University of Alberta

Content Adaptation Architecture for Universal Multimedia Access

by

Sunil Kumar Bandaru

A thesis submitted to the Faculty of Graduate Studies and Research in
partial fulfillment of the
requirements for the degree of *Master of Science*

Department of *Electrical and Computer Engineering*

Edmonton, Alberta
Spring 2004

Your file Votre référence
ISBN: 0-612-96447-7
Our file Notre référence
ISBN: 0-612-96447-7

Abstract

The use of multimedia data is growing at a rapid rate. Bringing multimedia services to terminals with limited capabilities such as limited bandwidths, and limited resolutions is a challenge to be dealt with. The features of these terminal devices generally vary in terms of storage capacity, memory, resolution, processing speed and bandwidth. Therefore an efficient scheme is required for adapting the multimedia content for delivery to the devices with limited resources. In this thesis we propose a novel distributed adaptation architecture suitable for resource-limited multimedia terminals as well as wired connections with high bandwidths. A novel part of the proposed architecture is efficient use of the cached data at the proxy server. We propose a cache replacement policy for efficient cache management in the proxy. The proposed policy provides superior performance compared to other existing cache replacement policies.

Acknowledgements

I would like to thank my supervisor Dr M. K Mandal for his financial support and invaluable guidance in my research. Without his vision and encouragement this work would not have been possible. Dr Mandal helped me by providing new ideas about implementation of the architecture and also some methodologies for evaluating cache policies. These were very crucial for completion of my thesis.

I would like to thank all the members of MCCL laboratory for their co-operation and help. I would especially like to thank Patrick Sessanga for his suggestions, which helped me significantly in my research work.

Finally, I would like to thank my family for their continuous support and encouragement to finish my thesis.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AF	Aging Factor
CC/PP	Composite Capability/ Preference Profiles
GIF	Graphic Interchange Format
HDC	Hand Held Device
HTTP	Hyper Text Transfer Protocol
JPEG	Joint Photo Experts Group
LAN	Local Area Network
LRU	Least Recently Used
LFU	Least Frequently Used
MMCM	Multimedia Content Model
MRMM	Multiple Resolution Multiple Modalities
MTTR	Mean Time-To-Re-access
PDA	Personnel Digital Assistant
QOS	Quality of Service
TACC	Transformation, Aggregation, Caching and customization Architecture
TCP	Transmission Control protocol
TS	Traffic Shaper
UDP	User Datagram Protocol
UMA	Universal Multimedia Access
WWW	World Wide Web

Chapter 1

Introduction

The usage of the Internet has increased enormously causing almost an exponential increase in the web content authored for the users. Recent developments have seen a tremendous increase in the usage of network appliances and mobile devices; the devices include hand-held personal computers (HPCs), personal digital assistants (PDAs), set-top boxes, smart cellular phones and network computers. The consumers use these devices to access rich multimedia content through the Internet. However some of these devices have limited resources, and are connected to networks of limited bandwidth. The features of these devices normally vary in terms of storage capacity, memory, resolution, processing speed and downlink bandwidth. For example screen sizes for personal computers (PCs) vary between 800x600 to 1024x780 pixels, for HPCs between 48x240 to 640x240 range, and for PDAs between 160x160 to 320x240 range. Screen colors range from 24 bit and 8 bit color to 4 bit and 2 bit gray level. These also use a variety of network connections ranging from cable to mobile, with varying bandwidth, connection characteristics and costs. Most of the multimedia content authored today is targeted towards PCs as a client. World Wide Web (WWW) documents, which have rapidly become the largest form of multimedia, are also authored specifically for personal computers with reasonable

high bandwidths. With the rapid development in wireless devices and large variety of the mobile devices available commercially the content providers are targeting to provide rich multimedia to these low bandwidths devices. The diversity of these devices makes it difficult and expensive to author multimedia content separately for each individual type of device. Content providers, therefore have to use different adaptation approaches for better Quality of Service (QOS) and faster responses. There are many important factors that can affect QOS such as network traffic, bandwidth, caching, and loading of media server or proxy server. A number of solutions have been proposed in an attempt to address the problems discussed above. Several caching techniques have been developed but their performance needs a significant improvement to address the QOS concerns in these architectures.

1.1 Motivation

Most of the content adaptation algorithms are implemented either in a media server or proxy server, resulting in overloading of the server or proxy. Current proxy-based architectures do not use the cached data at the proxy efficiently for adaptation. The proxy caching is an approach, which reduces the network traffic on the Internet back bone. The performance of the proxy caching is a major factor that can improve the performance and reduce the server latency. There is no reported work addressing the cache replacement policy for Universal Multimedia Access (UMA) based architectures. The approach of distributed adaptation of

content with efficient use of cached data in the proxy can greatly improve the performance of content adaptation architecture.

1.2 Major Contributions

The major contributions of this thesis are as follows

- An efficient architecture, which can deliver content for mobile and desktops users with improved QOS and reduced server latency.
- An efficient proxy cache replacement policy that can provide superior performance then the existing proxy cache replacement policies.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 2 presents review of the background work that includes different adaptation architecture for universal multimedia access, cache replacement schemes that are used in proxy servers and their drawbacks, and synthetic workload generator. Chapter 3 presents the proposed adaptation architecture and the cache management policy. Chapter 4 shows the performance evaluation of the proposed architecture and the cache management policy. The conclusions and directions for future work are presented in chapter 5.

Chapter 2

Review of Related Works

In this chapter, we present a comprehensive review of multimedia content, universal multimedia access and then data adaptation techniques. We also review a few selected cache replacement policies.

2.1 Multimedia Documents

Multimedia is one of the most exciting developments in the field of content representation. Multimedia refers to the simultaneous presentation of information using more than one mode of information transfer. Media include text, audio, graphics, animation, and video. Rich multimedia content is used for better interaction and communication.

Multimedia Documents [1] are most popular documents on web. A multimedia document consists of a number of different content types, such as text, still images, videos, audios and 3D scenes. The description of some of the content types is given below.

Images, audio, and video are most widely accessed content in the internet. Some of the common image formats are listed in Table 2.1. A video is a set of images, which are displayed sequentially. video track can have an accompanying audio track. Some of the movie file formats are listed in Table 2.2.

Table 2.1: Image Formats

Image Formats	Comments
Monochrome Image	1.Each pixel is stored as a single bit (0 or 1)
Gray Scale Image	1.Each pixel is usually stored as a byte (value between 0 to 255)
24-bit color Image	1.Each pixel is represented by three bytes (e.g., RGB). 2.Supports 256 x 256 x 256 possible combined colors (16,777,216)
8-bit color Image	1.One byte for each pixel 2. Supports 256 out of the millions colors possible, acceptable color quality 3.Requires Color Look-Up Tables (LUTs)
GIF (Graphics Interchange Format)	1.Uses the Lempel-Ziv Welch algorithm 2.Limited to only 8-bit (256) color images, suitable for images with few distinctive colors. 3.Supports *interlacing*
JPEG (Joint Photographics Experts Group)	1. Takes advantage of limitations in the human vision system to achieve high rates of compression. 2. Lossy compression, which allows user to set the desired level of quality/compression.
TIFF (Tagged Image File Format)	1.TIFF is a lossless format. 2.It does not provide any major advantages over JPEG and is not as user-controllable. It appears to be declining in popularity
JPEG2000	1.This format uses wavelet-based compression. 2.It also has progressive coding scheme.

Table 2.2: Video Formats

Video Formats	Comments
AVI (Audio Video Interleave)	1.It is an audio video standard designed by Microsoft. 2.Files in this format have an .AVI extension. 3.These files are limited to 320 x 240 resolution, and 30 frames per second
MPEG (Motion Picture Experts Group)	1.MPEG audio and video are the standard formats used on Video CDs and DVDs. 2.Very popular on the Internet due to its combination of high quality and high compression ratio
Quick Time	1.Developed by Apple computer 2.QuickTime supports most encoding formats, including Cinepak, JPEG and MPEG. 3.The file extension is MOV
Real Video	1.Streaming technology developed by Real Networks. 2.RealVideo uses a variety of data compression techniques and works with both normal IP connections as well as IP Multicast connections. 3.One of leading formats in Internet audio and video, giving access to more than 85% of the streaming media programming on the Web
Shockwave	1.Shockwave is the method used to display Macromedia's Director movies over the web. Director is multimedia tool that uses the "Lingo" scripting language. 2.It allows extremely complex multimedia displays to be created.

2.2 Concept of Universal Multimedia Access

The concept of Universal Multimedia Access (UMA) deals with delivery of images, video, audio and multimedia content under different network conditions, user and publisher preferences, and capabilities of terminal devices; such as Internet, wireless LAN or others from any type of terminals with varying capabilities such as mobile phones, personal computers and television sets. The primary function of UMA services is to provide the best QOS or user experience by either selecting appropriate content format or adapting the content format directly. The concept of the UMA has two aspects. From the user side, UMA allows users access to a rich set of multimedia content through various connections such as Internet, optical Ethernet, DSL, satellite and others, with different terminal devices. From the content or service provider side, UMA promises to deliver timely multimedia contents with various formats to a wide range of receivers that have different capabilities and are connected through various access networks. A major motivation behind UMA is to enable terminals with limited communication, processing, storage and display capabilities to access rich multimedia content.

Figure. 2.1 shows the concept of UMA. The UMA application suits the next generation mobile and wireless systems, as seen in the developments of third generation systems such as the European Universal Mobile Telecommunications System (UMTS) and the efforts of the Third Generation Project Partnership (3GPP). For these applications, UMA will enable users access to future services

independently on their choice of access technology, terminal equipment and usage preferences.

The mobile devices typically have limited memory, power and resolution while the mobile networks are mostly of low bandwidths and unreliable connectivity. Some of the problems are discussed in detail.

Limited Resolution: The mobile phones and PDA's have very small screens with limited resolutions and color depths. When a user using these devices accesses the conventional web page, the images on the page cannot be displayed due to the lack of resources. Hence content has to be modified to meet the requirements of the user's mobile devices.

Limited Power and Memory: The mobile devices have limited memory and hence are not suitable for displaying websites with huge data items such as images and videos. Power consumption in mobile device is another constraint, huge data requires more processing time to display and this indeed consumes more power.

Limited Bandwidth and unreliable connectivity: The mobile networks have a limited bandwidth. Hence if users want to download a webpage with rich multimedia content it takes considerable amount of time to download the entire page. The connectivity is also unreliable because of the packet loss.

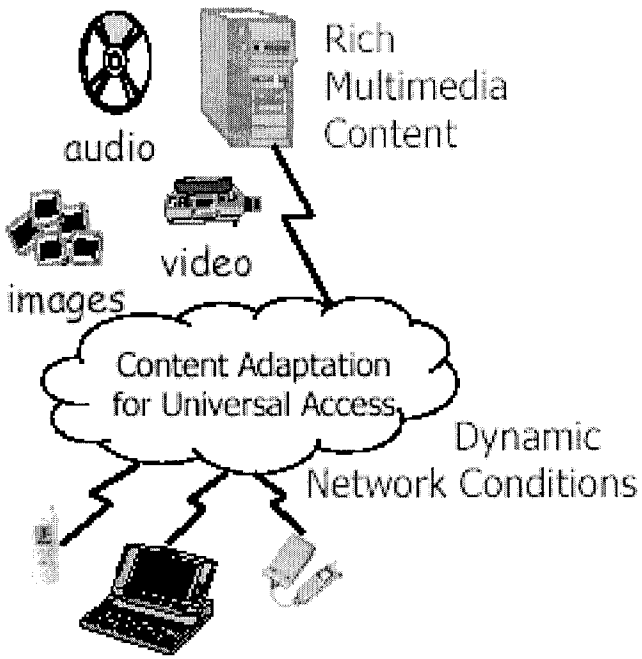


Figure 2.1 Universal access concept [2]

2.3 Content Adaptation

The adaptation is the device or mechanism that is changed or changes so as to become suitable to a new or special application or situation. The phenomenon of changing the content according to the client resources is defined as Content adaptation. The purpose of adaptation is providing better Quality of Service to the client. There are number of ways to adapt data, some of them are transcoding, distillation, compression etc. Typical examples of trancoding are conversion within media types, such as an image encoded in one standard is transcoded to another standard. The distillation process involves in data compression, it can be lossy or lossless.

Some research has been done in the area of content adaptation for low bandwidth terminals. Most web content adaptation is done at the server, at the client, or at the proxy. The adaptation at the client is not a good approach in mobile networks, these networks have very low bandwidth, which results in slow transmission of rich multimedia content. The low computational power of the device also makes the content adaptation at the device slow.

2.3.1 Adaptation at the Server

Adaptation at the server is a common approach but expensive. This is because the content has to be rewritten in multiple resolutions and/or multiple modalities. The server selects the appropriate resolution and modality based on the client resources and sends the data to the client.

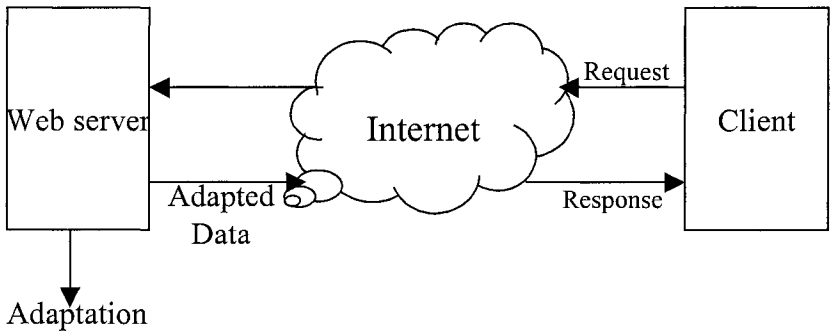


Figure 2.2 Schematic of server based adaptation

The infopyramid scheme was proposed by Mohan et al. [3] for web servers. The infopyramid is a framework used to i) combine the individual components of multimedia content-description and ii) methods and rules for handling the content and content descriptions. Figure 2.3 shows the infopyramid scheme. The infopyramid describes content in different modalities, at different resolutions and

at multiple abstractions. In addition, it may define methods for manipulation, translation, transcoding, and generation of the content. Primarily the infopyramid provides a hierarchy for content descriptors in order to guide search and retrieval.

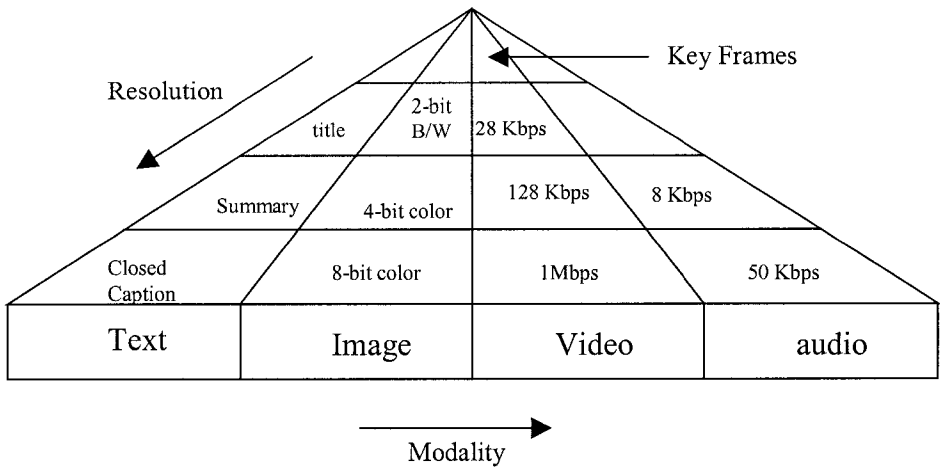


Figure 2.3 InfoPyramid scheme [3]

Multi-modal representation

Multimedia is usually not in multiple formats, or modalities. A video clip may contain some data in raw format, including closed captions in two or more languages. For certain query and retrieval tasks, the appropriate content modality may not be available. The required modality may be obtained by transforming other modalities. For example, a video clip can be transformed into images showing key-frames, while text can be synthesized into speech.

Multi-resolution presentation

Each multimedia content component can also be described in numerous resolutions. There exist various, resolution techniques to construct image and video pyramids. For example, "Flashpix" [4] provides mechanisms for storing and retrieval of still images at multiple resolutions. Figure 2.4 shows the Multiple resolution format. The features and semantics are obtained at different resolutions from the raw data, thus resulting in a feature or semantic pyramid. The features and semantics can also be obtained by using the transformed data at different resolutions between the client and the server. The server stores the data in multiple modalities and multiple resolutions. On receiving a request, it selects the required resolution and modality, which best meets the user requirement.

(a) High resolution (b) Medium Resolution (c) Low Resolution

Figure 2.4 Multi-resolution format [4]

Mohan *et al.* [3] have proposed a technique for adapting the multimedia content to optimally match the capabilities of the client devices of diverse nature. They

proposed a progressive data representation, called infopyramid scheme, in which web pages are transcoded in multiple resolutions and modalities. Here, the multimedia data is represented in the server in multiple resolutions (e.g., low, medium or high), and modalities so that they can be accessed different devices. When a client tries to access a web page, the client information is received at the server along with the request sent by the client. Based on the resources available, the server determines the modality and the resolution of the data to be sent to the client. The selection of the modality and resolution is also based on the user's resources such as bandwidth, terminal resolution, and memory. Although the infopyramid scheme defines the content in different modalities and resolutions, it does not offer the flexibility to react to the carrying adaptation needs on demand. In other words, the options are limited and fixed due to the pre-defined resolutions and modalities.

2.3.2 Adaptation at the proxy

Proxy based adaptation is a popular approach used by most content adaptation architectures. The adaptation technique requires a transparent device called proxy server to be placed between the client and the Web server. Figure 2.5 shows the schematic of the proxy based architecture. In this approach the data is stored in the server and upon request from the client the data from the server is sent to the proxy and the proxy adapts accordingly to meet the client's requirements and finally sends it to the user. The important issue the time required by the proxy to adapt the data.

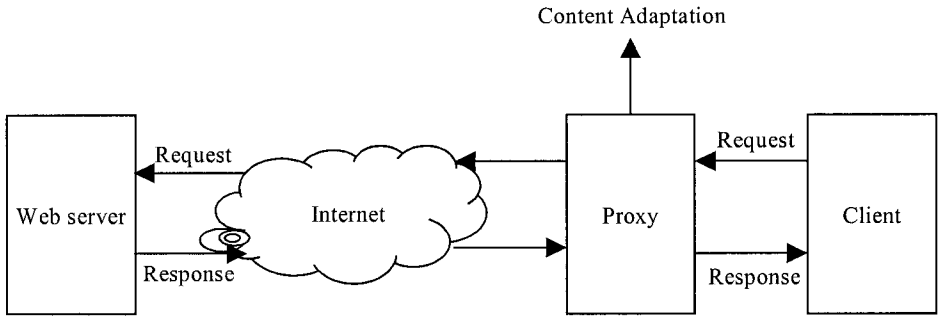


Figure 2.5 Schematic of proxy based adaptation

Mesto *et al.* [5] have proposed the media wrapper technique to adapt the web-data in the proxy server. The media wrapper uses a multimedia content model (MMCM) to represent the documents in a layered format. In addition, it employs adaptation taxonomy to adapt the multimedia data stored in MMCM format. Figure 2.6 shows the schematic of processing agent in a MMCM model. The media wrapper adapts the multimedia content based on a 10-step procedure, which is controlled by the adaptation manager. The adaptation manager receives the client information and asks the reasoner to determine the output type and releases the data information to the reasoner. The reasoner gets the information about the terminal and client, and decides output format and all other related issues and sends the information to the adaptation manager, which in turn invokes the adaptation selector. The adaptation selector asks the planner to select the agents to be used for the adaptation. The scheduler delegates the adaptation tasks to the selected agents and finally the agents convert the data into required format. The data is sent back to the adaptation manager that delivers it to the client.

