Experimental Evaluation of PFS Continuous Media File System *

Wonjun Lee, Difu Su, Duminda Wijesekera, Jaideep Srivastava,
Deepak Kenchammana-Hosekote† and Mark Forest‡
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455,
IBM Almaden Research Center, San Hose CA 94086†,
Rome Laboratory, Griffs Air Force Base, Rome NY 14853‡.
{WJLEE,SDF,WIJESEK,SRIVASTA, KENCHAM†}@CS.UMN.EDU, FORESTIM@RL.AF.MIL‡

Abstract

Large sizes of continuous media files, e.g. video, require multimedia applications to access data from secondary and/or tertiary storage during execution, making the file system the bottleneck of overall performance. Futhermore, such access require some application specific *Quality of Service (QoS)* parameters be met. These include retrieval rate, its variation, timing drift, average and bursty errors, and the degree of synchronization. Unavilability of such QoS parameters in conventional file systems result in failures to guarantee QoS parameters and thereby take advantage of application's tolerances. This paper presents an experimental evaluation of the Presto continuous media file system (PFS), that has been implemented in the context of a distributed multimedia application development environment that has been prototyped.

1 Introduction

The Unix File System (UFS) [19] has been the landmark achievement in file system design, and practically every modern file system borrows heavily from it.

1.1 Inadequacy of Current File Systems for Continuous Media

While the mechanisms provided by the UFS have been sufficient for most applications, there are important classes of applications where it is not been so, e.g. [21]. An increasingly important class of applications where UFS is not suitable are those requiring storage and retrieval of *continuous media (CM)*, i.e. audio, video, animation, etc. [20]. Following are some unmet file system needs of continuous media,

- Treatment of files as a sequence of bytes by current file systems preclude modeling and thereby preventing optimized timely retrieval of time sensitive CM data such as audio and video. Untimely jittery retrieval cause user dissatisfaction [20], and in conventional file systems these increase with load.
- Given the high level of redundancy and temporal autocorrelation of continuous media, its human consumption is very tolerant to controlled errors. The system should be able to understand specifications of acceptable tolerances, also called *Quality of Service*, and take advantage of them in resource management.

1.2 Design Issues in Continuous Media File Systems

Based on the above observations, the following issues must be addressed in designing a continuous media file system:

- *Placement and Storage Structures:* The placement of CM data should be such that it is capable of dealing with large sizes with real time retrieval needs that are embodied in QoS metrics.
- *Scheduling:* The goal of disk/server scheduling for continuous media is to satisfy QoS requirements by meeting deadlines of periodic I/O requests generated by some server resident CM stream manager with minimum buffers [11].

The key to designing high performance file systems to support CM requires that we use information about (i) the inherent temporal nature of continuous media, and (ii) QoS oriented nature of media access, to develop new solutions to these problems.

1.3 Related Work & Contribution

The last few years have seen a growing degree of interest in I/O issues for multimedia in general, and CM in particular [20, 6]. The initial work in the area, e.g. [1, 5, 13, 23], focused on the principles behind continuous media storage and retrieval, and led to the development of the *incremental retrieval model*. Recent work, e.g. [10], has developed the model more completely and used analytical techniques to elucidate the behavior of the model. A number of investigations, e.g. [3, 2, 17, 9], have developed techniques for continuous media access, and have carried out their evaluation, usually by means of simulation. Similarly, work in the area of file placement includes [18, 4, 16, 10], where different approaches to storing continuous media data have been proposed.

*This work is supported by Air Force contract number F30602-96-C-0130 to Honeywell Inc, via subcontract number B09030541/AF to the University of Minnesota.

CIKM '97 Las Vegas Nevada USA
Copyright 1997 ACM 0-89791-970-x/97/11..$3.50

In some cases simulation-based evaluation has been performed.

While some good incremental retrieval models exist, and a number of techniques for CM storage and access have been developed, there is a serious dearth of actual file system implementations which are suited for a broad range of multimedia applications. A number of efforts have focused on developing I/O storage and management systems for CM to fulfill the needs of specific applications, e.g. [22, 4]. Conversely, given the fact that *under low resource utilization whether the underlying system is cognizant of an application's real-time needs or not does not matter*, there have been efforts to use high-performance general purpose file systems to manage CM data, as evidenced by the Tiger-Shark file system [7] which has shown good performance in some multimedia applications. In our opinion this approach suffers from two problems. First, it is not scalable since keeping the load low means building the system with very high capacity. Second, such a system will have a seriously unacceptable price-performance ratio.

