The Gigaview Multiprocessor Multidisk Image Server

B. A. Gennart* B. Krummenacher* L. Landron* R. D. Hersch*
B. Saugy§ J.-C. Hadorn§ D. Müller§

Abstract. Professionals in various fields such as medical imaging, biology and civil engineering require rapid access to huge amounts of pixmap image data. Multimedia interfaces further increase the need for large image databases. In order to fulfill these requirements, the GigaView parallel image server architecture relies on arrays of intelligent disk nodes, each disk node being composed of one processor and one disk. This contribution reviews the design of the GigaView hardware and file system, compares it to other storage servers available on the market, and evaluates fields of applications for the architecture.

1. Introduction

In the fields of scientific modeling, medical imaging, biology, civil engineering, cartography and graphic arts, there is an urgent need for huge storage capacities, fast access and real-time interactive visualization of pixmap images.

While processing power and memory capacity double every two years, disk bandwidth increases at a much slower rate. Interactive real-time visualization of full color pixmap image data requires throughputs of 2 to 10 MBytes/s. Parallel input/output devices are required in order to access and manipulate image data at high speed.

A high-performance high-capacity image server must provide users located on local or public networks with a set of adequate services for immediate access to images stored on disk arrays. Basic services include real-time extraction of image parts for panning purposes, resampling for zooming in and out, browsing through 3-d image cuts and accessing image sequences at the required resolution and speed.

Previous research focussed on increasing transfer rates between CPU and disks by using Redundant Arrays Of Inexpensive Disks (RAID) [1]. Access to disk blocks was parallelized, but block and file management continued to be handled by a single CPU with limited processing power and memory bandwidth. In a more recent research project [2], the RAID concept was further extended to offer very high bandwidth disk arrays directly hooked onto high-speed networks (HIPPI based networks).

In this paper, we use a different approach: the multiprocessor multidisk (MPMD) approach we propose aims at associating disks and processors into an array of intelligent disk nodes capable of applying parallel local preprocessing operations before sending data from the disk to the client workstation. We have shown that such preprocessing operations are highly valuable in the case of image accesses: large pixmap images can be reduced into displayable size images at disk reading speed [3, 4]. In the MPMD approach, pixmap image data is partitioned into rectangular extents, each extent having a size which minimizes global access time. In order to ensure high throughput, image extents are stored on a parallel array of disk nodes. Each disk node includes one disk-node processor (T800 transputer), cache memory (6 MBytes) and one disk (400 to 1000 MBytes).

The authors have implemented a MPMD image server, called the GigaView. Through its SCSI-2 interface, it sustains throughputs of up to 5MBytes/sec., which allows to browse through images and maps of arbitrary size at the rate of three to four 512-by-512 full color image visualization windows per second [5].

This contribution describes the design of the Gigaview image server : the hardware architecture, the multi-dimensional file system (MDFS), and the server's data redundancy scheme. It analyzes the performance of the architecture through sim-

*Laboratoire de Systèmes Périphériques, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne. [gennart,landron,hersch]@di.epfl.ch
§BSI, Bureau de Service et d'Ingéniérie, Primerose, 27, CH-1007 Lausanne

ulation and experimentation, and compares the performance to existing storage servers. The multimedia behavior of the GigaView has been studied in [6].

Section 2 describes the hardware architecture, the MDFS file system and the server's redundancy scheme. Section 3 analyses the Gigaview performance under single-request and multiple-request. Section 4 compares the performance of the GigaView to existing storage servers. Section 5 describes two application fields for the GigaView parallel image server : geographical information systems, and medical imaging. Section 6 summarizes the results of this contribution and describes the directions of future image server research.

2. GigaView design

2.1 Hardware architecture

The parallel image server consists of a server interface processor connected through a crossbar switch to an array of disk nodes (Figure 1). The server interface processor provides the network interface. Each disk node consists of a standard disk connected through a SCSI-II bus to a local processing unit. The disk-node processors are transputers (T800 in the current versions, and T9000 when they become available). They provide both processing power and communication links. The number of links between the interface processor and the disk array is 4, equal to the number of links of a single transputer. The disk-node local processor supports disk access, image part extraction, image reduction, data compression and decompression.

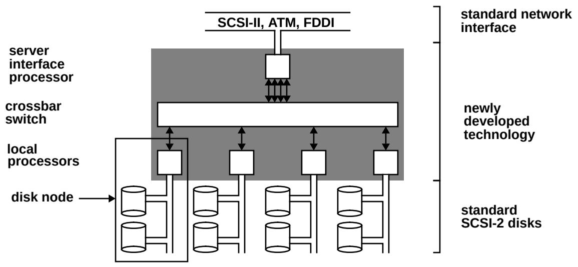

Figure 1: GigaView 8-disk architecture

2.2 Multi-dimensional file system

In order to access images in parallel, images are partitioned into rectangular *extents* (Figure 2). The Multi-Dimensional File System (MDFS) stores 1-dimensional (1-D), 2-D and 3-D images divided into 1-D, 2-D and 3-D extents respectively, and provides excellent access performance, regardless of the size of the accessed file and of the architecture on which it is executed [4]. Image access performances are heavily influenced by how extents are distributed onto a disk array. In a previous publication [3], we show that the extent size should be between 12 and 48 Kbytes, and describe algorithms to allocate extents efficiently on a disk array.