Although representing the data in MMCM format makes the adaptation easier, the media wrapper has so many phases and requires excessive processing time to perform all the processes before adapting, this leads to significant delay.

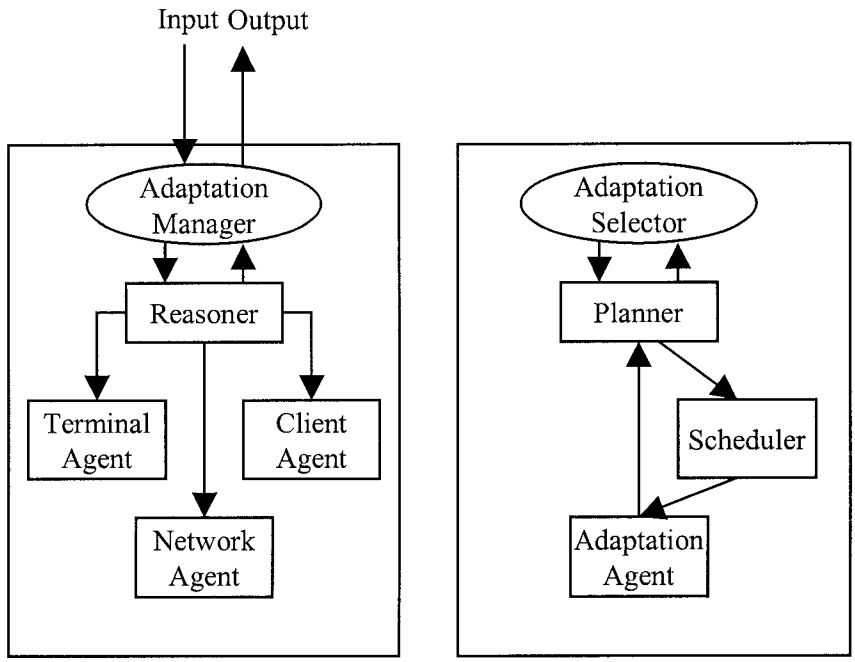


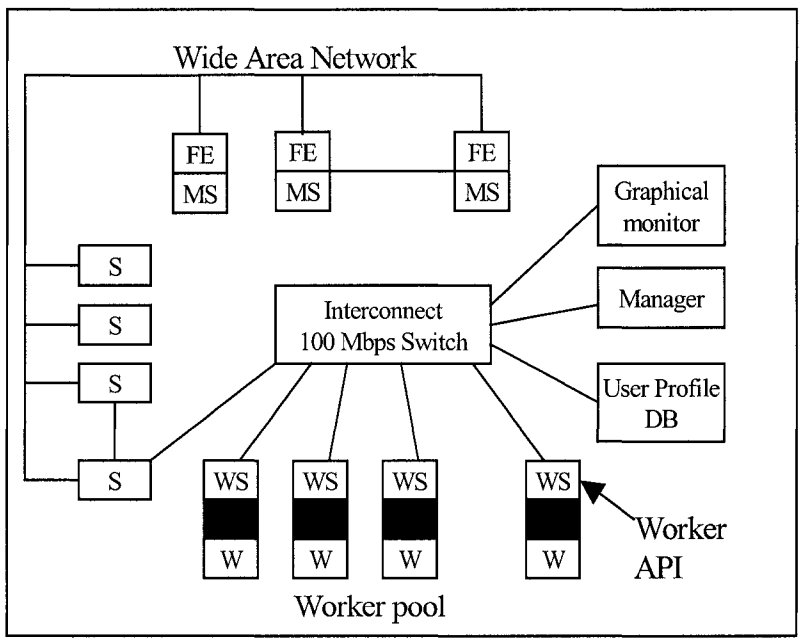
Figure 2.6 Schematic representation of the processing agents in MMCM model [5]

Cluster-based TACC server Architecture

Fox *et al.* [6] have proposed a proxy-based adaptation with "Data type specific Distillation". Here, the lossy compression mechanisms are used such that they behave differently for different data types. This approach is better than "generic" compressors, because the generic compressors cannot make intelligent decisions about what information to throw away because they do not understand the semantics of the data. The authors also proposed a cluster based proxy server Architecture, known as TACC (Transformation, Aggregation, Caching and

customization architecture). This server is actually made up of server workstations working together as a group.

Figure 2.7 shows the architecture of a cluster based TACC server. The Front Ends (FEs) are the connections to the http servers. The workers (W) are the workstations doing one or many specific tasks assigned to them. The manager assigns jobs to workers based on the functions to be performed. The cluster-based approach is a scalable, and cost effective solution. But the scheduling of the workers in cluster is time consuming and difficult task. This adds up to the transmission delay of the data. This problem can be clearly observed in streaming applications, which are becoming popular in the Web. In addition, managing the cluster of workstations to perform distributed processing is difficult.



WS: Worker Stub, S: Cache, MS: Manager stub

Figure 2.7 Architecture of a cluster based TACC Server [6]

Dynamic Adaptation (Image Transcoding proxy)

Han *et al.* [7] have proposed an framework for determining whether and how much to transcode an image to transmit to the client with lower bandwidths. The transcoding is done generally when the total time to transmit the client is reduced as a result of it. In this approach the transcoding proxy adapts the image according to the network resources while trying to meet an upper bound on the delay tolerated by the end user.

The basic architecture of the transcoding proxy is shown in the Figure 2.8. This proxy is built by integrating a transcoding sub-system into an HTTP proxy. There are two primary components of the transcoding subsystem 1) the policy module and 2) transformation module. The transformation module modifies the data before sending it to the client. The policy module makes the decision which transcoding algorithm to use. This decision is based on several criteria such as

1. The bandwidth between the "client and proxy" and "proxy and server".
2. The client device's resources such as display capabilities etc.
3. The preference of the user concerning the preferred rendering of the data.

Han *et al.* [7] have proposed an analytical framework for deciding when to transcode and when not to. The objective of the framework is accurately predict image transcoding time, output size of the image and network bandwidth between the proxy, client and the server. The policies discussed are implemented on images in JPEG and GIF format.

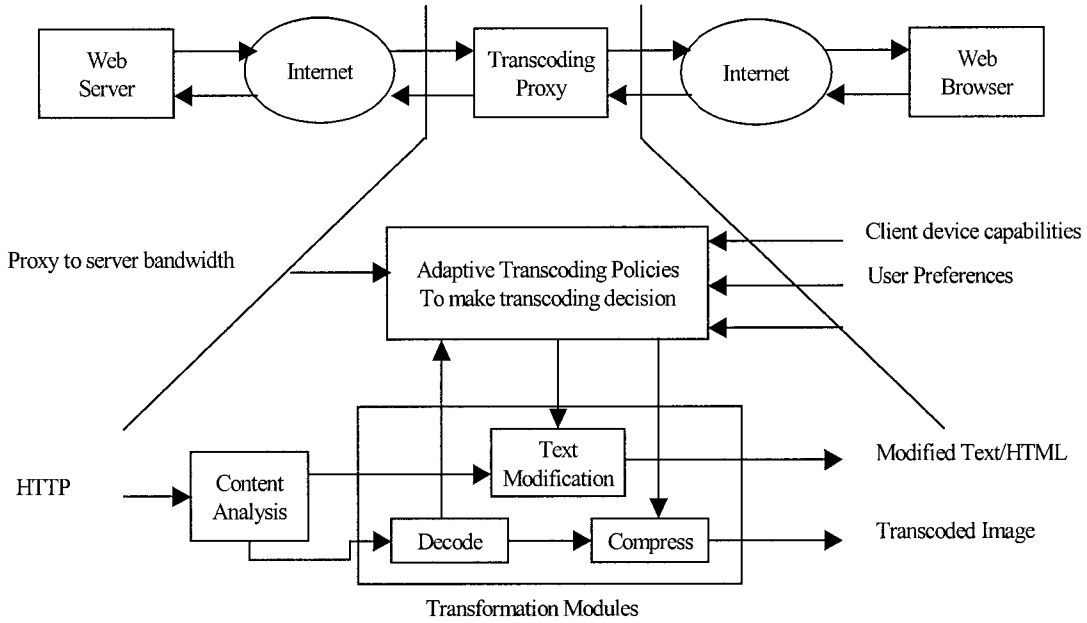


Figure 2.8 Architecture of Image transcoding proxy [7]

S = Original size of the image

S_p = Transcoded size of the image

B_{sp} = Bandwidth between server and proxy

B_{pc} = Bandwidth between proxy and client

D_p = Image transcoding delay

In an automated store and forward image transcoding proxy, transcoding is performed if the following condition is satisfied.

$$D_p(S)+S/B_{sp}+S_p(S)/B_{pc}<S/\min(B_{pc},B_{sp}) \qquad (2.1)$$

When proxy to client link is a bottleneck in a store-and-forward proxy system (ie $B_{pc} < B_{sp}$), transcoding is performed in [7] only if

$$D_p(S) + S / B_{sp} < [S - S_p(S)] / B_{pc} \qquad (2.2)$$

The Eqs. (2.1) and (2.2) denote that transcoding is done if round trip transmission time with proxy adaptation is less than round trip transmission time with no proxy adaptation.

Distributed Adaptation Technique

Khan *et al.* [8] proposed a distributed adaptation technique (DAT) for satellite and cellular networks. Figure 2.9 shows the schematic of DAT technique. In the DAT, the data is stored in multiple resolutions and multiple modalities (MRMM) at the server (similar to the infopyramid scheme). But, it distributes the load between the server and the proxy, and has the flexibility to react to the on demand request. Although, the DAT provides an overall better performance compared to infopyramid scheme, it has not considered any cache management at the proxy. Note that the proxy caching reduces the response time significantly as the proxies are located closer to the clients. Hence, an efficient cache management is crucial to achieve good performance. The performance of the distributed adaptation is not evaluated in the work proposed by khan *et al*, which is very critical in an architecture.

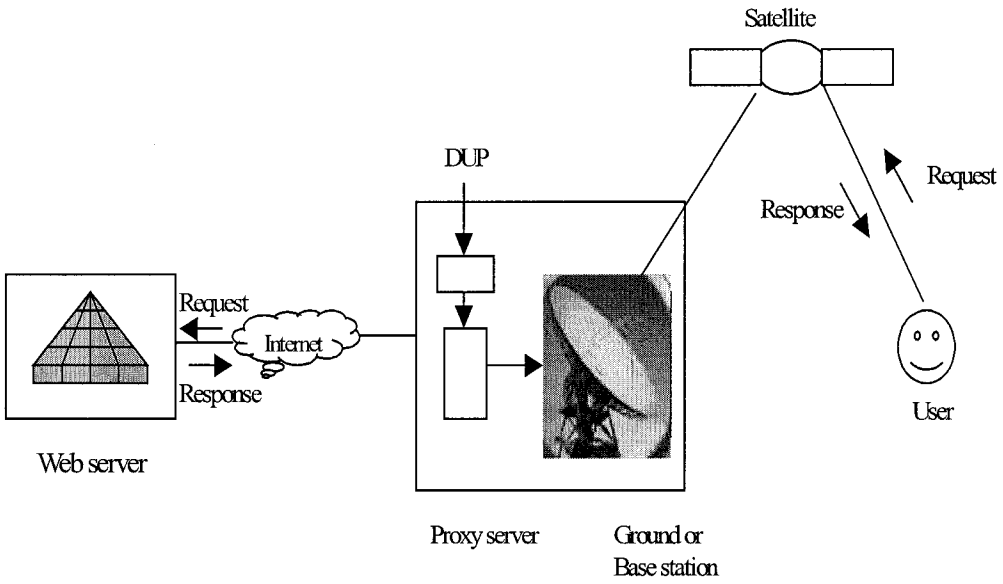


Figure 2.9 Distributed adaptation technique for cellular and satellite networks [8]

2.4 Cache

Caching is a technology that is already familiar in the context of hardware. Many hardware devices cache frequently used instructions and data in order to speed processing tasks. The WWW can be considered as a large distributed information system that provides access to shared data objects. The WWW is growing exponential in time, two of the major problems that today's web users are suffering from are the network congestion and server overloading. Researchers [11-15] have been working on how to improve web performance since early 90's. Caching popular objects at locations close to the clients has been recognized as one of the effective solutions to alleviate web service bottlenecks, reduce traffic over the Internet and improve the scalability of the WWW system.

The rapid increase in the usage of the Internet is crashing its backbone with huge amount of traffic and packet loss due to packet collisions. Figure 2.10 shows the effects of network congestion and how caching can reduce it. The caching servers mounted closer to the client that are shown in Figure 2.10 can clear reduce the stress on the Internet backbone by many folds. This kind of caching results in faster response to the clients, which improves the overall QOS of the system.

Proxy caching has become one of the useful approaches for reducing the network congestion and latency. There are several advantages of using proxy caching

1. Proxy caching reduces bandwidth consumption, thereby decreases network traffic and network congestion.
2. Proxy caching reduces access latency due to two reasons
    a) Frequently accessed documents are fetched from nearby proxy cache instead of remote data servers, the transmission delay is minimized.
    b) Because of the reduction in network traffic, those objects which are not cached can also be retrieved relatively faster than without caching due to less congestion along the path and less workload at the server.

3. Proxy caching reduces the workload of the remote web server. The number of requests sent from a proxy to the server will be reduced since much of the data is available in the proxy's cache.

If the remote server is not available because of server crash or network breakage, the client can obtain a cached copy at the proxy. Thus, the robustness of web service is enhanced.

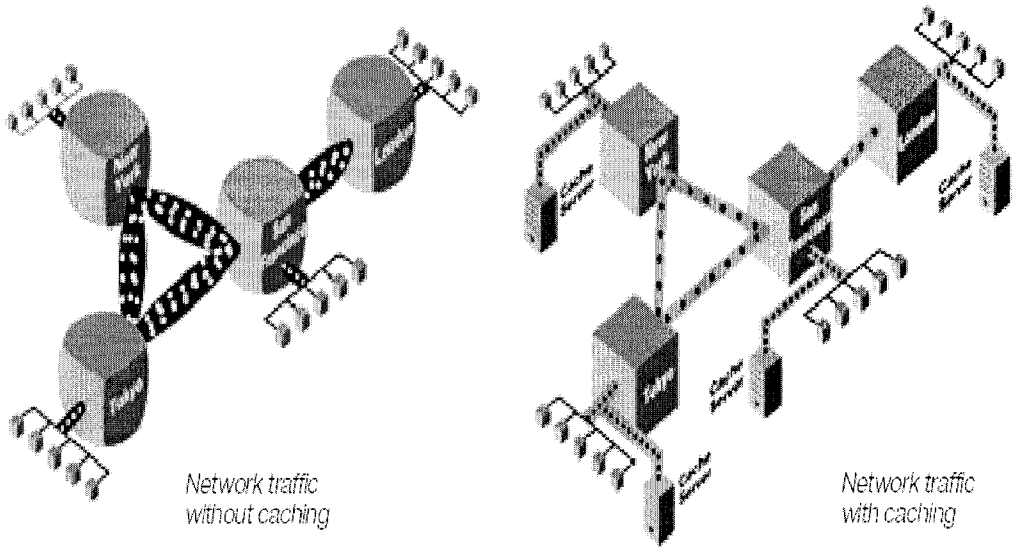


Figure 2.10 Scenario showing the effectiveness of Caching in a Network [16]

In general, a proxy server has a fixed amount of storage. When the storage fills up, the proxy must choose one or more media objects based on a certain caching replacement policy. The goal of the replacement policy is to make the best use of available resources, including disk and memory space as well as network bandwidth. To achieve this goal, the cache replacement policy should be able to

accurately predict future popularity of objects and determine how to use its limited space in the most advantageous way.

Cache Replacement policy is one of the key components in the cache management. The Cache Replacement Policy decides which object to be removed from the cache when it is full. There are different types of cache replacement policies. Some of them are listed

1. *Least Recently Used* (LRU) [17],
2. *Least Frequently Used* (LFU) [17]
3. *Size* [18]

These policies use access time, frequency and size as the criteria for eviction of the objects. The LRU policy evicts the objects that are least recently used. That means that it uses the time at which the object is accessed as criteria to evict. The LFU policy evicts the objects that are least frequently used. The size-based policy evicts the object based on size and evicts the larger objects. These caching policies are more suitable for static data where data adaptation is not required.

With the recent growth in multimedia applications, the future proxies are required to support multiple data types. The proxy based approach has become popular in Universal Multimedia Access (UMA) Architecture. The current cache replacement policies are not efficient in this type of architecture because the criteria considered for eviction of objects do not depend on the network conditions or the device resources. In UMA based architecture the network and client resources are important factors to consider because the data is changed

according to these conditions. The adaptation time and round trip time of the multimedia objects are important factors to be considered to achieve superior proxy performance.

A few caching policies have been proposed recently, which can work for both continuous (e.g., audio and video) and discrete (e.g., text and image) media data. Yu *et al.* [19] have proposed a network adaptive cache policy for mixed media. This policy prioritizes the objects based on the media type. In this policy priority, tendency, and frequency are used to calculate the weight of the object. Note that priority represents the importance of an object whereas tendency is the probability of the request hitness. When a new object comes in, if there is no free space in the cache, the proxy flushes out the object with the lowest weight. The weight function in this approach is defined as follows:

$$w = priority \times (\beta \times tendency + (1 - \beta) \times frequency)$$

where *priority* represents the importance of an object. The priority depends on the type of the object and may be on type of application. The *tendency* indicates the impact of the current request on the following requests according to the continual characteristics of the media. *Frequency* represents the popularity of the object to be accessed. β is the control parameter balancing the impact between *tendency* and *frequency*. Tendency shows the probability of the following request hitness.

$$\text{Frequency} = \frac{1}{MTTR}$$

where MTTR (mean time-to-re-access) is measured as the weighted sum of the inter-arrival times between the previous accesses.

$$w(i) = \alpha \times w(i-1), \alpha \leq 1 \text{ and } w(0) = 1-\alpha$$

$$MTTR = \sum_{i \geq 0} (t_i - t_{i-1}) \times w(i)$$

Thus for a given time t_0 $MTTR(t_0) = (1-\alpha)(t_0 - t_1) + \alpha \times MTTR(t_1)$

This policy suffers from cache pollution, which is very common in WWW. Cache pollution can be defined as a popular object suddenly becoming unpopular in the cache, but can exist in cache with its previous popularity. This is a major problem in UMA architectures because of the size of multimedia data relatively larger than static data which in turn occupy huge disk space to store these data.

Xiang *et al.* [20] proposed a cost-based cache replacement policy for wireless networks. In this policy cost value is calculated based on media distortion cost (source and channel distortion), startup latency cost and network fetching cost.

$$C(g_s) = p_t \times C_t(g_s) + p_d \times C_d(g_s) + p_q \times C_q(g_s)$$

where p_t, p_d, p_q are the weight parameters that stand for the unit price of network cost, latency and media distortion cost respectively.

$$C_t(g_s) = size(g_s) \times Dist(RTT \times Frequency)$$

where Size(g_s) is the size of the media object, Dist(RTT) is the distance between the remote server and the proxy.

$$C_d(g_s) = \begin{cases} delay(RTT \times frequency) & if \quad t < D_{threshold} \\ 0 & else \end{cases}$$

where *delay*(*RTT* X *frequency*) is the delay for delivering prefix from the server to the proxy which is a function of RTT between server and proxy.

Media distortion cost considers source distortion and channel distortion. The media source distortion cost can be calculated by

$$C_q(g_s) = distortion(g_s) \times frequency$$

where *distortion* (g_s) is the quality distortion of the media object. In general distortion is measured in PSNR.

Although cost-based policy performs well for wireless channels, it suffers when used in the wired networks because the factors considered in this policy have significant impact in wireless networks than wired networks. This policy also suffers from cache pollution, and hence it is not suitable for the UMA architecture.

2.5 Workload Generation

Williamson *et al.* [21] developed a synthetic workload generator for generating synthetic traces, which can be used for evaluating different cache replacement policies. Figure 2.11 shows the graphical user interface of the workload generator.