Overall, we believe that a continuous media file system *must* use information about the inherent nature of continuous media in all its functions to ensure both high performance and a good price-performance ratio. Hence, it must be built on the principles of incremental retrieval, which is emerging as the model to capture the inherent nature of continuous media. To our knowledge, the file systems that fit this criteria are [15, 14], where some principles of the incremental retrieval model have been used, especially in the design of the admission control strategy. However, one issue that is still lacking from any study we've seen is the evaluation of any file system from the viewpoint of QoS metrics. After all, a continuous media application's performance (and possibly correctness) needs are expressed in terms of its QoS specification, and hence the underlying system's performance must be measured in terms of how successful it is in attaining the specified QoS. In the multimedia networking community this is today the accepted way of evaluating communication protocols for continuous media. We believe that the file system community must do the same.

To the best of our knowledge, our study is the first one to evaluate file system performance for managing continuous media in terms of how well the specified QoS is attained. We have compared the experimental behavior of the Unix file system, as instantiated in the Solaris 2.5 operating system, with that of the Presto file system. The experimental data used conformed to that specified for motion JPEG video. Our principal findings are [12]:

- For general file operations, i.e. non-continuous media access, the maximum aggregate throughput obtained from PFS was about 80% more than that for UFS. While the throughput of either file system increases by increasing the size of the block accessed from the disk in each read (at the expense of having larger memory buffers), the range in which PFS can take advantage of this is much larger than that for UFS.
- For motion JPEG data, where the mean size of a frame is about 18KB, where PFS outperforms UFS by about 25%.
- For QoS parameters related to time drifts, PFS outperforms UFS by a huge margin.
- For QoS parameters related to frame loss, the loss suffered by PFS is less than one third of that suffered by UFS.

The structure of this paper is as follows: in section 2 we describe the architecture of the Presto file system and provide details of its implementation. In section 3 we provide an overview of the Quality of Service model used. Section 4 describes our experimental comparison of PFS and UFS for basic file operations, while section 5 does the same for continuous media access. In section 6 we conclude the paper.

2 Architecture of PFS and its Implementation

This section describes design objectives and requirements of PFS, and compares it with conventional file systems, e.g. UFS.

2.1 Requirements and Design Objectives

To handle continuous media (CM) efficiently we need to store and retrieve large amounts of multimedia information with continuous playback, providing user-specified QoS. The CM file system should provide larger data unit abstractions, such as *video frame* and *audio sample group*, unlike conventional file systems, e.g. UFS, which provide only a byte-oriented abstraction. A *unit* is a user-defined logical chunk of data, e.g. a frame for video data and a sequence of audio samples for audio data (Figure 1). Henceforth, the abstraction can support CM application's retrieval and storage needs via units in a CM stream. In addition, it should allow an application to randomly seek to a unit within a stream.

Another design objective for the CM file system is to provide efficient I/O access to the disk. This objective is crucial for CM retrieval and storage where real-time continuous delivery is required and where high volume I/O bandwidth is required. To do so, new storage and access strategy should provide timely delivery to applications via optimizing accesses, buffer management, and interface with the I/O scheduler.

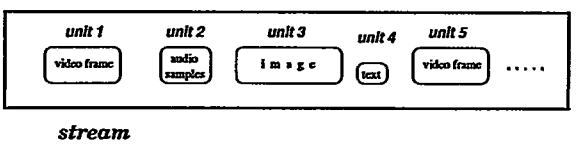

Figure 1: Unit and Stream in PFS

2.2 Comparison with UFS

Conventional storage managers such as UFS were designed for non-real-time access to byte(record)-oriented data. They have the following drawbacks [8]:

- UFS provides *byte-oriented* abstraction while CM applications (such as an application that manipulate video streams) prefer *unit(eg. a sequence of video frames)-oriented* abstraction.
- UFS is efficient in handling small size record-based file accesses, but is expensive for accessing larger files because of tree-structured index of the file system. Hence it is not appropriate for video and image data that require high volume.
- UFS does not provide functionality for real-time continuous delivery of data.
- The main buffering strategy used in UFS (i.e. LRU) may not be efficient in case sequential access is needed by CM applications.

2.3 Implementation Details

PFS is implemented on a raw disk partition of UNIX so as to by pass the UNIX block buffer cache and allow the imposition of customized access structures (Figure 2).