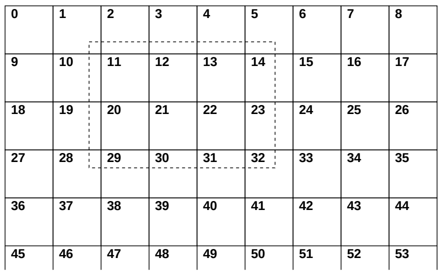

Figure 2: Division of an image into extents

The server interface processor runs the image server master process receiving image access requests from the network and issuing image access calls to the parallel image file server. The parallel file server includes a file system master process responsible for maintaining overall parallel file system coherence (directories, file index tables, file extent access tables) and extent serving processes running on disk node processing units. Extent serving processes are responsible for serving extent access requests, for maintaining the free block lists and for managing local extent caches. Local image processing tasks required for image presentation such as image data reduction for zooming purposes are located on disk node processing units.

The parallel low-level file system supports a single directory containing all the files stored on one MPMD cluster. Files are accessed through a directory entry which points to the *file distribution information block* (DIB). The DIB contains information relative to the file size, the file extension in x and y dimension, the extent width and height, the number of continuous extents per disk, the number of disks, a table with the successive disk numbers contributing to this file, and for each disk, a pointer to the *file local extent index table* (FLEIB) containing the local pointers to the data extent blocks (Fig. 3). At file opening time, the file system returns part of the content of the DIB. Directories and DIB have a fixed, maximal size. For safety reasons, they are duplicated on each of the disks in the cluster.

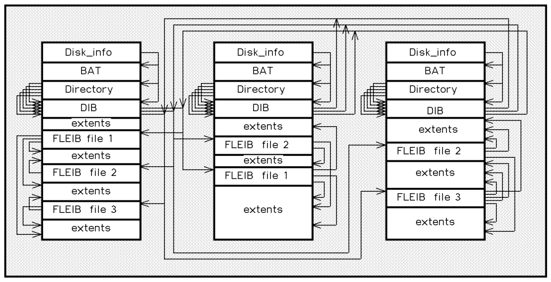

Figure 3: File system organization

At file opening time, each extent process reads the DIB and the FLEIB from its disk. Once the DIB and FLEIB are stored in memory, read and write operations on a given file can be executed at the rate of one disk access per extent. With an extent size between 12Kbytes to 48Kbytes, the throughput between disk and extent server is close to the maximal disk transfer rate.

2.3 Redundancy scheme

The redundancy scheme on the Gigaview server differs from the approach taken on RAID servers. RAID servers compute the redundancy information as the data is stored on the disk. This costs a write-access delay penalty (4 disk accesses are required for every user write operation), but ensures almost complete reliability. The GigaView server takes into account the improved reliability of single devices — up to 1,000,000 hours mean-time between failure (MTBF) for modern disk drives — to design a less restrictive redundancy scheme.

The *delayed parity* scheme (DPS) implemented on the GigaView enables the redundancy information to be computed *sometime after* the data has been written on disk. This assumes that the single-disk reliability is high, and that some recently written data may be lost in the event of a single disk failure. The following analysis will justify the DPS approach. The mean-time to data-loss for a RAID-5 server is given by the following formula [7] :

$$MTTDL = \frac{MTTF}{N} \frac{1}{\frac{MTTR}{\left(\frac{MTTF}{N-1}\right)}}$$

where $MTTDL$ is the disk-array Mean Time To Data Loss, $MTTF$ is a single-disk Mean Time Between Failure, N is the number of disks in the array (including the parity disk) on which the data is distributed and $MTTR$ is the single-disk Mean Time To Repair. The formula is written as the product of the MTBF of the array without redundancy $\frac{MTTF}{N}$, multiplied by a term showing the effect of the parity scheme. Considering a MTTR of 1 hour, 8 disks in the array, and a MTBF of 1,000,000 hour, we get a MTTDL of 13.89 billion hours, or 1.5 million years. Even without redundancy, the MTBF of an 8-disk array is 125,000 hours or 14 years. The MTTDL of a disk-array featuring delayed parity is given by the following formula :

$$MTTDL = \frac{MTTF}{N} \frac{1}{PTR\left(\frac{MTTR}{\left(\frac{MTTF}{N-1}\right)}\right) + (1 - PTR)}$$

where PTR (Parity Time Ratio) measures the fraction of time during which the parity information is available for the whole data. For example a 90%-PTR disk array is an array for which parity on the whole data is available 90% of the time. In this formula, the correction term consists of two parts, corresponding to periods of time where parity is (resp. is not) available. Considering a PTR of 0.1, the MTTDL for an 8-disk array is 1.1 million hour, similar to the MTBF of a single disk (over a hundred years), which is more than sufficient.

