"Don't hide power"

Peter Bosch, Sape J. Mullender
University of Twente
Enschede, Netherlands

Abstract

A standard PC today has an I/O throughput in excess of 100MBps. It can, therefore, be used as a continuous-media server. Since PC hardware is also cheap, it is surprising that not many continuous-media servers today are built on PC hardware.

This paper describes two things. First, we give a summary of I/O measurements on a 200MHz Pentium-Pro based machine. Second, we present the design and implementation of a continuous-media server called *Clockwise* whose design was partially dictated by the measured characteristics of the hardware.

1 Introduction

A typical PC, in its standard configuration, has a 33Mhz 32-bit PCI bus making I/O speeds possible of up to 126MBps. Intrigued by this large performance number we performed I/O performance experiments on such a machine to learn if the theoretical performance is practically feasible.

The reason for performing the measurements was to learn whether an Intel-based Personal Computer can be used as a continuous-media server machine. Initially we set our goal to the simultaneous recording and playback of 20 media streams of between one and eight Mbps[1]. This choice was based on our network technology (155 Mbps ATM). However, when we started measuring, we quickly learned that we were too conservative: When (well-designed) 622 Mbps ATM network interface cards become available, we claim that we can send up to 60 eight-Mbps media streams through the network.

By performing the I/O experiments, we learned how to organize system software such that media (or other I/O hungry) applications are offered an efficient I/O data path. The key issue is that applications and system software are organized such that the CPU is never in the data path of bulk data transfers; DMA devices are much better at moving data through a machine. One of the design principles for the continuous-media server, therefore, became to avoid data copying at all cost. Hence, we are interested in zero-copying system software.

The remainder of this paper is organized as follows. Section 2 summarizes the performance measurements. The actual measurement results and analysis will be made available soon [1]. Next, Section 3 describes Clockwise, the continuous-media server we have constructed that allows media applications a direct data path to the hardware. Section 4 summarizes this paper.

2 I/O measurements

To learn of the issues in transferring large quantities of data through a machine, a PC with a 200Mhz

* This work is funded by the PEGASUS-II project (ESPRIT LTR-21917).

The phrase "Don't hide power" was used by Butler Lampson in [6]

† Contact: Peter Bosch <peterb@huygens.org>

[1] Throughout, we use MBps to indicate megabytes per second and Mbps to indicate megabits per second.

Pentium-Pro processor was set up as experimentation machine[2].

The first experiment was to find out the sustained performance of a single disk. For this, we equipped our machine with a NCR53c810a Fast SCSI-II controller and we attached a 7,200 RPM Quantum Atlas-II disk to the SCSI bus. The Fast SCSI-II bus is a 40MHz bus, and uses a minimum of four cycles per byte transferred. This means that this bus has a maximum throughput of 9.5 MBps. The attached Quantum Atlas-II has a maximum throughput of 9.8 MBps which basically means that this disk cannot be used optimally. We learned that the measured performance of the disk is 8.7 MBps for reading a megabyte from the outer zone and 9.3 MBps for writing it back. The difference of 0.6 MBps is caused by a disk read policy: writes (of full tracks) are implemented more efficiently than reads.

Next, we equipped the measurement machine with several NCR53c810a SCSI-II bus controllers, each with a private Quantum Atlas-II disk. The goal of this measurement was to learn if aggregate disk performance scales linearly with the number of disks. We ran the same measurements as for the single disk setup, this time by starting all controllers at the same time.

This measurement showed that near-linear speedups happen with up to three disks for writing and up to four disks for reading. The throughput for reading two disks is 2.0 times that of one, for three disks the number becomes 2.95 and for four disks it is 3.9. For writing these numbers are 2.0, 2.9, and 3.4.

The reason for the worse-than-linear speedup of disk writes of more than three disks is due to contention on the PCI bus and memory. When data are read on a Pentium Pro, the host memory controller (the Intel 440FX [2]) first makes sure that none of the CPUs has dirty copies of the requested memory address. This verification consumes approximately 340 nanoseconds for the first memory references in a PCI transaction. Later memory references in the same PCI transaction are performed in streaming mode. Given that the NCR53c810a transfers at most 64 bytes per PCI transaction approximately 40% of the PCI bandwidth is wasted on this verification. The maximum achievable performance with NCR53c810a controller is approximately 34MBps, which resembles the measured performance.