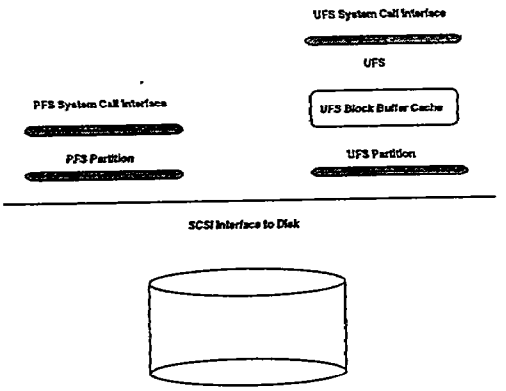

Figure 2: Disk Access via PFS and UFS

2.3.1 Disk Layout

A PFS partition is divided into one or more *extents*, and each extent consists of a number of *units*. A CM stream is stored in an extent. Information about each stream is stored in a structure referred to as the *inode*. A *super block* is used to maintain the formatting data and extent map of free extents. The overview of disk layout is shown in Figure 3.

2.3.2 Super Block

The formatting information in the super block includes: (1) number of extents in partition, (2) size of each extent in blocks, (3) size of each block in physical sectors.

The free extents in a partition are indexed by a bitmap. A first fit algorithm is used to allocate a free extent to a stream. A command called *Pformat* is used to format the raw partition and create the super block.

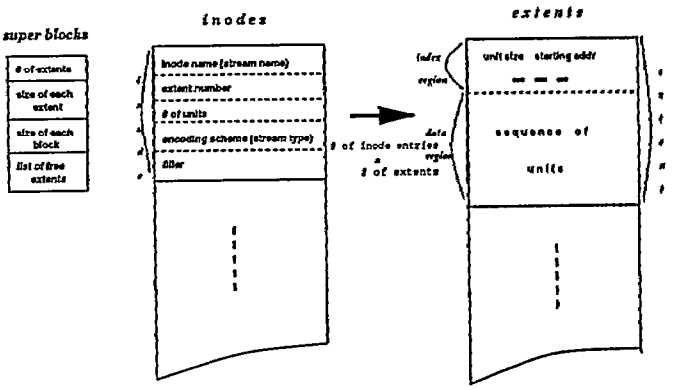

Figure 3: Disk Layout in PFS

2.3.3 Inodes

There is a unique inode assigned to each extent. Each inode is 128 bytes long, and includes the following fields: (1) name of a stream, (2) name of extent, (3) number of units in the stream, (4) stream type.

2.3.4 Extents

An *extent* consists of a number of units. The unit size is variable. Each extent is divided into two regions. The first region is the index region, storing the size and starting address of each unit. The other region is the data region storing a sequence of units, with each unit starting on a physical sector boundary.

2.4 PFS Application Programming Interfaces

PFS has a layered programming interface. In this paper, we discuss only the the stream layer interface. The stream layer interface is similar to the one in UFS. Some of the major functions in this API are as follows:

- **Pcreate**(*stream-name*): create a stream.
- **Popen**(*stream-name*): open a stream to read or write; the stream's read/write position is set to the beginning.
- **Pstart**(*stream-name, rate*): a (non-blocking) call that informs the stream server that it should start providing data from the stream at the specified rate.
- **Pclose**(*stream-name*): close a stream.
- **Pread**(*stream-name, num-units*): read the specified number of units from a stream.
- **Pwrite**(*stream-name, num-units*): write the specified number of units to a stream.
- **Pseek**(*stream-name, num-pos*): move the stream's read/write position forward by num-pos positions (backward if negative).

3 QoS Model for Continuous Media

The QoS metrics used to evaluate PFS and UFS, called the continuity parameters for CM streams in [24], are reviewed in this section.

Continuity of a CM stream is metrized by three components; namely *rate, drift* and *content.* For the purposes of describing these metrics, we envision a CM stream as a flow of frames. The ideal rate of flow and the maximum permissible deviation from it constitute our *rate* parameters. Given the ideal rate and the beginning time of a CM stream, there is an ideal time for a given frame to arrive/ be displayed. Given the envisioned fluid-like nature of CM streams, the appearance time of a given frame may deviate from this ideal. Our *drift* parameters specify aggregate and consecutive non-zero drifts from these ideals, over a given number of consecutive frames in a stream. For eg., first four frames of two example streams with their expected and actual times of appearance, are shown in Fig. 5. In the first example stream, the drifts are respectively 0.0, 0.8, 0.2 and 0.2 seconds; and accordingly it has an aggregate drift of 1.2 seconds per 4 time slots, and a non-zero consecutive drift of 1.2 seconds. In the second example stream the largest consecutive non-zero drift is 0.2 seconds and the aggregate drift is 0.3 seconds per 4 time slots. The reason for a lower consecutive drift in stream 2 is that the unit drifts in it are more spread out than those in stream 1.

