The Tiger Video Fileserver

William J. Bolosky, Joseph S. Barrera, III,
Richard P. Draves, Robert P. Fitzgerald,
Garth A. Gibson, Michael B. Jones, Steven P. Levi,
Nathan P. Myhrvold, Richard F. Rashid

April, 1996

Technical Report
MSR-TR-96-09

Microsoft Research
Advanced Technology Division
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Paper presented at the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 96), April, 1996.

The Tiger Video Fileserver

William J. Bolosky, Joseph S. Barrera, III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson [1],
Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold, Richard F. Rashid

Microsoft Research, Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

bolosky@microsoft.com, mbj@microsoft.com

Abstract

Tiger is a distributed, fault-tolerant real-time fileserver. It provides data streams at a constant, guaranteed rate to a large number of clients, in addition to supporting more traditional filesystem operations. It is intended to be the basis for multimedia (video on demand) fileservers, but may also be used in other applications needing constant rate data delivery. The fundamental problem addressed by the Tiger design is that of efficiently balancing user load against limited disk, network and I/O bus resources. Tiger accomplishes this balancing by striping file data across all disks and all computers in the (distributed) system, and then allocating data streams in a schedule that rotates across the disks. This paper describes the Tiger design and an implementation that runs on a collection of personal computers connected by an ATM switch.

1. Introduction

As computers and the networks to which they are attached become more capable, the demand for richer data types is increasing. In particular, temporally continuous types such as video and audio are becoming popular. Today, most applications that use these types store the data on local media such as CD-ROM or hard disk drives because they are the only available devices that meet the necessary storage capacity, bandwidth and latency requirements. However, the recent rapid increase in popularity of the internet and the increases in corporate and university networking infrastructure indicate that network servers for continuous media will increasing become feasible. Networked servers have substantial advantages over local storage, principally that they can store a wider range of content, and that the stored content can be changed much more easily.

Serving continuous media over a network presents a host of problems. The network itself may limit bandwidth, introduce jitter or lose data. The load presented by users may be uneven. The content on the server must be easily changeable, and the server must be highly available. The server must be inexpensive and easy to maintain.

The paper describes Tiger, a continuous media fileserver. Tiger's goal is to produce a potentially large number of constant bit rate streams over a switched network. It aims to be inexpensive, scalable and highly available. A Tiger system is built out of a collection of personal computers with attached high speed disk drives connected by a switched broadband network. Because it is built from extremely high volume, commodity parts, Tiger achieves low cost by exploiting the volume curves of the personal computer and networking industries. Tiger balances the load presented by users by striping the data of each file across all of the disks and all of the machines within the system. Mirroring yields fault tolerance for failures of disks, disk interfaces, network interfaces or entire computers.

This paper describes the work done by Microsoft Research on the Tiger prototype. Most of the work described in this paper was done in 1993 and 1994. Any products based on Tiger may differ from the prototype.

2. The Tiger Architecture

Tiger is organized as a real-time distributed system running on a collection of computers connected by a high speed network. Each of these computers is called a "cub." Every cub has some number of disks dedicated to storing the data in the Tiger file system as well as a disk used for running the operating system. All cubs in a particular system are the same type of computer, with the same type of network interface(s) and the same number and type of disks. Tiger stripes its file data across all of the disks in the system, with a stripe granularity chosen to balance disk efficiency, buffering requirements and stream start latency. Fault tolerance is provided by data mirroring.

The fundamental problem solved by Tiger is to efficiently balance the disk storage and bandwidth requirements of the users across the system. A disk

[1] Garth Gibson is at Carnegie Mellon University.

device provides a certain amount of storage and a certain amount of bandwidth that can be used to access its storage. Sourcing large numbers of streams requires the server to have a great amount of bandwidth, while holding a large amount of content requires much storage capacity. However, most of the bandwidth demanded may be for a small fraction of the storage used; that is to say, some files may be much more popular than others. By striping all of the system's files across all of its disks, Tiger is able to divorce the drives' storage from their bandwidth, and in so doing to balance the load.

