DYNAMIC LOAD BALANCING ACROSS MIRRORED MULTIMEDIA SERVERS

Ashwatha Matthur, Padmavathi Mundur

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250, USA
{ashwath1, pmundur}@csee.umbc.edu

ABSTRACT

The purpose of this paper is to present protocols for efficient load balancing across replicated multimedia servers in a Metropolitan Area Network. Current multimedia infrastructures, even when they use mirrored servers, do not have standardized load balancing schemes. Existing schemes frequently require participation from the clients in balancing the load across the servers efficiently. We propose two protocols in this paper for fair load balancing without any client-side processing being required. Neither protocol requires any change to the network-level infrastructure. Using network packet loss and packet transmission delay as the chief metrics, we show the effectiveness of the protocols through extensive simulations.

Keywords: Media streaming, Mirrored Servers, Load Balancing.

1. INTRODUCTION

Streaming applications such as directly usable online video and audio are becoming increasingly popular on the Internet. With this increasing popularity, several mechanisms have been introduced to improve the quality and availability of streaming media. The mechanisms are critical since streaming applications require more stringent QoS than usual Web services. In this paper, we provide two protocols for load balancing and QoS in mirrored multimedia servers for streaming applications.

The first of the two protocols we propose uses a centralized load distribution algorithm. The different mirrored servers communicate the degree to which they are loaded to a central server. All client requests are received by the central server, which uses the information it has about the global state to distribute client requests evenly. The second protocol does not use a central server and has a token passing scheme to split and distribute *each* client request equally across the servers. The distribution achieves diffusion of the traffic across several different routes. Through simulation experiments, we evaluate both protocols against an infrastructure with no load balancing in effect. The metrics used to evaluate the quality of streaming are the average packet loss rate and the average packet transmission latency. The results show significant decrease in the packet loss rates and latencies with our protocols.

In this and the following paragraph, we describe research that is relevant to the current work. Starting with techniques related to mirrored servers and load balancing, Myers, Dinda and Zhang[1] evaluate the techniques for selecting a server given a set of mirrored servers. Crovella and Carter use probes in [2] to evaluate bandwidth and dynamically select a server. Conti, Gregori and Panzieri[3] propose several load distribution mechanisms to achieve higher availability and quality of data. Bunt et al[4] couple load balancing with caching to handle requests to clustered Web servers under very high loads. Colajanni, Yu and Dias[5] use DNS as a centralized scheduler for load balancing. Nguyen and Zakhor[10] use a set of mirrored servers which co-ordinate using rate allocation and packet partition algorithms to achieve high throughput. Padmanabhan et al[11] use a scheme where data is distributed among the clients, which forward the data in case of a server overload. The above approaches differ from the present one because we focus on multimedia streaming and thus take packet loss into consideration as a QoS parameter. We also eliminate client-side processing for higher efficiency.

For Internet simulation, we use the Tiers topology generation mechanism, proposed in Doar[6]. The topology is generated in terms of several tiers such as LAN, MAN and WAN. Mellia, Carpani and Cigno[8] illustrate tools such as Tstat to collect data about Internet traffic on large scales. Recent research indicates that it is possible to provide general models that fairly represent traffic on the World Wide Web, using distributions such as M/Pareto for sessions and Poisson for arrival patterns[7]. Trace collection for video traces is illustrated in Fitzek and Reisslein[9].

The rest of the paper is organized as follows.Section 2 explains our load balancing protocols in detail. Section 3 provides the details of the simulation model, simulation parameters and results. We conclude this paper in Section 4.

2. PROTOCOLS

In this section, we present details of the proposed protocols for load balancing and QoS. The Centralized Control Protocol achieves load balancing by having a central server distribute client requests across a set of video servers. In the Distributed Control Protocol, each client request is split into several substreams and different video servers process

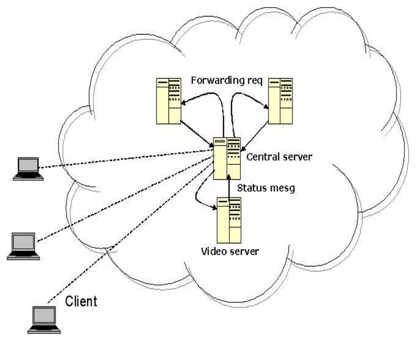

Figure 1: Centralized Control Protocol

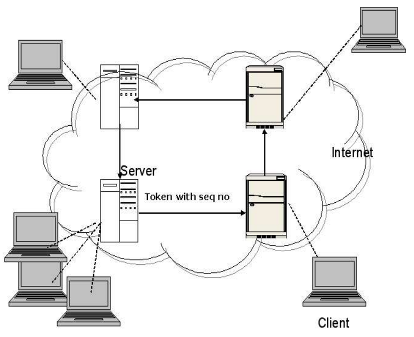

Figure 2: Distributed Control Protocol

different substreams.