The top portion of the GUI allows the user to set parameters to control the workload characteristics, prior to hitting 'generate' button. An entry box at the

top of the GUI specifies the name of the trace file to be generated. Sliding scale widgets are used to specify the number of references desired in the generated workload, as well as the number of distinct web objects and the number of one timers. Separate sliders are used to specify the slope for the Zipf-like document popularity distribution, the slope for pareto tail of the document size distribution, and the degree of statistical correlation between the size and popularity of web objects. Positive correlation means that larger objects are more likely to be referenced. Negative correlation means that smaller objects are more likely to referenced. The default setting of zero correlation means that document size and document popularity are independent characteristics.

The workload generator generates the trace in the format shown in Table 2.3. The first column is a time stamp representing the time in seconds at which a specific web object (URL) is requested. The second column is a document identifier, a unique integer assigned to each URL represented in the workload trace. The third column represents the size in bytes of a specific web object. This size is fixed throughout the trace for a given document id.

Table 2.3: A typical synthetic trace

TIMESTAMP	DOC_ID	SIZE
0.03245	0	1958
2.73954	9	366
3.47710	4	2536
4.1692	0	1958

The trace generated depends on different parameters that should be given as an input. The parameters are zipf slope [20], popularity bias, and Pareto tail index. In recent years there have been many studies on page request distribution. Numerous studies have found that this distribution follows Zipf's law. Zipf's law expresses a power-law relationship between the popularity "P" of an item and relative rank "r" among the referenced items, based on frequency of occurrence. The relationship is of the form $P = \frac{c}{r^{\beta}}$, where c is a constant and β is often close to 1. The Workload generator provides control over five key workload characteristics, namely one-time referencing, file popularity, file size distribution, correlation between file size and popularity, and temporal locality.

The advantage of using this workload generator is that it allows the user to generate various traces with different combinations of input parameters. This makes the evaluation of cache management polices more efficient.

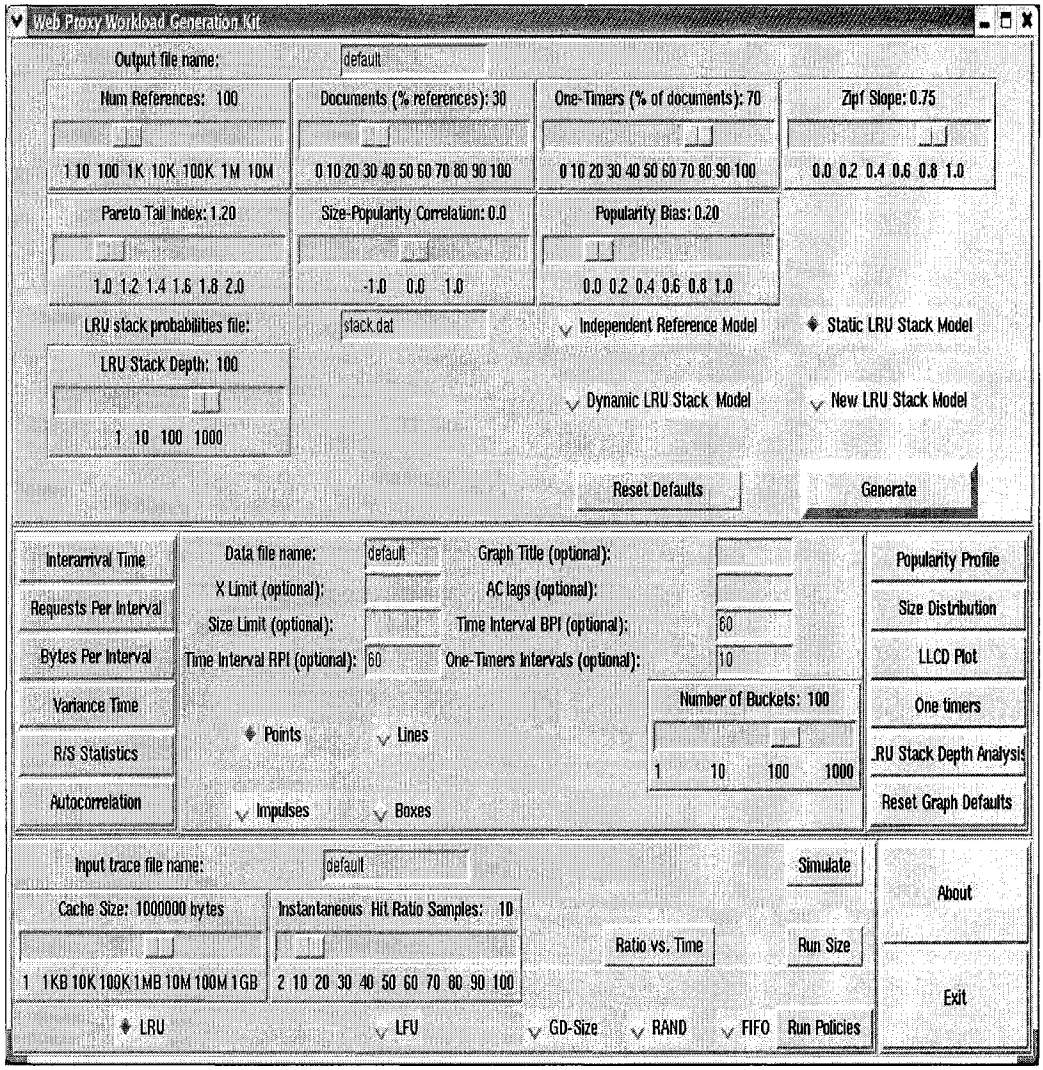


Figure 2.11 Snapshot of the Workload Generator [16]

2.6 Summary

A comprehensive review of various content adaptation architecture and their drawbacks were presented in this chapter. Various cache replacement policies that are used in the WWW were also reviewed. Performance setbacks of these policies when used in UMA based architectures were also discussed. A detailed review on the workload generator that will be used later in this thesis for performance evaluation of the cache replacement policies was presented.

Chapter 3

Proposed Architecture

In the previous chapter, several content adaptation architecture were presented. It was mentioned that these architecture are not efficient in providing better QOS. In this chapter we are going present the proposed architecture for content adaptation. For Most existing architectures, content adaptation is performed at the proxy or at the server. This results in extra processing load at the server or proxy and hence results in a slower response. This slow response degrades the overall performance when congestion occurs. Most of the adaptation architectures do not exploit the cached data in the proxy to the maximum level. The efficient usage of cached data at the proxy results in faster response time. Proxies are typically located closer to the clients.

The proposed architecture exploits the concept of distributed adaptation where partial adaptation is performed at the web server and remaining adaptation is performed at the proxy with efficient caching mechanism. This reduces the network latency, load on the server and improves the performance of the proxy. The proxy has the capability to adapt the cached data efficiently.

3.1 Adaptation Architecture

The schematic of the proposed architecture is shown in Figure. 3.1. The architecture has three main components – the server, proxy and cache, which are briefly discussed in the following.

3.1.1 Web-Server

The web server in the proposed architecture is a critical component. It has been observed in Chapter 2 that the infopyramid is a flexible format for representing web-data. The data represented in infopyramid can be accessed by a variety of client devices. However, the processing load at the server may still be high, and the format is not flexible for on-demand adaptation. In addition, it would be difficult to re-write the existing web pages in infopyramid format because it requires huge disk space for this representation. In the proposed architecture, the web server has the data is represented in a multi-resolution and multiple modalities (MRMM) format. The MRMM format is a simplified version of the infopyramid representation where the processing at the server is kept at a minimum level, and most of the required processing is performed at the proxy.

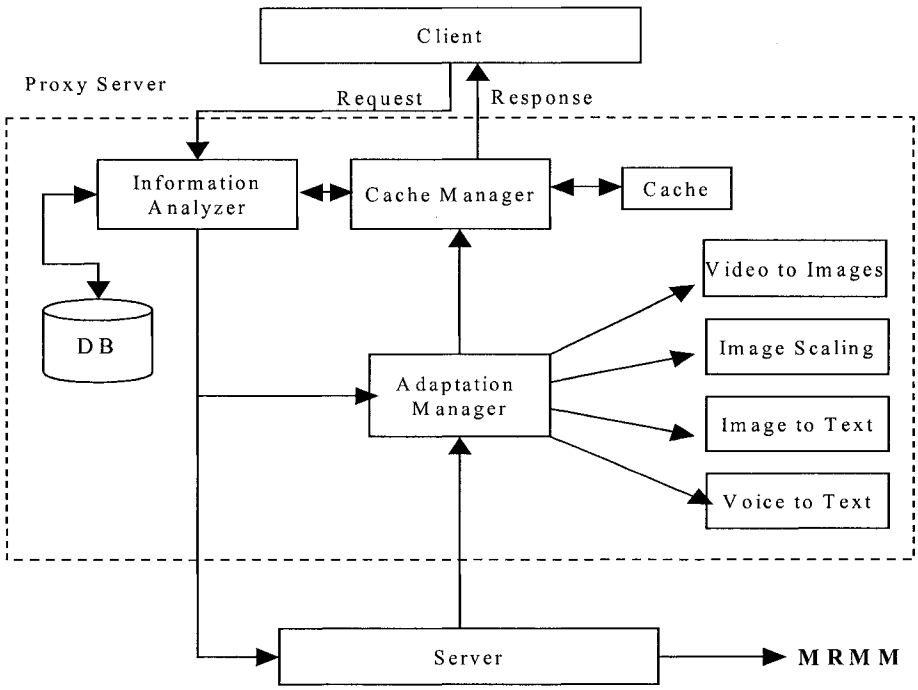


Figure 3.1 Proposed Content Adaptation Architecture

3.1.2 Proxy Server

The proxy server in the proposed architecture is a transparent agent that communicates with both the web server and the client. As shown in Figure 3.1 the proxy server has several modules. The information analyzer in the proxy server receives the client request and the request is sent to the cache manager to check whether the requested file is stored in the cache. The information analyzer gets the response from the cache manager if the file exists in the cache or not. If the file does exist in the cache the information manager gets the client information from Database Manager (DB) and then sends it along with the requested file to the adaptation manager. The adaptation manager makes the decision whether to adapt the cached data or not. Once the decision is made the

data is sent to the cache manager. The cache manager assigns the cost using the proposed cache replacement policy and stores the data in the cache. If the file does not exist in the cache the information manager sends the client profile and the request to the server. The server adapts the data according to the client profile and sends the file to the proxy. The adaptation manager makes the decision whether to still adapt the data or not based on the client profile. The adaptation manager adapts the data using one of appropriate modules that is suitable for requested data. The adaptation manager sends the adapted data to the cache manager. The adapted data is stored in the cache and then it is sent to the client.

The detailed functionality of each module is explained below

Adaptation Manager

Adaptation Manager is one of the key components of the architecture. It uses a decision making mechanism to accurately calculate the estimated transmission time between Server, Proxy and the client. The adaptation manager can choose a module like Image transcoding or compression or Video to Images or Images to Text or Voice to Text based on the client resources.

Cache Manager

Cache Manager is an important component in the proposed architecture. The proxy caching is a better approach for alleviating network congestion and reduce latency through distributed network load. Cache replacement policy is a key component in the cache management. The performance of the replacement policy can be determined by the hit ratio, byte hit ratio and traffic saving cost metrics.

Cost based replacement policy replaces the objects based on the cost of the Media objects. If a new object is requested and if the proxy is not full the object is simply cached in the proxy. If the proxy is full, the object with lowest cost will be removed. This operation stops only when the cost of the requested object is lower than those of all the objects in the cache.

Information Analyzer

This module checks the response from the cache manager whether the requested file is there in the cache or not. If the requested file is there in the cache, the Information Analyzer will get the client profile from the Database Manager and then send the file along with client profile to the adaptation manager. The adaptation manager decides whether to adapt the data or not. If the requested file is not there in the cache, the analyzer sends the client request along with the client profile to the server. Figure 3.2 shows the flow chart of the Information analyzer.

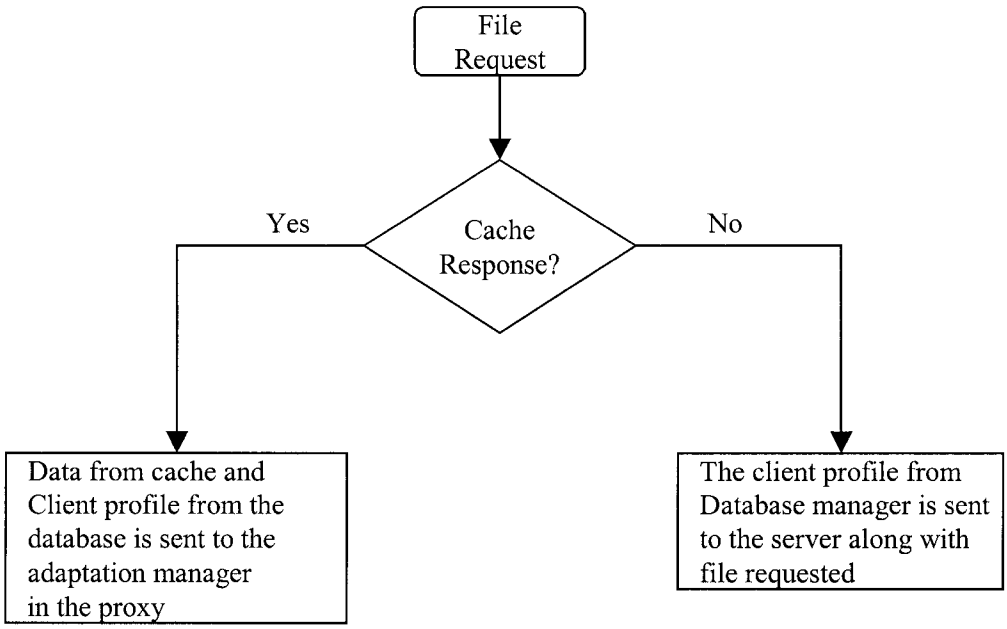


Figure 3.2 Flow chart for the Information Analyzer Decision

3.2 Timing analysis of the proposed architecture

Adaptation Manager makes the decision whether to adapt the data at the proxy or not. If the data has to be adapted it will request the module selector to select the appropriate module to adapt the data. The adapted data is send back the adaptation manager and from there it is stored in the cache by the cache manager. If the data do not need adaptation then it will be sent directly to the client. The proposed architecture proxy can also adapt the cached data, which is efficient mechanism to improve the performance. The timing analysis for decision making for adaptation of Data is given by the following equations.

Data fetched from the server

$$T_{sp} = \frac{D_s}{B_{sp}}$$

$$T_{Ser} = \frac{X_S}{B_S}$$

$$T_{pc-p} = \frac{D_t}{B_{pc}}$$

$$T_{pc-up} = \frac{D_s}{B_{pc}}$$

$$T_p = \frac{D_s X_p}{S_p B_t}$$

$$T_{WA} = T_{Ser} + T_{sp} + T_p + T_{pc-p}$$

$$T_{NA} = T_{sp} + T_{pc-up}$$

where

D_s :	Data Size of the Original Image
D_t :	Data Size of Transcoded Image.
T_{ser}:	Time taken by the data to get processed at the server.
B_{sp} :	Bandwidth between Server and Proxy.
B_t :	Internal Bandwidth of the Transcoding Proxy.
B_s :	Internal Bandwidth of the Server.
B_{pc} :	Bandwidth between Proxy and Client.
B_{sc} :	Bandwidth between Sever and Client.
T_p :	Time taken by the data to get processed at the proxy
T_{sp} :	Time taken by the data to be sent from server to proxy.
$T_{pc\text{-}up}$:	Time taken for un-processed data to be sent from the proxy to client.
$T_{pc\text{-}p}$:	Time taken for processed data to be sent from the proxy to client.
X_p :	Load on the proxy server (in-terms of the number of jobs being processed)
X_s :	Load on the server
S_p :	Speed of the processor used in the Proxy server
T_{WA}:	Total time taken by the data to reach the client after adaptation
T_{NA} :	Total time taken by the data to reach the client without adaptation

Timing Analysis for Cached Data

The decision whether to adapt the cached data or not is done by using the following timing analysis.

$$T_{cpc-p} = \frac{D_{tc}}{B_{pc}}$$

$$T_{cpc-up} = \frac{D_{sc}}{B_{pc}}$$

$$T_{pc} = \frac{D_{sc} X_p}{S_p B_t}$$

$$T_{WA} = T_{pc} + T_{cpc-p}$$

$$T_{NA} = T_{cpc-up}$$

we will define the following parameters used in the decision making process for cached data.

D_{tc} :	Data Size of Transcoded cached Image.
T_{cpc-up} :	Time taken for un-processed cache data to be sent from the proxy to client.
T_{cpc-p} :	Time taken for processed cache data to be sent from the proxy to client.
T_{pc} :	Time taken by the cache data to get processed at the proxy

The data will be adapted in the proxy if the total transmission time with adaptation (T_{WA}) is less than the total transmission time with no adaptation (T_{NA}).

$$T_{WA} < T_{NA}$$

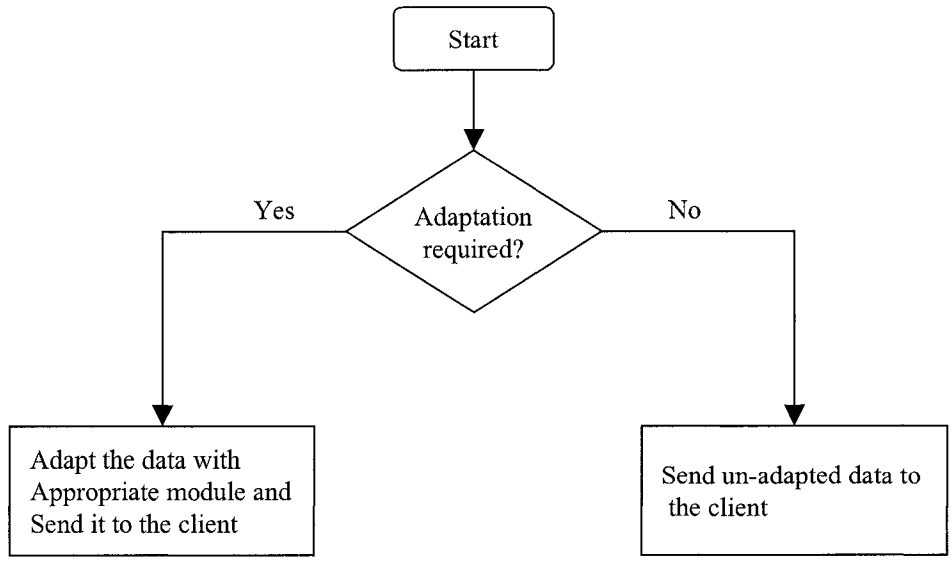


Figure 3.3 Flow chart for the Adaptation Decision

Case Study

Consider a scenario where the bandwidth between the server and proxy is 2MB, the bandwidth between the proxy and the client is 20kbps, the internal bandwidth for processing is 1MB. The data size that has been transmitted to the proxy is 300 KB. Size of the data after adaptation is 150KB

$$T_{sp} = \frac{D_s}{B_{sp}} = \frac{300K}{2M} = 0.15\,\text{sec}$$

$$T_{pc-p} = \frac{D_t}{B_{pc}} = \frac{150k}{20k} = 7.5\,\text{sec}$$

$$T_{pc-up} = \frac{D_s}{B_{pc}} = \frac{300k}{20k} = 15\,\text{sec}$$

$$T_p = \frac{D_s X_p}{S_p B_t} = \frac{300k \times 4}{1M} = 1.2\,\text{sec}$$

$$T_{WA} = T_{sp} + T_p + T_{pc-p}$$
$$= 0.15 + 7.5 + 1.2 = 8.85 \text{sec}$$

$$T_{NA} = T_{sp} + T_{pc-up}$$
$$= 0.15 + 15 = 15.15 \text{sec}$$

If the data exists in the cache then the calculation can be done as follows

$$T_{cpc-p} = \frac{D_{tc}}{B_{pc}} = \frac{150k}{20k} = 7.5 \text{sec}$$

$$T_{cpc-up} = \frac{D_{sc}}{B_{pc}} = \frac{300k}{20k} = 15 \text{sec}$$

$$T_{pc} = \frac{D_s X_p}{S_p B_t} = \frac{300k \times 4}{1M} = 1.2 \text{sec}$$

$$T_{WA} = T_{pc} + T_{cpc-p}$$
$$= 7.5 + 1.2 = 8.75 \text{sec}$$

$$T_{NA} = T_{cpc-up}$$
$$= 15 \text{sec}$$

Because $T_{WA} < T_{NA}$ the proxy adapts the data and sends it to the client.