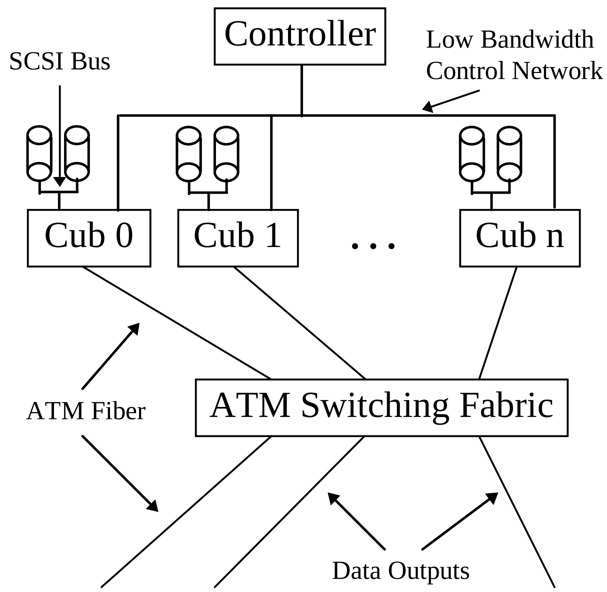

Figure 1: Basic Tiger Hardware Layout

In addition to the cubs, the Tiger system has a central controller machine. This controller is used as a contact point for clients of the system, as the system clock master, and for a few other low effort, bookkeeping activities. None of the file data ever passes through the controller, and once a file has begun streaming to a client, the controller takes no action until the end of file is reached or the client requests termination of the stream.

In order to ensure that system resources exist to stream a file to a client, Tiger must avoid having more than one client requiring service from a single disk at a given time. Inserting pauses or "glitches" into a running stream if resources are over committed would violate the basic intent of a continuous media server. Instead, Tiger briefly delays a request to start streaming a file so that the request is offset from all others in progress, and hence will not compete for hardware resources with the other streams. This technique is implemented using the Tiger schedule, which is described in detail in section 3.

Because the primary purpose of Tiger is to supply time critical data, network transmission protocols (such as TCP) that rely on retransmission for reliability are inappropriate. By the time an omission was detected and the data was retransmitted, the data would no longer be useful to the client. Furthermore, when running over an ATM network, the quality of service guarantees provided by ATM ensure that little if any data is lost in the network. Tiger thus uses UDP, and so any data that does not make it through the network on the first try is lost. As the results in section 6 demonstrate, in practice very little data is lost in the network.

Tiger makes no assumptions about the content of the files it serves, other than that all of the files on a particular server have the same bit rate. We have run MPEG-1, MPEG-2, uncompressed AVI, compressed AVI, various audio formats, and debugging files containing only test data. We commonly run Tiger servers that simultaneously send files of different types to different clients.

We implemented Tiger on a collection of personal computers running Microsoft Windows NT. Personal computers have an excellent price/performance ratio, and NT turned out to be a good implementation platform.

3. Scheduling

The key to Tiger's ability to efficiently exploit distributed I/O systems is its schedule. The schedule is composed of a list of schedule slots, which provide service to at most one viewer. The size of the schedule is determined by the capacity of the whole system; there is exactly one schedule slot for each potential simultaneous output stream. The schedule is distributed between the cubs, which use it to send the appropriate block of data to a viewer at the correct time.

The unit of striping in Tiger is the block; each block of a file is stored on the drive following the one holding its predecessor. The block size is typically in the range of 64KBytes to 1MByte, and is the same for every block of every file on a particular Tiger system. The time it takes to play a block at the given stream rate is called the *block play time*. If the disks are the bottleneck in the system, the worst case time that it takes a disk to read a block, together with some time reserved to cover for failed disks, is known as the *block service time*. If some other resource such as the network is the bottleneck, the block service time is made to be large enough to not overwhelm that resource. Every disk in a Tiger system walks down the schedule, processing a schedule slot every block service time. Disks are offset from one another in the schedule by a block play time. That is, disks proceed through the schedule in such a way that every block play time each schedule slot is serviced by exactly one disk. Furthermore, the disk that services the schedule slot is the successor to the disk that serviced the schedule slot one block play time previously.

In practice, disks usually run somewhat ahead of the schedule, in order to minimize disruptions due to transient events. That is, by using a modest amount of buffer and taking advantage of the fact that the schedule is known well ahead of time, the disks are able to do their reads somewhat before they're due, and to use the extra time to allow for events such as disk thermal recalibrations, short term busy periods on the I/O or SCSI busses, or other short disruptions. The cub still delivers its data to the network at the time appointed by the schedule, regardless of how early the disk read completes.