The NCR53c875 Ultra SCSI PCI [5] controller has a maximum PCI transaction of 512 bytes. This means that the time to perform the dirty data verification can be divided over 4 times as many bytes, which reduces the overhead substantially. Theoretically, the NCR53c875 controllers can transfer data with speeds up to 114 MBps.

We repeated the multi-disk experiments with one to four NCR53c875 Ultra SCSI controllers, each equipped with between one to three 10,000 RPM Seagate Cheetah disks which have a sustained (measured) transfer rate of 13.1 MBps. We performed two experiments. The "3" experiments assigned disks naively to the available SCSI controllers: the first three disks were assigned to the first SCSI controller, the next three disks to the next SCSI controller, et cetera. Since the (theoretical) performance of three Seagate Cheetahs is more than the Ultra SCSI bus can sustain, the experiment "2+1" assigned the first two disks to the first SCSI controller, before using the next SCSI controller. To measure the performance of nine or more disks in the "2+1" configuration, the third disks on each SCSI controller was used.

Figure 1 shows the performance of the disks under the two experiments. We measured a maximum performance of 100.1 MBps for reading a megabyte from 12 parallel disk and 90.0 MBps for writing a megabyte to 12 parallel disks. It turns out that up to approximately 80 MBps (8 disks), reads and writes perform equally well depending on the disk allocation strategy. The Figure shows that, as expected, the "2+1" strategy performs better for the first 8 disks with a performance improvement of approximately 10MBps.

However, when the ninth disk is assigned to the first string, performance of the "2+1" configuration quickly deteriorates below the "3" configuration. We think this is caused by contention on the PCI bus. We have simulated the behavior of the PCI bus transfers for the two configurations. First,

[2] All measurements were performed on Nemesis, a kernel with a continuous media scheduler, which is built as a part of the Esprit Pegasus project by the University of Cambridge and a Linux/FreeBSD/NetBSD NCR SCSI device driver that was ported to the Nemesis kernel.

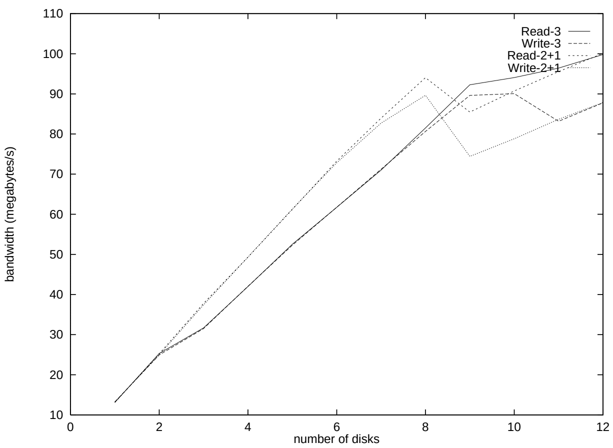

Figure 1: Seagate Cheetah performance

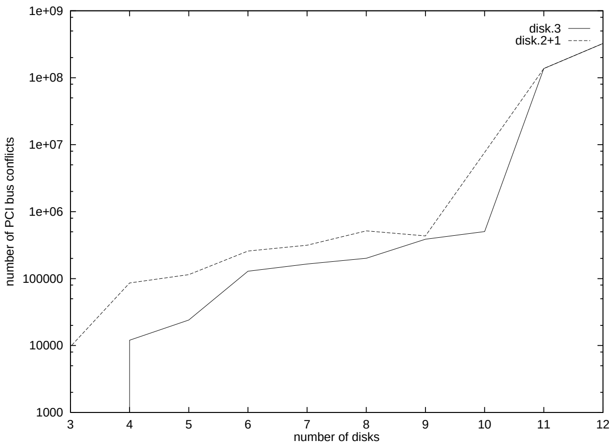

Figure 2: PCI bus conflicts

we defined a bus conflict when two or more SCSI controllers want to access the PCI bus simultaneously and next, we simulated the execution of a large number of MB reads and writes on the parallel disks.