3.2.1 Control Flow in the Architecture

Figure 3.4 shows the control flow of the architecture when a client request for a file. The client ID is grabbed by the information analyzer to check for the client profile information from the database. The information analyzer also checks with the cache manager for the existence of the requested file in the cache. Based on

the response from the cache manager, the file request goes to the server or the adaptation manager in the proxy for further processing.

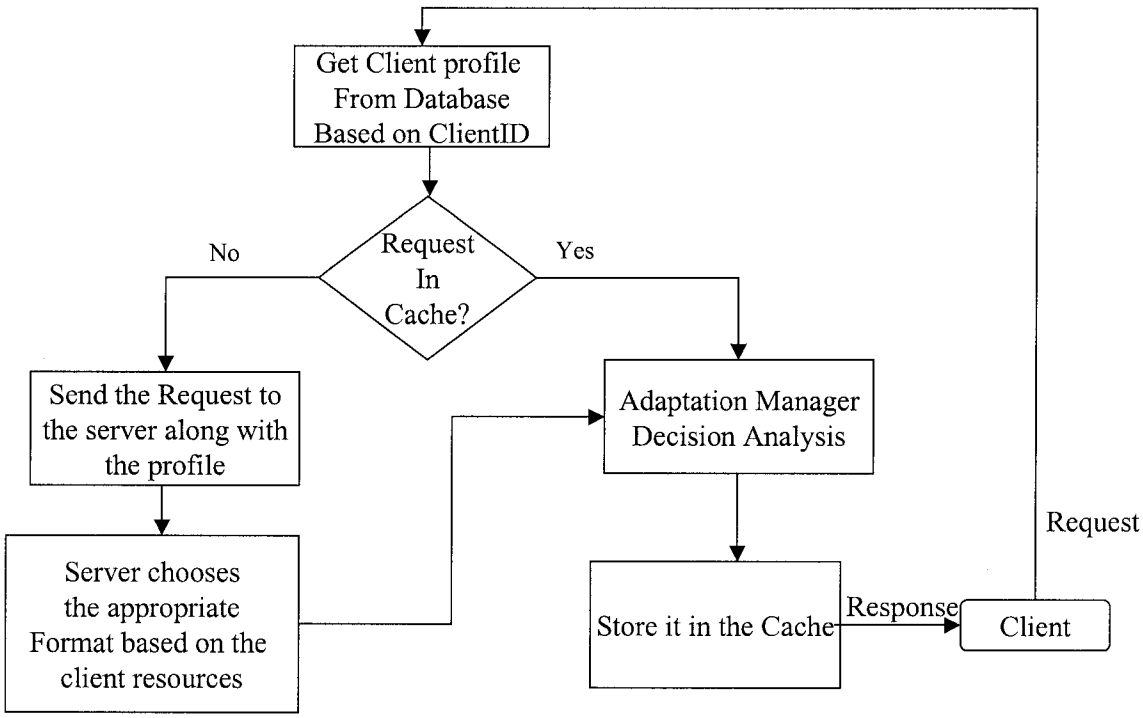


Figure 3.4 Flow chart of event execution in the Architecture

3.3 Cache Management Criteria

The key component of proxy caching in the proposed architecture is the cache replacement policy. We briefly mentioned the network adaptive cache policy [19] and the cost-based cache replacement policy [20] in chapter 2. Although, these policies generally perform well, they suffer from cache pollution that is very common in the WWW. Hence these policies are not suitable for the UMA-type architecture.

In this thesis, we propose a cost based cache replacement policy where we make best use of the available resources such as disk space and network bandwidth. It considers the following factors that have significant impact on the different data types.

1. Round trip time for fetching the object,

$$RTT_M = \frac{Size\ of\ the\ object}{Bandwidth\ \ between\ \ proxy\ \ and\ server}$$

2. Adaptation Time at the proxy server (if adapted)

$$T_{AM} = \frac{Size\ of\ the\ object \times complexity}{Internal\ Bandwidth\ \ of\ \ proxy}$$

3. Request Frequency of the object
4. Aging Factor

Round trip time is consider because it will change with the type of the media transmitted. The order of size variation for different types of media can be given as Text<Image<Audio<Video. Preference can be given based on the type of object that is transmitted. And also RTT gives the distance of the server to proxy. The cost of fetching an object from local server is less than the cost of fetching from a remote server.

Adaptation time is critical factor in the proxy because importance should be given to object based on the complexity of the adaptation process.

Frequency is used to determine the popularity of the object. *Aging factor* is used to decrease the cache pollution in the proxy cache.

The cost calculation in the proposed policy includes two aspects: i) fetching cost of the media object from the server, and ii) the adaptation cost of media objects. The *Fetching Cost* (C_F) is calculated as follows:

$$C_F = p_f \times RTT_M \times f_M$$

where RTT_M is the round-trip-time to fetch the media object from the server to the proxy, f_M is the request frequency of the media object, and p_f is the unit cost of the fetching time.

Note that the data adaptation time depends on the proxy internal bandwidth and adaptation complexity (which generally depends on the type of adaptation and media). Hence, the adaptation cost (C_A) is calculated using the following equation.

$$C_A = p_a \times T_{AM} \times f_M$$

Where T_{AM} is time required to adapt a media object, and p_a is the unit price of the adaptation.

Cache pollution is a common problem in the WWW, where a popular object suddenly becomes unpopular, but the object still remains in the cache because of the previous popularity. In order to reduce the cache pollution, we add an *Aging Factor* to the cost value. Aging factor determines how recently the object has been accessed. So when an object is brought to the cache, the aging factor is zero. When there is a hit, the aging factor of the requested object is set to the current time and is added to the total cost of the object. The aging factor of an object is calculated as follows. Assume that an object was first put in the cache at time t_0,

which is set to zero. After that the object has been accessed N times at time $[t_1, t_2, t_3, ...t_N]$. The aging factor (AF) will then be:

$$AF = \sum_{t=1}^{N} (t_k - t_{k-1})$$

For instance the object has been accessed 3 times from a cache then the AF is will be

$$\begin{aligned} AF &= (t_1 - t_0) + (t_2 - t_1) + (t_3 - t_2) \\ &= t_3 - t_0 \\ &= t_3 \end{aligned}$$

AF is nothing but the current time at which the object is accessed. Incorporating the aging factor, the final cost value C_T can be calculated as follows.

$$C_T = C_F + C_A + p_{af} \times AF$$

Where p_{af} is the unit price of proxy access time.

The cache manager calculates the cost (C_T) of each object. When the cache is full, the cache manager removes the objects with lowest C_T value in order to create space for new objects having higher cost values.

3.4 Application Scenario

Figure. 3.5 shows the application scenario of the proposed architecture. The proposed architecture is flexible to serve multiple clients with different resources. For instance a desktop user request a file that does not exist in a cache, the proxy sends the request along with the client information to the server. The server uses its decision making scheme to decide which format of data should be sent. As the

request is from a desktop user who has lot of resources, the server sends a higher resolution data. Proxy gets it from the server and it cache the data, and sends it to the client. If the next user is a PDA requesting for the same file then proxy gets from the cache, analyzes the resources of the client and adapts if it is necessary accordingly and sends it to the client. This mechanism provides faster response time to the client. The adaptation of the cached data can improve the performance of the proxy server and QOS of the all system. This approach reduces load on remote servers by considerable amount.

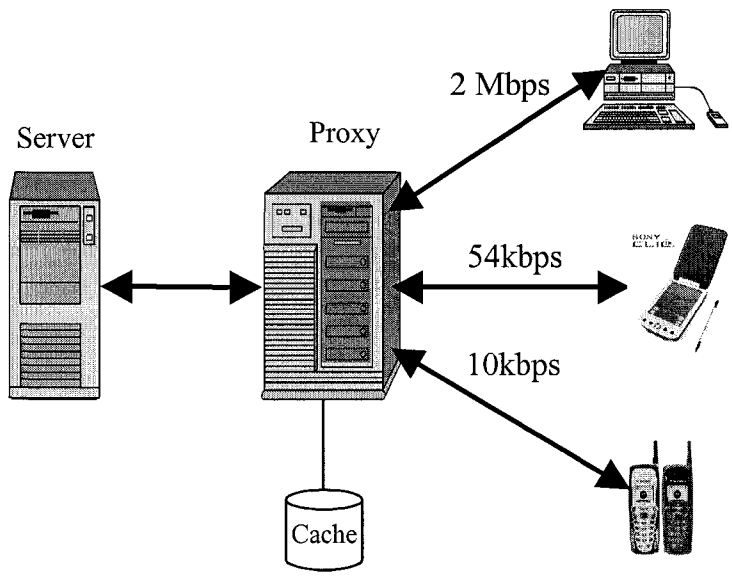


Figure 3.5 Application Scenario of the proposed architecture

3.5 Advantages of Proposed Architecture

The proposed architecture has several advantages.

1. Bandwidth of the network is saved when reduced resolution and a few modalities are sent to the proxy server. The bandwidth and transmission time are saved.
2. Flexibility can be obtained by allowing the users profile to be changed if a client wants to use different resources with same terminal.
3. The adaptation is flexible enough. The bandwidth of the channel is further reduced, the filtering or "data type specific distillation" can be performed at the proxy to ensure that the data rate suites the users requirements exactly.
4. Caching in the proxy server reduces the round trip time involved in fetching the document if the document is available in the cache.
5. Adaptation of the cached objects saves the server bandwidth and increases the proxy performance and the QOS.
6. If this architecture is used in satellite networks, it saves the bandwidth of the satellite channel uplink (from the user to the ground-station) as only an ID has to be sent by the client because the architecture maintains a client profile database, based on the client ID the client profile can be retrieved from the database.
7. Proxies are located closer to the clients, and hence the response time is reduced significantly.
8. Reduce the cost of building multiple configurations for different networks.

3.6 Summary

In this chapter a content adaptation architecture for universal multimedia access has been proposed. Different components of the architecture and their functionalities have been presented in detail. An efficient cache replacement policy for UMA based architecture has been proposed. A decision making scheme which calculates transmission time and uses them to make decisions for adaptation manager has been presented. The various advantages of the proposed architecture have been presented.

Chapter 4

Performance Evaluation

In previous chapter, we presented the proposed architecture for content adaptation. In this chapter we will evaluate the performance of the proposed architecture. First we evaluate the proposed cache replacement policy using the trace driven simulator developed as a part of the work proposed. We then evaluate the content adaptation architecture using the simulation test bed developed in our laboratory.

4.1 Performance Evaluation of the Cache Policy

Figure. 4.1 shows the simulation setup for performance evaluation. A synthetic workload generator developed by Williamson *et al.* [21] is used to generate the traces for the simulation. The traces generated are used by the simulator to evaluate the performance of the proposed replacement policy. The trace driven simulator is written in Java language. The trace driven simulator can calculate the hit ratio, byte hit ratio and traffic saving cost metrics of different policies.

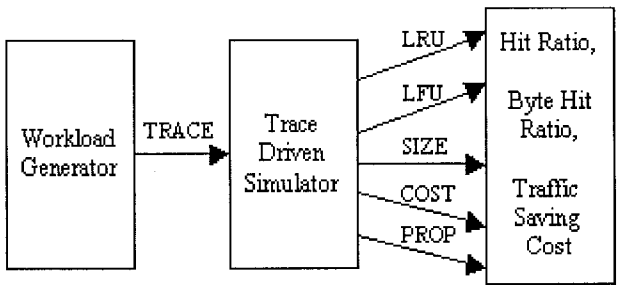


Figure 4.1. Simulation set up for performance evaluation

Performance evaluating metrics definitions

Hit Ratio: Is the percentage of number of hits from the cache over the total number of request.

$$\eta_{HR} = \frac{\text{Number of Hits}}{\text{Number of Hits} + \text{Number of Miss}} \times 100 \qquad (4.1)$$

Byte Hit Ratio: Is the percentage of number of bytes transferred from the cache over the total number of bytes requested.

$$\eta_{BHR} = \frac{\text{Numberof ByteHit}}{\text{Numberof Bytes Hit} + \text{Numberof Bytes Miss}} \times 100 \qquad (4.2)$$

Traffic saving cost: Is a metric for measuring the reduction in the network congestion cost. Traffic saving cost is measured in terms of giga bytes of data that has been transferred from the proxy as a result of hits in the cache.

4.1.1 Assumptions for the simulation

The bandwidth between the proxy server and the server is assumed to be fixed and is high. The internal bandwidth of the proxy is assumed to be fixed. Note that the internal bandwidth of the proxy server is an important factor that determines the adaptation time required for an object of given size in the proxy.

Table 4.1 shows the different input parameter given to the workload generator for generating synthetic traces. In the past studies on document request patterns, researchers [23-28] have found that the *zipf* slope value varies between 0.65 to 0.8 in a typical proxy server. Zipf slope is denoted by 'α'. Hence, in this thesis we have used traces with α=0.75 and α=0.80.

Table 4.1: Trace Specifications

	Trace 1	Trace2
No of references	0.1 million	0.5 million
Document (% references)	30	40
One Timers	70	70
Zipf Slope (α)	0.80	0.75
Pareto tail Slope	1.20	1.20
Size popularity correlation	0.03	-0.03
Popularity Bias	0.20	0.20

The zipf slope gives relationship between document rank and popularity. This parameter is crucial because it varies with positioning of cache in the network. The value of zipf slope is different if the cache resides in the web server and if it resides in the proxy. Size popularity correlation is also varied to check the performance with larger and smaller size objects in the trace1 and trace 2. Figure. 4.2 shows the popularity profile of two different traces with α = 0.80 and α=0.75 respectively. These traces are used as input for the trace driven simulator developed for evaluating the performance of the different cache replacement policies.

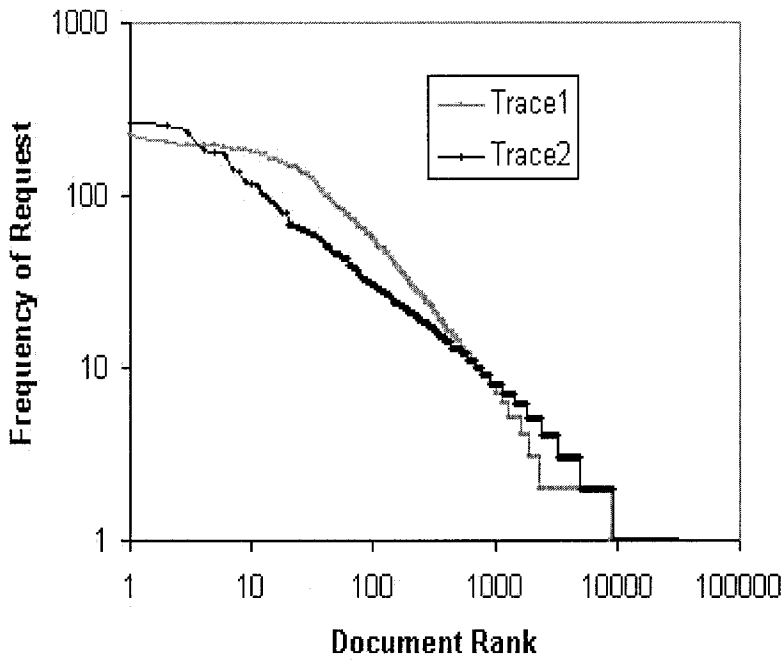
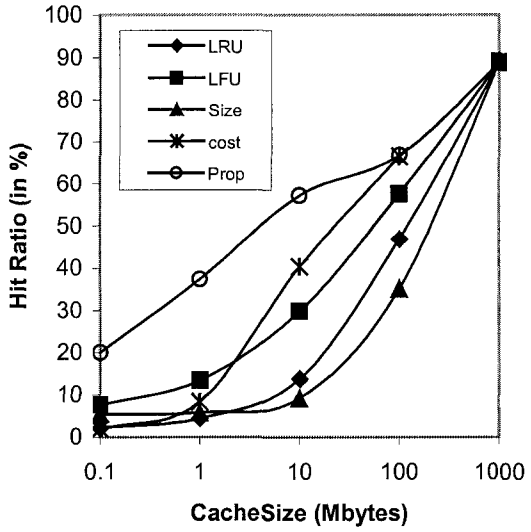


Figure 4.2 Document Popularity profile Trace 1 with α=0.80, and Trace 2 with α=0.75.

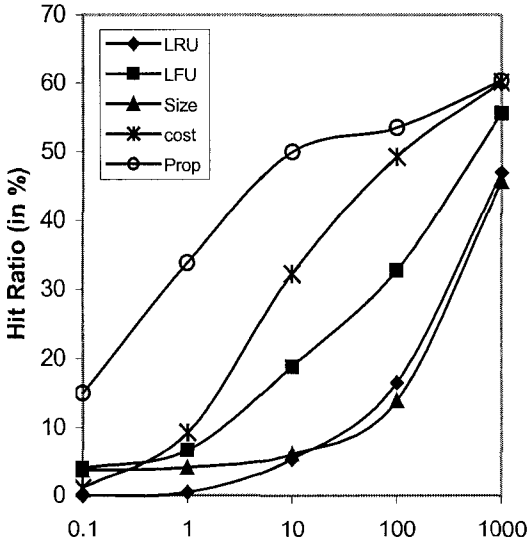
To evaluate the performance of our proposed replacement policy, we compare it with the LRU, LFU, Size, and cost based policy. Hit Ratio and Byte Hit Ratio can vary depending on the trace used for the simulation. A trace might have high hit ratio but it does not imply that it should have high byte hit ratio because the hits might on smaller size objects. In the same manner a trace having a high byte hit ratio might not have a high hit ratio. This metrics some times may be identical for some traces, so evaluate the performance using these metrics is very crucial.

We vary the cache size to study the *hit ratio* of the proxy caching for different polices for the two traces. Figure. 4.3 shows the hit ratio of the different cache replacement policies. It is observed that the hit ratio increases with the increasing cache size for all policies, which is expected as large cache can store more

objects. It is also observed that the proposed replacement policy performs better than the LRU, LFU, Size, and cost based replacement policies.



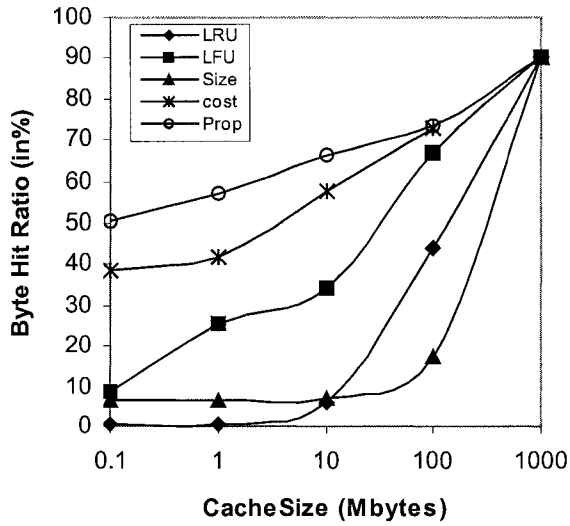
(a)



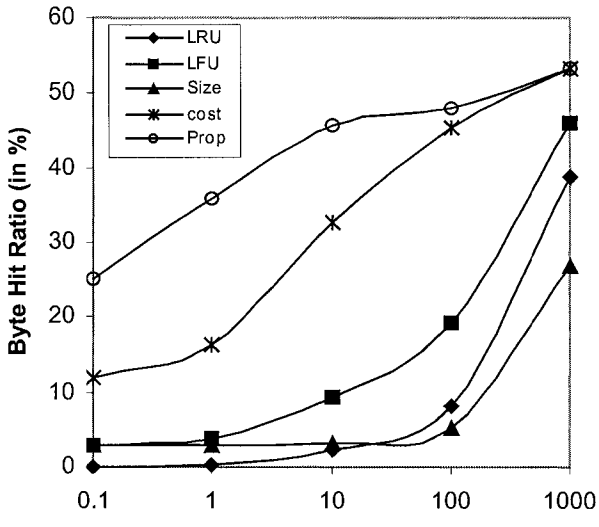
(b)

Figure 4.3. Hit ratio versus cache size. a) trace1 and, b) trace2.

