Optimal Display Sequence for Multi-clip Queries

by

Paul Sik Leung Shum

B. Sc., The University of British Columbia, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

December 1996

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date DECEMBER 3, 1996

DE-6 (2/88)

Abstract

With the increased significance of multimedia applications, the performance of multimedia systems used in these applications becomes more important. This thesis investigates the possibility to improve the throughput of a system serving multi-clip queries. A multi-clip query requests multiple video clips be returned as the answer of the query. In many multimedia applications, the order in which the video clips are to be delivered does not matter that much to the user. This thesis discovers that the throughput of a system can be improved significantly for queries if the display sequence can be rearranged using the "piggybacking" technique. It describes two optimization criteria: maximizing the number of piggybacked clips and maximizing the impact on buffer space, to find an optimal display sequence. This thesis shows that the display sequences can be found using the bipartite matching. Two corresponding admission control algorithms, MaxPVC and MaxIBS, are presented. It further extends the technique to video clips of variable buffer size. Finally, it presents the MaxPP algorithm, which is used to generate display

sequences for queries of variable length video clips. Experimental results show that the piggybacking technique can improve the system throughput significantly. Depending on the total number of available video clips, the throughput of the MaxPVC and MaxIBS algorithms can increase over 300%. The MaxPP algorithm can increase the throughput of the system to 30%.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr. Raymond Ng, who has been guiding me in this research. He has not only been giving me many valuable suggestions, but he also taught me how to do a research in an academic environment.

I would also like to thank my second reader, Dr. Gerald Neufeld, who has provided many useful suggestions to this thesis.

In particular, I must thank Dr. Nick Pippenger who has pointed me in the right direction in finding the appropriate graph algorithm to solve some of the problems encountered in this thesis.

In addition, I would like to thank the following people who have helped me in completing this thesis: Dr. Norm Hutchinson, for clarifying some confusion on the object oriented programming during the development of my simulation software, and Roger Tam, for his time in proof reading my thesis and his many useful suggestions.

I would like to express my thanks to my wife, Lisa, who has fully supported me in completing my Master degree. She also provided me with the

insight on my thesis topic.

Finally, I would like to thank my Lord and Savior, Jesus Christ, who led me through my studying in UBC and taught me many valuable lessons with His love and patience.

PAUL SIK LEUNG SHUM

The University of British Columbia

December 1996

Chapter 1

Introduction

With the current advancements of image processing, disk management and network management technologies, multimedia computing becomes a feasible and popular field of study. Many multimedia applications, which could not be implemented in the past due to hardware and software limitations, can now be implemented. For example, an experimental video on-demand application which provides access to 250 movies will be put into service in Orlando [FR94, RBE94]. In the near future, many multimedia applications will become an everyday tool to everyone.

Multimedia applications can cover a large part of our everyday life. For example, the world wide web is one of the multimedia applications which has been used in many areas such as ordering consumer products, retrieving public information and viewing images of different tourist places. In addition, it is an active area of research and development to deliver educational courses

through the Internet which allows people to attend a course from a remote location. This application is called "tele-learning". Advertisement and entertainment industries are now exploring the full power of multimedia computing to introduce a new era of living style.

Recently, Rogers Communications Inc. introduced pay per view movie for cable television. The idea of allowing subscribers to select their movie choice becomes more popular because business trend is moving towards a customer driven oriented industry. Due to the high cost of delivering movie on-demand service to subscribers, pay per view is only an intermediate solution. However, with the continuous development of new technologies, the movie on-demand service will eventually arrive to every subscriber home in the future [FR94]. News on-demand is also another service which is quite similar to the movie on-demand. It allows subscribers to select when and what news to be shown. For example, subscribers can choose to watch the business news in the morning and all the sports news in the evening. These are some important multimedia applications which will affect our future living style significantly.

1.1 Multi-clip Queries in Multimedia Systems

In a multimedia system, a user request or *query* is defined as a user's request of one or more video clips. A *video clip* (or *clip*) is defined as the abstraction of a display unit by its content. If a query selects five pieces of news, this query will consist of five video clips. On the other hand, if a query

selects a movie, this query will be considered to have only one video clip.

Multi-clip queries are queries returning two or more video clips. In returning the multiple clips to the user, the display order of the clips must be determined. There are three types of display order: fixed order, partial order and flexible order. A *fixed order query* specifies the display order of all clips. The specification of the order may be given by the user or by the system based on some information that the system keeps on the clips. A *partial order query* is a query which has specified some clips to be delivered in a specific order but other clips can be displayed in any order. In addition, if a partial order query specifies only the display order of the first few clips, the query is also called a *prefix order query.* A *flexible order query* is a query which does not specify the display order of the clips.

The display order of a multi-clip query can be important because some display orders are not acceptable. For example, in a tele-learning application, suppose there are three concepts, A, B, and C, to be explained. If concepts B and C are built based on concept A, then the student will not be able to understand B or C unless A has been displayed earlier. Thus, A must be displayed first. Further, suppose that the display order of B and C does not matter because they are independent of each other. Considering all three video clips together, this query has a prefix order because concept A must be displayed first and the rest does not matter.

Some applications have fixed order queries. For example, the display order of a series of shows about the same story cannot be changed; otherwise,

the viewer cannot understand the whole story. Another example can be found in the tele-learning application, because the display order of the lessons may not be logical if the order is reversed.

There are some applications which have prefix order queries such as the advertisement application and the news on-demand application. In the advertisement industry, the advertiser may set different prices for different positions in the display order. If a company pays the most expensive price, the advertisement video clip of this company will be shown first. The display order of other advertisement video clips can then be shown in any order. The news on-demand application also has prefix order queries. For example, a subscriber may specify certain news to be shown first and then the display order of the rest does not matter.

Finally, there are also some flexible order queries applications such as the news on-demand application and the world wide web application. Often times, a subscriber of the news on-demand application does not care about the display order of the news as long as all the news is shown. In a world wide web application, a browser may display the data on the screen when it has received all the data from the server. Sometimes, it may display some data in random order such as showing several different pictures. Therefore, the server can send the data in whatever order it finds the most efficient.

For each query, the system constructs a schedule such that the disk and buffer requirements will not be exceeded during the display of the video clips. A *schedule* of a multi-clip query for a user defines which video clip is to be

displayed at what time after the query has been admitted to the system. The determination of a schedule of a multi-clip query is subject to the following two important constraints:

1. No two clips for the same query can be shown simultaneously. If clip C_i is displayed at time t_i, and the length of the clip C_i is l_i, then no other clip can be scheduled to be displayed within the time interval $[t_i, t_i + l_i]$. This is the *non-overlapping* constraint.

2. No gap exists between the two successive clips. If clip C_i and clip C_j are two successive clips, and the display time and the length of C_i are t_i and l_i respectively, then the display of clip C_j must start at the time $t_i + l_i$. This is the *continuity* constraint.

For a multi-clip query, the *display sequence* defines the display order of the video clips. If the display sequence of a query has been determined, the schedule of the query can also be established because of the non-overlapping and continuity conditions. Thus, throughout this thesis, we focus only on the display sequence of a multi-clip query.

By definition, a fixed order query has a single display sequence. However, many feasible display sequences are possible for a partial or flexible order query. Not all display sequences that meet the user specification are equal. A display sequence is better when it consumes fewer system resources. The following example shows that some display sequences are better than others.

Example 1.1 Considering a multimedia system which currently has one active query displaying video clips in the sequence <A, B, C, D, E>, a new query which allows flexible order arrives with a request of clips {C, D, E, F, G}. At least two possible display sequences can be chosen. In Table 1.1, the

Period	1	2	3	4	5
Active Query Already in the System	A	B	C	D	E
First Display Sequence	C	D	E	F	G
Second Display Sequence	F	G	C	D	E

Table 1.1: Piggybacked Display Sequences

second display sequence is better than the first one because the new query only requires buffer space and disk resources for two periods of time, while the first display sequence requires the buffer space and disk resources for five periods of time. With the second sequence beginning in the third period, the new query does not require any additional resource because the active query is requesting the same set of video clips, and thus these two queries can be delivered together. □

In the paper [GLM95], the authors refer to the technique mentioned in Example 1.1 as "piggybacking." When more than one query can be delivered at the same time, the buffer and disk consumption can be reduced. This is possible because instead of using two units of buffer and disk resources, only one unit is required. In turn, the throughput of the system can be increased.

1.2 Problem Definition and Complications

The goal of this thesis is to find a display sequence such that the throughput of a multimedia system serving prefix order and flexible order multi-clip queries can be optimized. Only constant bit rate video clips are considered. The negotiation of the quality of services is out of the scope of this thesis.

One natural optimality condition is to maximize the total number of piggybacked clips. To find a display sequence which can maximize the total number of piggybacked video clips is not just to overlap the video clips scheduled to be shown in the system. For example, the video clips of a query can only be shown at one period. If there are two periods which this video clip can be shown, the assignment of the video clip to the period will directly affect the assignment of other video clips. Example 1.2 demonstrates the difference in choosing different periods.

Example 1.2 A new query requests a set of video clips {A, D, E, F, G}. Table 1.2 shows a list of active queries. The video clip E can be chosen in either

Period	1	2	3	4	5
Active Query 1	A	B	C	D	E
Active Query 2	C	D	E	F	G

Table 1.2: Active Queries Display Sequence

period 3 or period 5. If the video clip E is piggybacked at period 5 instead of

period 3, then the video clip G cannot be piggybacked because video clip G can be chosen at period 5 only. Therefore, a better clip display sequence needs to consider the display sequence as a whole instead of individual video clips. □

Furthermore, even when a display sequence of maximum number of piggybacked video clips has found, it may still have other display sequences which can further improve the throughput of the system.

Example 1.3 Consider two display sequences as shown in Table 1.3, both the

Period	1	2	3	4	5
Active Query 1	F	B	H	G	D
Active Query 2	I	F	D	E	G
First Display Sequence	A	(B)	(D)	(E)	C
Second Display Sequence	A	(B)	C	(E)	(D)

Table 1.3: Different Display Sequences Leading to Different Performance

display sequences piggyback three video clips, which are enclosed in parentheses. But they have different effects on the system performance. If the multimedia system does not have any buffer resource in period 5, the second display sequence is better than the first display sequence. The second display sequence piggybacks the video clip D such that no buffer resource is required because the active query 1 has already read the video clip D at period 5. On the other hand, if the system does not have any buffer resource in period 3, then the first display sequence is better than the second display sequence because the active query 2 has already read the video clip D at period 3. □

Therefore, not all display sequences which maximize the piggybacked video clips have the same effect on the throughput of the system. Another sensible optimality condition is to maximize the impact on the buffer space in each period. In addition, there are many other considerations before a display sequence can be determined.