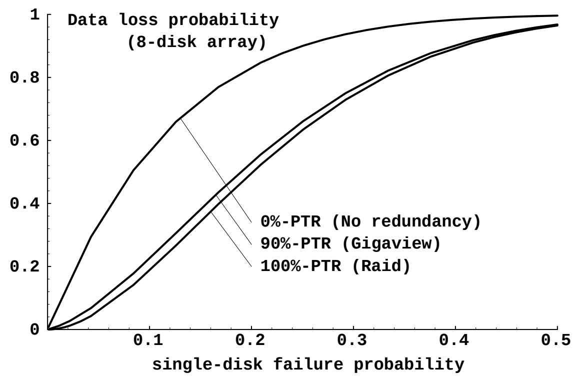

Figure 4: Data loss prob. vs. PTR

This theoretical analysis assumes that the loss of a single bit amounts to a data loss. However, although the delayed parity scheme does not guarantee total data integrity, it guarantees that most of the data (and in most cases, the whole data) can be recovered in the event of a single disk failure. Only some recently written data may be lost as a result of a single disk failure.

This analysis justifies the delayed parity redundancy scheme adopted in the GigaView design. Another approach is to study the effect of external causes on data integrity. For this analysis, we assume that an external cause (e. g. power supply breakdown) increases the probability of disk failure. In an n-disk system where each disk has a probability p to fail, the probability that exactly f disk fail $P(F=f)$ is given by the binomial probability law : $P(F=f) = C_n^f p^f (1-p)^{n-f}$.

Without parity, the data loss probability is the probability that one or more disks fail. With parity (RAID server approach), the data loss probability is the probability that two or more disks fail. With delayed parity (GigaView approach), the data loss probability is the weighted average of the with-parity and without-parity data loss probabilities. We plot as a function of the single disk failure the array data-loss loss probability, with and without redundancy scheme. Figure 4 shows the array data loss probability in three cases, no parity (0%-PTR), 90%-PTR, and 100%-PTR (equivalent to RAID-5 parity). In the case of a failure due to external causes, it confirms that the reliability of a GigaView with 90%-PTR is almost as good as a Raid server reliability.

3. GigaView performance analysis

This section analyzes through simulation the performance of the GigaView image server. It describes the simulation model (section 3.1). The performance under single request is modeled in terms of throughput and latency (section 3.2). The performance under multiple-request is shown to be dependent on the single request delay and the single request utilization (section 3.3).

3.1 Simulation model

Figure 5 describes the modeled behavior of the GigaView. Reading a visualization window from the GigaView consists of decomposing a window request into extent requests. As soon as an extent request is generated by the interface processor, it is transferred down the appropriate transputer link to the disk where the extent is located. The extent is fetched from the disk and transferred up a transputer link back to the interface processor, where it is merged with the other extents to form the visualization window.

The simulation model assumes that the disk access-time, the transputer-link transfer-time, and the transputer memory-to-memory copy-operations obey simple linear formulas of the form $Delay = Latency + (DataSize/Throughput)$.

3.2 Single-request behavior

This section shows by simulation that it is possible to describe the behavior of a parallel storage server using two numbers, latency and throughput. This is similar to the way secondary storage devices are described by two numbers, seek-time and throughput.

```
component GigaView is
    InterfaceT Interface ;
    LinkT DownLink[NUMBER_OF_LINKS] ;
    LinkT UpLink[NUMBER_OF_LINKS] ;
    DiskT Disk[NUMBER_OF_DISKS] ;
    procedure Read (WindowT window) ;
end GigaView ;

procedure GigaView.Read (WindowT window) is
begin
    Interface.Decompose (window, ERs) ;
    foreach ER in ERs do
        -- ER : extent request
        DownLink[ER.Link].Transfer (ER) ;
        Disk[ER.Disk].Access (ER, Ext) ;
        UpLink[ER.Link].Transfer (Ext) ;
        Interface.Merge (Ext, window) ;
    end foreach ;
end GigaView.Read ;
```

Figure 5: Simulation model

The approach is to measure the delay of the parallel storage server for increasing visualization window sizes, to linearize the delay using a least-square fit (Mathematica), and get a formula of the type :

$$\text{AccessTime} = \text{Latency} + \frac{\text{RequestSize}}{\text{Throughput}}$$

The GigaView architecture performance is sensitive to the extent allocation scheme. In particular, the extent size and the row offset have to be chosen carefully to reach the best performance. As shown in section 2.4., an extent size of 128-by-128 pixels and an extent row offset of 3 are effective for a wide range of visualization window sizes and optimum for a visualization window size of 512-by-512 pixels. In this experiment, the T800 transputers are modeled with a memory bandwidth of 18MBytes/s and each communication link has a throughput of 1.6MBytes/sec. The disks are T800-Quantum-SCSI2, whose seek-time and throughput have been measured experimentally at respectively 20msec. and 2.28MBytes/sec.

The linearization approach has proved particularly effective, regardless of the data allocation and the architecture of the system. Using the linear model of the performance of the GigaView, it is easy to demonstrate the effect of the number of disk-nodes in the architecture on the performance of the system. Figure 6 shows the access-time to a visualization window of increasing sizes for 4 architectures : 1-disk-node, 2-disk-node, 4-disk-node and 8-disk-node architecture.