Figure. 4.4 shows the byte hit ratio of different cache replacement policies. We vary the cache size to study the *byte hit ratio* performance. It is observed that the proposed replacement policy performs better than other replacement policies.



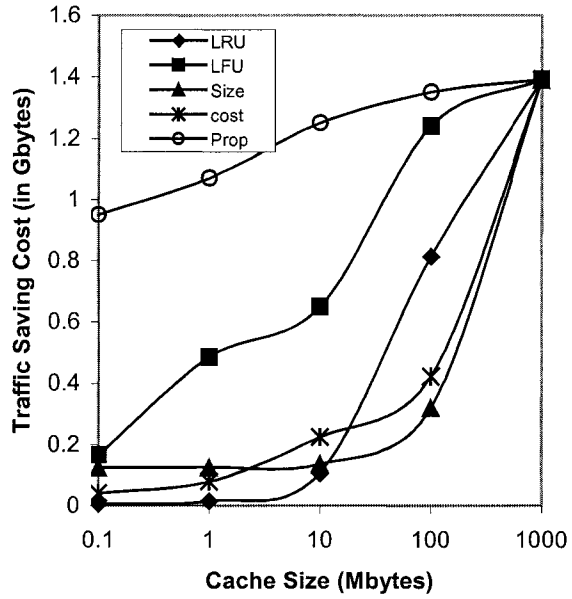
(a)



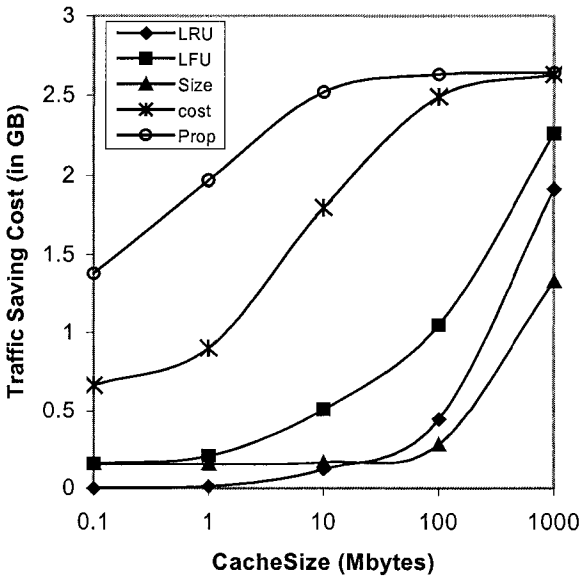
(b)

Figure 4.4 Byte hit ratio versus cache size of a) trace1 and b) trace2.

Figure. 4.5 shows the traffic saving cost in giga bytes for different cache replacement policies. The cache size is varied to study the traffic saving metric. It is observed that the proposed replacement policy provides higher network cost saving value than other replacement policies.



(a)



(b)

Figure 4.5 Traffic saving cost versus cache size. a) trace1, and b) trace2.

In some cases the other policies can perform better than the proposed policy where the trace parameters are different. The traces used in the simulations have the parameters, which are most common traces have on the web. The proposed policy might suffer when the trace has objects with smaller size and more one timers. The generation of such a trace which has very large one timers and smaller size is difficult task. So in this thesis we could not determine the exact performance degradation of the proposed policy in such scenario. The cache replacement policy is designed for UMA architectures in which most of the objects are multimedia objects whose size usually large.

4.1.2 Description of the Simulator

Figure. 4.6 shows the snapshot of the simulator, which is written in the Java programming language. The name of the trace should be typed in the input field "Name of Trace", cache size should be specified in the next field. The required cache policy should be selected from a drop down menu list. If the user clicks on the button "Get Hit Ratio", then the simulator will display the number of hits for the trace, number of misses, and the percentage (i.e. the hit ratio). If the user clicks the button "Get Byte Hit Ratio", then the simulator will display the number of byte hits, number of byte misses and the percentage (i.e. the byte hit ratio). The number of byte hits gives the number of bytes transferred from the proxy as a result of the hit, which gives the traffic saving cost.

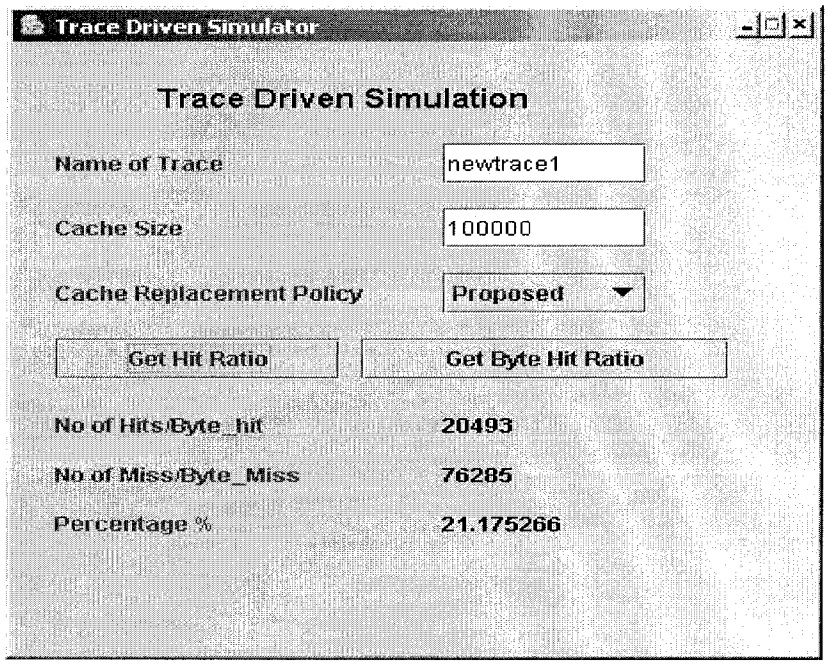


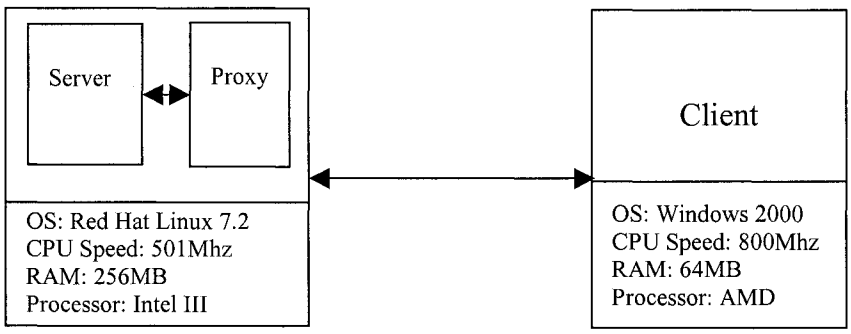
Figure 4.6 Snapshot of the simulator

4.2 Performance of the Overall Architecture

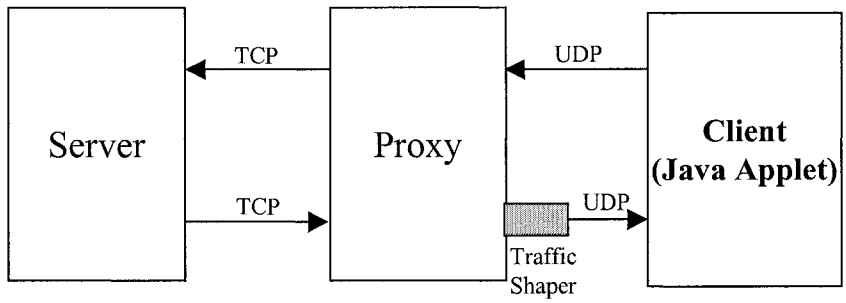
The performance of the proposed architecture is evaluated by creating a low bandwidth environment. Low bandwidth environment is created at client side by reducing the transmission rate at the proxy. This is done on the Linux workstation by using a traffic shaper (TS) [40]. Note that the TS acts as a virtual device that can control the outgoing traffic of the system. The TS creates a pseudo-networking device that is used by the system, but it relies on the underlying Ethernet interface to actually carry the traffic. Using the TS, the bandwidth can be varied from 10-250 kbps. The mobile devices generally have a bandwidth ranging from 9-30 kbps, and hence the bandwidth is varied between 10-100 kbps in this simulation.

The client, proxy and server programs are written in Java, as it gives the flexibility to run the simulation on different operating environments. In the simulation the proxy and server runs on the same workstation, and the bandwidth between them is very high.

Table 4.2 shows the configuration of the shaper device used for controlling the traffic. The line-1 of code attaches the shaper module to the Ethernet, which is the physical transmission device. The line-2 of code set the bandwidth of the shaper at what rate it has to transmit the data. The bandwidth is varied by changing speed in the line-2. The line-3 configures the system IP address to the shaper. The line-5 deletes the existing gateway configuration. The line-6 will add the new gateway configuration with the traffic shaper and this makes the traffic go through the traffic shaper device.



(a)



(b)

Figure 4.7. Simulation setup for performance evaluation a) System specification b) schematic of the simulation setup.

Using simulation setup shown in Figure 4.7, we evaluated the following scenarios - i) the adaptation at the server without any adaptation at proxy, and ii) the adaptation of cached data at the proxy. The proxy transmits the image data to the client through UDP. The packet size of the UDP can be specified in the proxy.

Table 4.2: Traffic shaper configuration

Configuration Commands
shapecfg attach shaper0 eth0
shapecfg speed shaper0 10000
Ifconfig shaper0 129.128.197.172 netmask 255.255.255.0 up
Route add –net 129.128.197.0 netmask 255.255.255.0 dev shaper0
route delete default eth0
route add default gw 129.128.197.1 shaper0

Figure. 4.8 shows the snapshot of whole system developed for simulation. The client interface has several options for requesting GIF, JPEG2000 and JPEG Images. The client can request various image qualities. It displays the total time taken for receiving the images at client side. The system uses UDP and TCP based connection to communicate between the client, proxy and the server

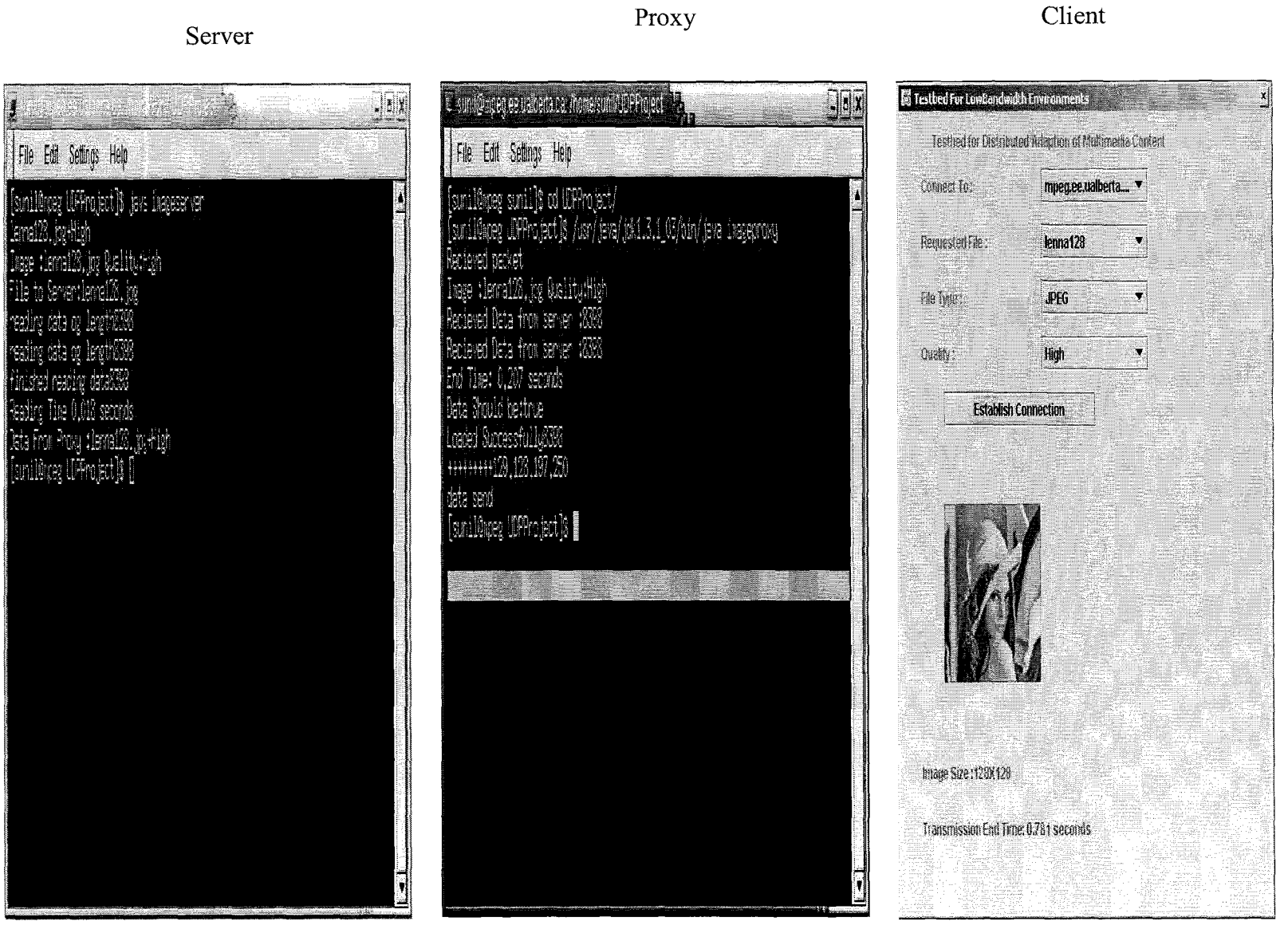


Figure 4.8 Snap shot of the entire 3-tier System developed for simulation

Figure. 4.9 shows the JPEG images used in the simulation. Figure. 4.10 shows the transmission time for JPEG-compressed images. We observe in Figure 4.10 that at lower bandwidths $T_{WA} < T_{NA}$, and hence the adaptation of the JPEG images reduces the transmission time. As the bandwidth increases, $|T_{NA} - T_{WA}|$ decreases, and eventually will become zero. Let us denote this bandwidth as the critical bandwidth. In other words, at the critical bandwidth, $T_{WA} = T_{NA}$. Note that below the critical bandwidth, the data adaptation reduces the transmission time (at the cost of the image quality). The overall transmission time of the JPEG image at various bandwidths is shown in Fig. 4.11. If a lower quality image is acceptable to the client, the adaptation manager adapts the data before transmission; otherwise, the proxy sends the unadapted data with a larger transmission time. This is shown by two sets of transmission time (when bandwidth < critical bandwidth) in Fig. 4.11. If the available bandwidth is above the critical bandwidth, then $T_{WA} > T_{NA}$, and hence adaptation is not beneficial.

(a)High Resolution (b) Medium Resolution (c)Low Resolution

Figure 4. 9. JPEG compressed images of size 256x256 stored in the server in MRMM format.

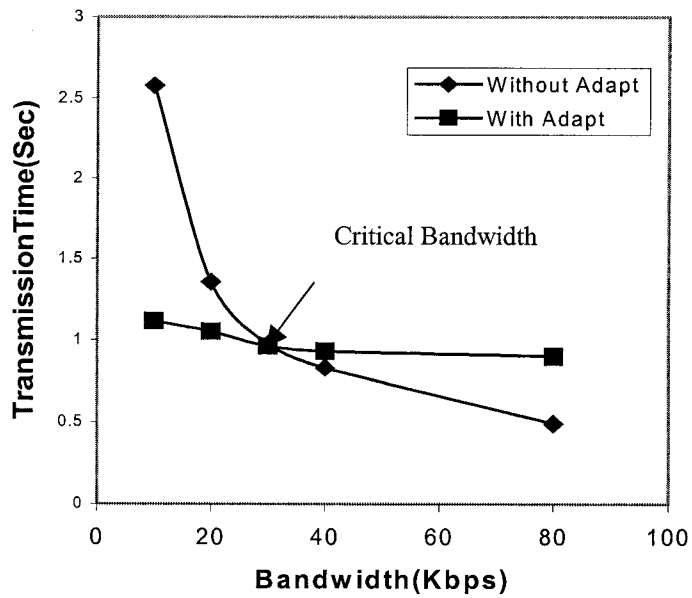


Figure 4.10. Total transmission time of JPEG images at different bandwidths.

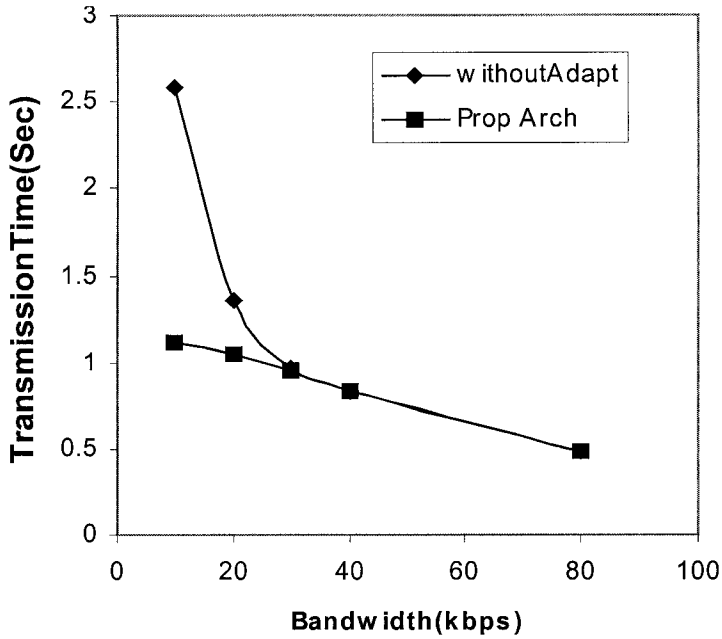


Figure 4.11. Total transmission time of JPEG images at different bandwidths. Adaptation decision is made using condition $T_{WA}<T_{NA}$.

Figure. 4.12 shows the JPEG2000 compressed images used in the simulation. Note that the JPEG2000 provides embedded bitstream, and hence the data can be considered to be in MRMM format. This reduces the problem of storing images of different resolution at the server. Figure. 4.13 shows the performance of the architecture with varying bandwidths. As the data is in the embedded form, the server decides the size of the bitstream to be sent to the client based on the client resources. It is observed that there is a trade-off among the bandwidth, transmission time and image quality. The proposed architecture has the ability to decide the best policy depending on the user requirement.

(a) (b) (c)

Figure 4.12 JPEG2000 Compressed images using KAKADU software with varying bits/pixel a) High Resolution (1 bits/pixel) b) Medium Resolution (0.7bits/pixel) c) Low Resolution (0.5 bits/pixel)

Figure. 4.14 shows the transmission times when the requested data is available in the proxy-cache. As the data is available in the cache, the transmission times in Figure. 4.14 are smaller compared to Figure. 4.13. It shows the proxy caching mechanism improves the performance and provides faster response time to the clients.