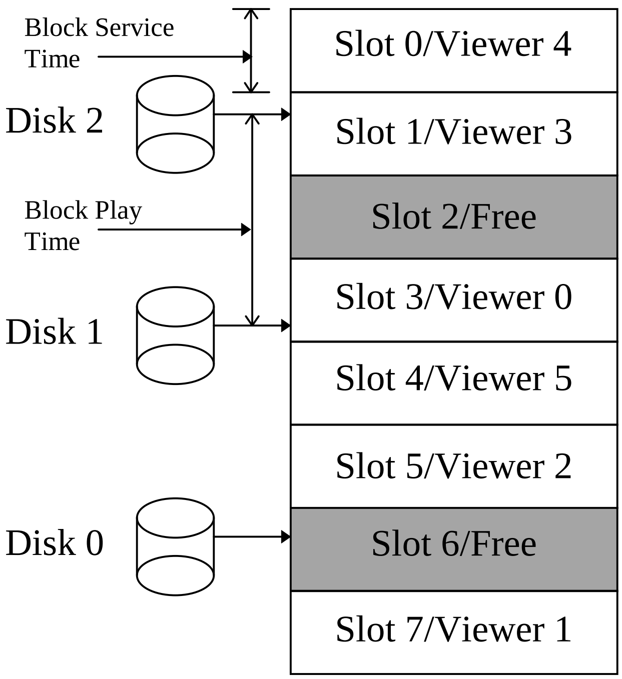

Figure 2: Typical Tiger Schedule

Figure 2 shows a typical Tiger schedule for a three disk, eight viewer system. The disks all walk down the disk schedule simultaneously. When they reach the end of the slot that they're processing, the cub delivers the data read for that viewer to the network, which sends it to the viewer. When a disk reaches the bottom of the schedule, it wraps to the top. In this example, disk 1 is about to deliver the data for viewer 0 (in slot 3) to the network. One block play time from the instant shown in the figure, disk 2 will be about to hand the next block for viewer 0 to the network. Because the files are striped across the disks, disk 2 holds the block after the one being read by disk 1.

Striping and scheduling in this way takes advantage of several properties of video streams and the hardware architecture. Namely, that there is sufficient buffering in the cubs to speed-match between the disks and output streams; that files ("movies") usually are long relative to the product of the number of disks in the system and the block play time; and that all output streams are of the same (constant) bandwidth.

Buffering is inherently necessary in any system that tries to match a bursty data source with a smooth data consumer. Disks are bursty producers, while video rendering devices are smooth consumers. Placing the buffering in the cubs rather than the clients makes better use of memory and smoothes the load presented to the network.

Because files are long relative to the number of disks in the system, each file has data on each disk. Then, even if all clients are viewing the same file, the resources of the entire system will be available to serve all of the clients. If files are short relative to the number of disks, then the maximum number of simultaneous streams for a particular file will be limited by the bandwidth of the disks on which parts of the file lie. Because the striping unit is typically a second of video, a two hour movie could be spread over more than 7,000 disks, yielding an upper limit on scaling that is the aggregate streaming capacity of that many disks; for 2 Mbit/s streams on Seagate ST15150N (4GByte Barracuda) disks, this is over 60,000 streams. Beyond that level, it is necessary to replicate content in order to have more simultaneous viewers of a single file. Furthermore, the network switching infrastructure may impose a tighter bound than the disks.

Having all output streams be of the same bandwidth allows the allocation of simple, equal sized schedule slots.

When a viewer requests service from a Tiger system, the controller machine sends the request to the cub holding the first block of the file to be sent, which then adds the viewer to the schedule. When the disk containing the first block of data processes the appropriate schedule slot, the viewer's stream will begin; all subsequent blocks will be delivered on time. When a cub receives a request for new service, it selects the next unused schedule slot that will be processed by the disk holding the viewer's first block. This task is isomorphic to inserting an element into an open hash table with sequential chaining. Knuth[Knuth 73] reports that, on average, this takes $(1+1/(1-\alpha)^2)/2$ probes, where α is the fraction of the slots that are full. In Tiger, the duration of a "probe" is one block service time. Since service requests may arrive in the middle of a slot on the appropriate drive, on average an additional one half of a block service time is spent waiting for the first full slot boundary. This delay is in addition to any latency in sending the start playing request to the server by the client, and any latency in the network delivering the data from the cub to the client. Knuth's model predicts behavior only in the case that viewers never leave the schedule; if they do, latency will be less than predicted by the model.

Tiger supports traditional filesystem read and write operations in addition to continuous, scheduled play. These operations are not handled though the schedule, and are of lower priority than all scheduled operations. They are passed to the cub holding the block to be read or written. When a cub has idle disk and network capacity, it processes these *nonscheduled* operations. In practice, because the block service time is based on worst case performance, because extra capacity is reserved for operation when system components are failed (see section 4), and because not all schedule slots will be filled, there is usually capacity left over to complete nonscheduled operations, even when the system load is near rated capacity.

