The Design of a Variable Bit Rate Continuous Media Server

Gerald Neufeld, Dwight Makaroff and Norm Hutchinson

Department of Computer Science,
University of British Columbia
Vancouver, Canada

Abstract. This paper describes the design and implementation of a file server for variable bit rate continuous media. Most continuous media file servers have been designed for constant bit rate streams. We address the problem of building a server where each stream may have a different bit rate and, more importantly, where the bit rate within a single stream may vary considerably. Such servers will be come increasingly more important because of Variable Bit Rate (VBR) compression standards such as MPEG-2.

1 Introduction

The motivation for the design of a specialized file server for continuous media such as video and audio is well established. Most existing work in this area has assumed constant bit rates for the media stream. Recent work has been done in analyzing variable rate servers[1]. However, such analysis still assumes that each *individual* stream has a constant bit rate. Compression methods such as motion JPEG or MPEG-2 produce streams whose bit rates vary considerably within the stream. The server describe here is intended to support such variable streams. The primary area of complexity is in the admissions control algorithm and in I/O scheduling. As well, the server described here does not assume any single syntax such as MPEG-2.

2 Architecture

The design of the file server is based on a set of server nodes, each with a processor and disk storage on multiple local SCSI Fast/Wide buses. Each node is connected to the ATM network for delivering continuous media data to the client systems (See Figure 1).

Reading a media stream from the server is done via three RPC requests: *open*, *prepare* and *read*. The open request provides basic administration support, while the prepare request primes the connection in preparation for subsequent reads by transmitting an initial amount of data.

For example the client application issues open requests for a video, audio and text streams. Each open request will create an XTP[2] connection – which is rate

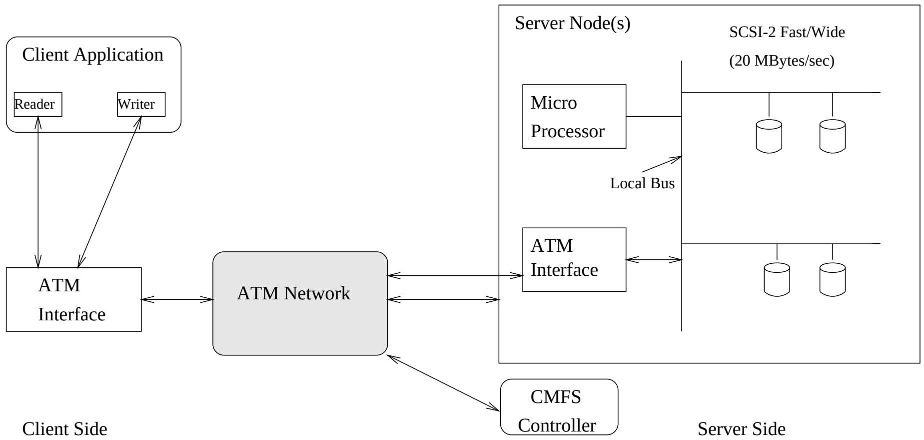

Fig. 1. Organization of System

controlled but without error recovery – from the server node containing the file to the client. The XTP connect request (FIRST packet) contains information about the minimum size of the window and the required bit rate. If the client accepts the request window size (i.e., reserves that amount of buffer space) then the subsequent local read operations will not be delayed. During the transmission of the media over the XTP connection, the client periodically sends RESERVE packets back to the server indicating that it has more buffer space available. The client must send a RESERVE packet if the amount of media data buffered at the client falls below the minimum required for continuous operation.

3 Admissions control algorithm and I/O scheduler

In order to schedule variable bit rate (VBR) streams, the server defines fixed length cycles or slots for all streams[1]. Each disk stream is then divided into these slots. A stream vector is created which contains the number of blocks which must be read for each slot for the stream. Because the stream may be VBR, the values for the vector may vary considerably.

The admissions control algorithm must know the minimum number of blocks that the server can read in a single slot. This value is calculated by running a calibration program. The minimum number is calculated by uniformly spacing the blocks across the disk thus maximizing the seek times (assuming a SCAN algorithm). This value more accurately reflect the actual capacity of the server since it includes all transfer delays (through SCSI bus and I/O bus to memory as well as server software overhead).

A simple admissions algorithm then is simply to sum the vectors for the active streams plus the new stream. If any slot in the resulting vector is greater

[1] A reasonable time for such a slot may be 500 milliseconds.

than the minimum number of blocks the server can read per slot, the admission fails. This is a very conservative estimate however. Any single slot may have less blocks to read than the server could read. In this case, the server is idle at the end of a slot. In order to admit more streams, we permit the server to read ahead as fast as it can subject to buffering constraints. By permitting the server to read ahead we can admit a stream where the total number of blocks required to be read within a slot is greater than the minimum number of blocks the server can read for a slot. The amount of such over allocation is proportional to the number of blocks the server has read ahead. The minimum number of blocks a server can read per slot is also required for accurately calculating the buffering requirements.