In addition to timing and rate, ideal contents of a CM stream are specified by the ideal contents of each frame. Due to loss, delivery or resource over-load problems, appearance of frames may deviate from this ideal, and consequently lead to discontinuity. Our content metric is designed to measure the average and bursty deviation from the ideal specification. A loss or repetition of a frame is considered a *unit loss* in a CM stream. For a more precise definition envision the flow of CM frames as a train of slots with successive slot numbers, as given in Fig 4. Some slots may be filled with contents of frames. We define a unit loss only for slots that have some occupying contents, Suppose $s(k)$ is the media frame at slot $s(i)$ of stream s. Suppose the immediately previous non empty slot to $s(i)$ is $s(i-l)$, where $l > 0$, and it is occupied by contents of frame $F(j)$. In case there are no skips, repeats or misses, if slot $s(i)$ is occupied by media frame $F(k)$, then slot $s(i-l)$ should be occupied by contents of frame $F(k-l)$. Hence the unit loss incurred at slot $s(i-l)$ due to skips and repeats is $\|k-l-j\|$. The unit loss due to missing frames at slot $s(i)$ is $(l-1)$, precisely because there are $(l-1)$ empty slots in between slots $s(i)$ and $s(i-l)$. Hence the maximum of unit losses due to skips, repeats and misses at slot $s(i)$, say $USL(i)$, is $\max\{\|k-l-j\|, l-1\}$. Consequently, we define $\max\{\|k-l-j\|, l-1\}$ to be the *unit loss* at slot $s(i)$. In order to measure the unit loss at the beginning of a stream, we assume that every stream has a hypothetical slot $s(-1)$ with a hypothetical frame $F(-1)$.

The aggregate number of such unit losses is the *aggregate loss* of a CM stream, while the largest consecutive non-zero unit loss is its *consecutive loss.* In the example streams of Fig. 5, stream 1 has an aggregate loss of 2/4 and a consecutive loss of 2, while stream 2 has an aggregate loss of 2/4 and a consecutive loss of 1. The reason for the lower consecutive loss in stream 2 is that its losses are more spread-out than those of stream 1.

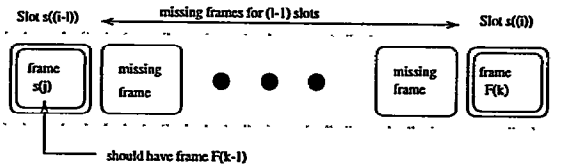

Figure 4: Unit Loss of a Stream

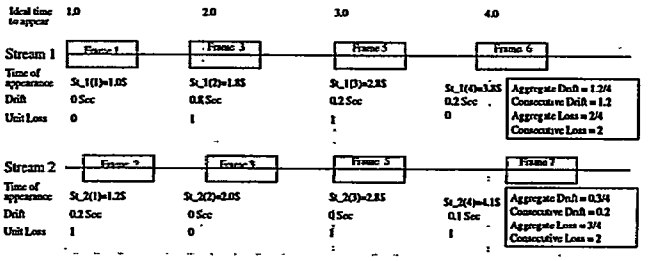

Figure 5: Streams Used to Explain Continuity Parameters

4 Comparison of PFS and UFS on Standard File Operations

In this section, we demonstrate the advantages of PFS over UFS based on experiments that use standard file operations. The experiments were carried out on a Sun Ultra Sparcstation with a 4GB SCSI disk. The focus of our experiments was to verify that for standard file operations the performance of PFS is comparable to that of UFS.

4.1 Evaluation Metrics

The metrics we consider in this section are disk throughput, buffer requirement and service cycle length.

1. **Disk Throughput :** Disk throughput is measured by the data rate of a single read operation, and its value is obtained by computing the ratio of the size of an average read to the time taken for it.

2. **Buffer Requirement:** Buffer Requirement is measured by the I/O buffer size. The data in a single read is put into I/O buffers. We call the amount of data in a single read the buffer size.

3. **Response Time and Service Cycle:** *Response time* is the time between a read request's initiation and its completion. When there is concurrent stream access in the system, the I/O schedule is divided into service cycles. In each cycle, the appropriate amount of data is fetched for each stream, and its length provides a lower bound on the response time.

Since we are interested in the performance of the disk system, it was important to ensure that the file reads were not serviced from the file cache. Hence, we used different files in each test, and chose to use each file only once to bypass the effects of system I/O buffering. To eliminate the effects of random fluctuations, we repeat each operation several times and use the average value or should it be minimum value.

4.2 Experiment 1 : The Effect of Unit Size on Disk Throughput

4.2.1 Experimental Design

In both PFS and UFS, the relationship between buffer size(b) and data access time(t) is:

$$t = (1/r_o)b + t_o \quad (1)$$

$$r = r_o/(1 + r_o t_o/b) \quad (2)$$

Here, t is the access time, r is data rate, buffer size(b) is the data size in a single read, r_o is the possible maximum data rate, and t_o is the constant overhead for a single read. In Unix, r_o depends solely on the buffer size. In PFS, r_o depends on both the unit size and the number of units in a single read.