Figure 6 shows that latency decreases and throughput increases as the number of disk-nodes increases. With a T800-based architecture, adding more disk-nodes ceases being beneficial, since link communication bandwidth limits overall performance. Beyond 8 disk-nodes, the throughput increases only marginally and the latency does not decrease.

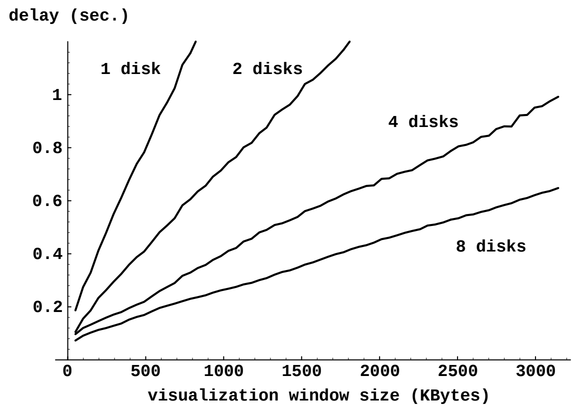

Figure 6: GigaView single request delay (T800-based architecture, simulation results)

It is possible to get a precise idea of the maximum number of disk-nodes the architecture supports by carrying out a single single-request experiment. The key concept is that of component *utilization*, defined as the ratio between a given component's active-time and the total simulation time. The component utilization is a simulation result, together with individual operation delays.

The simulation consists of requesting a single 512-by-512 3-byte-pixel visualization window, on a 4-disk-node T800-based architecture. In a 4-disk-node architecture, the average disk-node utilization is 86%, the links are 42%-utilized, and the interface processor is 33%-utilized. The ratio between disk-node- and link-utilization is $\frac{0.86}{0.42} \simeq 2$. This suggests that an 8-disk-node architecture provides an equal utilization of disk-nodes and links. The utilizations of disks and links in an 8-disk architecture are equal at 66% : the 8-disk architecture is said to be *balanced*. Simulations show that above 8 disks the throughput does not increase. Balancing the architecture should therefore be a design target.

The utilization data for the 8-disk architecture also shows that the maximum component utilization decreases significantly when stepping up the architecture from 4 to 8 disk-nodes. This explains why the delay of an 8-disk architecture (0.218s for a 512-by-512 3-byte-pixel visualization window) is more than half the delay of a 4-disk architecture (0.332s). Changing the data allocation scheme to improve the utilization by decreasing the extent size does not improve performance : the overhead due to the larger number of extents negates the effect of the improved data allocation.

3.3 Multiple-request behavior

This section describes the behavior of the GigaView under multiple requests. In order to provide a reference point, this study compares the behavior of the GigaView under multiple-request to the behavior of an abstract fixed-service-time server. It shows that, due to internal pipelining, the GigaView sustains higher throughput than the fixed-service-time server. The amount of additional throughput depends on the single-request utilization of the disk array.

Simulation characteristics. Requests to the GigaView represent a Poisson process. This means that individual requests are independent and that the number of requests in a given time interval only depends on the length of that interval. The interval between requests therefore follows an exponential distribution. The load on the system is expressed in terms of *requested throughput.* In our simulations, all users request a 512-by-512 3-byte-pixels visualization window (786 KBytes). Therefore, a requested throughput of 3MBytes/sec. corresponds to 4 window requests per second. The Poisson process hypothesis also ensures that, for a given load, the number of users requesting windows from the system has no effect. Only the requested throughput affects the average response time of the system. For a given system architecture, each simulation consists of requesting 5000 visualization windows at random positions in an image, for a given load. Each configuration is simulated for 20 loads chosen in the range of loads sustainable by the architecture. The result of each simulation is the delay average over the 5000 requests. For these simulations, the architecture consists of T9000 transputers and Quantum-SCSI-2 disks. Since the T9000 transputers were not yet available at the time of submission, their performance was conservatively estimated at 36MBytes/s memory bandwidth and 8MBytes/s link transfer rate. The Quantum-SCSI-2 latency and throughput are measured experimentally at 20msec. and 2.23MBytes/sec.

Fixed service-time server. The *fixed-service-time server* provides a reference point for the GigaView simulations. Its only property is its service-time, equal to the service-time of a single visualization window request. If a new request occurs while a request (the *current* request) is being served, the new request is delayed until the current request is completely served. Requests to the reference server follow the same distribution as requests to the GigaView. For example, a T9000-based 4-disk-node GigaView architecture satisfies a 512-by-512 3-byte-pixel visualization window request in 0.305 sec. The maximum throughput sustainable by the fixed-service-time server is (MST = *maximum sustainable throughput*) :

$$MST = \frac{\text{SRS}}{\text{SRD}} = \frac{768 KBytes}{0.305 sec.} = 2.62 MBytes/sec.$$

where *SRS* is the Single Request Size and *SRD* is the Single Request Delay. Figure 7 shows the performance results of the GigaView. The continuous line represents the GigaView performance (delay average), whereas the crosses represent the performance of the fixed-service-time server (delay average).