1. In general, video clips may not have a constant buffer requirement. For example, in a news on-demand application, queries may request a set of news clips which are stored in different forms such as still image, video or audio. As each clip has a different form of data, the display rate of the clip is also different. In addition, if some news video clips are stored using different compression methods, the display rate for these clips are also different. Because of the difference in display rates, the buffer requirements for the clips may vary. If the system uses the maximum buffer requirement of all the video clips, it will waste a lot of resources because the buffer will not be fully utilized when displaying the video clips with lower buffer requirements. Therefore, in choosing the piggybacked video clips, the buffer requirements of video clips must also be considered.

2. Furthermore, video clips also may not have a constant length. With variable length video clips, piggybacking becomes extremely complicated because of the non-overlapping and continuity conditions of the schedule. For example, suppose there is an active query displaying <A, B, C, D> and a new query requests the {A, B, D, E} with the length of video clips specified in the Figure 1.1.

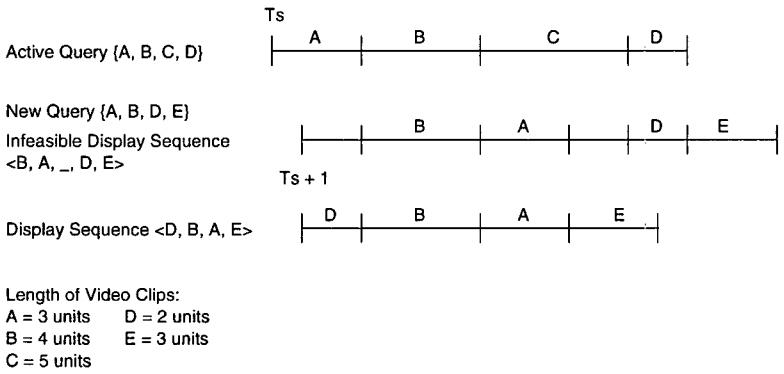

Figure 1.1: Synchronization Problem

The display sequence <B, A, _, D, E> has the maximum number of piggybacked video clips but it violates the continuity condition because E cannot fit in between the video clip A and D. But, the display sequence <D, B, A, E> is feasible. Determining the display sequence becomes much more complicated because the length of each video clip must be considered at the same time when the display sequence is constructed.

3. In a real life application such as news on-demand, there are some video clips which are more popular than the others. The clips which have a higher probability of being chosen are called *hot clips*. If the total number of hot clips is small enough to stay in the cache at all times, then no piggybacking algorithm is required. In the other extreme, if the total number of hot clips is very large, then the probability of piggybacking video clips is very small. In this case, piggybacking is not effective. Therefore, the effectiveness of applying the piggybacking algorithm also depends on the distribution of the hot clips.

4. With any algorithm developed to piggyback video clips, the computational time of the algorithm must be reasonable due to the real time aspect of multimedia applications.

1.3 Contributions of Thesis

In this thesis, different algorithms are proposed to find a display sequence of multi-clip queries such that the throughput of a multimedia system can be optimized. The contributions of this thesis are as follows:

1. As discussed in Section 1.2, finding the display sequence of which maximizes the total number of piggybacked video clips is not a simple task. In this thesis, we show that the maximum matching in a bipartite graph [MS91, HK73, CLR90, FT87] can be used to determine a display sequence with the maximum number of piggybacked clips. We show how to construct the appropriate bipartite graph. We also present a MaxPVC (Maximizing Piggybacked Video Clips) algorithm which can be used in the admission control of a multimedia server.

2. As discussed in Section 1.2, another sensible optimality condition is to maximize the impact on the buffer space. In this thesis, we show that the maximum weight matching in a bipartite graph with an appropriate weighting scheme can be used to determine a display sequence following a set of preference rules such that the impact on the buffer space can

be maximized. We also develop MaxIBS (Maximizing the Impact on Buffer Space) algorithm which can be used in the admission control of a multimedia server.

3. Since the buffer requirement of video clips varies as discussed in Section 1.2 (1), we demonstrate that the maximum weight bipartite matching can be used to find an optimal display sequence for video clips with variable buffer requirement.

4. The techniques mentioned so far are developed based on the assumption of constant length video clips. However, they cannot be used to find a display sequence for variable length video clips. In this thesis, we develop an efficient algorithm, which is called MaxPP (Maximizing Piggybacked Periods) algorithm, to find an optimal display sequence which can maximize the total number of piggybacked periods for variable length video clips.

5. With the distribution of hot clips and the real time aspect of the multimedia application as discussed in Section 1.2 (3) and (4), experiments are done to demonstrate the effectiveness of the MaxPVC, MaxIBS and MaxPP algorithms under a realistic environment. These experiments include:

   (a) queries selecting video clips using a CNN headline news distribution.

   (b) queries selecting video clips using a hot clips distribution.

(c) queries selecting video clips using a random distribution.

(d) prefix order queries instead of flexible order queries.

Our experimental results show that the throughput can increase significantly. Furthermore, the computational time of finding a display sequence is negligible.

1.4 Outlines of Thesis

Chapter 2 describes related work. Chapter 3 describes the MaxPVC algorithm. Chapter 4 describes MaxIBS algorithm. Chapter 5 discusses the experimental results for MaxPVC and MaxIBS algorithms. Chapter 6 discusses how to deal with the variable buffer video clips. Chapter 7 discusses the MaxPP algorithm for variable length video clips. Chapter 8 concludes the thesis.

Chapter 2

Related Work

Many excellent studies have been conducted in the field of multimedia systems. These studies include the fundamental principles of implementing a multimedia system and their architectural design, the conceptual modeling of multimedia data, and different algorithms to increase and optimize the throughput of a multimedia system.

2.1 Multimedia Systems and Their Architectural Design

There are many differences between handling multimedia data and traditional data. One of them is the delay sensitive property of the multimedia data. The multimedia data will no longer be useful if the data cannot be dis-

played to users continuously. For example, when a video is watched, it is not acceptable to display the first few frames and then wait for several seconds before displaying the next few frames. Another difference is the size of the multimedia data. For example, a NTSC quality video with 640 $\times$ 480 pixels (8 bits/pixels) requires 8.7 megabytes/sec. Therefore, the total disk space requirement of a 30 minute video is 8.7 megabytes/sec $\times$ 60 seconds $\times$ 30 minutes = 15.66 gigabytes [GVKR95]. Even with the latest data compression technology, the difference of the size between multimedia systems and traditional database systems is still significant. Therefore, many techniques such as the flexible and adaptive buffer management [FNS95], hot set model [SS82], approximating block accesses technique [Yao77], and the flexible and predictor approach [CY89] which can be used by the relational database systems cannot be used in multimedia systems.

Because of the inapplicability of the traditional techniques, Gemmell and Christodoulakis present some principles on how to store and retrieve delay sensitive multimedia data [GC92]. In their paper, they discuss the reading, the consumption, and the buffer functions of a multimedia system. They also demonstrate how these functions can be used to implement the system. Rangan and Vin also present how to design file systems for multimedia data such as the digital video and audio data [RV91]. Anderson, Osawa and Govindan present a file system for continuous media [AOG92].

Another recent paper provides an overview of how to design multimedia storage servers [GVKR95]. Gemmell, Vin, Kandlur and Rangan describe the

architectures and the algorithms required in the implementation of multimedia storage servers. They present ideas for the placement of data in a single disk and in multiple disk configuration. They also present different disk scheduling algorithms and admission control algorithms which can be used in the implementation. In addition, Lougher and Shepherd present a paper on how to design a storage server for continuous media [LS93]. Neufeld, Makaroff and Hutchinson discuss how to design a variable bit-rate continuous media server for an ATM network [NMH96]. Rangan and Vin present some efficient storage techniques for multimedia data [RV93].

A multimedia system combines different areas of research in computer science such as database management, networking technology, compression, disk and image processing technology. To link these areas together requires both hardware and software architecture designs [RS91, LLW95].

Vin presents an overall architecture view of a multimedia system in [Vin94]. He divides the system into three subsystems: an information management subsystem, a storage subsystem and a network subsystem. The information management subsystem provides the answers for queries generated by users. The storage subsystem is used to store and retrieve the multimedia data in devices such as disk arrays or optical jukeboxes. The network subsystem handles the transmission of the multimedia data to users in a timely manner. He discusses different issues of implementing each subsystem.

In [FR94], Federighi and Rowe present a distributed hierarchical storage manager which is used for a video library. They describe the hardware

architecture of the system as well as the software architecture. Instead of an overview paper, they discuss in detail on how the video data is loaded to a video file server from a tertiary storage. In addition, they also describe the storage organization and a distributed cache management algorithm.

2.2 Conceptual Modeling and Indexing

In order to fulfill the continuity requirement of user queries, the traditional database model is no longer applicable. In addition, the huge size of the multimedia data requires a new structure of representing the data. Furthermore, the characteristics of multimedia data complicate the indexing structure of the data during searching and retrieving of the data.

Gibbs, Breiteneder and Tsichritzis present a new concept of representing the time-based data using "timed streams" as the basic abstraction for modeling the multimedia data [GBT94]. The conceptual model is still under development. There are still many problems in implementing a database system which is capable of supporting the time-based representation.

The complexity of the indexing structure for multimedia data is due to the information contained in the data. For example, a movie has numerous information that users may want to query. For example, the date of production, the actors, a particular shot showing an explosion scene and many more pieces of information can be requested. In [RBE94], Rowe, Boreczky and Eads divide them into three types of indexes: bibliographic, structural and content.

They attempt to solve the problem of locating a video of interest in a video database.

2.3 Performance Optimization

Many research papers discuss how to improve or optimize the performance of a multimedia system. In general, they are using two approaches: the data placement techniques, and the disk scheduling and buffer sharing algorithms.