The performance evaluation results in section 4.1 show that the proposed policy provides a better hit ratio than other policies. The proposed architecture uses this policy, which provides better overall performance than other policies. The fetching time from the server is saved due to the effective caching mechanism. Adapting of the cached data can reduce the response time, which can be seen in Figure. 4.13. The overall Quality of Service of the system is improved.

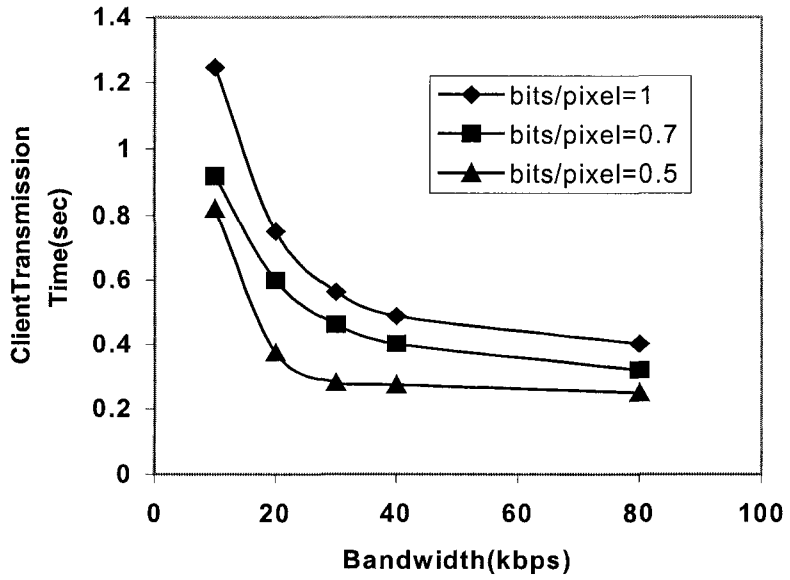


Figure 4.13. Client transmission time (server to client) of JPEG2000 compressed images at different bandwidth.

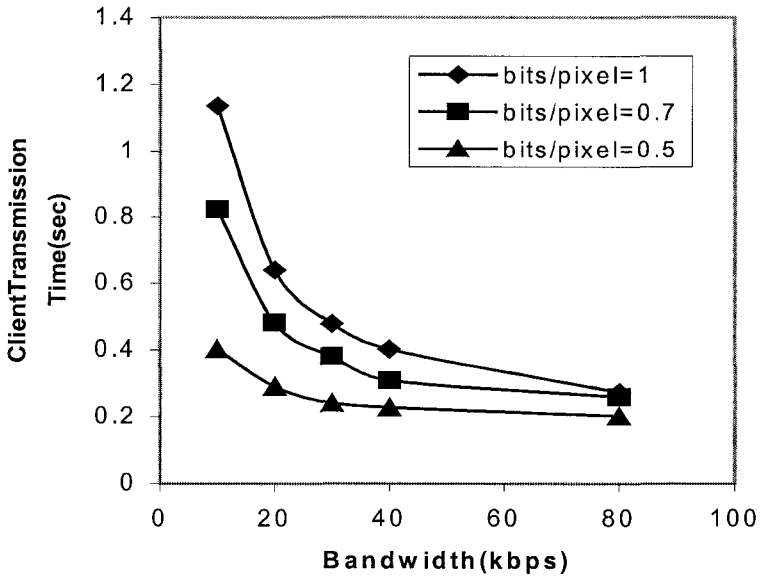


Figure 4.14. Client transmission time (proxy to client) at various bandwidth using cached data at the proxy.

The overall performance can be shown by evaluated by using the following analytical analysis.

Case study

From the performance results of section 4.1 for 10 MB of cache size the proposed policy can achieve 40% hit ratio. We can calculate the average time of the total system by using the formula

$$\text{Average Time} = 0.4 \times T_{PC} + 0.6 \times T_{SC}$$

where T_{PC} is the transmission time between the proxy and the client and T_{SC} is the transmission time between the server and the client.

From the Figures. 4.13 and 4.14 the time taken for a high quality image at 20 Kbps takes 1.34 sec if there is a miss in the proxy cache and takes 1.18 sec if there us a hit in the proxy cache.

Average time $= 0.4*(1.18)+0.6*(1.34)$

$= 0.472 + 0.824$

$= 1.296$ sec

The average time taken for a server to transmit a high quality image at 20 kbps in 2-tier architecture using other existing cache policy is 1.37 sec. The proposed architecture provides faster response times.

Figure. 4.15 shows the performance of the proposed architecture when compared with a 2-tier architecture. The decision-making scheme in the proxy helps to decide which data should be adapted and which should not be. From the transmission results we can observe that the proposed architecture can transmit

varying resolutions of images based on client resources. The client response times is reduced considerably when compared with respective to the 2-tier architecture. The effectiveness of the scheme makes a considerable impact on the QOS of the overall system.

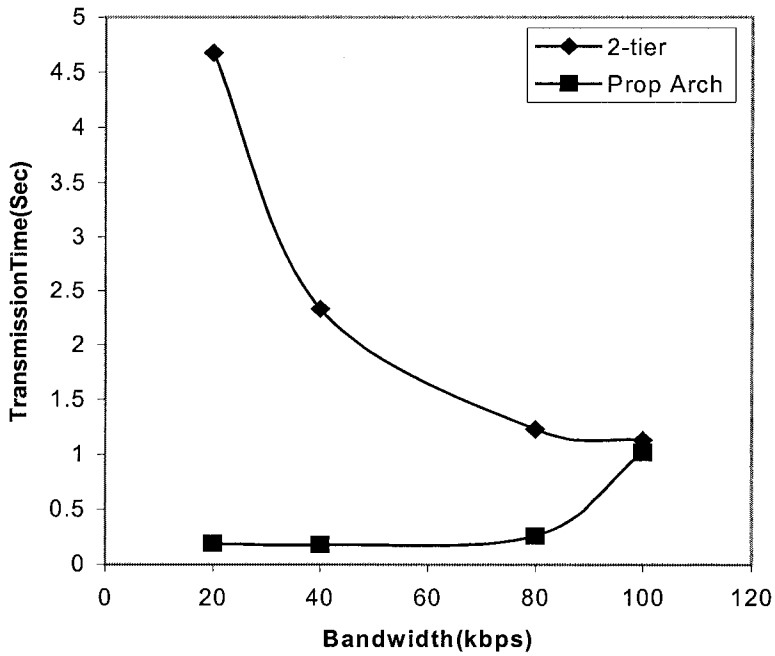


Figure 4.15 Performance of proposed architecture when compared with a 2-tier architecture

4.3 Summary

Performance of the proposed cache replacement policy was evaluated by comparing the performance metrics with the other existing replacement policies. The results show that the proposed replacement policy provided superior performance. The proposed content adaptation architecture was evaluated using

the simulation set up. Different transmission times were noted to evaluate the performance of the system with cached data at the proxy, with adaptation at the server and adaptation of cached data at the proxy server. The results show that the proposed architecture along with proposed cache replacement policy can perform better than existing architectures.

Chapter 5

Conclusions and Future work

In this thesis an efficient content adaptation architecture for Universal Multimedia Access has been proposed. The content adaptation is performed on the fly. The proposed architecture adapts data both at the proxy and content server. The distributed adaptation reduces the load on the proxy and the content server.

The proposed architecture [44] efficiently uses the cached data in the proxy, which reduces the network congestion, latency and improves the proxy performance. The simulation results show that the response time is reduced and server side latency is also reduced.

An efficient cache replacement policy [45] that provides superior performance than the existing cache replacement policies has also been proposed. The simulation results prove that the proposed cache replacement policy will provide better performance. The proposed cache replacement policy also improves the efficiency of the proxy server by allowing more request served from the proxy cache.

The proposed architecture has an efficient decision making scheme which decides when to adapt the data. This decision-making criteria makes the

architecture more robust. The proposed adaptation architecture with the caching mechanism will be very efficient in mobile and wired networks especially for large size multimedia data.

Future Work Recommendations

The performance of the proposed architecture has been evaluated using a simulator under lighter load. It would be useful to evaluate the performance under heavy client loads. Although simulation gives an indication of how the architecture works, it would be important to implement and evaluate the architecture in real time environment. There is also a need to develop a load balancing strategy for effective distribution of client loads between the content server and the proxy. Client profiles can also be implemented using the CC/PP (Composite Capability/ Preference Profiles) protocol [46]. CC/PP protocol gets the client resource from a mobile device in the form of a XML document, which can be stored in the database.

Bibliography

1. M. K Mandal, "Multimedia Systems and Signals," Kluwer Academic, Jan 2003.
2. http://www.ercim.org/publication/Ercim_News/enw48/perkis.html
3. R. Mohan, R. Smith, and C. S. Li, "Adapting Multimedia Internet Content for Universal Access," *IEEE Transactions on Multimedia,* Vol. 1, No.1, pp. 104-114, Mar 1999.
4. http://www.kodak.com/US/en/digital/flashPix/
5. M. Mesto, A. Koivisto and J. Sauvola "The Media Wrapper in the Applications of Multimedia Content for Mobile Environments," *Proceedings of SPIE: Multimedia Systems and Applications III,* Vol. 4209, pp. 132-139, Boston, MA, 2001.
6. A Fox, S. D Gribble, and E. A Brewer and E. Amir; "Reducing www latency and bandwidth requirements by real-time distillation," *Proceeding of 5th Int. www Conf.,* Paris, France, 1996.
7. R. Han, P. Bhagwat, R. Lamaire, T. Mummert, V. Perret and J. Rubs, "Dynamic Adaptation in a Image Transcoding Proxy For Mobile Web Browsing," *IEEE Personal Communications*, December 1998. PP 8-17.
8. S. Khan and M. K. Mandal, "Distributed Adaptation Technique for Mobile Web Browsing," *Proc. of SPIE: Internet Multimedia Management System III*, Vol. 4862, pp. 231-230, Boston, July 29-Aug 2, 2002.

9. A. Lera, A. Molinaro, and S. Marano, "Wireless broadband applications: the teleservice model and Adaptive QOS provisioning," *IEEE Communication Magazine,* Vol. 7, No.10, pp. 71-75, Oct 1999.

10. M. Kojo, K. Raatikainen, and T. Alanko, "Connecting Mobile Workstations to the Internet over a Digital Cellular Telephone Network," *Proc. of the Mobidata Workshop on Mobile and Wireless Information Systems,* Rutgers University, NJ, Nov 1994.

11. J. Wang, "A survey of webcaching schemes for the internet," *ACM computer communication review,* Vol 29(5), October 1999.

12. G. Abdulla, E. A. Fox, M. Abrams, and S. Williams, "Caching Proxies: limitations and potentials," *Proceedings of the 4th International WWW conference,* Boston, USA, December 1995.

13. C. Aggarwal, J. L. Wolf, and P.S. Yu, "Caching on the World Wide Web," *IEEE Transactions on Knowledge and data Engineering,* Vol.11, No. 1, January 1999.

14. P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, "Changes in Web client access patterns: characteristics and caching implications, " *World Wide Web (Special issue on Characterization and Performance Evaluation)*, 1999.

15. S. Bhattacharjee, K. Calvert, and E. W. Zegura, "self organizing wide-area network caches," *IEEE Infocom '98,* April 1998.

16. Caching for Improved Content Delivery by Intel corp.

17. D. L. Willick, D. L. Eager, and R. B. Bunt, "Disk cache replacement policies for network file servers," *Proceedings of the 13th Intl. Conf. on Distributed Computing System*, pp.2-11, May 1993.

18. S. Williams, M. Abrams, C. R. Standridge, G. Addulla, and E. A. Fox, "Removal policies in network caches for World Wide Web documents," *Proceeding of SIGCOMM*, pp. 293-305, Stanford, CA, Aug 1996.

19. F. Yu, Q. Zhang, W. Zhu and Y. Zhang "Network-Adaptive Cache Management schemes for Mixed Media," *The second IEEE Pacific-Rim Conference on Multimedia (IEEE-PCM) 2001*, Beijing, China, Oct 2001.

20. Z. Xiang, Q. Zhang, W. Zhu and Y. Zhang "Cost-Based Replacement Policy for Multimedia Proxy across Wireless Internet," *IEEE Globecom'01*, San Antonio, USA, Nov 2001.

21. N. Markatchev and C. Williamson, "WebTraff: A GUI for Web proxy cache Workload Modeling and Analysis," *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Fort Worth, TX, pp. 356-363, October 2002.

22. M. Busari and C. Williamson, "ProWGen: A synthetic workload Generation tool for simulation evaluation of web proxy caches," *Computer Networks*, Vol. 38, No. 6, pp. 779-794, June 2002.

23. X. Yang and G. D Veciana, "Zipf law and Effectiveness of Hierarchical Caching," *Proceedings of Communication Networks and Distributed Systems 2002 (CNDS02),* San Antonio, USA, Jan 2002.

24. L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shanker, "Web caching and Zipf like distribution; Evidence and Implications", *Proceedings of IEEE INFOCOMM conference,* New York, NY, pp. 126-134, March 1999

25. M. Busari and C. Williamson, "On the sensitivity of web proxy cache performance to workload characteristics," *Proceedings of IEEE INFOCOMM,* Anchonage, Al, pp. 1225-1234, April 2001.

26. C. Murta and V. Almedia, "Using performance Maps to understand the behaviour of web caching policies," *The second IEEE workshop on Internet Applications(WIAPP01),*Sanjose, USA, July 2001.

27. S. Gadde, J. Chase, and M. Rabinovch, "Web caching and content distibution: Aview from the interior," *Proceedings of 5th International web caching and content delivery workshop*, 2000

28. R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, " Web proxy caching; the devil is in the details," *Workshop on Internet server performance,* Madison, USA, June 1998.

29. R. Tewari, H. Vin, A. Dan, and D. Sitaram, "Resource based caching for web servers," *Proceedings of SPIE/ACM conference on Multimedia Computing,* January 1998.

30. H. Kobayoshi and S. Z Yu, "Performance models of web caching and prefetching for wireless Internet Access," *International conference on performance evaluation: Theory, Techniques and Applications (PERETTA 2000),* Fukushima, Japan, September 2000.

31. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the implications of Zipf's law for web caching," *Technical Report,* 1998

32. M. Arlitt and C. Williamson, "Internet webservers: workload characterization and performance implications," *IEEE/ACM transaction on Networking,* Vol 5, pp 631-645, October 1997.

33. F. Yu, Q. Zhang, W. Zhu, and Y. Q. Zhang, "QoS-adaptive proxy caching for multimedia streaming over the Internet," *Proceedings of First IEEE Pacic-Rim Conference on Multimedia*, Sydney, Australia," Dec. 13-15, 2000

34. V. Korolev and A. Joshi, "An End-End Approach to Wireless Web Access," *International Workshop on Wireless Networks and Mobile Computing*, Scottdale, Arizona, USA, April 2001

35. R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, pages 3-10, Port Jeerson, NY, June 2001. ACM Press.

36. A. Hafid, G. Bochmann, and R. Dssouli, " Distributed Multimedia Applications and Quality of service," *Electronic journal on networks and distributed processing,* 1999.

37. L. Villard, C. Roisin and N. layada, "A XML based multimedia document processing model for content adaptation," *proceedings of digital documents and electronic publishing (DDEP00)),* 2000.

38. A. Kitamoto, " Multiresolution cache management for distributed satellite image database using NACSIS-Thai international link'" *proceedings of the 6thIinternational workshop on academic information networks and systems (WAINS),*pp. 243-250, 2000.

39. I. Ari, A. Amer, E. L. Miller, S. Brandt, D. Long, "Who is more adaptive? ACME: adaptive caching using multiple experts," *Workshop on distributed data and structure (WDAS 2002),* Paris, France, March 2002

40. www.linux.com (Information about traffic shapers)

41. www.java.sun.com

42. www.w3c.org

43. http://www.kakadusoftware.com

44. S. Bandaru and M. K Mandal, "Content Adaptation architecture with usage of cached data in a Multimedia proxy," presented at *SPIE: Internet Multimedia Management Systems*, Orlando, USA, Sep 2003.

45. S. Bandaru and M. K Mandal, "Cache Replacement Policy for UMA Architectures," submitted to *Transaction of IEEE Multimedia.*

46. M. Nilson, J. Hjelm, and H. Ohto, "Composite Capabilities/Preference profiles: Requirements and Architecture," *W3C Working Draft*, March 2003.

APPENDIX A

In Chapter 4 the simulation set up used for evaluating proposed content adaptation architecture is described. The internal details of the communication in the simulation setup are not explained in detailed. In this appendix a detailed description about the communication of client, proxy and server is explained. Fig A.1 shows the handshake model of the client, proxy and server programs used in the simulation. The client and proxy uses a UDP based connection to communicate. This gives the ability to uses user defined packet size. The proxy and server programs communicate using TCP based Connection. TCP is a persistent connection and more reliable then UDP. The proxy program waits for the request from the client program and once the request is received it checks for the requested file in the cache. If the requested exists in the cache then the proxy sends the UDP packets to the client. If the requested file does not exist in the cache then the proxy uses a TCP based connection to connect to the server. The server transmits the requested file to the proxy. The proxy then sends the requested file to the client using the UDP based connection.

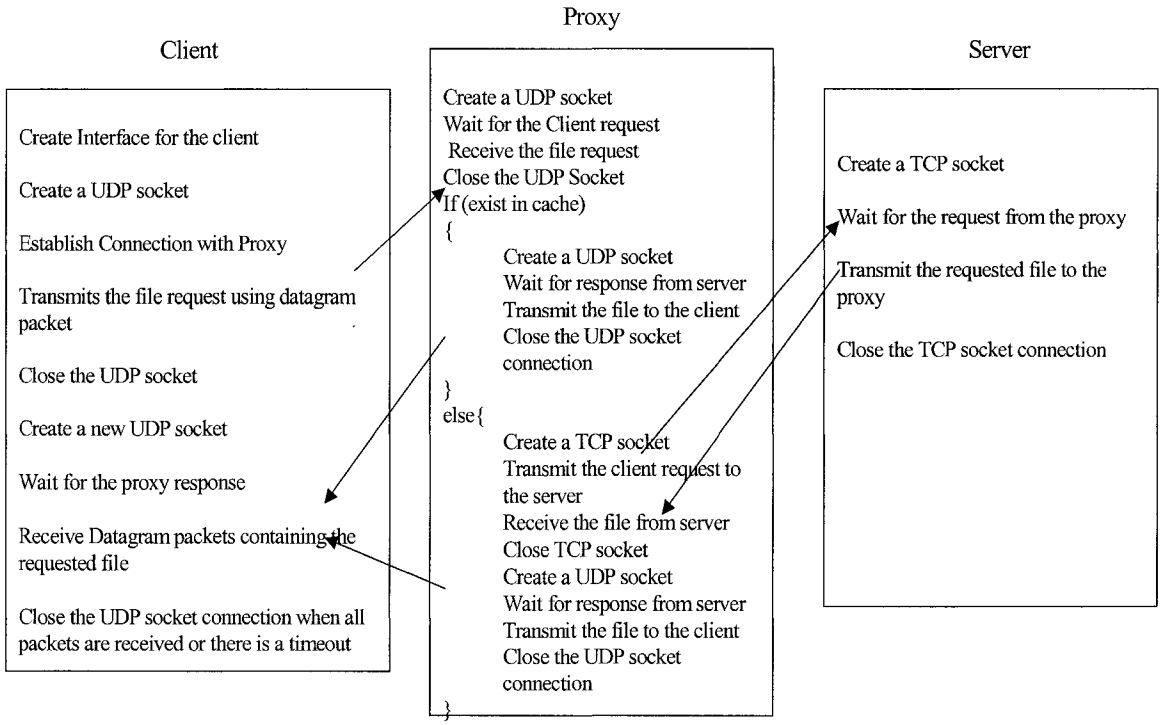


Figure A.1 Handshake Model of 3 –tier architecture implemented using java sockets

Appendix B

In Chapter 4 the description about the client interface, proxy and server was not in detailed. The internal flow of these programs is not explained in detailed. This appendix explains in detail about the functionality and program flow of the client, the proxy and the server programs developed.