Nonscheduled operations are used to implement fast forward and fast reverse (FF/FR). A viewer requests that parts of a file be sent within specific time bounds, and Tiger issues nonscheduled operations to try to deliver the requested data to the user. If system capacity is not available to provide the requested data within the provided time bounds, the request is discarded. In practice, the clients' FF/FR requests require less disk and network bandwidth than scheduled play, and almost always complete, even at high system load and when components have failed.

4. Availability

Tiger systems can grow quite large, and are usually built out of off-the-shelf personal computer components. Large Tiger systems will have large numbers of these components. As a result, component failures will occur from time to time. Because of the striping of all files across all disks, without fault tolerance a failure in any part of the system would result in a disruption of service to all clients. To avoid such a disruption, Tiger is fault tolerant. Tiger is designed to survive the failure of any cub or disk. Simple extensions will allow tolerance of controller machine failures. Tiger does not provide for tolerating failures of the switching network. At an added cost, tiger systems may be constructed using highly available network components.

Tiger uses mirroring, wherein two copies of the data are stored on two different disks, rather than parity encoding. There are several reasons for this approach. Because Tiger must survive not only the failure of a disk, but also the failure of a cub or network interface card, using commercial RAID [Patterson et al. 88] arrays (in which fault tolerance is provided only in the disk system) will not meet the requirements. Building RAID-like stripe sets across machines and still meeting Tiger's timeliness guarantees would consume a very large amount of both network bandwidth and buffer memory when a disk is failed. Furthermore, in order to provide the bandwidth to supply a large number of streams, Tiger systems will have to have a large number of disks even when they're not required for storage capacity. Combined with the extremely low cost per megabyte of today's disk storage, these factors led us to choose mirroring as the basis for our fault tolerance strategy.

A problem with mirroring in a video fileserver is that the data from the secondary copy is needed at the time the primary data would have been delivered. The disk holding the secondary copy of the data must continue to supply its normal primary load as well as the extra load due to the failed disk. A straightforward solution to this problem would result in reserving half of the system bandwidth for failure recovery. Since system bandwidth (unlike disk capacity) is likely to be the primary determiner of overall system cost, Tiger uses a different scheme. Tiger declusters its mirrored data; that is, a block of data having its primary copy on disk i has its secondary copy spread out over disks i+1 through i+d, where d is the *decluster factor*. Disks i+1 through i+d are called disk i's *secondaries*. Thus, when a disk fails, its load is shared among d different disks. In order not to overload any other disk when a disk fails, it is necessary to guarantee that every scheduled read to a failed disk uses an equal amount of bandwidth from each of the failed disk's secondaries. So, every block is split into d pieces and spread over the next d disks.

There are tradeoffs involved in selecting a decluster factor. Increasing it reduces the amount of network and I/O bus bandwidth that must be reserved for failure recovery, because it increases the number of machines over which the work of a failed cub is spread. Increasing the decluster factor also reduces the amount of disk bandwidth that must be reserved for failed mode operation, but larger decluster factors result in smaller disk reads, which decrease disk efficiency because the same seek and rotation overheads and command initiation time are amortized over fewer bytes transferred. Larger decluster factors also increase the window of vulnerability to catastrophic failure: the system fails completely only when both copies of some data are lost. When a disk fails, the number of other disks that hold its secondaries, or whose secondaries it holds is twice the decluster factor, because any disk holds the secondaries for the previous d disks; the next d disks after any disk hold its secondaries. We expect that decluster factors from 4 to 8 are most appropriate with a block size of .5Mbytes if the system bottleneck is the disks. Backplane or network limited systems may benefit from larger d.

Bits stored on modern disks must occupy a certain linear amount of magnetic oxide (as opposed to distending a fixed angle). Since the tracks on the outside part of a disk are longer than those on the inner part, more bits can be stored on the outer tracks. Because the disk rotates at a constant speed, the outer tracks store more data than the inner ones and the data transfer rate from the outer tracks is greater than that from the inner ones. Tiger exploits this fact by placing the secondaries on the inner, slower half of the disk. Because of declustering, during failures at least *d* times more data will be read from the primary (outer) region of any given disk than from its secondary (inner) region. So, when computing the worst case block service time and taking into account the spare capacity that must be left for reading secondaries, the primary would be read from the middle of the disk (the slowest part of the primary region) and the secondary from the inside. These bandwidth differences can be considerable: We measured 7.1MB/s reading sequentially from the outside of a 4GB Seagate Barracuda ST15150N formatted to 2Kbyte sectors, 6.6MB/s at the middle and 4.5MB/s at the hub. Given our measured worst case seek and rotation delays of 31ms from end to end and 27ms from outside to middle, the time to read both a 1/2 MByte block from the outer half of the disk and a 1/8 Mbyte secondary from the inner half is 162ms. Reading them both from near the inner half of the disk (the worst case without the Tiger inner/outer optimization) would take 201ms.