The following example illustrates this method. Assume that the server is capable of reading a minimum of 10 (fixed size) disk blocks per slot. Figure 2 shows the current schedule for the server and a new stream to be admitted. In

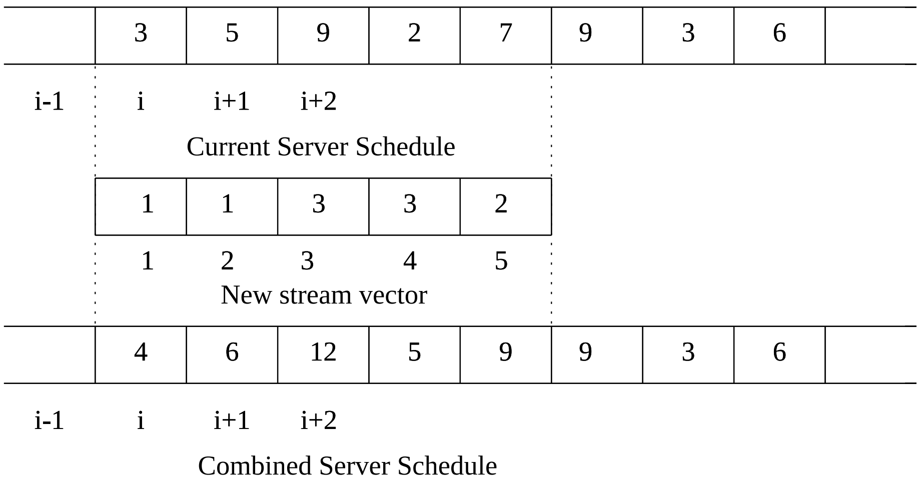

Fig. 2.

these vectors, the numbers represent the number of disk blocks that must be read in each slot. For instance, in the current server schedule, at slot i the server must read 3 disk blocks, at slot $i+1$ the server must read 5 disk blocks etc. These blocks represent the total number of blocks for all active streams. The vector for the new stream to be admitted represents the blocks which must be read just for that new stream. For instance, in the first slot for the new stream the server must read 1 block, then 1 again for the next slot, etc.

In the conservative admissions control algorithm, we would simply add the new stream vector to the current schedule. In this case, the $i + 2^{nd}$ slot would have a value of 12 which is higher than the minimum number of blocks the server can read in that slot (10). However if we permit the server to read more than a single slots worth, the first two slots worth of disk blocks would be read in one slot time. This read-ahead permits the given schedule to be accepted. By the time the server reads slot $i+2$ it will still be in the 2^{nd} slot of time assuming the server reads at the minimum number of blocks per slot.

The server in practice reads more than the minimum blocks per slot. This read-ahead is taken into account when a new vector is added. The number of blocks that were read-ahead are the actual number of blocks the server was able to read per slot (between minimum and maximum blocks per slot). The algorithm starts with this number of actual read-ahead blocks and then continues assuming the server will read a minimum number of blocks.

So far in this discussion we have assumed that there are an arbitrary number of buffers. That is, the server can read ahead with out fear of running out of buffers. Clearly this is not the case. We therefore have to stop read-ahead in the admissions algorithm once we have run out of buffers. For purposes of buffer consumption we assume the server reads a minimum number of blocks per slot. As buffers are transmitted on the ATM network they are freed. We therefore factor in the number of buffers that are being freed into the admissions algorithm. A detailed description of this algorithm is given in [3].

4 Conclusions

A multiple node version of the file server has been implemented. We have created a client that supports video (using the Parallax JPEG card), audio and text. Synchronization is accomplished using the described methods to schedule the prepare calls and using a real-time schedule to maintain the synchronization of the presentation (lip-sync). The server runs on a IBM RS 6000 (350) over an 100 Mbps ATM link to a client running on a Sun Sparc II. The system environment uses a real-time threads package developed for this project which operates within a single Unix process [4]. This package also provides an operating system shield to native systems which do provide some form of real-time threads such as AIX 4.1.

References

1. Bikash, S., Ito, M. and Neufeld, G.: The Design and Performance of a Continue Media Server for a High-Speed Network *to appear* IEEE MultiMedia Conference, Boston, (May 1995)
2. Strayer, W. T., Dempsey B. J., and Weaver A. C.: XTP: The Xpress Transfer Protocol, Addison Wesley Publishing, 1992
3. Neufeld, G., Makaroff, D., Hutchinson, N.: Internal Design of the UBC Distributed CMFS, Technical Report, (1995)
4. The UBC Real-Time Threads package, Technical Report, July,1994