Figure 2 shows the number of bus conflicts for both configurations. It shows that for 9 disks there are slightly more conflicts on the PCI bus, and for 10 disks there are an order of magnitude more conflicts on the PCI bus. Further, when 9 disks are used, the “3” configuration only uses 3 controllers, while the “2+1” configuration uses 4 controllers. We think that the combination of these two effects are the reason for the dramatic performance drop of the “2+1” configuration.

In any case, the key to attain such I/O speeds is by using the DMA devices extensively. For all experiments, the CPU has never been in the data path. As has been shown by McVoy et al. [9], including the CPU in the data path can substantially hinder the data movement through a machine.

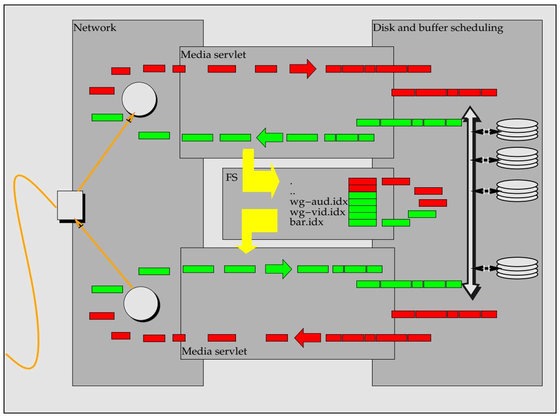

Figure 3: Clockwise’s architecture

3 Clockwise

Clockwise is a continuous-media file server that allows media applications (or other I/O hungry applications) to take advantage of the hardware capabilities of the underlying machinery. The structure of Clockwise is such that when transferring data between networks and storage devices (critical devices of a continuous-media server), the CPU is not required for bulk data transfer.

The structure of Clockwise is shown in Figure 3. This Figure shows that a Clockwise hierarchy consists of three layers: the disk layer, the application layer and the network layer. The disk layer is responsible for the scheduling, storage and retrieval of so-called dynamic partitions. The application layer (servlets), is responsible for interpreting the data stored in the dynamic partitions and the network layer is responsible for transferring data on and off the network.

Dynamic partitions in Clockwise are a way of grouping related blocks (of a megabyte) together, much like the grouping of blocks in Loge [4] and Logical Disk [3]. Dynamic partitions are administrated by the disk layer in a partition table. This partition table lists for each dynamic partition which blocks are allocated to the dynamic partition and on which disk the blocks are allocated. The placement of blocks on disks can be determined by the applications themselves.

An application can "open" a partition to read and write blocks from the dynamic partition with a set of Quality of Service parameters. These parameters specify the average bandwidth with which the application transfers data and the amount of buffering it requires to operate. If, for example, a constant bit rate video stream is played from a Clockwise dynamic partition, the bandwidth parameter represents the bit rate of the stream. Also, the application presents how many transfer buffers it requires to maintain a constant flow of data (e.g. by double buffering). The Clockwise scheduler uses the bandwidth parameters for a schedulability test, and to order the requests in an EDF queue [7].

Once an application has opened a dynamic partition, the Clockwise disk layer provides the application with an efficient data path to the I/O hardware. An application can present work to the disk layer, and the only thing Clockwise does is to verify the memory address(es) that are presented by the user application, to assign physical disk addresses to the request based on the user supplied dynamic partition address and to start the I/O devices when the operation is due – Clockwise does not touch the bulk data.

Clockwise also manages the physical memory on the Clockwise server machine. The reason for this is two-fold; First, each application requires memory to run and secondly, Clockwise needs to protect itself against malicious clients. It is expected that two types of applications run on a Clockwise server: applications that require vast amounts of memory such as Unix-like file systems and media applications that require memory for buffering of media data. Since physical memory is allocated through Clockwise, Clockwise can mediate between memory requests from both type of clients.

The second reason to implement a memory manager inside Clockwise is to protect Clockwise (and other applications) against malicious clients. Since applications pass data pointers to Clockwise that are directly inserted in the SCSI chips, Clockwise needs to verify the validity of the address. For this, Clockwise keeps a list of valid addresses per client application.

Clockwise is currently available with a few applications: a file-system application and a media application. The file-system application implements a number of existing file systems: McKusick's FFS [8], Lunis' EXT2FS, and an implementation of BSD LFS [10, 11], each with their own NFS server. The media application allows the recording and playback of digitized audio and video data.