2.1. Centralized Control Protocol (CCP)

In this protocol, the video servers periodically send state information to the central server, indicating their current load. The central server maintains the global state, and thus has knowledge about how loaded each of the video servers is. The precision of this knowledge depends on how often the video servers send information about their loads to the central server. Whenever a connection request comes in, the central server forwards it to the least loaded video server. Irrespective of the streaming protocol, TCP is used for inter-server communication so as to prevent losses of messages from the video servers to the central server and vice versa.

If a video server fails to send any message about its state information to the central server, the video server is assumed to be inactive or overloaded. No client requests are forwarded to the video server until it becomes active again and resumes sending status messages. The protocol can be thought of as a *single-stream, single-server* protocol, since exactly one server services each client request, and there is no breaking up of the stream among servers.

2.2. Distributed Control Protocol (DCP)

Most centralized control solutions suffer from the overhead of having a single point of failure. Limited resource availability at the central server can also be an overhead under high load. However, they have the advantage of simplicity. To evaluate these factors, we present our second protocol using a distributed control architecture.

In the distributed control protocol, a set of video servers form a token passing arrangement to serve each client request. Let the number of servers be *K*. When a connection request comes to a server from a client, it breaks up the requested stream into *K* segments, where *K* is the number of servers. Let each segment take *n* seconds to stream. Each segment has a sequence number *i*, $1 \leq i \leq K$. The server streams the first segment. A period *D* seconds before it completes transmission of the first segment, it hands over control to another server, chosen at random. The second server processes the second segment, and hands over control to a third server, and so on. While forwarding the request, a sequence number and a *server-list* are also forwarded. The sequence number indicates the segment that is to be processed by the next server. The *server-list* contains a list of servers which have already processed segments of this stream. While choosing a new server, a server not in *server-list* is chosen. While forwarding the request, each server appends its id to the *server-list*. This protocol can be thought of as *multiple-server, single-stream*, since a single stream is serviced in several segments and each request is processed by several servers.

In case the new server does not respond, a different server not present in *server-list* is chosen. In case none of the servers not in *server-list* respond, a server already in *server-list* has to be chosen.

Event: Connection request from client to a server.

1. Start processing segment 1. (i=1)

2. *D* secs before completion, send to a server chosen at random, $\{$Connection req, Seq num *i+1* (next segment), *server-list* = (Server-id) $\}$

Event : $\{$Connection req, seq num *i*, *server-list*$\}$ received from another server.

1. start processing segment *i*.

2. *server-list* = *server-list* + (Server id)

3. if *(i=K)*, end of stream. Otherwise,

4. *D* secs before completion, choose a server not in *server-list*, and forward $\{$conn req, *i+1*, *server-list*$\}$ to the server.

3. SIMULATION SETUP AND RESULTS

Network Simulator 2.0 is used for simulating the network and the multimedia architecture. The multimedia files are simulated by traces of actual videos encoded in MPEG-4 format[9].

3.1. SIMULATION SETUP

A Metropolitan scale network with LANs being connected to different corporate networks is used for simulation. The topology is simulated using the topology generator Tiers 1.1[6]. Realistic values for different network parameters, such as nodes/LAN, LANs/MAN, are used as illustrated in Table 1. In order to model background Internet traffic,

Parameter	Details
Network	MANs, generated by Tiers1.1
Number of nodes	Approx. 1100
Traffic arrival pattern	Poisson, Mean 50-300/hr
Telnet and FTP sessions	M/Pareto $0.9 < a < 1.1$
Background traffic	M/Pareto ON/OFF periods
Simulation period	500 hours
Link capacity	10 Mbps
Ave Length of video	15 min; heavy-tailed
MANs	$N_m = 10$
Nodes/MAN	$S_m = 10$
LANs/MAN	$N_L = 5$
Nodes/LAN	$S_L = 20$
Total number of links	approx 1250
Intranetwork redundancy	$R_M = 2, R_L = 1$

Table 1: Simulation Parameters

we note a few general characteristics of Web traffic: Traffic on the Internet is characterized by a large proportion of TCP traffic and a relatively smaller proportion of UDP traffic. TCP traffic, notably Telnet and FTP, have a Poisson arrival pattern. Telnet and FTP sessions can be modeled by a Pareto distribution with heavy tail. Background traffic can be modeled as superpositions of ON/OFF periods expressed as Pareto distributions; $0.9 < a < 1.1$[7]. We use all of the above factors to generate realistic background traffic. Results from Tstat, a network monitoring tool described in [8] are used to provide quantitative information regarding the proportion of TCP and UDP flows and other data in our simulation. Table 1 summarizes information about the simulation. Results from Doar[6] are used to apply various parameters for the topology of the network. The intranetwork redundancy parameter refers to the number of edges between nodes of the same type.