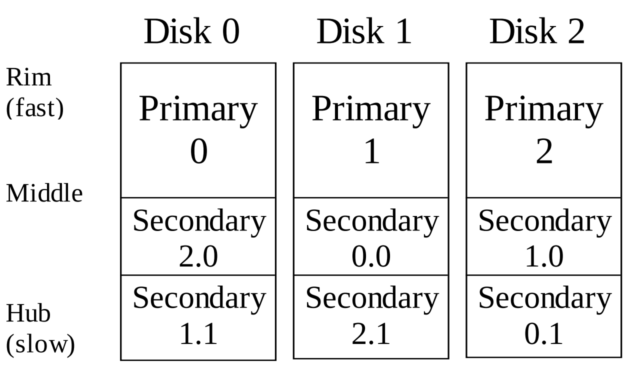

Figure 3: Disk Layout, Decluster 2

Figure 3 shows the layout of a three disk Tiger system with a decluster factor of 2. Here, "Primary n" is the primary data from disk n, while "Secondary n.m" is part m of the secondary copy of the data from disk n. Since d=2, m is either 0 or one. In a real Tiger layout, all secondaries stored on a particular disk (like 2.0 and 1.1 stored on disk 0) are interleaved by block, and are only shown here as being stored consecutively for illustrative purposes.

If a disk suffers a failure resulting in permanent loss of data, Tiger will reconstruct the lost data when the failed disk is replaced with a working one. After reconstruction is completed, Tiger automatically brings the drive on line.

The failure of an entire cub is logically similar to losing all of the disks on the cub at the same time. The only additional problems are detecting the failure and dealing with connection cleanup, as well as reconnection when the cub comes back on line. Failure detection is accomplished by a deadman protocol: each cub is responsible for watching the cub to its left and sending periodic pings to the cub to its right. Because drives are striped across cubs (i.e., cub 0 has drives 0, n, 2n, etc., where n is the number of cubs), drives on one cub do not hold mirror copies of data for another drive on that cub unless the decluster factor is greater than or equal to the number of cubs, in which case the system cannot tolerate cub failures.

5. Network Support

The network components of Tiger provide the crucial function of combining blocks from different disks into single streams. In other words, Tiger's network switch performs the same function as the I/O backplane on supercomputer-based video servers: it recombines the file data that has been striped across numerous disks. Tiger systems may be configured with any type of switched network — typically either ATM or switched ethernet. ATM has the advantage of providing quality of service guarantees to video streams, so that they will not suffer degradation as other load is placed on the switch, but is not designed to handle a single data stream originating from multiple sources.

Microsoft has evangelized a multipoint-to-point ("funnel") ATM virtual circuit standard in the ATM forum. In this standard, a normal point-to-point virtual circuit is established, and after establishment other machines are able to join in to the circuit. It is the responsibility of the senders to assure that all of the cells of one packet have been transmitted before any cells of any other packet are sent. When Tiger uses an ATM network, it assures packet integrity by passing a token among the senders to a particular funnel, thus preventing multiple simultaneous senders on a funnel.

Tiger's only non-standard kernel-mode code is a UDP network software component that implements the token protocol as well as sending all of the data packets. This special component allocates the buffer memory used to hold file data and wires it down. It is able to perform network sends without doing any copying or inspection of the data, but rather simply has the network device driver DMA directly out of the memory into which the disk places the file data. Because there is no copying or inspection of Tiger file data, each data bit passes the I/O and memory busses exactly twice: once traveling from the disk to the memory buffer, and then once again traveling from memory to the network device.

6. Measurements

We have built many Tiger configurations at Microsoft, running video and audio streams at bitrates from 14.4Kbits/s to 8Mbits/s and using several output networks ranging from the internet through switched 10 and 100 Mbit/s ethernet to 100Mbit/s and OC-3 (155Mbit/s) ATM. This paper describes an ATM configuration set up for 6Mbit/s video streams. It uses five cubs, each of which is a Gateway Pentium 133 MHz personal computer with 48 Mbytes of RAM, a PCI bus, three Seagate ST32550W 2Gbyte drives and a single FORE systems OC-3 ATM adapter. The cubs each have two Adaptec 2940W SCSI controllers, one controller having two of the data disks and the boot disk, and the other having one data disk. The data disks have been formatted to have a sector size of 2Kbytes rather than the more traditional 512 bytes. Larger sectors improve the drive speeds as well as increasing the amount of useful space on the drive. The Tiger controller machine is a Gateway 486/66. It is not on the ATM network and communicates with the cubs over a 10Mbit/s Ethernet. The controller machine is many times slower than the cubs. The cubs communicate among themselves over the ATM network. Ten 486/66 machines attached to the ATM network by 100Mbit/s fiber links serve as clients. Each of these machines is capable of receiving several simultaneous 6Mbit/s streams. For the purpose of data collection, we ran a special client application that does not render any video, but rather, simply makes sure that the expected data arrives on time. This client allows more than one stream to be received by a single computer.