The first experiment is designed to verify this relationship. For PFS experiments, in a single run, we fix number of units fetched in a single read, vary the unit size and measure the access time. We repeat this for different numbers of units in a single read. In UFS, we regard the whole buffer as a unit and carry out the same experiment.

4.2.2 Experimental Results

Fig.6 shows the relationship between the data rate and the unit size. The graph shows that PFS has a higher maximum possible data rate.

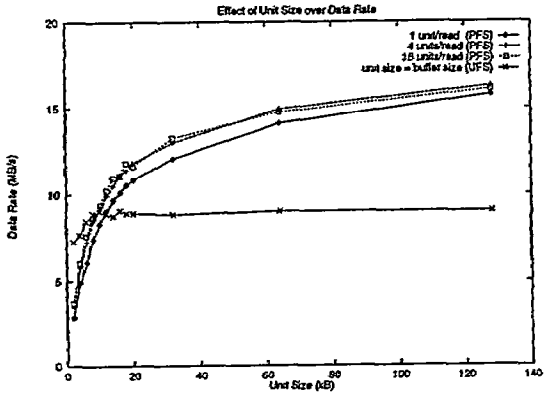

Figure 6: Effect of Unit Size on Data Rate

In PFS, data rate increases with unit size, and the more number of units per read, the higher the achievable data rate. In UFS, our experimental observations are that when the buffer size is smaller than 10KB, the data rate increases with the buffer size. When the buffer size is larger than 10KB, the data rate remains constant.

When the unit size is small (below 8KB), UFS is faster than PFS. When the unit size is above 10KB, PFS is faster than UFS. In a CM Stream, we store a frame of CM data as a unit. UFS is optimal for small frames while PFS is optimal for large frames. For 640 by 480 JPEG frames, where the size of the compressed frame is about 18KB, PFS has a higher data rate than UFS.

From Fig.6, we determine two ways to improve the throughout of PFS. One is to group a number of frames into a unit. The other is to read out more units in a single read, called *group read*. The data rate increases with group size.

4.3 Experiment 2: Effect of Concurrent Streams on Buffer Size

4.3.1 Experimental Design

When there are requests for multiple streams from PFS, they compete for I/O bandwidth and other system resources, and therefore require larger buffers to handle the context switch overhead. Hence, to maintain a fixed data rate we need a longer service cycle.

In this experiment, we assume that all CM streams are JPEG Streams with frame size 18KB and data rate to be 30 frames per second (i.e.540 KB/s). In PFS, each frame is stored in a unit. In a service cycle, the same number of units are read out for each stream. The experiment is as follows: First, we fix the number of concurrent streams and perform a group read. The group size begins with one unit and is gradually increased until the required data rate is achieved. We repeat this for increasing number of concurrent streams, until the system is saturated, and thus unable to satisfy the required data rate. For UFS, we consider frame size as unit size and carry out the same experiment.

4.3.2 Experimental Results

Fig.7 shows the relationship between the number of concurrent streams and the buffer size for each stream.

UFS can support up to 16 concurrent JPEG streams, each at the rate of 540KB/s. When the number of concurrent streams is more than 16, the required data rate is not satisfied. The buffer size of each stream remains the same as the frame size while the number of streams goes from 1 to 16. Under the same conditions, PFS can support up to 23 concurrent JPEG streams, while the buffer size of each stream monotonically increases with the number of streams.

In PFS, when there are less than three concurrent streams, the buffer is the same as frame size. As more concurrent streams enter, we must increase the buffer size and do group read to meet the required consumption rate.

4.4 Experiment 3 : Effect of Concurrent Streams on Service Cycle Length

4.4.1 Experimental Design

This experiment is designed to determine the effect of concurrent CM streams on the length of the service cy-

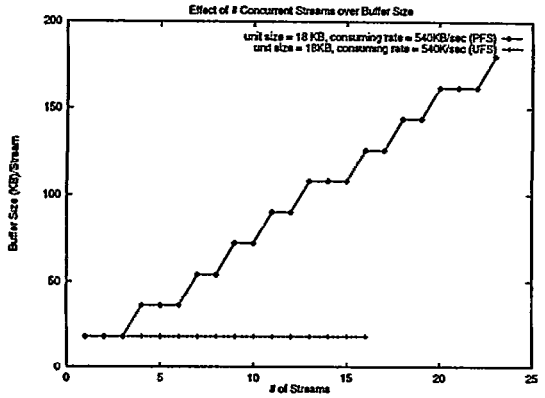

Figure 7: Effect of Number Concurrent Streams on Buffer Size

cle. This experiment can be performed in the same way as the previous one.