3.1 Media applications

The media application communicates with the Clockwise server to store and retrieve so-called media blocks from disks. Media blocks hold the raw digitized and possibly compressed audio and video. It is quite likely that a media contains many media blocks and the collection of media blocks is stored in a Clockwise dynamic partition.

When a media application is activated, it provides a number of Quality of Service parameters to the network, CPU, disk and memory. The network QoS and CPU QoS are negotiated with the network driver and the host operating system, respectively. The disk and memory Quality of Service parameters are presented to Clockwise. Clockwise allocates disk bandwidth by applying the aforementioned schedulability tests, and it allocates a number of memory buffers.

We use Fore Systems AVAs and ATVs to record and render digitized audio and video. An AVA is device that connects to an ordinary (CCD) camera on one side and to an ATM link on the other side. It digitizes the analogue audio and video and it is able to compress the video by using motion-JPG compression [12]. Once activated, an AVA produces a continuous flow of AAL5 packets with digitized audio and video. An ATV is the reverse device: it renders all AAL5 packets it receives on a television set. Note that the AVA and ATV do not use any synchronization: they simply digitize the current frame or render the current ATM input.

The AVA does not digitize entire video frames, but rather a number of scan-lines. A possible configuration is that the AVA outputs an AAL5 ATM packet whenever it has digitized a small number of scan-lines, a few *tiles* of video. The advantage of this approach is that end-to-end latency between source and sink is minimized: rather than having to wait for the transmission of the entire digitized frame, a rendering device can update the screen whenever an AAL5 packet has been received with only a few scan lines.

The disadvantage of the transmission of the small number of scan lines is that AAL5 packets can be quite small when motion-JPG compression is enabled. Consider for example a configuration where the 768x288 television screen is transmitted as 72 AAL5 packets (a common configuration for the AVA) to a Clockwise server. This means that every 384x8 pixels are transmitted as a separate AAL5 packet, with an uncompressed size of approximately 3KB. If, however, motion-JPG is enabled, the resulting packet size can be between 200 bytes and 3KB.

To avoid copying of data to store the received media consecutively on disk, Clockwise allows the reading and writing of data through scatter-gather lists. Rather than pointing to a single large buffer, a scatter-gather lists points to a number of (small) buffers that together form a dynamic partition block. Since the underlying hardware is capable of dealing directly with such scatter-gather lists, Clockwise only makes sure that the presented data pointers are part of the Clockwise memory before it transmits the list to the hardware.

Index information to the raw AVA media data are required to allow later interpretation of the media dynamic partition. The information consists of three types: a time track, a tile track and a frame track. The time track records the relative time of a frame within the movie. The tile track records the sizes of the tiles within a frame and the frame track records the addresses of independent frames within the movie. When an AVA movie is recorded, the AVA media application generates these index tracks on the fly. The reason for recording a separate tile index file is because the tiles can have a varying size when motion-JPG is recorded.

The index files themselves are stored on a Unix-like file system (described below) and can be accessed by clients through the NFS interface.

3.2 File system applications

There exist a large number of Unix-like file systems today that all optimize for different characteristics. McKusick's Fast File System (FFS) [8] optimizes by grouping groups of cylinders and files that are located in the same user directory are stored in the same cylinder group. If a directory is read, the disk heads are already located at the correct cylinder for reading a file from that directory.

Rosenblum's Log-structured File System (LFS) [10] recognizes that by using large read caches, disks are dominantly used for write operations. By optimizing write accesses to disk, the entire file system is faster. Writes can be optimized by only performing batches of writes to the end of a file-system log. Every once in a while, the log is cleaned and unaccessible data is removed.

It is a tremendous task to make each file system type available under Clockwise. Fortunately, the NetBSD[3] Unix-like system implements a *virtual-file system (VFS)* with a number of Unix-like file systems incorporated. By making VFS available under Clockwise, each of the incorporated file systems are also available under Clockwise. Currently the following file systems are incorporated: McKusick's FFS, Lunis' EXT2FS, and an implementation of BSD LFS [11].