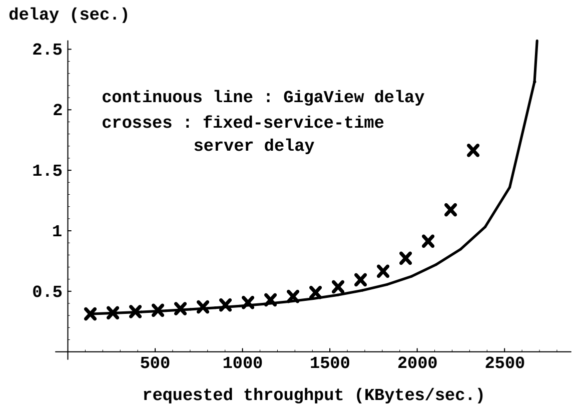

Figure 7: Performance under multiple requests (T9000-based architecture)

Figure 7 shows that the performance of the GigaView is *superior* to the performance of a fixed-service-time server. This result is not difficult to explain. During a single-request experiment, no component of a 4-disk-node GigaView architecture is used more than 90% of the time. Therefore, under multiple-request, some amount of internal pipelining occurs, making the GigaView able to sustain higher loads than the fixed-service-time server.

One can match the behavior of the fixed-service-time server and the GigaView by scaling the x-axis of the fixed-service-time server performance curve by a factor equal to the inverse of the single-request utilization of the GigaView. This suggests that the GigaView MST must be defined as :

$$\text{MST} \simeq \frac{\text{SRS}}{\text{SRD}} \cdot \frac{1}{\text{SRU}}$$

where *SRU* is the Single Request Utilization. In this formula, the single-request size is a simulation parameter ; the single-request delay and utilization are simulation results. The formula holds true regardless of the single request size. A single single-request simulation is enough to evaluate an architecture's maximum sustainable throughput.

Effect of the number of disk-nodes. Figure 8 shows the effect of the number of disk-nodes on the performance of the GigaView. Adding disk-nodes to the architecture improves the delay of each request and the GigaView's ability to sustain higher loads.

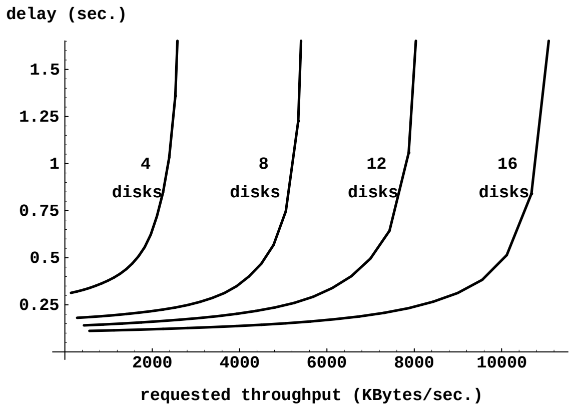

Figure 8: Effect of the number of disk-nodes (T9000-based architecture)

Consider a requested throughput of 6 MBytes/sec. The average delay for a 12-disk-node architecture is around 400msec, whereas a 16-disk-node architecture satisfies requests on average within 200msec, i.e. an improvement by a factor of 2. This seems to be in contradiction with the single request analysis of the same architecture.

The single-request analysis applied to a T9000-based architecture (Figure 9) shows that the maximum throughput is reached for a 12-disk-node architecture. The 16-disk-node architecture offers very little benefit over the 12-disk-node architecture, in terms of single-request throughput or access delay. The major difference between the two architectures lies in the utilization of 12-disk and 16-disk architectures under single-request. In a 12-disk-node architecture, disk-node components are utilized on average 76% of their time, and in a 16-disk-node architecture, they are used on average 61% of their time.

Using the MST formula introduced earlier, we find that the maximum sustainable throughputs are 2.98 MBytes/s (respectively 5.97 MBytes/s, 9.04 MBytes/s, 11.94 MBytes/s) for a 4-disk (respectively 8-disk, 12-disk, 16-disk) architecture.

Although the single request throughput does not increase above 12 disks, the maximum sustainable throughput under multiple requests increases linearly with the number of disks, for up to 16 disks. Above 16 disks, the interface processor becomes saturated and the maximum sustainable throughput does not increase anymore.

As the throughput approaches the MST, the access delay increases exponentially. At 6MBytes/sec, the 12-disk architecture is closer to its MST than the 16-disk architecture. Hence the access delay is much higher for the 12-disk architecture.

number of disks	4	8	12	16
delay (ms)	354	189	143	135
latency (ms)	123	75	95	44
thr. (MB/s)	4.01	7.97	12.8	11.8
utilization (%)	93	87	76	61
MST (%)	2.98	5.97	9.04	11.94

Figure 9: GigaView single-request analysis (T9000-based architecture)

4. Measured performance comparisons

This section compares the access delays of four storage systems. The first configuration is an actual SparcClassic workstation and its local disk and the second configuration is an actual RAIDER-5 system connected to a Sparc server 1000. The third system is an actual RAID level-3 system connected to a Cray Y-MP. The fourth system is an actual 4-disk-node GigaView system. The SparcClassic local disk is a 1GB Quantum with SCSI interface, 10msec seek-time and 2.93MBytes/sec. sustained throughput. The RAIDER-5 system is a RAID-5 architecture consisting of (4+1) WREN-9 disks having a latency of 12.9 msec. and a wide SCSI-2 interface. The RAID level-3 system consists of 10 disks (8 + 2 spare) Hitachi DK-516-15.