This 15 disk Tiger system is capable of storing slightly more than 6 hours of content at 6Mbits/s. It is configured for .75Mbyte blocks (hence a block play time of 1s) and a decluster factor of 4. According to our measurements, in the worst case each of the disks is capable of delivering about 4.7 primary streams while doing its part in covering for a failed peer. Thus, the 15 disks in the system can deliver at most 70 streams. Each of the Fore ATM NICs is able to sustain 17 simultaneous streams at 6Mbits/s because some bandwidth is used for control communication and due to limitations in the FORE hardware and firmware. Since 20% of the capacity is reserved for failed mode operation, the network cards limit the system to 68 streams. We measured the PCI busses on the cubs delivering 65 Mbytes/s of data from disk to memory; they are more than fast enough to keep up with the disk and network needs in this Tiger configuration. Therefore the NICs are the system bottleneck, giving a rated capacity of 68 streams.

We ran two experiments: unfailed and failed. The first experiment consisted of loading up the system with none of the components failed. The second experiment had one of the cubs (and consequently all of its disks) failed for the entire duration of the run. In each of the experiments, we ramped the system up to its full rated capacity of 68 streams.

Both consisted of increasing the load on the server by adding one stream at a time, waiting for at least 100s and then recording various system load factors. Because we are using relatively slow machines for clients, they are unable to read as many streams as would fit in on the 100Mbit/s ATM lines attached to them. To simulate more load, we ran some of the clients as “black holes” — they receive more data than they can process, and ignore all incoming data packets. The other clients generated reports if they did not see all the data that they expected.

The system was loaded with 24 different files with a mean length of about 900 seconds. The clients randomly selected a file, played it from beginning to end and repeated. Because the clients’ starts were staggered and the cubs’ buffer caches were relatively small, there was a low probability of a buffer cache hit. The overall cache hit rate was less than 1% over the entire run for each of the experiments. The disks were almost entirely full, so reads were distributed across the entire disks and were not concentrated in the faster, outer portions.

The most important measurement of Tiger system performance is its success in reliably delivering data on time. We measured this in two ways in our experiments. When the server does not send a block to the network because the disk operation failed to complete on time or because the server is running behind schedule it reports that fact. When a non-black hole client fails to receive an expected block, it also reports it. In the no failure experiment, neither the server nor the clients reported any data loss among the more than 400,000 blocks that were sent. In the test with one cub failed, the server failed to place 94 blocks on the network and the clients reported a comparable number of blocks not arriving for a lost block rate just over 0.02%. Most of these undelivered blocks happened in three distinct events, none of which occurred at the highest load levels. The system was able to deliver steams at its rated capacity without substantial losses.

In addition to measuring undelivered blocks, we also measured the load on various system components. In particular, we measured the CPU load on the controller machine and cubs, and the disk loading. CPU load is as reported by the perfmon tool from Microsoft Windows NT. Perfmon samples the one second NT CPU load average every second, and keeps 100 seconds of history. It reports the mean load from the last 100 seconds. For the cubs, the mean of all cubs is reported; the cubs typically had loads very close to one another, so the mean is representative. Disk loading is the percentage of time during which the disk was waiting for an I/O completion.

A fourth measurement is the startup latency of each stream. This latency is measured from the time that the client completes its connection to the Tiger to the time that the first block has completely arrived at the client. Because the block play time is 1 second and the ATM network sends the blocks at just over the 6 Mbit/s rate, one second of latency in each start is due to the block transmission. Clients other than our test program could begin using the data before the entire block arrives. In this configuration the block service time is set to 221ms so as not to overrun the network, so that much time is added to the startup latency in order to read the block from the disk. Therefore, the smallest latency seen is about 1.3s. The measured latencies are not averages, but rather actual latencies of single stream starts, which explains why the curve is so jumpy. We do not expect that Tiger servers would be run at very high loads because of the increased startup latency, so the latencies at the highest loads are not typical of what Tiger users would see.