The simulation architecture consists of 4 identical video servers and 100 multimedia clients connected over a metropolitan area network. The multimedia clients generate requests in a Poisson arrival pattern. The average number of requests per hour is varied from 50 to 300 during the experiments.

Constraint on packet transmission periods: To determine the maximum acceptable packet delay, we use the following approach: Let playback begin at the receiver after *n* seconds worth of video has been cached. Let the average download rate be *X* bytes/sec, and let the playback rate be *B* bytes/sec. Let the instant at which playback begins be $t = 0$. Therefore, *nB* bytes of data have already been transmitted.

We derive an expression for a constraint that must hold for the packet to reach the destination in time and be usable. Consider the m^{th} byte of the stream. Since the playback is at *B* bytes/sec, it should be, ideally, played at $T_{play} = m/B$ sec. Since downloading is at *X* bytes/sec, the instant at which it will be transmitted is: $T_{sent} = (m - nB)/X$, because *nB* bytes have already been transmitted. The following inequality must hold if the packet is to reach the destination in time:

$(T_{sent} + packetlatency) < T_{play}$

Or, packet latency $< (\frac{m}{B} - \frac{m-nB}{X})$

While measuring packet loss rates, we identify packets that have too high a delay to be played back, and treat them on par with lost packets.

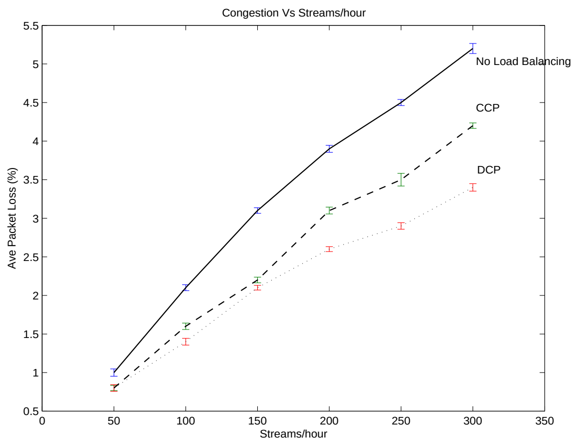

Figure 3: Congestion Vs Number of Streams/hr: Packet losses are reduced significantly with distributed control

3.2. RESULTS

Experiments are conducted using the above simulation setup to evaluate the performance of the two protocols. They are compared against a base scenario where no load balancing is involved. The base scenario uses the same infrastructure, but no attempts are made to distribute the traffic across the servers. The server which receives a client request processes it in its entirety. The average packet loss percentage and the average packet transmission delay are measured, with the number of streams per hour being increased from 50 to 300. The results are averaged over five trials; confidence interval analysis for 95% confidence is conducted over the sets of five experimental values. The graphs are provided with errorbars that show the half-widths of the confidence intervals.

Figure 3 shows the average packet losses as the number of streams is increased. The distributed protocol fares best, since it achieves the highest degree of load distribution across the network. Each client request is diffused across several different routes, and thus no particular route gets a higher amount of traffic than others. Packet losses due to local congestion conditions are minimized, since the traffic is distributed evenly across the servers. The centralized control protocol does slightly worse since the global state is updated only once in a while and thus, the load distribution is not ideal.

Figure 4 shows the average packet transmission delays for the same increases in the number of streams per hour. The distributed control protocol does slightly worse than the centralized control protocol, since each stream is serviced by all servers, both near and far from the client. It still does better than when there is no load balancing, since delays

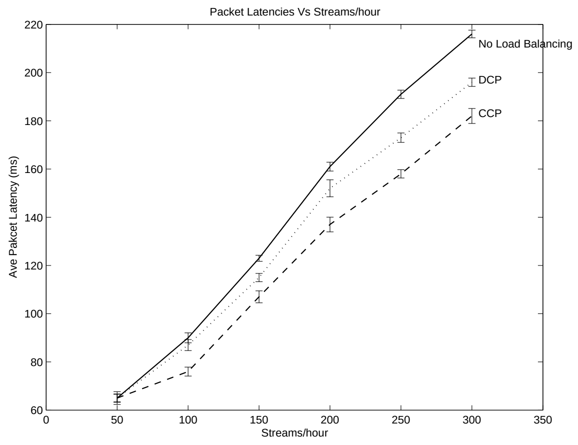

Figure 4: Packet delays Vs Number of Streams/hr: Packet delays are reduced significantly with centralized control