4.4.2 Experimental Results

Fig.8 shows the relationship between the number of concurrent streams, and the length of the service cycle.

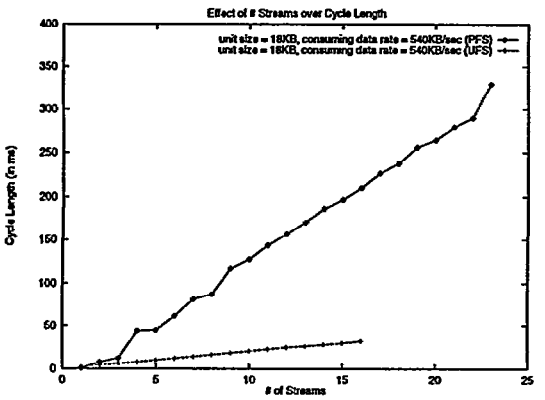

Figure 8: Effect of # of Concurrent Streams on Cycle Length

In UFS, as the number of concurrent streams increases, the length of the service cycle increases linearly. In PFS, when there is only one stream, the length of the service cycle is very small. As the number of concurrent streams increases, the length of the service cycle increases faster than that in UFS. This shows that the overhead of resource contention of concurrent streams in PFS is more than that in UFS.

This relationship between the number of concurrent streams and the length of service cycle is similar to the relationship between the number concurrent streams and the buffer size. We can deduce it from equation 1:

$$T > (sr_o t_o/(r_o - sR) \quad (3)$$

$$B > r_o R t_o/(r_o - sR) \quad (4)$$

$$S_{max} = r_o/R \quad (5)$$

Here, T is the length of the service cycle, B is the buffer size, S_{max} is the maximum number of concurrent streams, s is the number of concurrent streams, and R is the consumption rate. For a given number of streams, the length of the service cycle for PFS is longer than that for UFS, as shown in figure 8. This happens because PFS's approach to handling higher workload is to expand the length of the cycle, and thereby increase the efficiency of the I/O system by reducing context switch overhead. This is not without its drawback, since a larger buffer is required by PFS, compared to UFS, for a given workload (as shown in figure 7). However, as we can observe, this strategy is better overall since it allows PFS to handle a larger range of workload, i.e. 23 streams, as compared to a maximum of 16 streams for UFS.

5 Comparison of PFS and UFS on QoS Metrics

The QoS metrics we measured are unit, aggregate and consecutive drift factors and loss factors, as described in Section 3.

5.1 Experiment 4 : Measuring Drift Factors

This experiment is designed to measure the effect of UFS and PFS approaches to file management on drift profiles of streams, which specify the average and bursty deviation of schedules for frames from ideal expected points in time. We measured the difference between the ideal rendition time and the actual rendition time as a unit granule drift. The aggregate drift is the sum of unit granule drifts over some interval, and the consecutive drift is the sum of consecutive non zero drifts.

5.1.1 PFS's Suitability for Real-Time Applications

In table 1, an example history of stream scheduling based on timing metrics (drift profile) using PFS is shown. This experiment was performed for a single stream. Most UGD's are zero, and only in the time slots 9, 10, and 14 frames have time differences 5, 5, and 6, respectively. Hence the resulting ADF is $5+5+6=16$ per 15 granules, and the largest consecutive drift is 5 $+5 = 10$, between s(8) and s(10). Hence, the CDF is 10.

However, for UFS (in table 1), the drift statistics are quite different, i.e. most UGD's have larger times, and as a result the ADF (1125) and the CDF (1014) are much larger than those measured for PFS. Consequently, UFS is not well suited to support perceivable continuity of CM streams which are sensitive to drifts. Through this experiment, we can say that PFS preserves more faithfully the characteristics of real-time applications than UFS does (in Figure 9). Also, when we increase the number of concurrent streams (1 to 20), the ADF values are increased.