B.1 Client Program

This appendix gives a detailed description about the client program. This program generates the client for connecting to the proxy. This program also generates the graphical user interface. The program connects to the proxy using UDP connection. This program runs at port number 3000. The packets of the requested file are received at this port number. The interface has a dropdown menu bars which can allow the client to choose different servers it can connect, different image formats it can request. A timer program calculates the total time taken for serving the request to the client and is displayed on to the interface.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ImageClient2 extends JFrame implements
ActionListener
{
      JComboBox JCB1,JCB2,JCB3,JCB4;
      Container C;

// creates the graphical interface components
```

```
public ImageClient2()
{
        Vector v1=new Vector();
        v1.add("None");
        v1.add("mpeg.ee.ualberta.ca");
        v1.add("mccl.ee.ualberta.ca");
        v1.add("129.128.197.176");
        Vector v2=new Vector();
        v2.add("None");
        v2.add("lenna128");
        v2.add("lenna256");
        v2.add("lenna512");
        Vector v3=new Vector();
        v3.add("None");
        v3.add("GIF");
        v3.add("JPEG");
        v3.add("JPEG2000");
        Vector v4=new Vector();
        v4.add("None");
        v4.add("Low");
        v4.add("Medium");
        v4.add("High");
        JLabel title=new JLabel("Testbed for Distributed
                          Adaption of Multimedia Content");
        JLabel connection=new JLabel("Connect To :");
        JLabel FileName=new JLabel("Requested File :");
        JLabel FileType=new JLabel("File Type :");
        JLabel Quality=new JLabel("Quality :");
        JCB1=new JComboBox(v1);
        JCB2=new JComboBox(v2);
        JCB3=new JComboBox(v3);
        JCB4=new JComboBox(v4);
        JButton JB=new JButton("Establish Connection");
        C=getContentPane();
        C.setLayout(null);
        C.add(connection);
        C.add(FileName);
        C.add(FileType);
        C.add(Quality);
        C.add(JB);
        C.add(JCB1);
        C.add(JCB2);
        C.add(JCB3);
        C.add(JCB4);
        C.add(title);
        setTitle("Testbed For LowBandwidth Environments");
        connection.setBounds(30,40,100,25);
        FileName.setBounds(30,80,100,25);
        FileType.setBounds(30,120,100,25);
        Quality.setBounds(30,160,100,25);
        JB.setBounds(60,200,200,25);
        JCB1.setBounds(190,40,140,25);
        JCB2.setBounds(190,80,140,25);
```

```
        JCB3.setBounds(190,120,140,25);
        JCB4.setBounds(190,160,140,25);
        title.setBounds(45,10,350,20);
        setSize(500,600);
        setVisible(true);
        setResizable(false);
        JB.addActionListener(this);
    }
```

This method fires the events generated by button. This method sends the request to the proxy and then waits for the response from the proxy.

```
    public void actionPerformed(ActionEvent ae)
    {
        try{
        Timer time=new Timer();
        String ad=JCB1.getSelectedItem().toString();
        String FileName=JCB2.getSelectedItem().toString();
        String FileType=JCB3.getSelectedItem().toString();
        if(FileType.equals("GIF"))
            FileName=FileName+".gif";
        if(FileType.equals("JPEG"))
            FileName=FileName+".jpg";
        if(FileType.equals("JPEG2000"))
            FileName=FileName+".jp2";
        String Quality=JCB4.getSelectedItem().toString();
        String transmit=FileName+"+"+Quality;
        time.start();
        DatagramSocket socket=new DatagramSocket();
        byte[] buf=new byte[256];
        buf=transmit.getBytes();
        InetAddress add=InetAddress.getByName(ad);
        DatagramPacket packet=new
        DatagramPacket(buf,buf.length,add,5000);
        socket.send(packet);
        socket.close();
        DatagramSocket socket1=new DatagramSocket(3000);
        socket1.setReceiveBufferSize(25000);
        byte[] buffer1=new byte[64000];
        byte[] buffer2=new byte[16000];
        DatagramPacket packet1=new
        DatagramPacket(buffer1,0,buffer1.length);
        socket1.receive(packet1);
        System.out.println("Recieved packet");
        socket1.close();
        time.stop();

        String tim=null;
        if (FileType.equals("JPEG2000"))
        {
        FileOutputStream fis=new FileOutputStream(new
                            File(FileName));
```

```
                fis.write(packet1.getData());
                fis.close();
                JLabel Tim=new JLabel("Transmission "+tim);
                JLabel size=new JLabel("File is Wrote to the
                            HardDisk");
                C.add(Tim);
                C.add(size);
                size.setBounds(30,460,200,25);
                Tim.setBounds(30,500,250,25);
                try{
                      Process
                      P=Runtime.getRuntime().exec("/usr/java/jdk1.3.1
                      _03/bin/java JJ2KDecoder -i "+FileName);
                      DataInputStream dis=new
                      DataInputStream(p.getInputStream());
                String c;
                while((c=dis.readLine())!=null)
                      {
                      System.out.println(c);
                      }
                }catch (Exception e)
                {
                      System.out.println(e);
                }

                }
                else{
                ImageIcon icon=new ImageIcon(packet1.getData());
                String width=Integer.toString(icon.getIconWidth());
                String height=Integer.toString(icon.getIconHeight());
                String display="Image Size :"+width+"X"+height;
                JLabel Tim=new JLabel("Transmission "+tim);
                JLabel size=new JLabel(display);
                can canv=new can(icon);
                C.add(canv);
                C.add(Tim);
                C.add(size);
                canv.setBounds(30,250,200,200);
                size.setBounds(30,460,200,25);
                Tim.setBounds(30,500,250,25);
                }
                }catch(Exception pe)
                {
                      System.out.println(pe);
                }
        }
```

This method fires the events generated by action on the Dropdown menu box. Changes options according to the image format selected.

```
        public void itemStateChanged(ItemEvent it)
        {
                if(it.getItem().toString().trim()=="JPEG2000")
                {
```

```
                    System.out.println("Item is cahnged");
                    JCB4.setVisible(false);
                    Vector v=new Vector();
                    v.add("1");
                    v.add("2");
                    v.add("3");
                    v.add("4");
                    v.add("5");
                    JCB4=new JComboBox(v);
                    JCB4.setBounds(190,160,140,25);
                    C.add(JCB4);

                }else
                {
                Vector v4=new Vector();
                JCB4.setVisible(false);
                v4.add("None");
                v4.add("Low");
                v4.add("Medium");
                v4.add("High");
                JCB4=new JComboBox(v4);
                JCB4.setBounds(190,160,140,25);
                C.add(JCB4);
                }
        }
        public static void main(String[] args)
        {
                    new ImageClient2();
        }
}
```

B.2 Proxy Program

This program creates a proxy server that can serve the clients request based on the resources available. The program uses a UDP connection to get receive the request from the client. This program runs at port number 5000. This program connects to the server using a TCP based connection. This program also checks whether the requested file exist in the cache or not. The proxy program also implements the decision making scheme which is discussed in chapter 3.

```
import java.net.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.image.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.stream.ImageOutputStream;

public class imageproxy
{
      String Hostadd=null;
      String quality=null,name=null;
      boolean cache=true;

This constructor creates the UDP sockets and receives the request
from the client.

public imageproxy()
      {
      try{
            DatagramSocket socket=new DatagramSocket(5000);

            byte[] buf=new byte[256];
            DatagramPacket packet=new
```

```
                    DatagramPacket(buf,buf.length);
                    socket.receive(packet);
                    System.out.println("Recieved packet");
                    String p=new String(packet.getData(),0);
                    StringTokenizer st=new StringTokenizer(p,"+");
                    name=st.nextToken();
                    quality=st.nextToken().trim();
                    System.out.println("Image :"+name+"
                    Quality:"+quality);
                    ImageIcon h=(ImageIcon)connectserver(p);
                    InetAddress add=packet.getAddress();
                    int port=packet.getPort();
                    Hostadd=add.getHostAddress();
                    System.out.println("+++++++++"+Hostadd);
                    String m="bye Buddy";
                    buf=m.getBytes();
                    packet=new DatagramPacket(
                    buf,buf.length,add,port);
                    socket.send(packet);
                    System.out.println("data send");
                    }catch(Exception e)
                    {
                          System.out.println(e);
                    }
      }
```

This function connects to the server if the object is not found in the cache.

```
      public Object connectserver(String p)
      {
       Object m=null;
       Timer time=new Timer();
      try{
            if (cache==true)
            {
                  File f=new File(name);
                  int len=(int)f.length();
                  byte[] buf=new byte[len];
                  System.out.println("reading data of
                  length"+len);
                  time.start();
                  DataInputStream dis1= new DataInputStream(new
                  FileInputStream(f));
                  System.out.println("reading data of
                  length"+len);
                  dis1.read(buf,0,(int)len);
                  System.out.println("finished reading
                  data"+buf.length);
                  time.stop();
                  time.print("Reading Time");
                  int bits=Integer.parseInt(quality);
                  System.out.println("Quality"+bits);
                  switch(bits)
```

```
        {
                case 1: len=len/4;
                        break;
                case 2: len=len/2;
                        break;
                case 3: len=(2*len)/3;
                        break;
                case 4: len=(3*len)/4;
                        break;
                case 5: len=len;
                        break;
        }

System.out.println("Data size to be transmitted
"+len);
 DatagramSocket socket=new DatagramSocket();
 socket.setSendBufferSize(250000);
 InetAddress add=InetAddress.getByName
("mpeg.ee.ualberta.ca");
 DatagramPacket packet1=new
 DatagramPacket(buf,0,len,add,3000);
 socket.send(packet1);
 socket.close();
        }
        else{
        Socket soc=new Socket("localhost",2000);
        PrintStream dos=new
        PrintStream(soc.getOutputStream());
        time.start();
        DataInputStream dis1=new
        DataInputStream(soc.getInputStream());
        dos.println(p);
        int len=dis1.readInt();
        byte[] buf=new byte[len];
        dis1.read(buf,0,len);
        System.out.println("Recieved Data from server
        :"+len);
        System.out.println("Recieved Data from server
        :"+buf.length);
        time.stop();
        time.print("End Time:");
        soc.close();

 boolean adapt=adaptationManager(len);
 StringTokenizer ST1=new StringTokenizer(name,".");
 String name1=ST1.nextToken();
 String ext=ST1.nextToken();
 if (ext.equals("jp2"))
 {
 if(adapt==true)
        {
                if(quality.trim().equals("Low"))
        {
                len=len/5;
```

```
                    }

                    if(quality.trim().equals("Medium"))
                    {
                          len=len/2;
                    }

                    if(quality.trim().equals("High"))
                    {
                          len=len;
                    }

                    }
                    }
              int templen=len/2;
              byte[] temp1=new byte[templen];
              byte[] temp2=new byte[templen];
              for (int i=0;i<templen;i++)
              {
                    temp1[i]=buf[i];
                    temp2[i]=buf[templen+i];
        }
              System.out.println("Loaded Successfully");
              System.out.println("Loaded Successfully"+len);
              DatagramSocket socket=new DatagramSocket();
              socket.setSendBufferSize(250000);
              InetAddress
              add=InetAddress.getByName("mpeg.ee.ualberta.ca");
              DatagramPacket packet1=new
              DatagramPacket(buf,0,len,add,3000);
              socket.close();
              }
              }catch(Exception e)
              {
                   System.out.println(e);
              }
              return m;
      }

public static void write(ImageWriter writer, ImageWriteParam
imageWriteParam, IIOImage iioImage, String filename, float
compressionQuality) throws IOException
     {
ImageOutputStream out = ImageIO.createImageOutputStream(new
File(filename));
imageWriteParam.setCompressionQuality(compressionQuality);
writer.setOutput(out);
writer.write((IIOMetadata) null, iioImage, imageWriteParam);
out.flush();
out.close();
      }

This function decides whether to adapt the data or not.
public boolean adaptationManager(int length)
```

```
	{
		int Transdata=0;
		int bandpc=9;
		int bandinternal=1500;
		int bandsp=2300;
		float
Timeprocess=0,TimeUnprocess=0,Ncache=0,Wcache=0;
		float TimeAdapt=0,TimeUnadapt=0;
		boolean adapt=false;
			if(quality.equals("Low"))
			{
				Transdata=length/2;
			}

			if(quality.equals("Medium"))
			{
				Transdata=length*3/4;
			}

		if(quality.equals("High"))
			{
				Transdata=length;
			}
		Timeprocess=Transdata/bandpc;
		TimeUnprocess=length/bandpc;
		Ncache=Transdata/bandinternal;
		Wcache=length/bandinternal;

		TimeAdapt=Timeprocess+Ncache;
		TimeUnadapt=TimeUnprocess+Wcache;
		if(TimeAdapt<TimeUnadapt)
		adapt=true;

		System.out.println("Data Should be:"+adapt);


		return adapt;

	}

public static void main(String args[]) throws IOException
	{
		new imageproxy();
	}

}
```

B.3 Server Program

This program generates the server. The server runs at port number 2000. The server communicates using TCP based connection. The server has a images stored in multiple resolutions. The server decides which resolution should be sent to the client.

```
import java.net.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class imageserver
{
      Socket s;

This constructor creates the TCP socket for serving the request
to the clients.

      public imageserver()
      {
            String p;
            Timer time=new Timer();
            try{ServerSocket ss=new ServerSocket(2000);
            s=ss.accept();
            DataInputStream dis=new
DataInputStream(s.getInputStream());
            DataOutputStream ps1=new
            DataOutputStream(s.getOutputStream());
            String l=dis.readLine();
            System.out.println(l);
            StringTokenizer st=new StringTokenizer(l,"+");
            String name=st.nextToken();
            String quality=st.nextToken();
            System.out.println("Image :"+name+"
            Quality:"+quality);
                  if((quality.trim()).equals("Low"))
                  {name="low"+name;
                  System.out.println("Image quality is Low");
                  }
                  if((quality.trim()).equals("Medium"))
                  name="medium"+name;
                  if((quality.trim()).equals("High"))
                  name=name;

            System.out.println("File to Server:"+name);
            File f=new File(name);
            int len=(int)f.length();
```

```
        byte[] buf=new byte[len];
        //byte[] buf=new byte[(int)len];
        System.out.println("reading data og length"+len);
        time.start();
        DataInputStream dis1= new DataInputStream(new
        FileInputStream(f));
        System.out.println("reading data og length"+len);
        dis1.read(buf,0,(int)len);
        System.out.println("finished reading
        data"+buf.length);
        time.stop();
        time.print("Reading Time");
        ps1.writeInt((int)len);
        ps1.write(buf);
        System.out.println("Data From Proxy :"+l);
        ps1.flush();
        ps1.close();
        dis1.close();
        s.close();
        }catch(IOException e)
        {
         e.printStackTrace();
        }
}

public static void main(String[] args) throws IOException
{
        new imageserver();
}
```

APPENDIX C

C.1 Trace Driven Simulator

In Chapter 4 the cache replacement policies were evaluated using a trace driven simulator. The description about the design patterns is not explained in chapter 4. This appendix gives a detailed description about the design patterns used and program flow of the simulator. Fig A.2 shows the design pattern used by the trace driven simulator. Façade is an object oriented design pattern. In this pattern main class controls the sub classes. The trace driven simulator used in chapter 4 for evaluating the proposed cache replacement policy uses this pattern to integrate different components. The different components are nothing but the different policies that are used in performance evaluation.

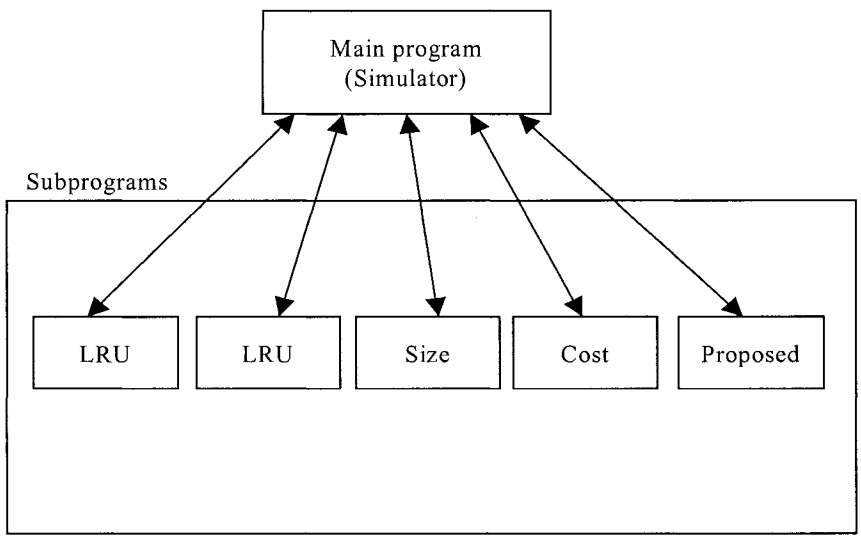


Figure A.2 Façade design pattern used by the trace driven simulator

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class TraceDrivenSim implements ActionListener
{
      JFrame jf;
      JTextField tf1,tf2;
      JLabel result=new JLabel();
      JLabel hit=new JLabel();
      JLabel miss=new JLabel();
      JComboBox jcb;
      JButton jb1,jb2;

Constructer creates the graphical interface components for the
trace driven simulator.

      public TraceDrivenSim() {
            jf= new JFrame("Trace Driven Simulator");
            tf1=new JTextField(30);
            tf2=new JTextField(30);
            jb1=new JButton("Get Hit Ratio");
            jb2=new JButton("Get Byte Hit Ratio");
            JLabel title=new JLabel("Trace Driven Simulation");
            JLabel TLabel=new JLabel("Name of Trace");
            JLabel CLabel=new JLabel("Cache Size");
            JLabel PLabel=new JLabel("Cache Replacement Policy");
            JLabel HLabel=new JLabel("No of Hits/Byte_hit");
            JLabel MLabel=new JLabel("No of Miss/Byte_Miss");
            JLabel RLabel=new JLabel("Percentage %");
            JPanel jp2=new JPanel();
            Vector v = new Vector();
            v.add("LRU");
            v.add("LFU");
            v.add("Size");
            v.add("Proposed");
            jcb=new JComboBox(v);
            title.setFont(new Font("arial",1,16));
            title.setForeground(Color.blue);
            TLabel.setForeground(Color.red);
            CLabel.setForeground(Color.red);
            PLabel.setForeground(Color.red);
            HLabel.setForeground(Color.red);
            MLabel.setForeground(Color.red);
            RLabel.setForeground(Color.red);
            jb1.setForeground(Color.red);
            jb2.setForeground(Color.red);
            jcb.setForeground(Color.red);

            jp2.add(title);
            jp2.setLayout(null);
            jp2.add(TLabel);
            jp2.add(tf1);
            jp2.add(CLabel);
```

```
            jp2.add(tf2);
            jp2.add(PLabel);
            jp2.add(jcb);
            jp2.add(jb1);
            jp2.add(jb2);
            jp2.add(HLabel);
            jp2.add(MLabel);
            jp2.add(RLabel);
            jp2.add(hit);
            jp2.add(miss);
            jp2.add(result);

            title.setBounds(70,20,200,25);
            TLabel.setBounds(20,60,100,25);
            tf1.setBounds(210,60,100,25);
            CLabel.setBounds(20,100,100,25);
            tf2.setBounds(210,100,100,25);
            PLabel.setBounds(20,140,180,25);
            jcb.setBounds(210,140,100,25);
            jb1.setBounds(20,180,140,25);
            jb2.setBounds(170,180,140,25);
            HLabel.setBounds(20,220,180,25);
            hit.setBounds(210,220,100,25);
            MLabel.setBounds(20,250,180,25);
            miss.setBounds(210,250,180,25);
            RLabel.setBounds(20,280,180,25);
            result.setBounds(210,280,100,25);
            jf.getContentPane().add(jp2);
            jf.setSize(400,400);
            jf.setVisible(true);
            jf.setResizable(false);
            jb1.addActionListener(this);
            jb2.addActionListener(this);

      }

//This function fires the events generated by the buttons and
also selects the appropriate policy for testing.

      public void actionPerformed(ActionEvent ae)
      {
            String file=tf1.getText();
            String cachesize=tf2.getText();
            String policy=jcb.getSelectedItem().toString();
            if(ae.getSource()==jb1)
            {
                  if (policy.equals("LRU"))
                  {
                  LRU l=new LRU(file,cachesize);
                  String percent=Float.toString(l.sunnyPrint());
                  String Nhit=Integer.toString(l.sunnyHit());
                  String Nmiss=Integer.toString(l.sunnyMiss());
                  hit.setText(Nhit);
                  miss.setText(Nmiss);
```

```
result.setText(percent);
}
if (policy.equals("LFU"))
{
LFU l=new LFU(file,cachesize);
String percent=Float.toString(l.sunnyPrint());
result.setText(percent);
String Nhit=Integer.toString(l.sunnyHit());
String Nmiss=Integer.toString(l.sunnyMiss());
hit.setText(Nhit);
miss.setText(Nmiss);
result.setText(percent);
}
if (policy.equals("Size"))
{
SizeBased l=new SizeBased(file,cachesize);
String percent=Float.toString(l.sunnyPrint());
String Nhit=Integer.toString(l.sunnyHit());
String Nmiss=Integer.toString(l.sunnyMiss());
hit.setText(Nhit);
miss.setText(Nmiss);
result.setText(percent);
result.setText(percent);

}
if (policy.equals("Proposed"))
{
proposedreplacement l=new
proposedreplacement(file,cachesize);
String percent=Float.toString(l.sunnyPrint());
String Nhit=Integer.toString(l.sunnyHit());
String Nmiss=Integer.toString(l.sunnyMiss());
      hit.setText(Nhit);
      miss.setText(Nmiss);
      result.setText(percent);
      result.setText(percent);
}
}

if(ae.getSource()==jb2)
{
      if (policy.equals("LRU"))
      {
      BH_LRU l=new BH_LRU(file,cachesize);
      String percent=Double.toString(l.sunnyPrint());
      String Nhit=Double.toString(l.sunnyHit());
      String Nmiss=Double.toString(l.sunnyMiss());
            hit.setText(Nhit);
            miss.setText(Nmiss);
            result.setText(percent);
      }
      if (policy.equals("LFU"))
      {
      BH_LFU l=new BH_LFU(file,cachesize);
```

```
                        String percent=Double.toString(l.sunnyPrint());
                        result.setText(percent);
                        String Nhit=Double.toString(l.sunnyHit());
                        String Nmiss=Double.toString(l.sunnyMiss());
                        hit.setText(Nhit);
                        miss.setText(Nmiss);
                        result.setText(percent);
                        }
                        if (policy.equals("Size"))
                        {
                        BH_Size l=new BH_Size(file,cachesize);
                        String percent=Double.toString(l.sunnyPrint());
                        String Nhit=Double.toString(l.sunnyHit());
                        String Nmiss=Double.toString(l.sunnyMiss());
                        hit.setText(Nhit);
                        miss.setText(Nmiss);
                        result.setText(percent);
                        result.setText(percent);

                        }
                        if (policy.equals("Proposed"))
                        {
                        BH_proposedreplacement l=new
                        BH_proposedreplacement(file,cachesize);
                        String percent=Double.toString(l.sunnyPrint());
                        String Nhit=Double.toString(l.sunnyHit());
                        String Nmiss=Double.toString(l.sunnyMiss());
                        hit.setText(Nhit);
                        miss.setText(Nmiss);
                        result.setText(percent);
                        result.setText(percent);
                        }
                }

        }

        public static void main(String[] args)
        {
                new TraceDrivenSim();

        }
}
```