The experiment consists of transferring visualization windows of increasing sizes from disk(s) to host memory, and measuring the transfer times. All architectures run MDFS, the multidimensional file system. The image from which the visualization windows are selected is 3072-by-2048 3-byte pixels in size, and is divided in 128-by-128 extents. On the first three configurations (single-disk station and the two RAID servers), the entire data is experimental.

For the GigaView performance measurements, it is assumed that transferring a visualization window from disk to host is a two-stage pipeline. The first stage of the pipeline transfers rows of extents from the disks to the GigaView server interface processor memory. The second stage transfers rows of extents from the server interface processor memory to the host memory.

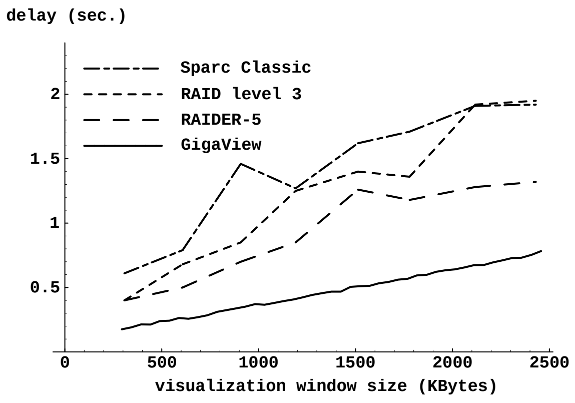

Figure 10: Experimental results

The access delay is a combination of (1) *actual delays* measured on the GigaView system (transfer of the whole visualization window between the disks and the GigaView interface processor) and (2) a conservative *estimate* of the transfer delay of one row of extents between the GigaView interface processor and the host memory. The formula giving the estimation of the SCSI bus transfer time is :

$$\frac{\text{RowOfExtentSize}}{\text{SCSIthroughput}} = \frac{\left\lceil \frac{\text{ImageWidth}}{\text{ExtentWidth}} \right\rceil \cdot ExtentSize}{5 \cdot 10^6}$$

The fact that the GigaView performance is superior to the RAID systems performance can be traced to the fact that the GigaView has an excellent control over extent allocation, which could not be achieved on the tested RAID-III and RAIDER-V systems. To achieve the best visualization window access times, it is necessary to control precisely the disk allocation of each image extent.

5. Applications

The authors consider two application fields for the GigaView image servers : geographical information systems, and medical imaging. Both fields require large amounts of pixmap data, as well as the ability to define relationships between various pixmaps (hypermedia document). Both fields also require the ability to display the information stored on the server. Multimedia techniques [6] can be used to provide the best presentation of the data.

5.1 Geographical information systems (GIS)

The EPFL and BSI Engineering develop civil engineering network planning facilities based on the superimposition of networks (road, gas, electricity) and scanned topographic maps.

Experience acquired during exhibitions and interactions with potential users led us to the conclusion that the Gigaview must support various layers of information such as orthophotos, scanned 1:25'000 topographic maps, 1:5'000 local maps, and 1:500 cadastral maps. For reference, topographic (resp. cadastral) maps scanned at 500 dpi covering the whole of Switzerland represent 37.5GBytes (resp. 27.5GBytes) uncompressed. In order to pack sparse scanned maps of a significant region onto a disk array of reasonable size (16 disks for example), there is an imperative need for using lossless compression techniques.

The GigaView uses several lossless compression algorithms tuned to the kind of data stored on disk. The algorithms are variations of the run-length coding algorithm and are optimized to provide high-speed software decompression, at the expense of compression efficiency.

Scanned topographic maps consist of 1-byte pixels. The BRL1 algorithm recognizes uniform sequences and divides each map in two kinds of runs : compressed uniform runs, and uncompressed runs. Scanned cadastral maps are predominantly white and very sparse bitmaps. They are compressed using two versions of a lossless compression algorithm called BRL2 and BRL3, working at the byte level. The BRL2 algorithm divides bitmaps in three kinds of byte runs : runs of black bytes ; runs of white bytes ; runs of gray bytes. The BRL3 algorithm takes into account the fact that in most cases, black and white runs are followed by a single gray byte : Each black or white run consists of several identical bytes followed by a single gray byte. In addition, the compression algorithms combines short white runs between two gray runs into a single longer gray run, to speed up decompression.

The compression and decompression facilities are integrated into the server's file system. The *data access pipeline*, i.e. the path from the compressed data on disk to a visualization window on the GigaView interface processor consists of four steps : moving the required compressed-image extents from disk to its disk-node processor cache ; decompressing the extents on the disk-node processor, and storing the uncompressed extents back in the disk-node processor cache ; transferring the uncompressed extents from the disk-nodes to the server interface processor ; and merging the decompressed extents into the visualization window buffer. The four steps are pipelined for extents extracted from the same disk node.