2.3.1 Data Placement Techniques

The data placement approach divides the multimedia data into different pieces and stores each piece of data in a different disk to increase the performance. This approach aggregates the total bandwidth of multiple disks to fulfill the continuity requirement of the multimedia system. As the data retrieval rate of a disk may not be capable of displaying high resolution multimedia data continuously without disruptions, storing the data in multiple disks allows the data to be retrieved in parallel such that the display of the video will not be disrupted [GR93, BGMJ94, LL95].

In [VRG95], Vin, Rao and Goyal present two data placement policies: fixed-size block and variable-size block. They describe how to store video streams on the disk array such that either each block contains a variable num-

ber of frames or a fixed numbers of frames. They present an analytical result showing that the fixed-size block placement policy is better than the variable-size block placement policy if the application involves frequent creation, modification and deletion of multimedia data. However, the variable-size block placement policy is better if the application does not allow editing.

2.3.2 Disk Scheduling and Buffer Sharing Algorithms

In a conventional system, queries are completed in a relatively short period of time. For example, in a relational database, if a user requests a record, the record will be located and returned by the system very quickly. However, in a multimedia system, the period in answering a query is comparatively longer than a traditional system. For example, if a user requests a two-hour movie, the query will not be completed until the whole movie has been shown. Therefore, the query remains active for two hours. At the same time, the disk can be used to serve other queries because the display rate of the query is slower than the data consumption rate. The disk can be optimized by serving multiple queries at the same time. In this situation, the movement of the disk head to retrieve data can be predicted in advance.

On the other hand, the disk cannot serve too many queries all at the same time. Otherwise, when all the data stored in the buffer has been displayed and the disk has not yet read another set of data, the query will not have data to display. This is called the "starvation" problem. The disk scheduling

approach seeks opportunities to optimize the performance such as maximizing the disk utilization or reducing the I/O demand in a multimedia system without the starvation problem. There are many papers presenting different disk scheduling algorithms to optimize the performance of a multimedia system.

Many disk scheduling algorithms have been proposed. The pre-seeking sweep algorithm [Gem93] splits the data into sorting sets to reduce the disk latencies. The SCAN-EDF algorithm [RW93] combines the shortest seek time first algorithm and the earliest deadline first algorithm to improve the performance of the disk. The grouped sweeping scheduling (GSS) algorithm [CKY94] re-orders the serving queries within the group so that a better performance can be obtained. A valuable resource in a multimedia system is the buffer space because the size of multimedia data requires a large amount of buffer space for storage. Ng and Yang present a buffer sharing scheme and propose two prefetching strategies: simple and intelligent [NY96]. The buffer sharing scheme allows multiple streams (or queries) to share buffers together. Makaroff and Ng present an implementation scheme of buffer sharing in multimedia servers [MN95].

In [GLM95], Golubchik, Lui and Muntz attempt to reduce the I/O operations using the techniques of "batching", "bridging" and "adaptive piggybacking". Their idea is to group different queries together and thus the resources can be optimized. They use their idea on the movie on-demand application only. The "batching" technique is to delay a query in the hope that another query for the same video will arrive within the batching interval.

The "bridging" technique is to hold the data in the buffer such that a later query can be served without accessing the disk. The "adaptive piggybacking" technique groups queries in progress together by adjusting the display rates of the queries.

2.4 Scheduling and Bipartite Matching

Numerous research papers have been discussed in the previous sections. However, none of them considers multi-clip queries. With the multi-clip queries application, the flexibility of determining the display sequences provides an opportunity to improve the throughput of the system.

To solve the display sequence, many sequencing and scheduling problems presented in [GJ79] are investigated. The *sequencing* and *scheduling* problems are to find out the best order of tasks which can be completed without violating other specified conditions. They include problems with one or multiple processors, with weights and without weights on each task to be scheduled, with deadlines and constraints on resources and waiting queues. However, the problem discussed in this thesis is not one of the above problems because the constraints change with time. Therefore, the sequencing and scheduling problems investigated are not useful in solving the display sequence problem. However, the display sequence problem can be simplified as a bipartite matching problem which will be shown in Chapter 3 and Chapter 4. As some parts of the problem in this thesis can be solved by the maximum bipartite matching

problem, a brief description about the bipartite matching problem is presented in the following paragraph.

A *bipartite graph* is an undirected graph, $G = (P, C, E)$, if and only if its vertex set can be partitioned into two subsets, P and C, such that each edge of the graph, $e \in E$ has one endpoint in P and the other in C [MS91]. Figure 2.1 shows an example of a bipartite graph.

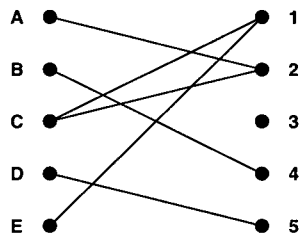

Figure 2.1: A Bipartite Graph

A *matching*, M in a bipartite graph is a subset of edges in which no two edges in M share an endpoint. A maximum bipartite matching is a matching with the maximum cardinality of M. In solving the maximum bipartite matching, two approaches are commonly used. One is using the augmenting path [MS91] and the other is using the maximum flow method [CLR90].

The bipartite matching problem can be solved in $\mathcal{O}(|E|\sqrt{|V|})$ time [NU, HK73] where V is the total number of vertices and E is the total number of edges in the bipartite graph. In addition to the maximum bipartite matching, there is also maximum weight matching which is similar to the maximum bipartite matching except for the fact that it requires maximizing the weight of the edges instead of the total number of matched edges. This problem can be solved in $\mathcal{O}(|E| \cdot |V|)$ time [NU, FT87]. Both of the above algorithms are

very useful in this thesis. The library used to solve the bipartite graphs in this thesis is LEDA library (Library of Efficient Data types and Algorithms) which is developed by Max-Planck-Institut fuer Informatik in Germany.

However, the above algorithms cannot be used to handle the variable length video clips because the variable length video clip does not fit into the bipartite graph model. Therefore, a new algorithm has been developed to solve the variable length video clips case.

Chapter 3

Maximizing the Total Number of Piggybacked Clips

For a flexible or prefix order multi-clip query, the piggybacking technique may be effective if the display sequence is re-ordered such that a video clip can be delivered at the same time to multiple queries. This may lead to a higher throughput of a multimedia system. This chapter presents the Max-PVC algorithm which uses the bipartite graph to find a display sequence with the maximum number of piggybacked video clips so that the sharing of buffer resources may be optimized.

In this chapter, some assumptions are made in finding the display sequence because these assumptions can simplify the analysis. Furthermore, there are some applications which do not violate these assumptions. Assumptions used in this thesis are as follows:

1. Each video clip takes exactly one unit of time, or one *period* to display.

2. Each video clip takes exactly one unit of buffer space resource.

3. All queries are served in first come first serve discipline.

4. The resource being considered in this thesis is the buffer resource only. As all proposed algorithms in this thesis do not affect the disk resource, algorithms proposed by other people such as [NY96] and [MN95] can be used to deal with the optimization of the disk resource.

The assumptions 1 and 2 will be removed in later chapters when discussing queries with variable buffer and variable length requirements.

This chapter will first describe the definitions of reading and candidate sets. It will then show how to construct an appropriate bipartite graph to find out a display sequence which has the maximum number of piggybacked video clips. A theorem is presented to show that the maximum matching from the constructed bipartite graph finds the maximum number of piggybacked video clips. The MaxPVC algorithm is presented at the end of this chapter.

3.1 Reading and Candidate Sets

The *reading set* of a period is defined as the set of video clips which will be read by the system in that period because of the requests of active queries. Only the video clips which are going to be read in a period is important to

solve the display sequence problem. But the information about which active query requesting which video clip at what time is not necessary. Table 3.1 shows an example of the reading sets of periods 1 to 5. The set of video clips

Period	1	2	3	4	5
Active Query 1	A	B	C	D	E
Active Query 2	C	F	G	A	D
Active Query 3	A	C	K		
Reading Sets	{A,C}	{B,C,F}	{C,G,K}	{A,D}	{D,E}
Candidate Sets	{A}	{B}	{G,K}	{A}	$\emptyset$

Table 3.1: Example of Reading and Candidate Sets

of a new query in each period which can be piggybacked is called the *candidate set.* The candidate set of a new query can be found by intersecting the reading set in that period and the set of video clip requested by the new query. For example, a new query requests the set of video clips {A, B, G, I, K}. The candidate sets of all periods for the new query is shown in Table 3.1.

3.2 Maximum Piggybacked Clips Display Sequence

Three steps are used to find a display sequence with the maximum number of piggybacked video clips:

1. Constructing the appropriate bipartite graph;

2. Running the bipartite matching algorithm; and

3. Building the final display sequence.

In the following, we consider these steps one by one.

3.2.1 Constructing the Bipartite Graph

In constructing the bipartite graph $G = (P, C, E)$ to find the maximum number of video clips which can be piggybacked, each node in P represents a period, which is called a *period node*. Each node in C represents a video clip requested by the new query, which is called a *clip node*. The edge $(p, c) \in E$ joining a period node, p, and a clip node, c, represents that the clip, c, is one of the elements in the candidate set in the period p.

Example 3.1 A new query requests {A, B, G, I, K} as in Table 3.1, the corresponding bipartite graph is shown in Figure 3.1. The edge (1, A) means

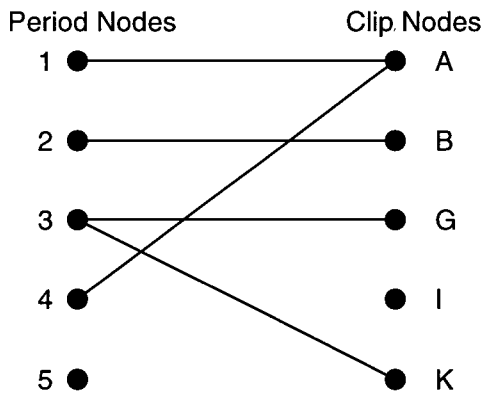

Figure 3.1: A Bipartite Graph Construction Example

that A is a video clip which is in the candidate set of period 1. The edges (3, G) and (3, K) mean that G and K are elements in the candidate set of period 3. □

3.2.2 Running the Bipartite Matching Algorithm

By running the bipartite matching algorithm, a maximum bipartite matching Max can be found. For example, the matching found for Example 3.1 may be $Max = \{(2,B), (3,G), (4,A)\}$. Each edge in Max is the assignment of the clip to the corresponding period. The edge (2, B) $\in Max$ means that the video clip B should be shown at period 2. Section 3.3 will prove that the maximum matching found using the bipartite matching algorithm will find the maximum number of piggybacked clips.