Figures 4 and 5 show the measured numbers for the normal operation and one cub failed tests, respectively. The three mean load measurements should be read against the left hand *y*-axis scale, while the startup latency curve uses the right hand scale.

An important observation is that the machine's loads increase as you would expect: the cub's load increases linearly in the number of streams, while the controller's is relatively constant. Most of the work done by the controller is updating its display with status information.

Even with one cub failed and the system at its rated maximum, the cubs didn't exceed 50% mean CPU usage. Tiger ran its disks at over 75% duty cycle while still delivering all streams in a timely and reliable fashion. Each disk delivered 4.25 Mbytes/s when running at load.

At the highest load, 68 streams at 6 Mbits/s were being delivered by 4 cubs. This means that each cub was sustaining a send rate of 12.75 Mbytes/s of file data, and 25.5 Mbytes/s over its I/O bus, as well as handling all appropriate overheads. We feel confident that these machines could handle a higher load, given more capable network adapters.

7. Related Work

Tiger systems are typically built entirely of commodity hardware components, allowing them to take advantage of commodity hardware price curves. By contrast, other commercial video servers, such as those produced by Silicon Graphics[Nelson et al. 95] and Oracle[Laursen et al. 94], tend to rely on super-computer backplane speeds or massively parallel memory and I/O systems in order to provide the needed bandwidth.

These servers also tend to allocate entire copies of movies at single servers, requiring that content be replicated across a number of servers proportional to the expected demand for the content. Tiger, by contrast, stripes all content, eliminating the need for additional replicas to satisfy changing load requirements.

Also, while other systems tend to rely on RAID disk arrays[Patterson et al. 88] to provide fault tolerance, Tiger will transparently tolerate failures of both individual disks and entire server machines. As well as striping across disks in an array Tiger stripes data across server machines.

[Berson et al. 94] proposes an independently developed single-machine disk striping algorithm with some similarities to that used by Tiger.

8. Conclusions

Tiger is a successful exercise in providing data- and bandwidth-intensive computing via a collection of commodity components. Tiger achieves low cost and high reliability by using commodity computing hardware and a distributed, real-time fault tolerant software design. Striping of data across many loosely connected personal computers allows the system to handle highly unbalanced loads without having to replicate data to increase bandwidth. Tiger's scheduling algorithm is able to provide very high quality of service without introducing unacceptable startup latency.

Acknowledgments

Yoram Bernet, Jan de Rie, John Douceur, Craig Dowell, Erik Hedberg and Craig Link also contributed to the Tiger server design and implementation.

References

[Berson et al. 94] S. Berson, S. Ghandeharizadeh, R. Muntz, X. Ju. Staggered Striping in Multimedia Information Systems. In *ACM SIGMOD '94*, pages 79-90.

[Knuth 73] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, pages 520-521. Addison-Wesley, 1973.

[Laursen et al. 94] Andrew Laursen, Jeffrey Olkin, Mark Porter. Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data. In *ACM SIGMOD '94*, pages 470-477.

[Nelson et al. 95] Michael N. Nelson, Mark Linton, Susan Owicki. A Highly Available, Scalable ITV System. In *SOSP 15*, pages 54-67.

[Patterson et al. 88] D. Patterson, G. Gibson, R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *ACM SIGMOD '88*, pages 109-116.

