiles will be delivered in roughly the requested order, but small variations from the requested order will occur. These variations must be accommodated by buffering or other strategies in the client application.

This approach can supply data to any sort of large-image browsing application, including applications for displaying large aerial-photo landscapes, satellite images, X-ray images, scanning microscope images, and so forth.

### 3.2 Video Server

Examples of video server applications include video players, video editors, and multimedia document browsers. A video server might contain several types of stream-like data, including conventional video, compressed video, variable time base, multimedia hypertext, interactive video, and others. Several users may be accessing the same video data at the same time, but may be at different frames in the stream. This affects many factors in an image server system, including the layout of the data on disks. Since there is a predictable, sequential pattern to the requests for data, the data would be placed on disk in a like manner. Large commercial concerns such as Time Warner and U.S. West are building large-scale commercial video servers such as the Time Warner / Silicon Graphics video server [6]. Our approach may address a wider scale, as well as a greater diversity, of data organization strategies so as to serve the diverse needs of schools, research institutions, and hospitals for video-image servers in support of various educational and research-oriented digital libraries.

### 3.3 Application to ISS Interface

Application access to the ISS is through a client-side library that accepts requests for data, and returns data to the application. The client library obtains from the ISS Master a list of ISS Disk Servers (q.v.) that have data for the area of interest. The client library establishes connections to all ISS Disk Servers containing that data set. The application specifies the location of a memory buffer to receive incoming data.

The current implementation provides access to large images, in which the unit of data is a tile. The application requests data in terms of lists of tiles, and the tiles sent by the ISS servers are placed into the application's buffer. (See Figure 2 (Application Architecture).)

---

3. MAGIC (Multidimensional Applications and Gigabit Internetwork Consortium) is a gigabit network testbed that was established in June 1992 by the U. S. Government's Advanced Research Projects Agency (ARPA)[11]. MAGIC's charter is to develop a high-speed, wide-area networking testbed that will demonstrate interactive exchange of data at gigabit-per-second rates among multiple distributed servers and clients using a terrain visualization application.

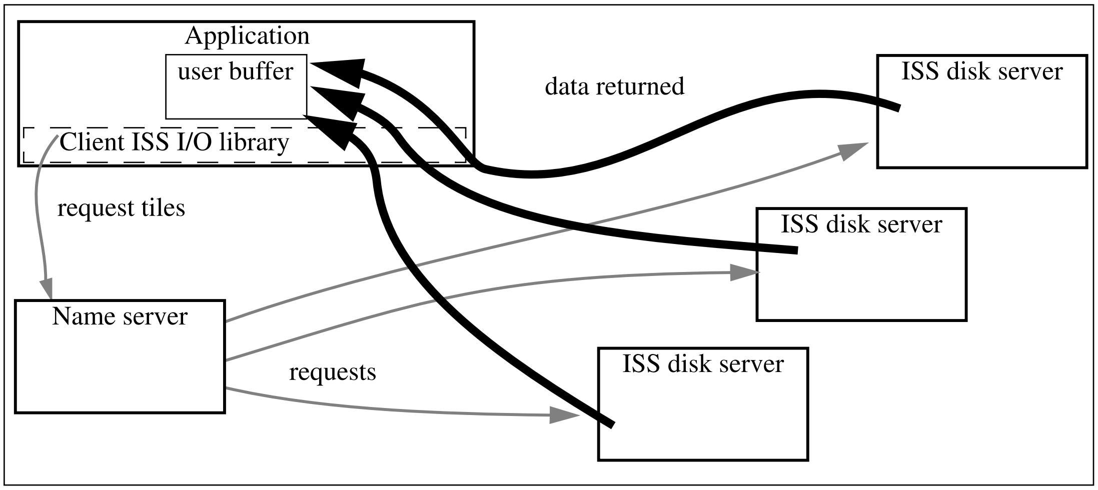



**Figure 2 Application Architecture**

### 3.4 Data Prediction

Data prediction is important to ensure that the ISS is utilized as efficiently as possible. By always requesting more tiles than the ISS can actually deliver before the next tile request, we ensure that no component of the ISS is ever idle. For example, if most of a request list's images were on one server, the other servers could still be reading and sending or caching images that may be needed in the future, instead of idly waiting. The point of the path prediction is to provide a rational basis for pre-requesting tiles.