3.2.3 Building the Final Display Sequence

After running the bipartite matching, a list of piggybacked clips is generated. The edges in Max show the periods when the piggybacked video clips should be displayed. However, there are some unmatched nodes in the bipartite graph such as period 5 or period 1 if $Max = \{(2,B), (3,G), (4,A)\}$ in Example 3.1. The rest of the unmatched video clips and periods can be assigned randomly to build a complete display sequence. For example, clip I can be displayed at period 5 and clip K can be displayed at period 1. Therefore,

the complete display sequence of the above new query is <K, B, G, A, I>.

3.3 MaxPVC-Bipartite Theorem

In this section, two definitions and one theorem are used to prove the MaxPVC - Bipartite Theorem. They are stated as below:

An *alternating path with respect to a matching M*, in a bipartite graph $G = (P, C, E)$, is a path which connects edges $e_1, e_2, e_3, \ldots, e_n$ where $e_{2k} \in M$ and $e_{2k+1} \notin M$. For example, when n is 3, e_1 and e_3 are $\notin M$ and e_2 is $\in M$. An alternating path with respect to M can have even or odd number of edges. For example, the odd-edge path in Figure 3.2 begins with an edge $(x, y) \notin M$, $(x, c) \in M$ and $(x', c) \notin M$. The even-edge path begins with an edge $(x, y) \notin M$, $(x, c) \in M$, $(x', c) \notin M$ and $(x', c') \in M$.

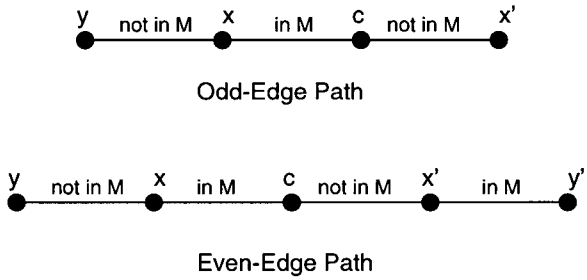

Figure 3.2: Odd-Edge Alternating path and Even-Edge Alternating Path

An *augmenting path with respect to a matching M*, in a bipartite graph $G = (P, C, E)$, is a path which meets the following conditions [MS91]:

1. The first and the last edges, e_{first} and e_{last}, of the path are not matched, that is $e_{first} \notin M$ and $e_{last} \notin M$.

2. Every second edge, e_{2n}, on the path is matched, that is $e_{2n} \in M$.

3. The first and the last vertices on the path are unmatched which means that no edge in M is connected to these vertices.

Theorem 3.1 (Augmenting Path Theorem [Ber57])

A matching is of the maximum cardinality if and only if the graph has no augmenting path with respect to the matching. □

Theorem 3.2 (MaxPVC-Bipartite Theorem)

The maximum bipartite matching of the bipartite graph constructed in Section 3.2.1 finds the maximum number of piggybacked clips of the query. □

Proof:

Suppose that $Max \subseteq E$ is the maximum bipartite matching of the constructed bipartite graph, and $P' = \{p : (p, c) \in Max\}$ and $C' = \{c : (p, c) \in Max\}$. Assume that there exists a clip y which can be piggybacked and will increase the total number of piggybacked video clips, but $y \notin C'$. Since y can be piggybacked, there must be a corresponding period x such that y is in the candidate set of the period x. Based on the construction of the bipartite graph in Section 3.2.1, (x, y) will be one of the edges in the bipartite graph. If $(x, y) \notin Max$, there are two possibilities:

1. The period x is matched with another clip, c. If this is the case, an alternating path can be constructed.

   - If the alternating path has even number of edges, adding (x, y) into Max will at least remove one edge in Max such that the total

number of piggybacked clips will not be increased. Therefore, it contradicts the assumption that the addition of (x, y) will increase the total number of piggybacked clips.

- If the alternating path has odd number of edges, the first and the last vertices of the path will not in P' and C' because the first and the last edges are not in Max. By the definition of the augmenting path, this alternating path is an augmenting path. By the Augmenting Path Theorem, Max cannot be a maximum bipartite matching which contradicts to the above assumption that Max is a maximum bipartite matching.

2. The period x is not matched. If this is the case, adding (x, y) into Max will result in a matching with a greater cardinality, which contradicts the maximum bipartite matching.

Combining the above two cases and by contradiction, the theorem is proved.

□

3.4 MaxPVC Algorithm

Previous sections show how to find the maximum number of piggybacked video clips using bipartite graph. This section describes the full MaxPVC algorithm.

Algorithm 3.1 (MaxPVC Algorithm)

1. All new queries are appended to a queue.

2. Find the candidate set of each period for the query at the head of the queue.

3. Check the candidate set in each period. If the candidate set is empty and all the buffer spaces have been allocated in that period, the query is put back to the head of the queue. The algorithm will be invoked again after the last period which the candidate set is empty and all the buffer spaces have been allocated.

4. Construct a bipartite graph as shown in Section 3.2.1.

5. Invoke the maximum bipartite matching algorithm. A maximum matching Max is generated.

6. Complete the display sequence as discussed in Section 3.2.3.

7. Check the display sequence to see whether it will exceed the maximum buffer space allowed in each period. If any one period exceeds the buffer space limit, the query will be put back to the head of the queue. The algorithm will be invoked again in the following period. Otherwise, the query will be activated and buffer space will be allocated for this query.

□

The purpose of step 3 is to find out whether it is possible to generate a display sequence of the query or not. For example, suppose that the candidate set of a new query at period 3 is empty and that all the buffer resources at period 3 have been allocated to other active queries. In this case, it is impossible to generate any feasible display sequence for the query before the time period 3. Therefore, the remaining steps need not be carried out.

The purpose of step 7 is used to verify that the final display sequence of a query will not violate the buffer resource constraint. This is important although step 3 has done the checking because there are cases where a display sequence can pass through step 3 but it may violate the buffer resource constraint. For example, a new query requests {A, B, C, D, E}. The candidate set of the query at all period is {C}. In this case, it may be able to generate a feasible display sequence. Therefore, it passes the checking of step 3. However, if there are two periods which have allocated all the buffer resource to other active queries, the generated final display sequence will definitely violate the buffer resource constraint. Therefore, step 7 is to confirm the feasibility of the final display sequence of the query.

3.5 Summary

The MaxPVC algorithm has been discussed in detail in this chapter. The MaxPVC-Bipartite theorem shows that the maximum number of piggy-backed video clips can be obtained by finding the maximum bipartite matching

of an appropriate bipartite graph. A full detail of the complete MaxPVC algorithm has also been presented.

However, the MaxPVC algorithm has a problem when the buffer requirement is considered together. Chapter 4 describes the MaxIBS algorithm which is used to handle the above problem. Chapter 5 will compare the experimental results of MaxPVC and MaxIBS algorithms.

Chapter 4

Maximizing the Impact on Buffer Space

The display sequence of a query which maximizes the total number of piggybacked video clips can be found by using the MaxPVC algorithm described in Chapter 3. As mentioned in Example 1.3, the buffer resource for each active query will directly affect the system performance. However, the MaxPVC algorithm does not take the effect of the buffer resources into consideration. In view of the insufficiency of MaxPVC algorithm, MaxIBS algorithm is developed in the hope that the throughput of the system can be further improved.

This chapter begins with the description of the buffer consumption graph. It discusses the criteria for choosing the piggybacked periods and lists the order of preference for the piggybacked periods. It then describes

how to find a display sequence, which chooses the piggybacked periods according to the order of preference, using the maximum weight bipartite graph with an appropriate weighting scheme. It shows that the maximum weight bipartite matching finds the display sequence which selects the piggybacked periods according to the order of preference. The complete MaxIBS algorithm is described at the end of this chapter.

4.1 Buffer Consumption Graph

A general system buffer consumption graph is shown in Figure 4.1. The

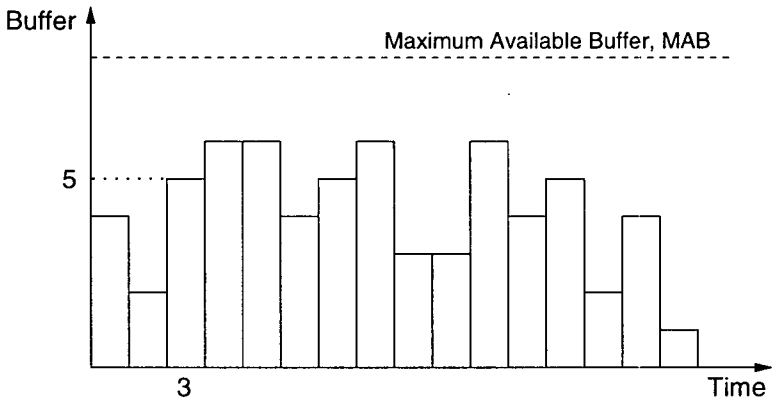

Figure 4.1: General Buffer Consumption Graph

maximum available buffer, MAB, is the maximum amount of buffer space available to the system. The *maximum allocated buffer*, MRB, is the total amount of buffer which has been allocated by queries in each period. For example, the maximum allocated buffer for period 3 is five units in Figure 4.1. The buffer consumption graph can be divided into three different types of

periods.

1. Type I periods are periods in which the buffer allocated has already reached the maximum allocated buffer MRB.

2. Type II periods are periods which meet the following two conditions:

   (a) They are not Types I periods; and

   (b) For all later periods $t_1 > t$, the amount of buffer space required in periods t_1 are less than or equal to that needed in t.

3. Type III periods are periods t where there exists a later period $t_1 > t$ such that the buffer reserved at t_1 is greater than the buffer reserved at t and they are not Type I or Type II periods.

Based on the above classification, the type of each period of a general buffer consumption graph is shown in Figure 4.2.