The authors evaluated the three decompression algorithms on three processor architectures : sparc-sun4m processor (in Sparc5 workstations) ; sparc-sun4c processor (in Sparc IPC workstations) ; T800 transputer (Figure 11). The delays of each algorithm are measured on each architecture, for windows of varying sizes. The delay curves are linearized and the slope of the linearized curve represents the algorithm throughput. The Sparc5 workstation is able to decompress at around 10MBytes/sec (40MBytes/sec peak), the SparcIPC at 2.5MBytes/sec (9MBytes/sec peak), and the T800 transputer (which has no internal cache) at the rate of 900KBytes/sec (2.5MBytes/sec peak).

architecture	BRL1 bytemap	BRL2 bitmap	BRL3 bitmap
sparc-sun4m	12.50	9.90	12.35
sparc-sun4c	2.31	2.51	2.70
t800	0.91	0.91	0.92
compression factor	4.06	4.57	4.57

Figure 11: BRL decompression performance (MBytes/sec.)

The authors also tested the 4-disk Gigaview architecture connected to a MacIntosh computer, for compressed and uncompressed maps, and for various zoom factors (Figure 12). A zoom factor of n is achieved by selecting one in n^2 pixels in a decompressed image. The experiment consists for each zoom factor to extract visualization windows of increasing size. When the zoom factor is increased, the visualization window sizes are not changed, and consequently, the size of the data fetched from the disks is increased. The experiments are done in compressed and uncompressed mode. In compressed mode, the compressed data is extracted from the disk, decompressed, and merged into the uncompressed visualization window. In uncompressed mode, the data is uncompressed throughout the experiment. Figure 12 reports four results as a function of the zoom factor : the interface processor (SIP) throughput in compressed mode, the total disk-node throughput in compressed mode, the SIP throughput in uncompressed mode, the total disk-node throughput in uncompressed mode. The total disk-node throughput is the sum of the uncompressed-data throughput through each disk-node.

In uncompressed mode, the current SCSI-MacIntosh interface limits the SIP throughput at 660KBytes/sec. At the disk-node level however, the throughput can reach up to 7.66MBytes/sec., enabling a complete topographic map (128MBytes uncompressed) to be visualized in less than 20sec. In compressed mode, the combined disk-access and decompression throughput reaches in the average case (typical cadastral map) 2.75 MBytes/sec, or 700KBytes/sec per processor in the GigaView architecture ; and in the best case (completely white cadastral map) 8MBytes, or the same throughput as in uncompressed mode. The next generation T9000 transputers will allow the decompression process to be completely transparent to the user.

	compr. data on disk		uncompr. data on disk	
zoom	SIP (decompr.)	disks (compr.)	SIP (uncompr.)	disks (uncompr.)
1	0.65	0.65	0.65	0.65
2	0.31	1.23	0.65	2.58
3	0.22	1.96	0.54	4.83
4	0.15	2.40	0.38	6.10
5	0.10	2.55	0.26	6.49
6	0.07	2.65	0.19	6.92
7	0.05	2.74	0.15	7.32
8	0.04	2.75	0.12	7.66

Figure 12: Gigaview access throughput (MBytes/sec.)

These results show the benefits of integrating decompression into the data access pipeline. Furthermore, the image-oriented file system can be ported to a high-end workstation with multiple processors and SCSI channels, while retaining excellent decompression performance.

5.2 Medical imaging

The authors acquired and stored on the GigaView a 3-D MRI scan (Magnetic Resonance Imaging). The image size is 100MBytes (384-by-512-by-512 1-byte pixels). Thanks to the GigaView, image views orthogonal to the main axes can be extracted at the rate of several frames per second. The image frames through which the user is browsing come directly from the disk, without the costly operation of preloading them in memory.

For this application, the 3-D images are divided in 3-D extents, which improve the locality of both disk and memory accesses. This feature is essential as access times depend almost completely on access locality. Let's assume an image width (X-axis), height (Y-axis), and depth (Z-axis) of W, H, and D pixels ; a visualization window width and height of w and h ; and an extent size of e pixels.

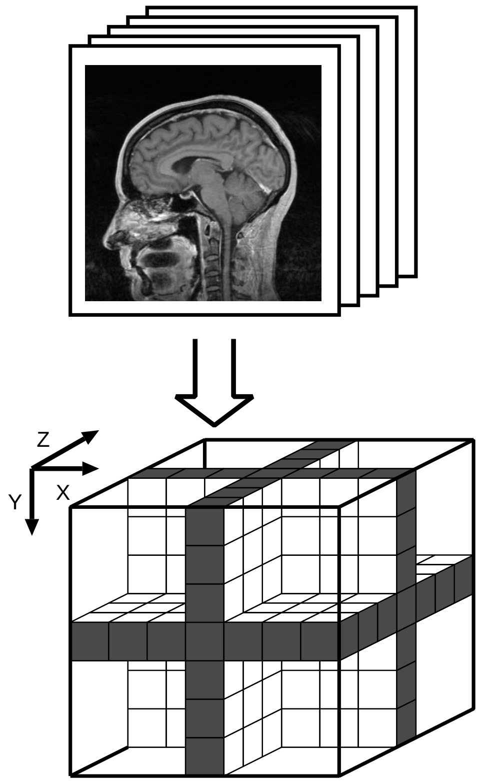

Figure 13: MRI scan and 3-D extents

Consider first the case where the 3-D images are stored as a set of 2-D images (X-Y planes stacked along the Z-axis, top of Figure 13). In this format, an extent (i.e. a few KBytes of data with good locality) has a width of W and a height of $\nu = e/W$. To fetch a visualization window along the XY plane requires accessing an extent for every ν lines in the visualization window. Along the XZ plane, it requires accessing an extent for every line in the visualization window. Along the YZ plane, it requires accessing one extent for every ν pixel in a visualization window line.

