

PERFORMANCE IMPROVEMENT IN MULTIMEDIA STREAMING BY PARALLELISING SERVER AND CLIENT ARCHITECTURES.

B.JayaPrakash Narayan, K.J.Krishna Ram, R.M.Venkatraman

B.E Computer Science & Engg. (IV Year) Anna University, Chennai.

kriscomp@bharatmail.com, jpnarayan@hotmail.com

ABSTRACT

The role of multimedia applications in our day-to-day life has dramatically increased in the last few years. In this paper, we present an architectural framework to improve the processing speed of multimedia streams. The architecture consists of media server and media client portions. We present an idea of implementing parallelism on both sides thereby increasing the performance.

Introduction to Protocol architecture:

The TCP/HTTP protocols were designed for the type of traffic with reliable transmission and minimal delay constraints. However, multimedia traffic requires the use of different protocols to provide the necessary services. The "slow start" TCP congestion-control mechanism can interfere with the audio and video "natural" play out rate. TCP doesn't provide timing information, a critical requirement for multimedia support. TCP reliable message delivery mechanism causes further Jitter and skews internally between frames and externally between associated video and audio streams.

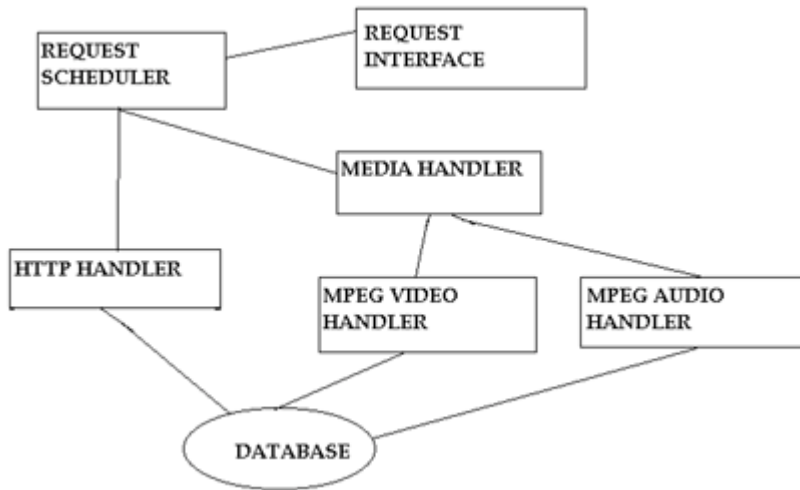
In order to improve the Quality of Service, different set of protocols are employed to handle continuous media. The protocols used are Real Time Streaming Protocols (RTSP) in the Application layer and Real Time Transport protocols (RTP) in the Transport layer. RTSP maintains the state of the media streams and has appropriate server control. RTP does not lay much stress on reliable delivery mechanism. It maintains timing information using the time stamp field, which helps in intra and inter frame synchronization and avoids Jitter. The performance of the systems implementing these protocols degrades during services like video-conferencing where the servers cannot handle the bulk of incoming requests and due to increased packet drop-rate.

In order to increase the Quality of Service in these cases we propose a parallel multiprocessor system. In this paper we discuss on

- 1) Server and client architecture
- 2) Parallel Approach using multi-processor system
- 3) Load balancing and scalability
- 4) Implementation

Server and Client Architecture:

Server Architecture



The main components of the server are shown in figure. They are:

1. The Request Interface
2. The Request Scheduler
3. The HTTP request handler,
4. The Continuous media handlers, and
5. The Database

Request Interface:

It receives client requests and passes the request to the *Request Scheduler*. The admission controller then determines or estimates the requirements of the current request, which may include network bandwidth and CPU load. It then makes a decision on

whether the current request should be served based on its knowledge of current conditions.

Request Scheduler

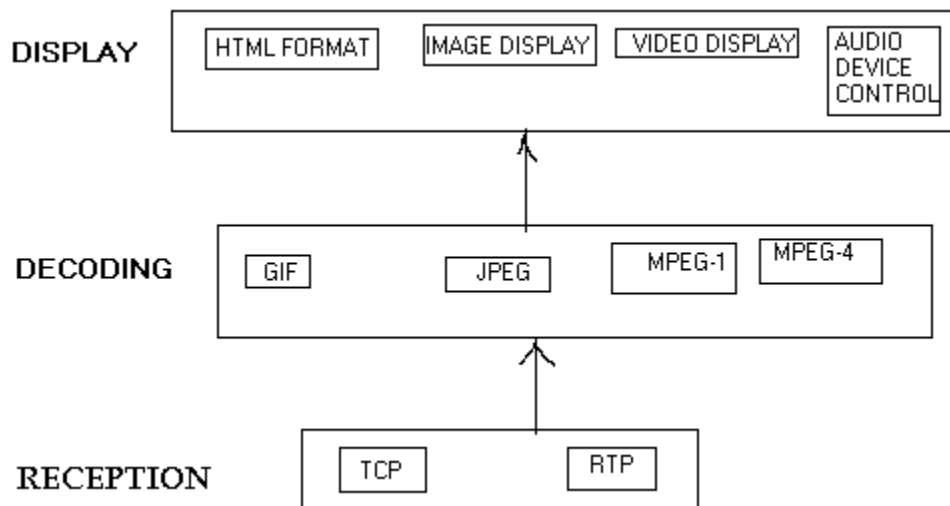
Once the system grants the current request, the main request dispatcher hands the request. The scheduler classifies the request and has handlers for: HTTP, MPEG Audio and Video. Each handler uses a different transmission protocol for handling requests.

HTTP requests use TCP/IP. MPEG video and audio use RTP.

Database

It is responsible for recording the request and transmission statistics.