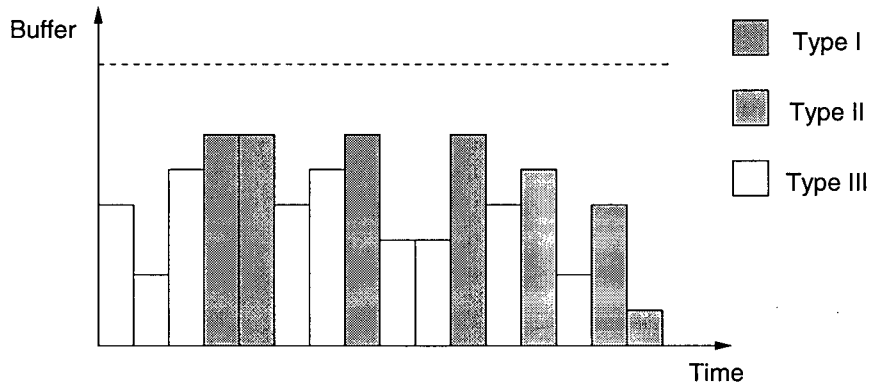

Figure 4.2: Classification of Periods in Buffer Consumption Graph

4.2 Order of Preference for Piggybacked Periods

If a video clip can be piggybacked at periods of either Type I, II or III, which periods should be chosen? The MaxPVC algorithm described in Chapter 3 can choose anyone because they will all lead to the maximum number of piggybacked video clips. However, in consideration of the buffer requirement, some choices are better than others.

Example 4.1 A new query arrives at period 1 and requests eight video clips which has only one video clip that can be piggybacked at either period 2 or period 8 as shown in Figure 4.3, the display sequence which piggybacks at period 2 is not a feasible schedule. The maximum available buffer in period 2

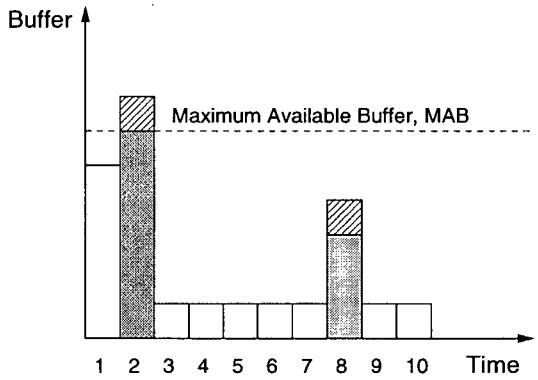

Figure 4.3: Choosing Different Piggybacked Periods

is equal to the maximum allocated buffer in period 2 which means that no more buffer resource is available at period 2. Therefore, the display sequence

which chooses to piggyback at period 8 is a better display sequence than the one which chooses to piggyback at period 2 because the query can be admitted into the system earlier. Thus, the throughput of the system may be increased. □

If a video clip can be piggybacked at more than one period, the choice of selecting the period will directly affect the throughput of the system as shown in Example 4.1. Therefore, an order of preference for piggybacked periods should be determined such that the generated display sequence will optimize the throughput of the system. The proposed order of preference is as follows:

1. Select periods of Type I from right to left in the buffer consumption graph.

2. Select the maximum number of periods of Type II from left to right in the buffer consumption graph after selecting all possible periods of Type I.

3. Select periods of Type III after selecting all possible periods of Type I and II.

In formulating the above order of preference, the maximum allocated buffer is taken into consideration. As the period reaches the maximum available buffer, if it cannot be piggybacked with any video clip, the query must wait after this period has passed. Therefore, it is more important to keep the maximum allocated buffer as minimum as possible so that new queries can be admitted regardless of the display sequences.

Another consideration is the future buffer allocation. If two periods of the same maximum allocated buffer can be piggybacked, it may be better to choose the one on the right hand side because the effect of the piggybacking lasts longer and more queries may be benefited. For example, if either period 1 or period 10 can be piggybacked and they both have the same maximum allocated buffer, it is better to choose period 10 because all queries requested during the period 2 to period 10 are benefited by this piggybacking. However, if period 1 is chosen which means that period 10 requires to increase the maximum allocated buffer, all queries arrive after period 2 will suffer the increase of the maximum allocated buffer at period 10.

The following are all possible cases when choosing the piggybacked periods of different types.

1. When there are two periods of Type I which can be piggybacked, the one in the right side will be chosen according to the above order of preference because of the future buffer allocation benefit.

2. When there are one period of Type I and one period of either Type II or Type III, the period of Type I is chosen because it has larger maximum allocated buffer over the others.

3. When there are two periods of Type II, the one in the left side is chosen because it has the larger maximum allocated buffer.

4. When there are three periods of Type II which two periods can be piggybacked if one is not chosen, it is better to piggyback two periods instead

of one even though the one is in the left side of the other two periods.

5. When there are one period of Type II and one period of Type III, and the Type III period is left of the Type II period, the Type II period is chosen because of the future buffer allocation benefit. If the Type III period is right of the Type II period, the Type II period is chosen because it has larger maximum allocated buffer.

6. When there are two periods of Type III, no order of preference is set for the simplicity of the algorithm.

4.3 Finding a Display Sequence with the Order of Preference

As the bipartite matching used in MaxPVC algorithm chooses the matching randomly as long as it finds the maximum number of piggybacked video clips, it cannot be used in choosing a display sequence with the above order of preference. However, the maximum weight bipartite matching algorithm can be used to find a matching which matches the periods with the above order of preference if an appropriate weighting scheme is used.

The procedure to find a display sequence with the order of preference for piggybacked periods is as follows:

1. Constructing the bipartite graph as described in Section 3.2.1;

2. Assigning the weight to each edge;

3. Running the maximum weight bipartite matching algorithm which is similar to the step described in Section 3.2.2 except that the maximum weight bipartite matching algorithm is used instead of the bipartite matching algorithm; and

4. Building the final display sequence as described in Section 3.2.3.

4.3.1 Weighting Scheme

The weighting scheme used must be able to show the order of preference for the piggybacked periods. That is, the higher the order of preference, the greater the weight is. In addition, three constraints are required.

1. To guarantee all the periods of Type I chosen before any periods of Type II or Type III requires the total weight of all periods of Type II and Type III smaller than any weight of any period of Type I.

2. To guarantee all the periods of Type II chosen before any periods of Type III requires the total weight of all periods of Type III smaller than any weight of any period of Type II.

3. If there are two periods of Type II which can be piggybacked if one period of Type II is not chosen, it is better to piggybacked two periods instead of one period. This requires the total weight of any two periods of Type II larger than any one period of Type II.

The following weight scheme is used to assign the weight according to the order of preference discussed in Section 4.2.

1. The weight of periods of type III is 1.

2. The weight of periods of type II is assigned in the following way. If there are k periods of type II and m periods of type III, the weight for rightmost period is:

$$W_1 = \begin{cases} k & \text{if } k > m \\ m+1 & \text{if } k \leq m \end{cases}$$

and the rest from the right to left is $W_2 = W_1 + 1, W_3 = W_1 + 2, \cdots$. In this case, the total weight of all periods of type II is:

$$\sum_{i=1}^{k} W_i = \frac{k}{2}(2W_1 + k - 1)$$

3. The weight assignment for periods of type I from the left to right is $2^k, 2^{k+1}, \cdots, 2^{k+i}$ where $2^k > \sum W_{typeII} + \sum W_{typeIII}$.

Figure 4.4 shows an example of assigning the weight to periods of different types. All periods of type III are assigned to 1 as described in step 1. There are nine periods of type III and three periods of type II. Therefore, the starting weight for periods of type II is equal to $(9 + 1)$ as shown in step 2. Based on the equation stated in step 2, the $\sum W_{typeII} = \frac{3}{2}(2 \times 10 + 3 - 1) = 33$. The starting weight for periods of type I is equal to $2^8 = 64$ because $64 > 33 + 9$ as described in step 3.

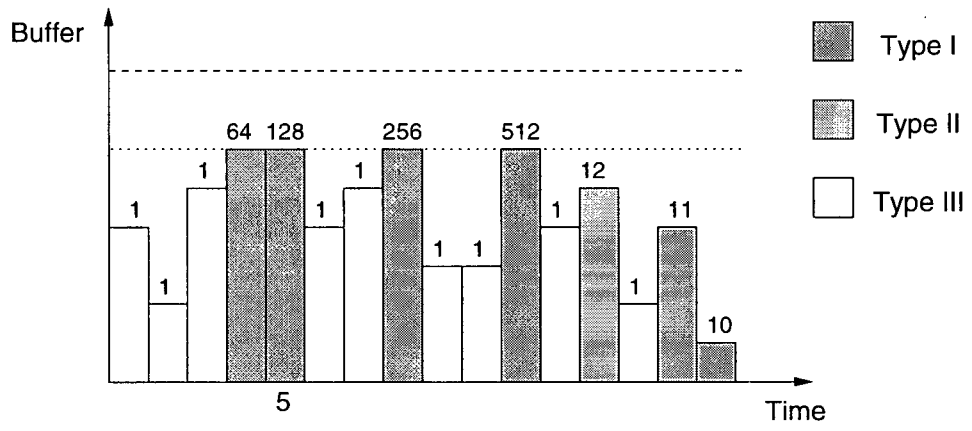

Figure 4.4: Weight Assignment for Different Types of Periods

4.3.2 Assigning Weight

The weight of the edges in the constructed bipartite graph is assigned based on the above weighting scheme. The weight of all the edges connected to the period will be assigned to the value calculated based on the above weighting scheme. For example, the weight of all the edges connected to the period 5 in Figure 4.4 is 128.

4.4 MaxIBS-Maximum Weight Bipartite Theorem

Theorem 4.1 (MaxIBS-Maximum Weight Bipartite Theorem)

The maximum weight bipartite matching of the bipartite graph constructed in Section 3.2.1 with the weighting scheme described in Section 4.3.1 finds the piggybacked periods of the query according to the order of preference

stated in Section 4.2. □

Proof:

Suppose that $Max \subseteq E$ is the maximum weight bipartite matching of the constructed bipartite graph, $P' = \{p : (p, c) \in Max\}$ and $C' = \{c : (p, c) \in Max\}$ and W is the total weight of the matching Max. Assume the followings:

1. There exists a period x which has a higher order of preference than one of the periods, say $x' \in Max$; and
2. x can replace x', and $x \notin P'$.