More formally, accessing images along the XY (resp. XZ, YZ) plane requires $\frac{h}{\nu}$ (resp. h, resp. $w \cdot \frac{h}{\nu}$) extent accesses. To give some numbers, assuming W, H, D at 2048 pixels, w, h at 512 pixels, e at 32768 pixels, a single extent access-time at 20msec and a single disk, we get an access-time of 160msec (resp. 10.24s, 327.68sec) along the XY (resp. XZ, YZ) plane. The access anisotropy is large. On the other hand, if we consider cubical extents, the number of extent accesses is identical along all 3 planes ($\frac{w}{\sqrt[3]{e}} \cdot \frac{h}{\sqrt[3]{e}}$). With the same image and visualization window, the access time becomes 5.12 sec. along any axis. Moreover, access to contiguous planes will be much faster, as the relevant extents can be maintained in disk-node cache. For example, 32 frames can be visualized in the same 5.12sec. If we consider an 8-disk architecture, the access time drops below 1sec.

Thanks to 3-D extents, the amount of data read from the disk depends only on the visualization window size. This last feature is essential, considering for example that the 3-D scan of a complete human body represents about 24GBytes of data (2048-by-2048-by-2048 3-byte pixels).

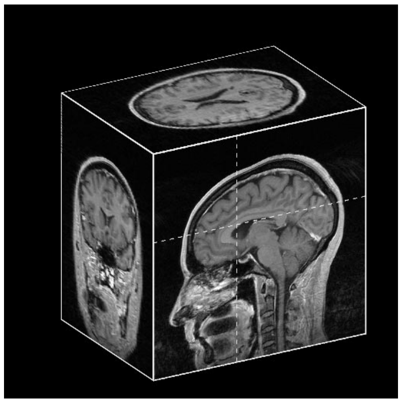

Figure 14: 3-D visualization of MRI scan

The authors are developing visualization algorithms that can be implemented efficiently on the GigaView architecture. These algorithms are based on the extraction, rotation and projection of planes of arbitary direction in the 3-D reference image (Figure 14).

6. Conclusion

This paper has presented the design, evaluation and applications of the GigaView multiprocessor multidisk image server. The GigaView is a dedicated multiprocessor architecture connected through a standard SCSI-bus to a workstation. It can interactively display 2-D and 3-D pixmap images accessed simultaneously from several disks. The division of data in extents gives excellent locality to random accesses of 2-D and 3-D pixmap images. The MDFS file system enables data access and processing to be pipelined, allowing for example decompression to be performed almost transparently to the user.

Future research aims at adapting the GigaView concept to multi-processor multi-disk workstations. Research will evaluate the modifications required to the file system in order to achieve the performance of the current GigaView server on a standard UNIX multiprocessor platform.

7. Acknowledgment

This project is supported by the Swiss SPP-IF impulse program in computer science (grant 5003-034345) and by the CERS (grant 2434.1). We acknowledge the help of the Swiss Office of Topography and of the Cadastre du Canton de Vaud which provided the maps used in the project.

REFERENCES

[1] D. A. Patterson, G. A. Gibson, and R. H. Katz. The case for RAID : redundant arrays of inexpensive disks. In *Proceedings ACM SIGMOD Conference*, pages 106–113, Chicago, IL, May 1988.

[2] E. K. Lee, P. M. Chen, J. H. Hartman, A. L. Drapeau, E. L. Miller, R. H. Katz, G. A. Gibson, and D. A. Patterson. RAID-II : a scalable storage architecture for high-bandwidth network file service. Technical Report 92/672, CSD, UC Berkeley, 1992.

[3] R. D. Hersch. Parallel storage and retrieval of pixmap images. In *Proceedings of the 12th IEEE Symposium on Mass Storage System*, pages 221–226, Monterey, 1993.

[4] R. D. Hersch, B. Krummenacher, and L. Landron. Parallel pixmap image storage and retrieval. In Grebe et al., editor, *Proceedings of the World Transputer Congress*, pages 691–699. IOS Press, 1993.

[5] B. A. Gennart, B. Krummenacher, L. Landron, and R. D. Hersch. GigaView parallel image server performance analysis. In IOS Press, editor, *Transputer Applications and Systems, Proc. World Transputer Congress*, pages 120–135, Como, September 1994.

[6] B. A. Gennart and R. D. Hersch. Multimedia performance behavior of the GigaView parallel image server. In *13th IEEE Symp. on Mass Storage Systems*, pages 90–98, Annecy, June 1994.

[7] Garth A. Gibson. *Redundant Disk Arrays : Reliable, Parallel Secondary Storage.* MIT Press, Cambrigde, MA, 1992.