5.1.2 History Based Stream Scheduling

Consider a example QoS specification, where the acceptable drift profile is (100ms//30, 20ms) and the acceptable rate profile is (30, 100). At any point in time, the

Time Slot (PFS)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UGD	0	0	0	0	0	0	0	0	5	5	0	0	0	6	0
ADF	0	0	0	0	0	0	0	0	5	10	10	10	10	16	16
CDF	0	0	0	0	0	0	0	0	5	10	0	0	0	6	0

Time Slot (UFS)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
UGD	741	29	9	6	89	7	6	2	89	7	8	21	0	7	4
ADF	741	770	779	785	874	881	887	889	978	985	993	1014	1014	1121	1125
CDF	741	770	779	785	874	881	887	889	978	985	993	1014	0	7	11

Table 1: Media Granule Renditions in PFS and UFS

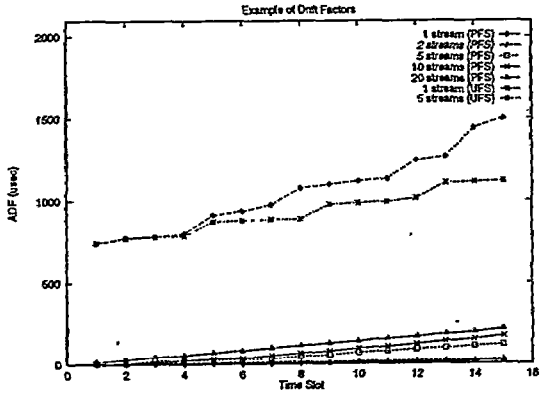

Figure 9: Unit Sequencing Drift Factors

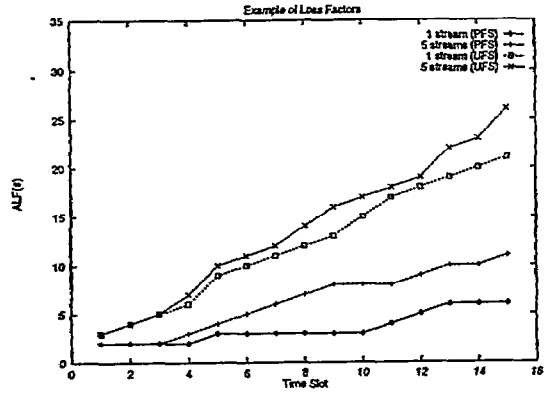

Figure 10: Unit Sequencing Loss Factors

time at which the next frame (granule) is to be scheduled is restricted by the history (at this point in time) of unit granule drifts [24]. To satisfy the drift profile, the next timing interval of rendition is given by (equation 6):

$$[T_i + \frac{1}{\rho} - \min\{\frac{\sigma}{\rho(\rho+\sigma)}, D(i)\}, T_i + \frac{1}{\rho} + \min\{\frac{\sigma}{\rho(\rho+\sigma)}, D(i)\}] \quad (6)$$

where, D(i) is the maximum drift slack available for $s((i+1))$, and satisfies $D(i) \geq 0$.

At a rate of 33 frames/sec, the ideal time of rendition for 16th frame is 495 ms. According to (6), the rendition interval for the next slot should be within the interval $[495-10, 495+10] = [485, 505]$. Such information can be used to determine how to assign priorities to streams at this point in time. We call such class of scheduling policies as *history-based scheduling policies*, as they take the history of stream progress into account in making their scheduling decisions. Our current investigation is experimenting with such policies.

5.2 Experiment 5: Loss Factors

This experiment is to measure the effect of loss factors which specify aggregate and consecutive frame loss ratios. We measured the difference between the ideal frame arrival time and the actual frame appearance time. Using these values, we calculate the gap between the ideal media granule and the presented media granule as unit sequencing loss. We set a threshold granule period of time slot, and if the gap is larger than the threshold value, we assume the frame is lost. If the gap is between 0 and the threshold, we consider the frame as being delayed. Otherwise, it arrives on time. The aggregate media granule loss is the sum of unit sequencing losses and the consecutive media granule loss is the sum of non zero consecutive unit sequencing losses.

In the current experiment to measure the effects of *content* based QoS metrics (loss factors: such as USL, ALF, and CLF), we observe that both PFS and UFS provide fairly uniform and stable results (in figure 10). As we see in experiment 4, in measuring drift factors for single and/or multiple stream(s), we get better thoughput results (less loss) in PFS than in UFS. Comparing single stream to multiple streams, the former situation leads to lower loss than the latter for both PFS and UFS.

6 Conclusions

Given the very large size of continuous media data, e.g. video, most multimedia applications must access data from secondary and/or tertiary storage during execution. This makes the performance of continuous media files systems critical to overall system performance. Additionally, access to continuous media requires that *some Quality of Service (QoS)* parameters (specified by the application) be met. These include specification of retrieval rate and its variation, allowable timing drift, acceptable average and bursty errors, and degree of synchronization. Conventional file systems do not have any notion of QoS, and thus fail on two counts. First, they are unable to provide any guarantees regarding QoS, and second they are not able to take advantage of the application's tolerance (specified by the QoS parameters) in optimizing the file system performance. In this paper we present results from the experimental evalu-

ation of a continuous media file system, called Presto File System (PFS), which has been implemented in the context of a distributed multimedia application development environment that we are prototyping. Our ongoing work is developing storage and access mechanisms that take advantage of QoS specifications to optimize system performance.