C.1.1 LRU Policy

This program implements the least recently used algorithm. This program takes the trace name and cache size as input arguments.

```
import java.io.*;
import java.util.*;

public class LRU
{
      static String TraceFile;
      static int cachesize=0;
      Hashtable Dataset_Time=new Hashtable();
      Hashtable Dataset_Size=new Hashtable();
      int Counter=0;
      int hit=0,miss=0;
      float percent=0;
      public LRU(String file,String csize)
      {
            String Data=null;
            String ID,Time_stamp,SIZE;
            TraceFile=file;
            cachesize=Integer.parseInt(csize);
            File trace=new File(TraceFile);
            int tempcache=0;
            try{
            DataInputStream dis=new DataInputStream(new
            FileInputStream(trace));
            while((Data=dis.readLine())!= null)
            {
            int data_size=0;
            StringTokenizer st=new StringTokenizer(Data," ");
            Time_stamp=st.nextToken();
            ID=st.nextToken();
            SIZE=st.nextToken();
            data_size=Integer.parseInt(SIZE);
            tempcache=tempcache+data_size;
            if(tempcache<cachesize)
            {
                  Dataset_Time.put(ID,Time_stamp);
                  Dataset_Size.put(ID,SIZE);
                  Counter++;
            }else
            {
            break;
            }
            }

            runTrace();
            percent=((float)hit/(hit+miss))*100;
```

```
            System.out.println("No of Hits:"+hit);
            System.out.println("No of Miss:"+miss);
            System.out.println("Hit Percentage"+percent);

            }catch(Exception e)
            {
                  System.out.println(e);
            }
      }
```

This function runs the traces, implements the least recently used technique.

```
      public void runTrace()
      {
            try{
            DataInputStream DIS=new DataInputStream(new
            FileInputStream(new File(TraceFile)));
            String Data=null;
            Enumeration e=Dataset_Time.keys();
            for(int i=0;i<Dataset_Time.size();i++)
                  {
                        String skip=DIS.readLine();
                  }

            while((Data=DIS.readLine())!= null)
            {
            StringTokenizer st=new StringTokenizer(Data," ");
            String Time_stamp=st.nextToken();
            String ID=st.nextToken();
            String SIZE=st.nextToken();

                  if(Dataset_Time.containsKey(ID))
                  {
                        hit++;
                        Dataset_Time.put(ID,Time_stamp);
                  }
                  else
                  {
                        miss++;
      if(Dataset_Time.size()==Counter)
      {
      while(e.hasMoreElements())
      {
      Object Element_Time=e.nextElement();
      float curr_time=Float.parseFloat(Time_stamp);
      if(curr_time >
      Float.parseFloat(Dataset_Time.get(Element_Time).toString())
      )
            {
            Dataset_Time.remove(Element_Time);
            Dataset_Size.remove(Element_Time);
            Dataset_Time.put(ID,Time_stamp);
            Dataset_Size.put(ID,SIZE);
```

```
                }
                }
                }else
                      {
                            Dataset_Time.put(ID,Time_stamp);
                            Dataset_Size.put(ID,SIZE);
                      }
                }
                }
                }catch(Exception e)
                {
                      System.out.println(e);
                }
        }
This function returns the Hit ratio value
        public float sunnyPrint()
        {
              return percent;
        }
This function returns the number of hits by the this policy
        public int sunnyHit()
        {
              return hit;
        }
This function returns the number of misses by the this policy
        public int sunnyMiss()
        {
              return miss;
        }
```

C.1.2 LFU Policy

This program implements the least frequently used algorithm. This program takes the trace name and cache size as input arguments.

```
import java.io.*;
import java.util.*;

public class LFU
{
      String TraceFile;
      Hashtable Dataset_Time=new Hashtable();
      Hashtable Dataset_Frequency=new Hashtable();
      Hashtable Dataset_Aging=new Hashtable();
      Hashtable Dataset_Size=new Hashtable();
      int Counter=0;
      int hit=0,miss=0;
      float percent=0;
      public LFU(String file,String csize)
      {
            String Data=null;
            String ID,Time_stamp,SIZE;
            TraceFile=file;
            File trace=new File(TraceFile);
            int cachesize=Integer.parseInt(csize);
            int tempcache=0;
            try{
            DataInputStream dis=new DataInputStream(new
            FileInputStream(trace));
            while((Data=dis.readLine())!= null)
            {
            int data_size=0;
            StringTokenizer st=new StringTokenizer(Data," ");
            Time_stamp=st.nextToken();
            ID=st.nextToken();
            SIZE=st.nextToken();
            data_size=Integer.parseInt(SIZE);
            tempcache=tempcache+data_size;
            if(tempcache<cachesize)
            {
                  Dataset_Time.put(ID,Time_stamp);
                  Dataset_Size.put(ID,SIZE);
                  Counter++;
            }else
            {
                  break;
            }
            }

            intialFrequency();
```

```
        runTrace();
        percent=((float)hit/(hit+miss))*100;
        System.out.println("No of Hits:"+hit);
        System.out.println("No of Miss:"+miss);
        System.out.println("Hit Percentage"+percent);
        System.out.println("No of Objects:"+Counter);
        }catch(Exception e)
        {
              System.out.println(e);
        }
}

public void intialFrequency()
{
        Enumeration e=Dataset_Time.keys();
        String Freq="1";
        while(e.hasMoreElements())
        {
              Dataset_Frequency.put(e.nextElement(),Freq);
        }

}

public float sunnyPrint()
{
        return percent;
}
public int sunnyHit()
{
        return hit;
}
public int sunnyMiss()
{
        return miss;
}
public float ageCalculation(int Frequency, float time)
{
        float age= (float)Frequency+time;
              return age;
}

public void runTrace()
{
        try{
DataInputStream DIS=new DataInputStream(new
FileInputStream(new File(TraceFile)));
String Data=null;
Enumeration e=Dataset_Frequency.keys();
for(int i=0;i<Dataset_Frequency.size();i++)
              {
                    String skip=DIS.readLine();
              }

        while((Data=DIS.readLine())!= null)
```

```
        {
        StringTokenizer st=new StringTokenizer(Data," ");
        String Time_stamp=st.nextToken();
        String ID=st.nextToken();
        String SIZE=st.nextToken();
        if(Dataset_Frequency.containsKey(ID))
        {
        hit++;
        int freq=Integer.parseInt(
                Dataset_Frequency.get(ID).toString());
        freq++;

        Dataset_Frequency.put(ID,Integer.toString(freq));
              }
              else
              {
                    miss++;
        if(Dataset_Frequency.size()==Counter)
        {
        Object key1=findLowestFreqObject(Dataset_Frequency);
              if(key1!=null)
              {
                    Dataset_Frequency.remove(key1);
                    Dataset_Time.remove(key1);
                    Dataset_Size.remove(key1);
                    Dataset_Frequency.put(ID,"1");
                    Dataset_Time.put(ID,Time_stamp);
                    Dataset_Size.put(ID,SIZE);
              }else
                    continue;
        }else
        {
              Dataset_Frequency.put(ID,"1");
              Dataset_Time.put(ID,Time_stamp);

        }

        }
        }
        }catch(Exception r)
        {
              System.out.println("Level 2 breach:"+r);
              r.printStackTrace();
        }
}

public Object findLowestFreqObject(Hashtable ht)
{
        Enumeration e=ht.keys();
        int minvalue=10000000;
        Object key=null;
        while(e.hasMoreElements())
        {
        Object key1=e.nextElement();
```

```
            int temp1=Integer.parseInt(ht.get(key1).toString());
            if (temp1<=minvalue)
                  {
                        minvalue=temp1;
                        key=key1;
                  }

            }
            return key;
      }
```

C.1.3 Size

This program implements the size algorithm. This program takes the trace name and cache size as input arguments.

```
import java.io.*;
import java.util.*;

public class SizeBased
{
      static String TraceFile;
      static int cachesize=0;
      Hashtable Dataset_Time=new Hashtable();
      Hashtable Dataset_Size=new Hashtable();
      int Counter=0;
      int hit=0,miss=0;
      float percent=0;
      public SizeBased(String file,String csize)
      {
            String Data=null;
            String ID,Time_stamp,SIZE;
            TraceFile=file;
            cachesize=Integer.parseInt(csize);
            File trace=new File(TraceFile);
            int tempcache=0;
            try{
            DataInputStream dis=new DataInputStream(new
            FileInputStream(trace));
            while((Data=dis.readLine())!= null)
            {
            int data_size=0;
            StringTokenizer st=new StringTokenizer(Data," ");
                  Time_stamp=st.nextToken();
                  ID=st.nextToken();
                  SIZE=st.nextToken();
                  data_size=Integer.parseInt(SIZE);
                  tempcache=tempcache+data_size;
                  if(tempcache<cachesize)
                  {
                        Dataset_Time.put(ID,Time_stamp);
                        Dataset_Size.put(ID,SIZE);
                        Counter++;
                  }else
                        {
                        break;
                        }
            }

            runTrace();
            percent=((float)hit/(hit+miss))*100;
```

```
        System.out.println("No of Hits:"+hit);
        System.out.println("No of Miss:"+miss);
        System.out.println("Hit Percentage"+percent);
        System.out.println("No of Objects:"+Counter);

        }catch(Exception e)
        {
              System.out.println(e);
        }
}

public void runTrace()
{
try{
DataInputStream DIS=new DataInputStream(new
FileInputStream(new File(TraceFile)));
String Data=null;
Enumeration e=Dataset_Time.keys();
for(int i=0;i<Dataset_Time.size();i++)
        {
              String skip=DIS.readLine();
        }

        while((Data=DIS.readLine())!= null)
        {
        StringTokenizer st=new StringTokenizer(Data," ");
        String Time_stamp=st.nextToken();
        String ID=st.nextToken();
        String SIZE=st.nextToken();
        int curr_Size=Integer.parseInt(SIZE);
        if(Dataset_Size.containsKey(ID))
        {
              hit++;
        }
        else
        {
        miss++;
        if(Dataset_Size.size()==Counter)
        {
        Object key=findLargestSize(Dataset_Size);
        Dataset_Size.remove(key);
        Dataset_Time.remove(key);
        Dataset_Size.put(ID,Integer.toString(curr_Size));
        Dataset_Time.put(ID,Time_stamp);
        }else
        {
        Dataset_Size.put(ID,Integer.toString(curr_Size));
        Dataset_Time.put(ID,Time_stamp);

        }

        }
        }
        }catch(Exception r)
```

```
                {
                        System.out.println(r);
                }
        }

        public Object findLargestSize(Hashtable ht)
        {
                Enumeration e=ht.keys();
                int largevalue=0;
                Object key=null;
                while(e.hasMoreElements())
                {
                Object key1=e.nextElement();
                int temp1=Integer.parseInt(ht.get(key1).toString());
                if (temp1>=largevalue)
                {
                        largevalue=temp1;
                        key=key1;
                }
                }

                return key;
        }
        public float sunnyPrint()
        {
                return percent;
        }
        public int sunnyHit()
        {
                return hit;
        }
        public int sunnyMiss()
        {
                return miss;
        }


}
```

C.1.4 Proposed

This program implements the proposed algorithm. This program takes the trace name and cache size as input arguments.

```
import java.util.*;
import java.io.*;

public class proposedreplacement
{
      static String TraceFile;
      static int cachesize=0;
      Hashtable Dataset_frequency =new Hashtable();
      Hashtable Dataset_weight =new Hashtable();
      Hashtable Dataset_Size=new Hashtable();
      Hashtable Dataset_Time=new Hashtable();
      int hit=0,miss=0;
      final int SP_BW=1000000;
      final int P_BW=100000;
      int Complexity=2;
      int Counter=0;
      float percent=0;
      public proposedreplacement(String file,String csize)
      {
      String Data=null;
      String ID,Time_stamp,SIZE;
      TraceFile=file;
      cachesize=Integer.parseInt(csize);
      File trace=new File(TraceFile);
      int tempcache=0;
      int No_objects=0;
      try{
      DataInputStream dis=new DataInputStream(new
      FileInputStream(trace));
      while((Data=dis.readLine())!= null)
      {
            int data_size=0;
            StringTokenizer st=new StringTokenizer(Data," ");
            Time_stamp=st.nextToken();
            ID=st.nextToken();
            SIZE=st.nextToken();
            data_size=Integer.parseInt(SIZE);
            tempcache=tempcache+data_size;
            if(tempcache<cachesize)
            {

                  Dataset_Size.put(ID,SIZE);
                  Dataset_Time.put(ID,Time_stamp);
                  Counter++;
                  }else
                  {
```

```
                    break;
                    }
              }

              No_objects=Counter;
              intialFrequency();
              intialCost();
              runTrace();
              percent=((float)hit/(hit+miss))*100;
              System.out.println("No of Hits:"+hit);
              System.out.println("No of Miss:"+miss);
              System.out.println("Hit Percentage"+percent);

              }catch(Exception e)
              {
                    System.out.println(e);
              }

        }

        public void intialFrequency()
        {
              Enumeration e=Dataset_Size.keys();
              String Freq="1";
              while(e.hasMoreElements())
              {
                    Dataset_frequency.put(e.nextElement(),Freq);
              }

        }

        public void intialCost()
        {
              Enumeration e=Dataset_Size.keys();
              while(e.hasMoreElements())
              {
              Object key=e.nextElement();
              int size=Integer.parseInt(
                        Dataset_Size.get(key).toString());
              float time=Float.parseFloat(
                           Dataset_Time.get(key).toString());
              float Cost= costCalculation(size,time);
              Dataset_weight.put(key,Float.toString(Cost));

              }
        }

        public float costCalculation(int size, float time)
        {
              float rtt=size/SP_BW;
              float Adapt_Time=Complexity*(size/P_BW);
//            float Adapt_Time=0;
              float Cost=(100*rtt+50*Adapt_Time)+time;
```

```
            return Cost;
        }

        public void runTrace()
        {
        try{
        DataInputStream DIS=new DataInputStream(new
        FileInputStream(new File(TraceFile)));
        String Data=null;
        Enumeration e=Dataset_weight.keys();
        for(int i=0;i<Dataset_Size.size();i++)
              {
              String skip=DIS.readLine();
              }
        while((Data=DIS.readLine())!= null)
              {
              StringTokenizer st=new StringTokenizer(Data," ");
              String Time_stamp=st.nextToken();
              String ID=st.nextToken();
              String SIZE=st.nextToken();
              float cost =costCalculation(
              Integer.parseInt(SIZE),Float.parseFloat(Time_stamp));
        if(Dataset_frequency.containsKey(ID))
        {
        hit++;
        int freq=Integer.parseInt(
        Dataset_frequency.get(ID).toString());
        freq++;
        Dataset_frequency.put(ID,Integer.toString(freq));
        }
        else
        {
              miss++;
              if(Dataset_weight.size()==Counter)
              {
              Enumeration ce=Dataset_weight.keys();
        while(ce.hasMoreElements())
        {
Object Curr_cost=ce.nextElement();
if(cost > Float.parseFloat
(Dataset_weight.get(Curr_cost).toString()))
{
        Dataset_weight.remove(Curr_cost);
        Dataset_frequency.remove(Curr_cost);
        Dataset_Size.remove(Curr_cost);
        Dataset_Time.remove(Curr_cost);
        Dataset_weight.put(ID,Float.toString(cost));
        Dataset_frequency.put(ID,"1");
        Dataset_Size.put(ID,SIZE);
        Dataset_Time.put(ID,Time_stamp);
}
}
}else
{
```

```
                Dataset_weight.put(ID,Float.toString(cost));
                Dataset_frequency.put(ID,"1");
                Dataset_Size.put(ID,SIZE);
                Dataset_Time.put(ID,Time_stamp);
                                }
                        }
                }
                }catch(Exception r)
                {
                        System.out.println(r);
                        }
                }
                public float sunnyPrint()
                {
                        return percent;
                }
                public int sunnyHit()
                {
                        return hit;
                }
                public int sunnyMiss()
                {
                        return miss;
                }
}
```