As an example of path prediction, consider an interactive video database with a finite number of possible branch points. (A "branch point" occurs where a user might select one of several possible play clips.) As a branch point is approached by the player, the predictor (without knowledge of which branch will be taken) will start requesting images (frames) along both branches. These images are cached first at the disk servers, then at the receiving application. As soon as a branch is chosen, the predictor ceases to send requests for images from the other branch. Any cached but unsent images are flushed as better predictions fill the cache.

For MAGIC's TerraVision, prediction is based on geometric characteristics of the path being followed, the limitations of the mode of simulated transport (that is, walking, driving, flying, etc.), the intended destination, and so on. The prediction results in a priority ordered list of tile requests being sent to the ISS. The ISS has no knowledge of the prediction strategy (or even if one has been used).

The client will keep asking for an image until it shows up, or until it is no longer needed (e.g. the application may have "passed" the region of landscape that involves the image that was requested, but never received.) Applications will have different strategies to deal with images that do not arrive in time. For example, TerraVision keeps a local low resolution data set to fill in for missing tiles.

Prediction is transparent to the ISS, and is manifested only in the order and priority of images in the request list. The prediction algorithm is mostly a function of the client application, and typically runs on the client. Prediction could also be done by a third-party application. The aforementioned interactive video database, for example, might use a third-party application for prediction.

## 4.0 Design

### 4.1 Goals

The following are some guidelines we have followed in designing the ISS:

- The ISS should be capable of being geographically distributed. In a future environment of large-scale, mesh-connected national networks, network-distributed storage should be capable of providing an uninterruptable stream of data, in much the same way that a power grid is resilient in the face of source failures, and tolerant of peak demands because of multiple sources multiply interconnected.
- The ISS approach should be scalable in all dimensions, including data set size, number of users, number of server sites, and aggregate data delivery speed.
- The ISS should deliver coherent image streams to an application, given that the individual images that make up the stream are scattered (by design) all over the network. In this case, "coherent" means "in the order needed by the application." No one disk server will ever be capable of delivering the entire stream. The network is the *server.*
- The ISS should be affordable. While something like a HIPPI-based RAID device might be able to provide the functionality of the ISS, this sort of device is very expensive, is not scalable, and is a single point of failure.

## 4.2 Research Issues

The design goals present several issues that need to be addressed. These include:

- How to store and retrieve image data at gigabit speeds using a storage system whose servers are widely distributed;
- How to place data blocks (tiles) such that image data will be distributed across many storage servers in a way that ensures parallel operation;
- How to handle high-speed IP transport over ATM networks to provide the parallel data paths needed to aggregate medium-speed disk servers into a logically integrated, high-speed image storage server. (Although ATM will probably become the Ethernet of the future, end-to-end networks will be heterogeneous for a long time to come, necessitating the use of an internetwork protocol, of which IP is the clear choice);
- Assessing how an ATM network will behave (or misbehave) under the conditions of multiple, coordinated, parallel data streams.

## 4.3 Approach: A Distributed, Parallel Server

The ISS design is based on the use of multiple low-cost, medium-speed disk servers which use the network to aggregate server output into a single high-bandwidth stream. To achieve high performance we exploit all possible levels of parallelism, including that available at the level of the disks, controllers, processors/memory banks, servers, and the network. Proper data placement strategy is also key to exploiting system parallelism. For placement of image tile data, an application-related program declusters tiles so that all ISS disks are evenly accessed by tile requests, but clusters tiles that are on the same disk[1]. This strategy is a function of both the data structure (tiled images) and the geometry of the access (e.g., paths through the landscape). Currently we are working on extending these methods to handle video-like data.

At the individual server level, the approach is that of a collection of disk managers that move requested data to memory cache. Depending on the nature of the data and its organization, the disk managers may have a strategy for moving other closely located and related data from disk to memory as a side effect of a particular data request. However, in general, we have tried to keep the implementation of data prediction (determining what data will be needed in the near future) separate from the basic data moving function of the server. Prediction might be done by the application, or it might be done be a third party that understands the data usage patterns. In any event, the server sees only lists of requested blocks.

As explained below, the dominant bottlenecks for this type of application in a typical UNIX workstation are, first, memory copy speed, and second, network access speed. For these reasons, an important design criterion is to use as few memory copies as possible, and to keep the network interface operating at full bandwidth all the time. Our implementation uses only three copies to get data from disk to network, so maximum server throughput is about (mem_copy_speed / 3).

Another important aspect of the design is that all components are instrumented for timing and data flow monitoring in order to characterize the ISS implementation and the network performance. To do this, all communications between ISS components are timestamped. We are using GPS (Global Positioning System) receivers and NTP (Network Time Protocol) [8] [9] to synchronize the clocks of all ISS servers and of the client application in order to accurately measure network throughput and latency.