As x can be piggybacked, there must exist at least one video clip y such that y is in the candidate set of the period x. Based on the construction of the bipartite graph in Section 3.2.1, (x, y) will be one of the edges in the bipartite graph. If $(x, y) \notin Max$, there are two possibilities:

1. The video clip y is unmatched with any other period. As $(x, y) \in E$ and all edges has a positive value in weight according to the weighting scheme described in Section 4.3.1, the addition of the edge (x, y) to the matching Max will have a total weight greater than W. This contradicts the assumption of the maximum weight bipartite matching Max which has the maximum weight.
2. The video clip y is matched with another period. An alternating path as defined in Section 3.3 can be constructed. If x can replace the period x', x' must be on one of these paths. Otherwise, they are independent to each other and cannot replace each other.

- If the alternating path has an odd number of edges, the last edge of the path must be unmatched. By choosing all the unmatched edges instead of the matched edges, it actually matches all the periods on the path. Therefore, the total weight by removing the matched edges from Max and adding all the unmatched edges to Max will be increased by the weight of the edge (x, y) which is greater than zero. This contradicts the assumption that Max has the maximum weight.

- If the alternating path has an even number of edges, the last edge of the path must be matched. The edge connecting to the period x' must be the last one on the path because all periods on the path will still be matched. If all unmatched edges are chosen, then x' is replaced by x.

  (a) If x and x' are periods of Type I, x must be on the right side of x'. According to the weight scheme, the weight for x must be greater than the weight of x'. By replacing x' with x, the total weight of the matching Max will be increased which contradicts to the assumption that Max has the maximum weight.

  (b) If x is a period of Type I and x' is a period of either Type II or Type III, the weight of x will always be greater than the weight of x'. Similarly, it contradicts the maximum weight matching assumption.

  (c) If x is a period of Type II, then x' must be either a period of

Type II which is on the right side of x or a period of Type III. According to the weighting scheme, the weight of x will be greater than the weight of x'. Therefore, it also contradicts the maximum weight bipartite matching assumption.

(d) If x is a period of Type III, there is no preference in choosing x or x'. Therefore, it contradicts that x has a higher order of preference than x'.

Combining the above two cases and by contradiction, the theorem is proved. □

4.5 MaxIBS Algorithm

Algorithm 4.1 (MaxIBS Algorithm)

The MaxIBS algorithm is almost exactly the same as the MaxPVC algorithm as described in Section 3.4 except that the edges in bipartite graph will be assigned weights according to the above weighting scheme before running matching algorithm and the maximum weight bipartite matching is used instead of the bipartite matching. □

4.6 Summary

The MaxIBS algorithm has been discussed in detail in this chapter. The MaxIBS-Maximum Weight Bipartite Theorem shows that the maximum weight bipartite matching finds the piggybacked periods of a query according to the order of preference stated in Section 4.2. In theory, both algorithms may improve the throughput of a system. Furthermore, the MaxIBS algorithm is better than the MaxPVC algorithm because MaxIBS algorithm takes the buffer requirement and the future buffer allocation into consideration. Chapter 5 tries to investigate the effectiveness of both algorithms through several experiments.

Chapter 5

Experimental Results

The MaxPVC and MaxIBS algorithms described in Chapter 3 and Chapter 4 may potentially improve the throughput of a multimedia server with multi-clip queries. Experiments must be carried out to simulate the real life environment in order to test the effectiveness of these algorithms. This chapter begins with the discussion about the simulator and the parameters used in the simulation. It then shows and analyzes the results of the experiments.

5.1 Simulator

The simulator used in the experiments is written in C++. It uses the LEDA library (Library of Efficient Data types and Algorithms) written by

Max-Planck-Institut fuer Informatik in Germany. Both the bipartite graph algorithm and the maximum weight bipartite graph algorithm have already been implemented by LEDA.

The simulator creates 64 users when it starts running. Each user generates queries requesting different video clips. Each query is then processed by the server in a first come first serve basis. A display sequence is generated based on the specified algorithm such as MaxPVC or MaxIBS algorithm. If there is no specified algorithm, the display sequence generated will be the same order as the query requests. Once the query has been admitted to the system, the user will generate another new query until the total number of queries for this simulation have been generated. The simulator stops running after completing all the requested queries.

5.2 Simulation Parameters

There are many parameters which can be set up during the simulation. As the buffer size and the length of each video clip in the following experiments are constant, their values are set to one. Each simulation generates 5,000 queries. The number of video clips requested per each query is ranged from 5 to 20. In order to generate reliable statistical results, each simulation is running with 10 trials with 5 different seed values for the random number generator.

5.3 Experiments

There are three sets of results for each experiment. One set of results is generated without using any algorithm to serve as the baseline for comparison. The other two sets of results are generated using MaxPVC and MaxIBS algorithms. Each experiment varies the values of maximum available buffer and the total number of available video clips to observe the effect of the algorithms in different situations.

Although the throughput of a system is measured by how many queries are processed per unit time, it can also be modified to measure how many periods are required to process a fixed number of queries without affecting the analysis of the result. The raw data obtained by the simulation is the total number of periods required to process 5,000 queries. Therefore, further processing of the raw data is not necessary. All results are shown as throughput ratios to the baseline data to simplify the analysis.

In order to find out the effectiveness of MaxPVC and MaxIBS algorithms under different types of queries, four experiments have been carried out with the following types of queries:

1. Queries with video clips from a CNN distribution;
2. Queries with video clips from a hot clips distribution; and
3. Prefix order queries.
4. Queries with video clips from a random distribution;

5.3.1 Experimental Result for Queries with Video Clips from a CNN Distribution

In this experiment, the distribution of clips chosen by each query follows the CNN Headline News. The distribution is quite close to a Zipf's distribution. That is, if clips are organized in descending order of appearance or requested frequencies, the frequency curve shows an exponential decay. The effectiveness of the MaxPVC and MaxIBS algorithms for CNN distributed queries is shown in Figure 5.1. The result is generated with 12 maximum

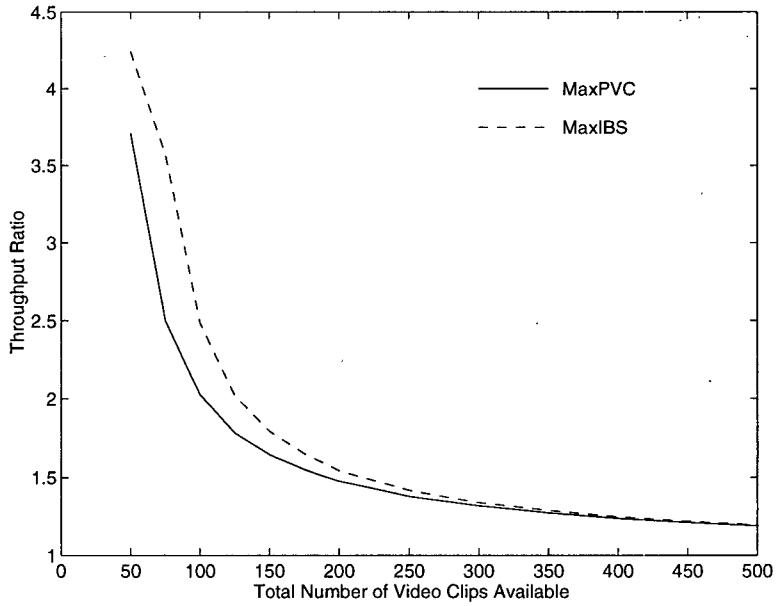

Figure 5.1: MaxPVC and MaxIBS Throughput Ratio for Queries with CNN Distributed Video Clips

available buffer in the server. Based on the above result, both algorithms, MaxPVC and MaxIBS, increase the throughput of the server from over 350%

to around 50% when the total number of video clips available to be chosen ranges from 50 to 200. Even when the total number of video clips available is 450, the increase in throughput is still over 20%. As the total number of video clips available to be chosen increases, the effectiveness of the algorithms decreases because the number of video clips which can be piggybacked decreases. When the queries choose the video clips using the CNN distribution, the MaxIBS algorithm performs better than the MaxPVC algorithm when the total number of video clips is below 200. When the throughput ratio between MaxPVC and MaxIBS is calculated, it ranges from 40% to less than 10% when the total number of video clips available ranges from 75 to 150.

The computation time of the MaxPVC algorithm per each query is about 1.3 milliseconds running on a Sun Sparc workstation while the computation time of the MaxIBS algorithm is about 1.6 milliseconds. The computation time of both algorithms are negligible.

When the maximum available buffer increases, the throughput ratio increases as shown in Figure 5.2. The graph shows that when the total number of video clips increases to 1,000, the increase in throughput is still over 20% if the maximum available buffer doubles from 12 to 24 units.

5.3.2 Experimental Result for Queries with Video Clips from a Hot Clips Distribution

In this experiment, the total number of hot clips is set to 50 for all the

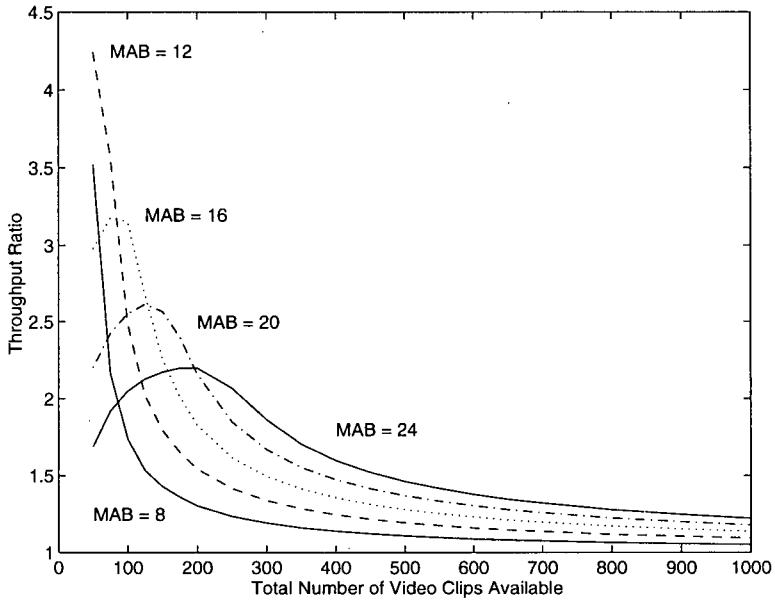

Figure 5.2: MaxIBS Throughput Ratios for Queries with CNN Distributed Video Clips with Different Maximum Available Buffer

simulation. The probability of choosing the hot clips is expressed as a ratio. For example, 1:4 means that there is four times higher chance in choosing the hot clips than choosing non-hot clips. The effectiveness of the MaxPVC and MaxIBS algorithm for queries with a hot clips distribution is shown in Figure 5.3. The result shows that when the probability of choosing the hot clips increases, the MaxIBS algorithm performs better than the MaxPVC algorithm. However, the difference is still minimal. When the probability of choosing the hot clips increases, the graph is shifting to the right side as shown in Figure 5.4. The ratios used in the graph are 1:1, 1:2, 1:4, 1:6, 1:8, 1:12, and 1:16. If the hot clips probability increases from 1:1 to 1:16, the throughput ratio of the MaxIBS can maintain 20% when the total available video clips increases from 300 to 900.