Client Structure



The client process the packet received in three portions in sequence. First portion (RECEPTION) gets the packet from the network interface. The second portion

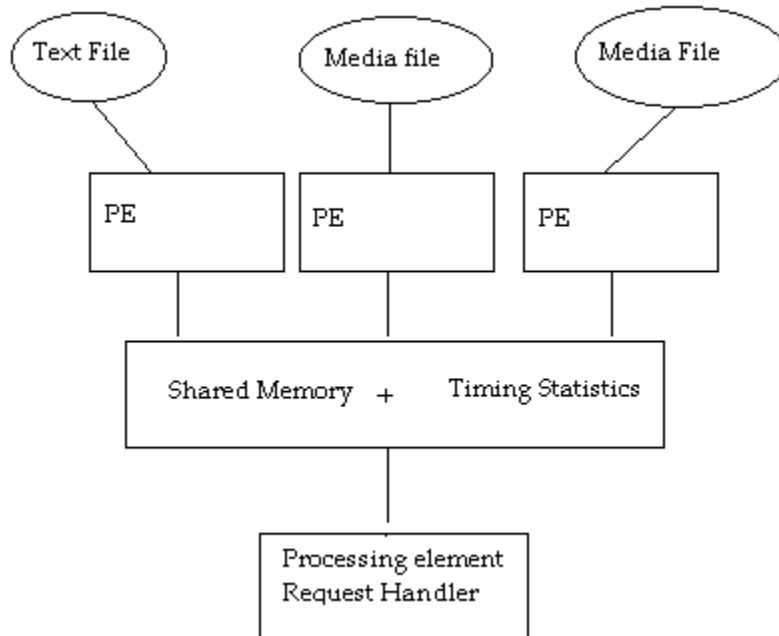
(DECODING) decodes the received packets using techniques like GIF, JPEG (for images), MPEG1; MPEG4 (for audio and video streams) The third portion (DISPLAY) includes traditional HTML formatting and inline image display, video display and audio play.

A PARALLEL APPROACH:

In order to parallelise the server and client operations we propose a multiprocessor system. We employ a centralized approach. The system must have implemented the RTSP and RTP protocols.

Scheduling in Server:

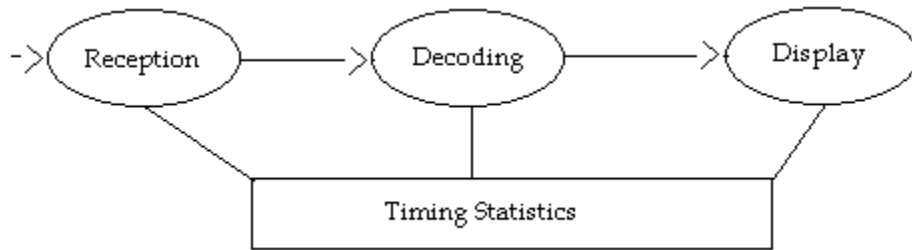
On receiving the request from the request interface, request handler acts like a server process which schedules requests to different processors based on their type (text, continuous media etc). The processor handling media requests can spawn number of processes based on the magnitude of the requested data. For further enhancement the media file is split as different streams and placed in various memory banks so that when a media file is requested different processes can read the file simultaneously and independently, thereby speeding up the processing time. The processor handling the media file uses compression techniques prior to packet transmission.



The timing statistics include the network usage and processor usage of each request, the quality of service data such as frame rate, frame drop rate, and jitter. This timing statistics is placed in high speed TLB.

Exploiting parallelism in client side:

In client side we propose pipeline scheme. This scheme includes 3 units connected in pipeline. The 3 units are reception, decoding, and display. These units run in parallel. When packets of single stream are received, the decoding unit uses the received packets and the display unit starts playing the packets previously decompressed. So the client side need not wait for entire media file retrieval.



The Display adjusts the media file playing rate based on the timing statistics indicated by Reception and Decoding units, and the user is oblivious to these adjustments.

Load Balancing:

In the multimedia streaming when packets arrive at a greater rate, packet drop rate becomes more prominent and dynamic at the nodes. So we propose a dynamic load-balancing scheme. By analyzing the packet drop rate at regular intervals of time (ToA) the master process can control the processors. The Time interval of Analysis (ToA) depends on the application. By using profile information and timing statistics the ToA is determined. If a processor remains idle or overloaded as indicated by the timing statistics, the master processor reschedules jobs based on timing statistics. If the server side is very busy, then the request handler sends appropriate feedback to the client (whose request could not be currently processed). On the other hand if the media packets' arrival rate at the client side is greater than its capacity then the reception and decoding module in the client sends a feedback to the server to reduce the media-streaming rate. In order to make efficient use of the bandwidth the Request Interface Module in the server creates RTSP packets of size depending on the available bandwidth.

Performance:

A simple test using two Intel celeron machines with 533mhz processors running the Unix OS connected by 10Mb/s Ethernet is performed. RTSP on Unix OS using JAVA is implemented. JPVM package is used to test the efficiency of multimedia streaming for streaming small video clips. The performance enhances for large video clips. Since Java is platform independent and has libraries for media and image handling the performance was noticeable. The JPVM uses message-passing libraries of java for communication. A considerable performance can be accomplished using JPVM package in case of multicast conferencing built on the parallel architecture described in the paper.

Reference:

- 1) "Multimedia Communication and Protocol" by Effelsberg
- 2) "Scheduling and Data Distribution in a Multiprocessor Video Server" – Narasima Reddy (IBM Almaden Research Center), IEEE-1995
- 3) "Real Time Transport Protocol: RTP",RFC-1889
- 4) "Real Time Streaming Protocol: RTSP" ,RFC-2326
- 5) "Jitter Control and Dynamic Resource Management for Multimedia Communication Over Broadband Network "Ahmed Bashandy & Edwin Chong ,Purdue University