## 5.0 ISS Architecture and Implementation

The following is a brief overview of a typical ISS operation. A data set must first be loaded across a given set of ISS hosts and disks, and a table containing disk/offset locations for each block of data is stored on each host. The application sends requests for data (images, video, sound, etc.) to the Name Server process on each Disk Server host, which does a lookup to determine the location (server - disk - offset) of the requested data. If the data is not stored on that host, the request is discarded with the assumption that another host will handle it; otherwise the list of locations is passed to the ISS Disk Server. Each Disk Server then checks to see if the data is already in its cache, and if not, fetches the data from disk and transfers it to the cache. Once the data is in the cache, it is sent to the requesting application.

In the following sections, we describe the basic software modules, their functions, how they relate to each other, and some of the terms and models that were used in the design of the ISS. Figure 3 (ISS Server Archi-

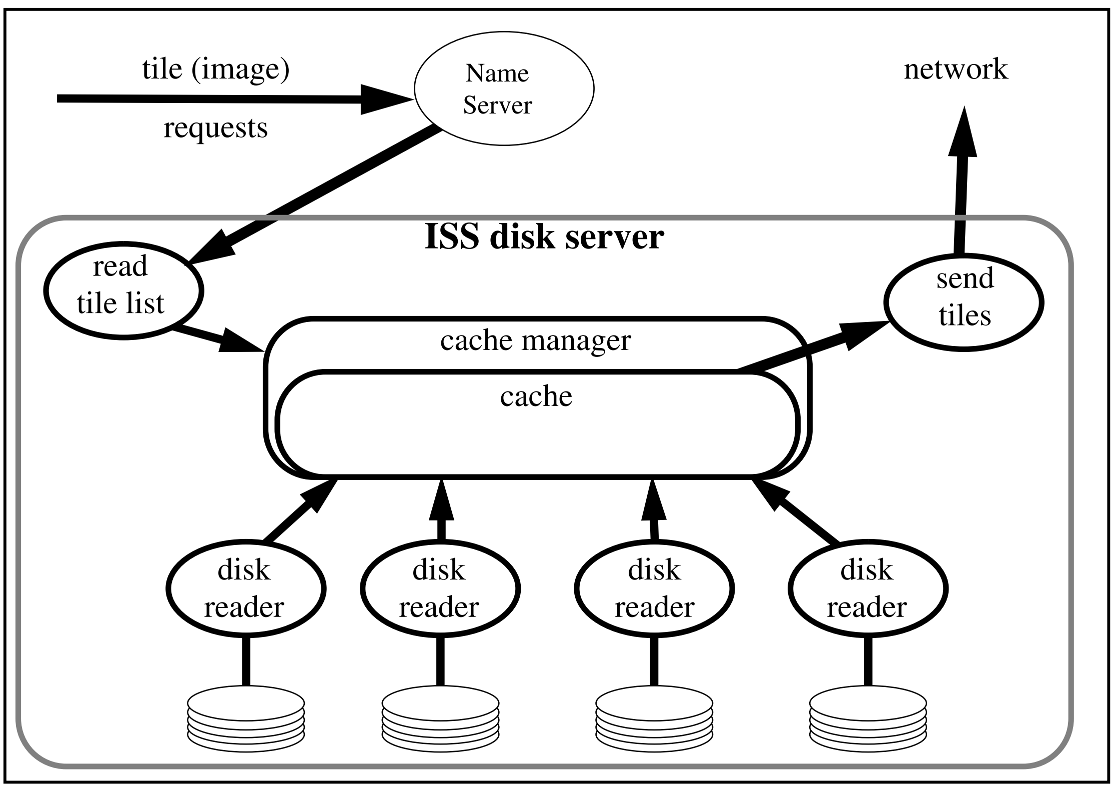



**Figure 3 ISS Server Architecture**

tecture) shows how the components of the ISS relate to each other.

### 5.1 ISS Master

The ISS Master process is responsible for application-to-ISS startup communication, Disk Server process initialization, performance monitoring, and coordination between multiple ISS Disk Servers. This includes the ability to collect performance and usage statistics of all ISS components. In the future, we plan to extend the functionality of the Master to dynamically reconfigure ISS Disk Server usage to avoid network or ISS Disk Server bottlenecks.