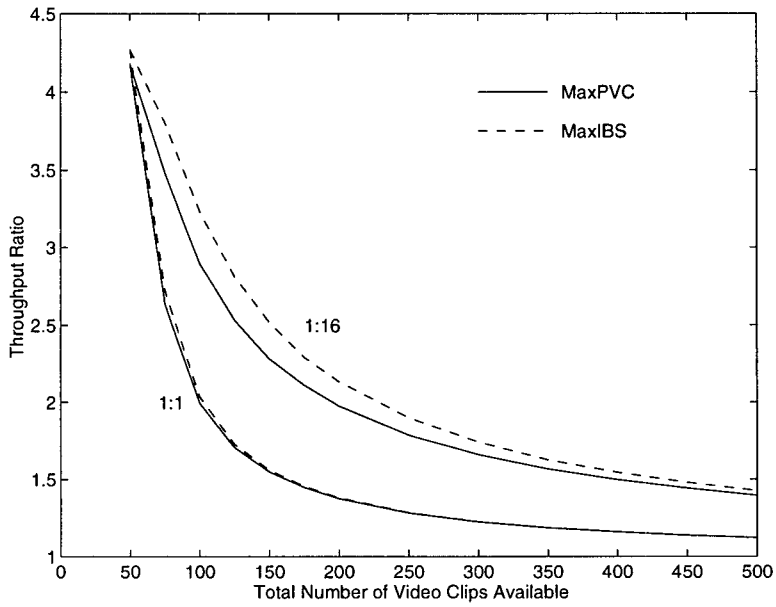

Figure 5.3: MaxPVC and MaxIBS Throughput Ratio with hot clip ratio 1:1 and 1:16

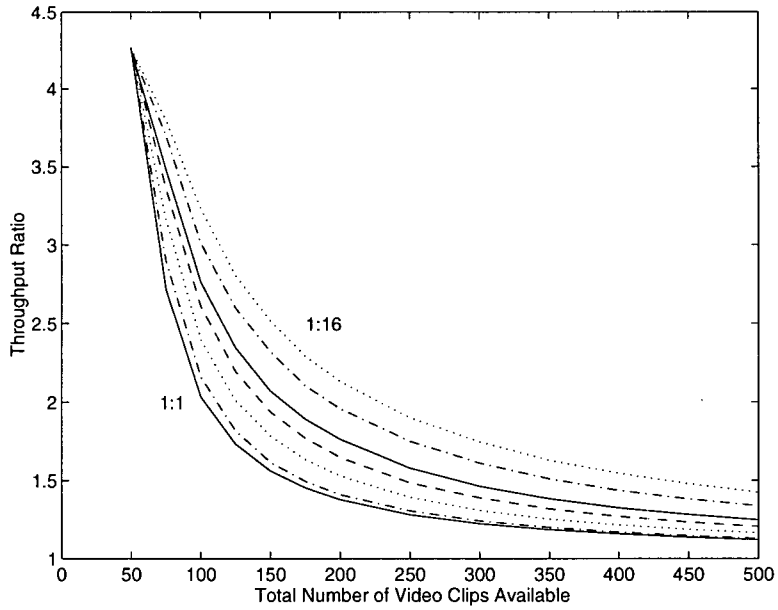

Figure 5.4: MaxIBS Algorithm with Queries Generated from a Hot Clips Distribution

5.3.3 Experimental Result for Prefix Order Queries

In this experiment, the display order of the queries is fixed in a certain percentage ranging from 0% to 100%. The effectiveness of the MaxPVC and MaxIBS algorithm for prefix order queries is shown in Figure 5.5 and Figure 5.6. The result shows that when 20% of the display order of the queries is fixed and the total number of video clips available is 150, the increase in the throughput ratio is over 20% in both MaxPVC and MaxIBS algorithms. However, when the percentage of prefix portion increases, the effectiveness of both algorithms decreases because the possibility of piggybacking decreases.

5.3.4 Experimental Result for Queries with Video Clips from a Random Distribution

The effectiveness of the MaxPVC and MaxIBS algorithms for randomly distributed queries is shown in Figure 5.7. The result shows that both algorithms, MaxPVC and MaxIBS, increase the throughput of the server from over 400% to 50% when the total number of video clips available to be chosen ranges from 50 to 150. Even when the total number of video clips available is 300, the increase in throughput is still over 20%. As the total number of video clips available to be chosen increases, the effectiveness of the algorithms decreases because the number of video clips which can be piggybacked decreases.

This experiment also finds out that the effectiveness of both algorithms

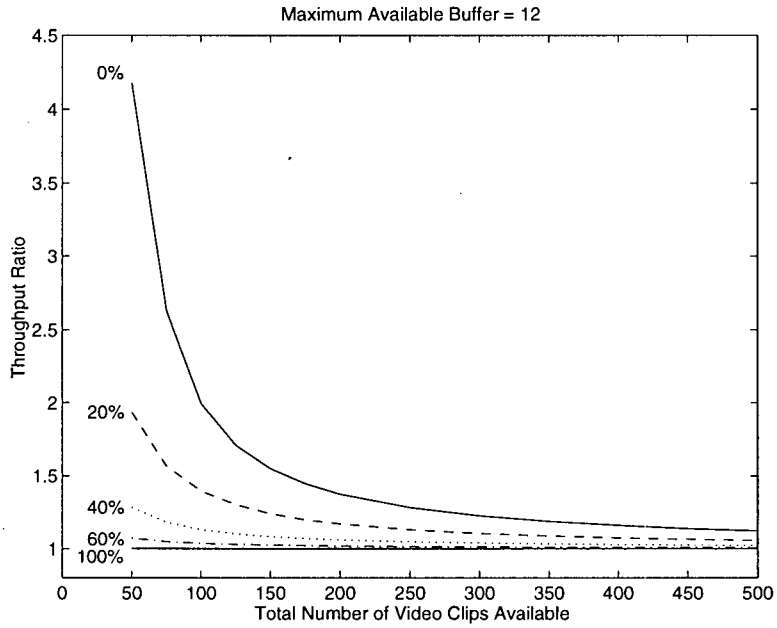

Figure 5.5: MaxPVC Throughput Ratios for Prefix Order Queries

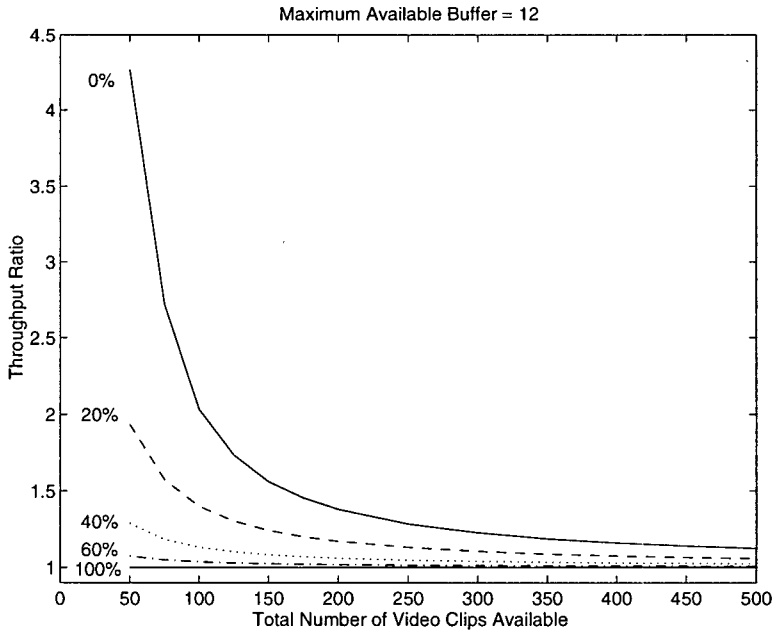

Figure 5.6: MaxIBS Throughput Ratios for Prefix Order Queries

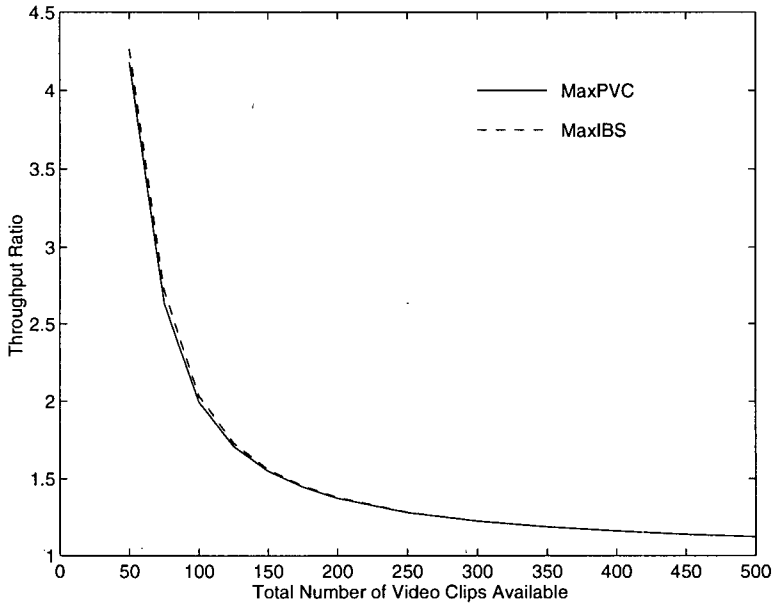

Figure 5.7: MaxPVC and MaxIBS Throughput Ratio for Queries with Random Distributed Video Clips

are very similar. When the total number of video clips available increases over 150, the difference of the throughput ratios is insignificant. The performance improvement of MaxIBS algorithm over the MaxPVC algorithm is less than 4% with less than 100 total available video clips.