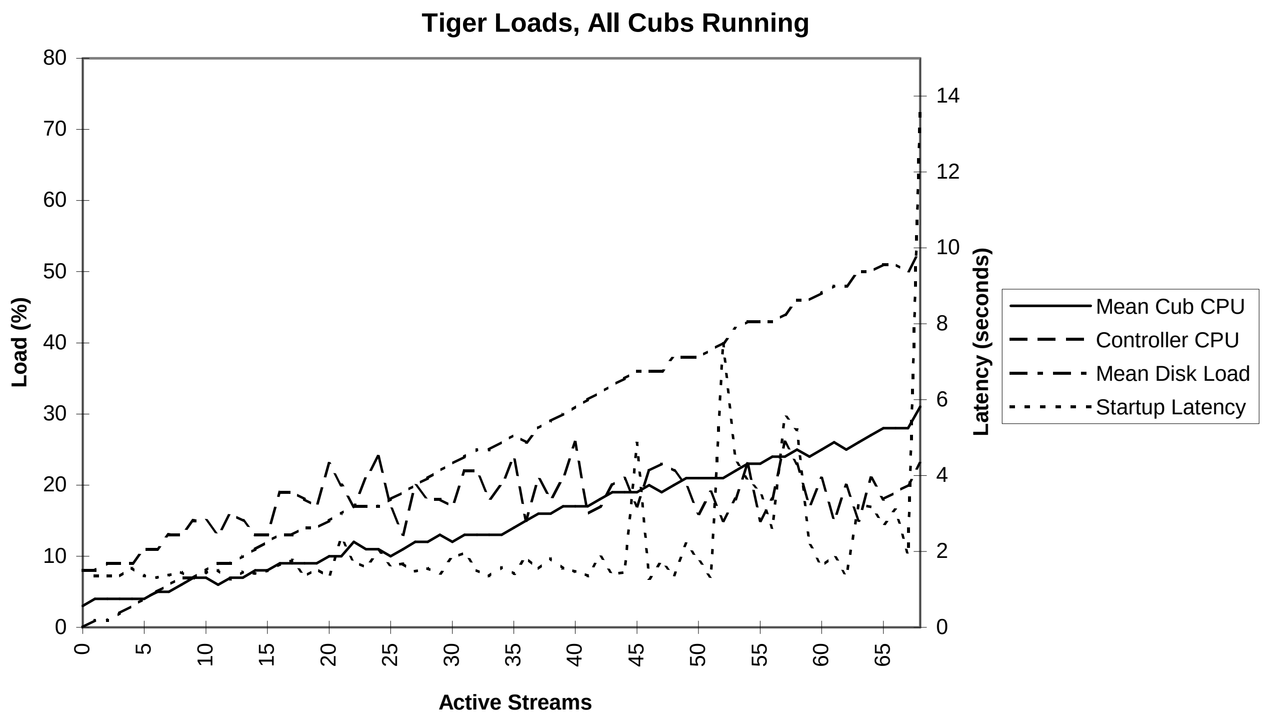

Figure 4: Tiger loads during normal operations

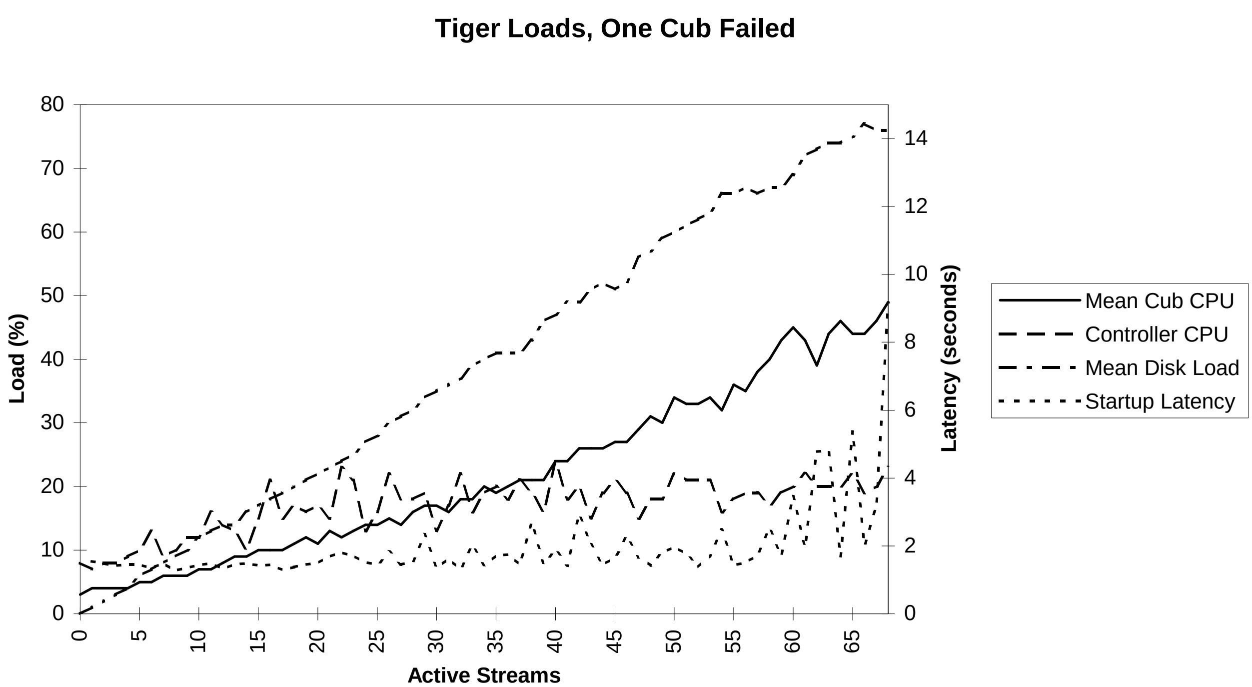

Figure 5: Tiger loads running with one cub of five failed and a decluster factor of 4