### 5.2 Name Server

The Name Server listens for tile request lists from the application. After receiving a list, the Name Server does a table lookup to determine where the data is located (i.e. which server, which disk, and the disk offset). The Name Server then passes this list to the ISS Disk Server.

### 5.3 ISS Disk Server

There is one ISS Disk Server process for each ISS host. It is responsible for all ISS memory buffer and request list management on that host. The Disk Server receives image requests from the Master process, and determines if the image is already in its buffer cache. If it is already in the buffer cache (which is kept entirely in available memory), the request is added to the “to send” list. Otherwise, it is added to a “to read” list. Tile requests that have not been satisfied by the time the next list from the Master process arrives are “flushed” (discarded) from the lists. All requests that haven’t been either read off of disk or written to the network interface are removed from all request lists, and any memory buffers waiting to be written are returned to the hash table. Note that if a tile read has completed, but the tile has not yet been sent to the network, the data stays in the cache, so that if that tile is in the next request list it will be sent first. Those buffers that were waiting to be filled with data from the disk are put at the head of an LRU (Least Recently Used) list so they may be used for requests in the newly arrived list. The Disk Server process also periodically sends status information to the Master.

ISS buffer management is very similar to that of the UNIX operating system, and many of the ideas for lists, hashing, and the format of the headers have been adopted from UNIX for use within the ISS [7]. A buffer can be freed from the hash table in one of two ways. If a buffer was allocated to a list (read/send) and that list was flushed, the buffer is returned to the head of the LRU list so that it is the next buffer to be reused. A buffer may also naturally progress through the LRU list until it has reached the end of the list, at which time it is recycled.

The Disk Server handles three request priority levels:

- high: send first, with an implicit priority given by order within the list.
- medium: send if there is time.
- low: fetch into the cache if there is time, but don't send.

The priority of a particular request is set by the requesting application. The application’s prediction algorithm should use these priority levels to keep the ISS fully utilized at all times without requesting more data than the application can process. For example, the application should send low priority requests to pull data into the ISS cache that the application will need in the near future; this data is not sent to the application until the application is ready. Another example is an application that plays back a movie with a sound track, where audio might be high priority requests, and video medium priority requests.

### 5.4 ISS Reader

The ISS Reader process reads data off of disk and puts it into the buffer cache that is managed by the Disk Server process. There is one Reader per physical disk. This process continually checks for requests in the “to read” list, starts a read operation on that disk if a request is pending, then waits for the data to be read. Once the data is read off of disk the request is moved into the list of data that is to be written out. There are two distinct lists of data that are to be written out, one for each of the high and medium priority levels described above.

### 5.5 ISS Sender

The ISS Sender process sends all data in the "to send" list out to the application that requested it. There is one sender per network interface. This process continually checks the list of data that is ready to be written out, looking for data that is of high or medium priority (as described above). Note that data of medium priority will only be sent if there is no data of high priority in the cache. However, it is possible for medium priority data to be written out before higher priority data, as in the case where the medium priority data is in the memory cache, and higher priority data is resident on disk.

## 6.0 Current ISS Status

All ISS software is currently tested and running on Sun workstations (SPARCstations and SPARCserver 1000's) running SunOS 4.1.3 and Solaris 2.3, DEC Alpha's running OSF/1, and SGI's running IRIX 5.x. Demonstrations of the ISS with the MAGIC Terrain Visualization application TerraVision have been done using several WAN configurations in the MAGIC testbed [11]. Using enough disks (4-8, depending on the disk and system type), the ISS software has no difficulty saturating current ATM interface cards. We have worked with 100 Mbit and 140Mbit TAXI S-Bus and VME cards from Fore systems, and OC-3 cards from DEC, and in all cases ISS throughput is only slightly less than *ttcp*[4] speeds.

Table 1 below shows various system *ttcp* speeds and ISS speeds. The first column is the maximum *ttcp* speeds using TCP over a ATM LAN with a large TCP window size. In this case, *ttcp* just copies data from memory to the network. For the values in the second column, we ran a program that continuously reads from all ISS disks simultaneously with *ttcp* operation. This gives us a much more realistic value for what network speeds the system is capable or while the ISS is running. The last column is the actual throughput values measured from the ISS. These speeds indicate that the ISS software adds a relatively small overhead in terms of maximum throughput.