7 Acknowledgements

The ideas implemented in PFS have benefitted a lot by discussion with a number of people. Specifically, we would like to thank Jim Richardson, Mukul Agarwal, and Jim Huang of Honeywell Inc., and Satya Prabhakar of SBC Inc.

References

[1] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Trans. Computer Systems*, 10(4):311–337, 1992.

[2] M. Chen, D.D. Kandlur, and P.S. Yu. Support for Fully Interactive Playout ina Disk-Array-Based Video Server. In *Proceedings ACM Multimedia 94*. ACM Press, October 1994.

[3] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic Batching Policies for an On-Demand Video Server. In *Proceedings ACM Multimedia 94*, pages 15–24. ACM Press, October 1994.

[4] C. Federighi and L.A. Rowe. The Design and Implementation of the UCB Distributed Video-On-Demand System. In *Proc. IS&T/SPIE 1994 Int'l Symp. electronic Images: Science and Technology*, 1994.

[5] D.J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Trans. Information Systems*, 10(1):51–90, 1992.

[6] D.J. Gemmell and J. Han. Multimedia Network File Servers: Multichannel Delay Sensitive Data Retrieval. *Multimedia Systems*, 1(6):240–252, 1994.

[7] R.L. Haskin and F.B. Schmuck. The Tiger Shark File System. In *COMPCON 96*, 1996.

[8] Jinadong Huang, Jim Richardson, Deepak R. Kenchamanna-Hosekote, and Jaideep Srivastava. Presto: Final Technical Report. Technical report, Honeywell Technology Center, august 1996.

[9] D.R. Kenchammana-hosekote and J. Srivastava. Scheduling Continuous media in a Video-On-Demand Server. In *Proceedings IEEE Conference on Multimedia Computing and Systems*, May 1994.

[10] D.R. Kenchammana-hosekote and J. Srivastava. Retrieval Techniques for Compressed Video Streams. In *Proceedings of Multimedia Computing and Networking SPIE IS&T*, January 1996.

[11] Wonjun Lee, Difu Su, Jaideep Srivastava, and D.R. Kenchammana-hosekote. QoS Driven Scheduling for Continuous Media File Servers. In *Proceedings of Multimedia Networks and Applications in SPIE Symposium on Voice, Video, & Data Communications*, November 1997.

[12] Wonjun Lee, Difu Su, Jaideep Srivastava, D.R. Kenchammana-hosekote, and Duminda Wijesekera. Experimental Evaluation of a Continuous Media File System. Technical report, Computer Science Department, University of Minnesota, Minneapolis, MN, May 1997.

[13] P. Lougher and D. Shepherd. The Design of a Storage Server for Continuous Media. *The Computer J.*, 36(1):32–42, 1993.

[14] C. Martin, P. S. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage Server. In Soon M. Chung, editor, *Multimedia Information Sotrage and Management*. Kluwer Academic Publishers, 1996.

[15] K.K. Ramakrishnan, L. Vaitzblit, and C. Gray. Operating System Support for a Video-On-Demand Service. In *Proc. 4th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video*, 1993.

[16] P. V. Rangan, H. M. Vin, and S. Ramanathan. Designing an On-Demand Multimedia Service. *IEEE Communications Magazine*, pages 56–65, July 1992.

[17] A.L. Reddy and J.C. Wyllie. I/O Issues in a Multimedia System. *Computer*, 27(3):69–74, 1994.

[18] L.A. Rowe, J. Boreczky, and C. Eads. Indexes for User Access to large Video Databases. In *Proc. IS&T/SPIE 1994 Int'l Symp. electronic Images: Science and Technology*, 1994.

[19] A. Silberschatz and P. B. Galvin, editors. *Multimedia: Computing, Communications and Applications*. Prentice Hall, New Jersey, 1995.

[20] R. Steinmetz and K Nahrstedt, editors. *Multimedia: Computing, Communications and Applications*. Prentice Hall, New Jersey, 1995.

[21] M. Stonebraker. Operating System Support for Database Management. *Computing Practices*, 1981.

[22] D.B. Terry and D.C. Swinehart. Managing Stored Voice in the Etherphone System. *ACM Trans. Computer Systems*, 6(1):3–27, 1988.

[23] H.M. Vin and P.V. Rangan. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Trans. Knowledge and Data Engineering*, 5(4):564–573, 1993.

[24] Duminda Wijesekera and Jaideep Srivastava. Quality of Service (QoS) Metrics for Continuous Media. *Multimedia Tools and Applications*, 3(1):127–166, September 1996.