VFS is part of the Unix-like kernel, while Clockwise is part of the Nemesis kernel. Since Nemesis is radically different from Unix-like systems, an interface layer is required that maps Unix-like kernel procedure calls onto Nemesis primitives. This layer implements procedures like sleep and wakeup, and routines to emulate calls to disable interrupts that are available inside the Unix-like kernels. Further it provides an interface to Clockwise that is inserted into VFS just like a Unix-like device driver.

File system caching in VFS is implemented by a simple LRU based block cache. Clockwise VFS is made such that it allocated memory buffers for caching from the Clockwise server rather than calling into the Unix-like memory allocator. Each VFS tries to allocate as much memory as possible to increase the number of cache hits and to reduce the number of disk operations. However, when Clockwise needs memory for new clients, or when a VFS uses too much memory in comparison with other clients, Clockwise may re-possess earlier allocated

[3] NetBSD is a public domain implementation of the Berkeley BSD4.4 definition.

memory buffers.

To allow remote clients to use a VFS, it can be NFS-mounted by remote clients.

3.3 Status

Clockwise has been implemented and is currently used for experiments. Since Clockwise does not copy bulk data once, we are confident that the measured application performance is close to the raw bandwidth as is presented in Section 2.

The use of standard file-system software as a user application on top of Clockwise has been proven invaluable: not a single file has been lost since the system was declared finished. The integration of an NFS server helped debugging the media applications since media index data could be processed by using standard Unix utilities on Unix machinery.

4 Summary

This paper shows two things: (1) a PC today can be used for applications that produce or consume high I/O loads and (2) this I/O performance can easily be "given" to applications by carefully structuring the system software. To demonstrate the I/O capabilities of PCs a number of I/O performance experiments have been performed. These measurements show that disk I/O performance is not the limiting factor in a PC, but rather network bandwidth and bus-centered devices now form a serious bottleneck.

By constructing a storage server that allows applications to exploit the DMA capabilities of the physical hardware, applications can be constructed elegantly and hardware can be used efficiently. The elegance is from the fact as far as applications are concerned, they communicate almost directly to the hardware and can use dynamic partitions as if they are using a raw disk. Clockwise is efficient because it does not introduce unnecessary data copying in the I/O path. Lastly, since the CPU is not in the bulk data path anymore, it can be used for other file system operations, such as off-line compression and decompression of media data. Clockwise does not "hide the power."

References

[1] Peter Bosch. PC I/O Measurements. Huygens report 98-XX. University of Twente, dpt. of Computer Science, Februari 1998.

[2] Intel Corporation. *Intel 440FX PCI set 82441FX PCI and memory controller (PMC) and 82442FX Data Bus Accelerator (BDX)*, May 1996.

[3] Wiebren de Jonge, M. Frans Kaashoek, and Wilson C. Hsieh. Logical disk: a simple new approach to improving file system performance. Technical report IR-325. Dept. of Mathematics and Computer Science,Vrije Universiteit, Amsterdam, 1993. Also MIT/LCS/TR-566 at MIT.

[4] Robert M. English and Alexander A. Stepanov. Loge: A Self-Organizing Disk Controller. *USENIX Conference Proceedings* (San Francisco, CA), pages 237–52. USENIX, Winter 1992.

[5] Symbios Logic Inc. *SYM53C875 PCI-Ultra SCSI I/O Processor*, June 1995.

[6] Butler W. Lampson. Hints for Computer System Design. *Proceedings of 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, New Hampshire). Published as *Operating Systems Review*, **17**(5):33–48, October 1983.

[7] C.L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, **20**(1):46–61, Januari 1973.

[8] Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, **2**(3):181–97, August 1984.

[9] Larry McVoy and Carl Staelin. lmbench: Portable Tools for Performance Analysis. *1996 Winter Usenix conference* (San Diego, January 1996), pages 279–94. Usenix Association, January 1996.

[10] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *Proceedings of the 13th ACM Symposium on Operating Systems Principles* (Pacific Grove CA (USA)), volume 25, number 5 of Operating Systems Review, pages 1–15, October 1991.

[11] Margo Ilene Seltzer. *File system performance and transaction support*. PhD thesis. University of California, 1992.

[12] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, **34**(4):30–44, April 1991.