**TABLE 1.**

| System | Max ATM LAN *ttcp* | *ttcp* w/ disk read | Max ISS speed |
|---|---|---|---|
| Sun SS10-51 | 70 Mbits/sec | 60 Mbits/sec | 55 Mbits/sec |
| Sun SS1000 (2 proc) | 75 Mbits/sec | 65 Mbits/sec | 60 Mbits/sec |
| SGI Challenge L | 82 Mbits/sec | 72 Mbits/sec | 65 Mbits/sec |
| Dec Alpha | 127 Mbits/sec | 95 Mbits/sec | 88 Mbits/sec |

## 7.0 Operation System Issues

### 7.1 Threads

Currently, the ISS Disk Server is implemented as a group of loosely-coordinated UNIX processes. We believe performance can be enhanced by transforming these processes into threads. Most of the gains arise from bypassing the overhead of the interprocess communication mechanisms needed to guarantee consistency of resources shared by the processes, e.g., the semaphores needed to ensure non-simultaneous access to the to-read and to-send lists. The same functionality can be achieved using thread-based mechanisms that are designed to be much faster, e.g., mutual exclusion locks.

The ISS Disk Server requires separate processes to receive from the network, read from disk, and send to the network. These processes must share certain resources, namely, the to-read lists, the to-send list, and the data cache. To ensure fair access to each of these resources, we force some processes to sleep for a short time: by

---

4. *ttcp* times the transmission and reception of data between two systems using the UDP or TCP protocols.

this mechanism, we guarantee that the operating system will perform a context switch. When any Disk Server process accesses a list or the cache, it first obtains a semaphore to guarantee exclusive access for the duration of the time it needs to perform its task. If other processes attempt to access the data, they are rejected and must, after a sleep-induced wait, try again.

Instead of the expensive semaphore mechanism, multiple threads guarantee exclusive access by using mutual-exclusion locks. The overhead of mutex locks is much less than that of semaphores, and checking mutex locks is much faster. Threads which cannot obtain a needed resource enter into a state of conditional waiting: this state eliminates the cycle of checking for the available resources, sleeping, and checking again, which characterizes processes attempting to gain a shared resource. Threads in conditional wait are simply put to sleep and signaled when the resource is available. Interthread communication is much faster than interprocess communication and threads consume fewer resources, since threads share the same text space with one another.

### 7.2 Real-Time Scheduling

An application like the Image Server System could benefit from real time scheduling. The ISS currently must attempt to coerce the UNIX scheduler to context-switch between the various competing ISS processes: trying to promote such context switching wastes time and reduces efficiency. A real-time operating system allows fine-grained control of the scheduler by means of thread prioritization and conditional waiting. Effectively, threads can take more or less processor time as necessary instead of arbitrarily taking a fixed slice of CPU time, and reducing competition with kernel-level or other user-level threads. This ability to vary the amount of time used by each thread is especially useful given that the ISS is driven by external events (the requests for the images) and must deliver the images back to the driving application within a predetermined time.

## 8.0 Experience

### 8.1 ATM Networking Issues

The design of the ISS is based upon the ability to use ATM network switches to aggregate cells from multiple physical data streams into a single high-bandwidth stream to the application. Figure 1 (Data Streams Aggregated by ATM Switches) shows multiple ISS servers being used to form a single high-speed data stream to the application.

Below is a list of what we have learned from our experience using ATM networks. Most of the experience reflected here comes from our work in the MAGIC gigabit testbed.

I) Hardware and Physical Layer:

- delivery of most ATM hardware has been delayed;
- the “infant mortality” rate has been high (several ATM interfaces and the ATM switch died in the first 60 days);
- it now seems clear that the workstation vendors have adopted multimode OC-3 as their preferred physical layer (we bought several 140 Mb/s TAXI, which were available six months ago);
- several of the multimode interfaces will drive single mode equipment (e.g., SONET terminals) by carrying the multimode fiber to the single mode interface (all examples of this have had active elements immediately behind the single mode interface);

II) Link Layer:

- not all of the ATM cell definitions (especially with respect to the Quality of Service (QOS) field) are uniform among manufacturers (QOS = 0 seems to be the point of agreement);

- there are many places where cell loss is apparent: these include switch output ports (next generation switches have much more buffering), and workstation interfaces, which are easily overrun for reasons not yet clear (it could be failure of kernel code to empty buffers fast enough);
- ATM device drivers are still fairly crude and buggy. This is especially true on multiprocessor systems, where the device drivers don't yet fully take advantage of the multi-threading capabilities of the operating system;

III) Network:

- Except for homogeneous switch environments, switched virtual circuits are not yet standardized, and in a large scale environment, use of PVCs makes set up and reconfiguration tedious and prone to errors;

IV) Transport:

- There are many throughput anomalies that are being investigated;
- Our work with timer-driven RTP shows some promise of it being a little more immune to cell loss than TCP.