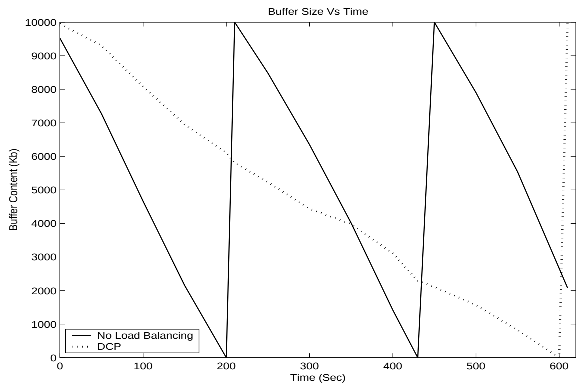

Figure 5: Buffer Level Vs Time: Buffer level is conserved more effectively with load balancing

due to congestion are minimized. In the centralized control protocol, the stream is either serviced by the nearest server or, in case the nearest server is highly loaded, a server that has lower load and thus can serve the stream faster.

Higher packet loss rates also lead to faster depletion of client buffers. This is because higher packet loss rates imply lower fill rates for the client buffer. Figure 5 shows the results of an experiment that measures the amount of data in the client buffer as a function of time. In the scenario with no load balancing, higher packet losses are involved. As a result, the client buffer runs out much sooner, and requires to be refilled more often. This leads to more frequent discontinuities in video playback. In the scenario with load balancing using DCP, there are fewer packet losses, leading to slower depletion of the client buffer. This means that there are less frequent discontinuities in playback. Figure 5 clearly illustrates this aspect.

4. CONCLUSIONS

In this paper, we have presented two protocols for load balancing in a mirrored multimedia server environment. The protocols avoid client-side processing. The only ability clients need to have is the ability to accept connections from more than one server. The protocols do not require any change to the network infrastructure either, since the inter-server communication occurs at the transport level. The two protocols have been evaluated in terms of their improvements to packet losses and packet latencies to the client. The reduction in packet loss is of particular significance to multimedia transmission. Packet transmission delays are also reduced significantly in both the protocols. Smaller packet transmission delay helps in maintaining a smaller buffer on the client side for streaming applications.

5. REFERENCES

[1] A. Myers, P. Dinda, and Hui Zhang, ”Performance characteristics of mirror servers on the Internet,” Proc. of IEEE INFOCOM, Mar 1999.

[2] M. Crovella and R. Carter, ”Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks,” Proc. of IEEE INFOCOM, 1997.

[3] M. Conti, E. Gregori, F. Panzieri, ”Load Distribution among Replicated Web Servers: A QoS-Based Approach,” in WISP 1999, ACM Press, Atlanta.

[4] R.B. Bunt, D.L.Eager, et al., ”Achieving Load Balance and Effective Caching in Clustered Web Servers,” The fourth International Web Caching Workshop, San Diego, CA, Mar-Apr 1999.

[5] M. Colajanni, P. S. Yu, and D. M. Dias, ”Analysis of task assignment policies in scalable distributed Web-server systems,” IEEE Transactions on Parallel and Distributed Systems, 9(6):585–600, 1998.

[6] M. Doar, ”A Better Model for Generating Test Networks,” Proc. of IEEE GLOBECOM, 1996.

[7] T.D. Neame, M. Zukerman, ”Modeling Broadband Traffic Streams,” Proc. of Globecom ’99, Rio de Janeiro, Brazil, Dec, 1999.

[8] M.Mellia, A.Carpani, R.Lo Cigno, ”Measuring IP and TCP behavior on an Edge Node,” Planet-IP and Nebula Joint Workshop, Jan 2002.

[9] Frank H.P Fitzek, Martin Reisslein, ”MPEG-4 and H.263 Video traces for Network Performance Evaluation,” IEEE Network Magazine, Vol. 15, No. 6, pages 40-54, Nov-Dec 2001 .

[10] T. Nguyen, A. Zakhor, ”Distributed Video Streaming over the Internet,” Proc. of MMCN 2002, San Jose, CA.

[11] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, ”Distributing Streaming Media Content Using Cooperative Networking,” ACM NOSSDAV, Miami Beach, FL, May 2002.