One of the things that is becoming apparent in our work with this architecture is that the conventional notion of QOS is not a good method for regulating tightly coupled applications like the ISS, and (for similar reasons) may not be good for distributed-parallel compute server systems. Problems frequently occur when several servers that normally operate asynchronously to provide data to a single source, suddenly synchronize to produce a burst of data that overloads the switch and interface on the single receiver, thereby causing everybody to slow down and retransmit, leading to severe throughput degradation. This is very similar to the problem of routing message synchronization described by Floyd and Jacobson[2].

### 8.2 Memory Copy Speed

The main bottleneck for the ISS Disk Server is the speed of moving data into and out of memory. A SPARCStation 10, for example, has memory copy speed of about 22 Mbytes/s (176 Mbits/s). When writing to the network, the situation is even worse because data is moved to the interface via UNIX "mbufs" [7], adding additional overhead. We have measured the speed of an mbuf copy as 19 Mbytes/s (152 Mbits/s), and there are two mbuf copies required to send a packet to the network. Along with the other overhead required to assemble packets, this limits the speed with which we can write to the network to 9.2 Mbytes/s (74 Mbits/s).

If the network sends were faster, e.g., 19.4 Mbytes/s (155 Mbits/s - the OC-3 rate, ignoring ATM overhead), the next bottleneck would be the disk reading speed, which in this configuration is 9 Mbytes/s (72 Mbits/s). This bottleneck is trivially removed by adding more disks. This brings us back to the "memcpy" limit of 22 Mbytes/s as the key bottleneck. The other bottlenecks are not likely to be relevant in the near future. Increasing the speed of workstation memory is the key to increased performance for this application.

## 9.0 Conclusions

The emergence of wide area high-speed networks enables many types of new systems, include distributed parallel data servers. We have designed and implemented a special purpose high-speed data server, called the ISS. The ISS is designed to be distributed across multiple hosts with multiple disks on each host, and should be capable of scaling to gigabit per second rates. Moreover, we believe that the core design is flexible enough so that only minor modifications need be made to adapt the ISS to different data types and access patterns.

In the process of implementing and using this system, we have learned many things about workstation and operating system bottlenecks, and using ATM networks. The main things we discovered are:

- memory to memory copy speed is the main I/O bottleneck on today's workstations;
- ATM networks still have many problems to be worked out before they are ready for general use.

## 10.0 References


[1] Chen, L. T. and Rotem, D., "Declustering Objects for Visualization", Proc. of the 19th VLDB (Very Large Database) Conference, 1993.

[2] Floyd, S. and Jacobson, V., "The Synchronization of Periodic Routing Messages", Proceedings of SIGCOMM '93, 1993.

[3] Ghandeharizadeh, S. and Ramos, L, "Continuous Retrieval of Multimedia Data Using Parallelism, IEEE Transactions on Knowledge and Data Engineering, Vol 5, No 4, August 1993.

[4] Hartman, J. H. and Ousterhout, J. K., "Zebra: A Striped Network File System", Proceedings of the USENIX Workshop on File Systems, May 1992.

[5] Leclerc, Y.G. and Lau, S.Q., Jr.,"TerraVision: A Terrain Visualization System", SRI International, Technical Note #540, Menlo Park, CA, 1994.

[6] Langberg, M., "Silicon Graphics Lands Cable Deal with Time Warner Inc.", San Jose Mercury News, June 8, 1993.

[7] Leffler, S.J., McKusick, M.K., Karels, M. J. and Quarterman, J.S., "The Design and Implementation of the 4.3BSD UNIX Operating System", Addison-Wesley, Reading, Mass., 1989.

[8] Mills, D., "Network Time Protocol (Version 3) specification, implementation and analysis", RFC 1305, University of Delaware, March 1992.

[9] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1361, University of Delaware, August 1992.

[10] Patterson, D., Gibson, G., and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in Proc. 1988 SIGMOD Conf., June 1988.

[11] Richer, I. and Fuller, B.B., "An Overview of the MAGIC Project," M93B0000173, The MITRE Corp., Bedford, MA, 1 Dec. 1993.

[12] Rowe, L. and Smith, B.C., "A Continuous Media Player", Proc. 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, CA, Nov. 1992.

[13] Schulzrinne, H. and Casner, S., "RTP: A Real-Time Transport Protocol", Internet Draft, Audio/Video Transport Working Group of the IETF, 1993.