5.4 Result Analysis

With the results of the experiments on the flexible order queries, the throughput of a multimedia server has increased significantly especially when the total number of the video clips available to be chosen is small. As the total number of clips available to be chosen increases, the effectiveness of the

algorithms decreases because the total number of clips which can be piggybacked decreases. However, even with a large number of video clips available, the throughput improvement is still significant if queries choose video clips from a CNN distribution or from a hot clip distribution. In general, queries are seldom with video clips from a random distribution.

The MaxPVC algorithm performs as well as the MaxIBS algorithm especially when the total number of available video clips increases. As the total number of available video clips increases, the possibility of any video clip which can be matched with more than one period decreases. Therefore, the matching found by the algorithms is likely to be identical. Therefore, the MaxIBS algorithm has the same performance as the MaxPVC algorithm.

The prefix order queries will decrease the effectiveness of the MaxPVC and MaxIBS algorithms because it minimizes the choice of piggybacking. When the percentage of the prefix portion increases, the effectiveness of the throughput decreases.

The increase in the computational time of finding a display sequence using MaxPVC and MaxIBS algorithms is in the order of milliseconds. Even though the increase in throughput is 10%, the algorithms are still worth to run because the cost of gaining the additional 10% is so insignificant.

As the experiments demonstrate that the MaxPVC and MaxIBS algorithms increase the throughput of a multimedia server significantly, the consideration of removing the constant buffer and constant length video clips limitation is worth to investigate. Chapter 6 and Chapter 7 will discuss how

to handle video clips of different buffer requirement and video clips of different length.

Chapter 6

Variable Buffer Video Clips

As discussed in Section 1.2 (1), video clips may have different forms of data. Thus, each video clip has different buffer requirement. In this thesis, a variable buffer video clip is considered as a video clip requiring multiple units of buffer. For example, video clip A and B may require one kilobyte and four kilobytes of buffer space respectively. If one unit of buffer is set to one kilobyte, video clip A and B will require one and four units of buffer space respectively. This chapter investigates how to generate a display sequence for variable buffer video clips which can improve the throughput of a multimedia system.

6.1 MaxPVC Algorithm and MaxIBS Algorithm

As the MaxPVC algorithm only considers to maximize the total number of piggybacked video clip, this algorithm can also be used to handle variable buffer video clips. When the final display sequence is built, the original MaxPVC algorithm does not care about the assignment order of the unmatched clips because all of them will increase the buffer unit by one. However, when the video clips have different buffer requirement, the assignment order affects the feasibility of the display sequence. For example, if period 1 and period 2 have one buffer unit and three buffer units available to be used respectively, and there are two unmatched video clips A and B which require one and three buffer units respectively, the display sequence <B, A> will be considered as infeasible but the display sequence <A, B> will be considered as feasible. Therefore, it is important to assign the unmatched video clips in the order from clips with the maximum buffer requirement to the minimum and from the earlier periods to the later periods. The order of assignment is called the *buffer order assignment*. The MaxPVC algorithm described in Section 3.4 can be used for variable buffer video clips except that the assignment order of the unmatched clips is changed from the random order to the buffer order.

The MaxIBS algorithm uses a weighting scheme to maximize the impact on buffer space. When the variable buffer video clips are considered, the buffer

allocated in each period does not only depend on the type of the piggybacked periods, but it also depends on the buffer requirement of the piggybacked video clips. Therefore, the weighting scheme used in Section 4.3.1 is insufficient to handle the variable buffer video clips.

To find a display sequence which takes the buffer requirement of video clips, the maximum impact on the buffer space of the system and the total buffer requirement altogether into consideration is difficult because all these parameters are interdependent. The weighting scheme which incorporates all the above parameters will be very complicated. However, as the buffer resource is scarce in every multimedia server, if video clips with a high buffer requirement can be piggybacked, the throughput of the servers may be increased. Therefore, for simplicity, the buffer weighting scheme is used. The *buffer weighting scheme* is to assign the buffer requirement as the weight to each video clip. The MaxIBS algorithm described in Section 4.5 can be used for variable buffer video clips except that the weighting scheme used is changed to the buffer weighting scheme and the assignment order of the unmatched clips is changed from the random order to the buffer order. The display sequence found by running the MaxIBS algorithm using the buffer weighting scheme shares the maximum number of buffer units for each query.

6.2 Experimental Result

An experiment has been carried out to investigate the effectiveness of the MaxPVC algorithm and the MaxIBS algorithm using the buffer weighting scheme. The experiment runs for 5 trials with 5 different seeds for the random number generator. The buffer requirement of the video clips ranges from 1 to 3 units in a random distribution. The maximum available buffer is 12 units. Queries request video clips from a random distribution. The result is shown in Figure 6.1. Based on the result, both algorithms increases the throughput

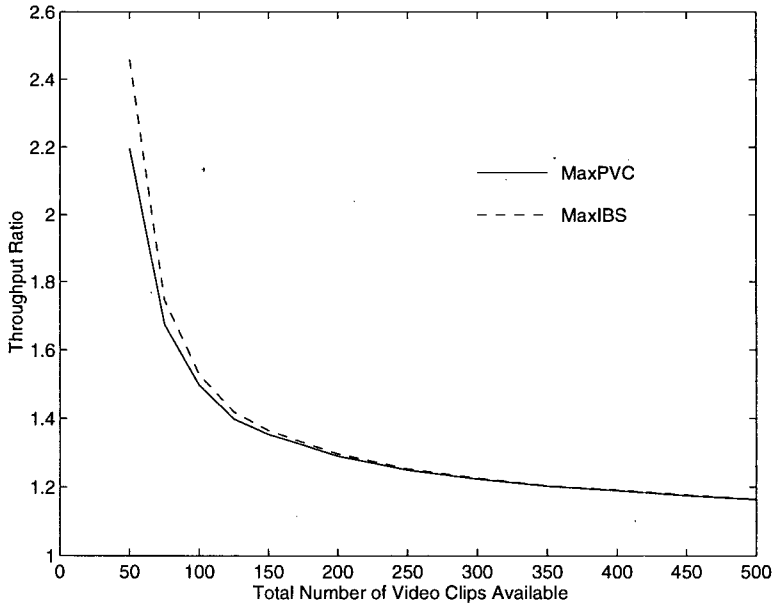

Figure 6.1: MaxPVC and MaxIBS Throughput Ratio for Variable Buffer Video Clips

of the server from 210% to 20% when the total number of video clips available to be chosen ranges from 50 to 350 respectively. The MaxPVC and MaxIBS

algorithms perform almost the same when the total number of video clips increases to 350. The increase in throughput of MaxIBS over the MaxPVC is over 10% when the total number of video clips available is 50 and over 4% when the total number of video clips available is 75. Therefore, MaxPVC algorithm is a very effective algorithm to improve the throughput of a server.

The computation time of the MaxPVC algorithm per each query is about one millisecond running on a Sun Sparc workstation while the computation time of the MaxIBS algorithm is also about one millisecond. The computation time of both algorithms are negligible.

Chapter 7

Variable Length Video Clips

As discussed in Section 1.2 (2), video clips may also have different lengths. When the video clips have different lengths, the MaxPVC and Max-IBS algorithms cannot be used because they assume that all video clips have the same length. A variable length video clip can be considered as a video clip of multiple units of length. For example, if a video clip is 15 minutes long and the unit length of the system is 5 minutes, then the video clip is considered to have 3 units of length. If another video clip is 10 minutes, then this video clip has 2 units of length. This aggregation can simplify the analysis without the loss of generalization. In fact, actual multimedia applications generally have video clips in multiple units of length. For example, the advertisement video clips shown on the television have lengths of 15 seconds or 30 seconds.

This chapter investigates how to handle variable length video clips. It describes a MaxPP algorithm which is used to find a display sequence for

which the total number of periods for the selected piggybacked video clips is the largest. At the end of the chapter, it shows the experimental results of using the MaxPP algorithm.

7.1 Insufficiency of Bipartite Matching

To find a display sequence using the MaxPVC or MaxIBS algorithms, a bipartite graph is created as described in 3.2.1. However, when the length of video clips is greater than one, the bipartite graph constructed can no longer represent the actual situation.

Example 7.1 Suppose that a query has five video clips which have a different length. Figure 7.1 shows the length of each video clip. If (1, A) is chosen, then

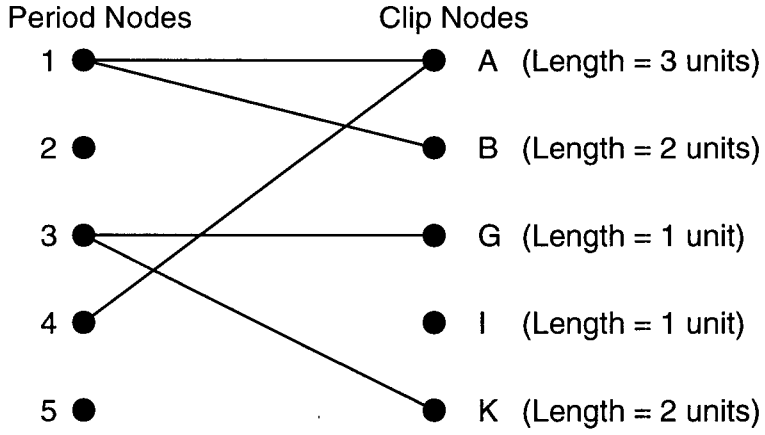

Figure 7.1: Insufficiency of Bipartite Matching for Variable Length Video Clips

the edges (1, B), (3, G) and (3, K) are invalid because the video clip A requires three periods. The bipartite graph fails to show this relationship. □

Example 7.2 Suppose that supplementary edges are added to Figure 7.1 which is shown in dash lines. Although the supplementary edges show the

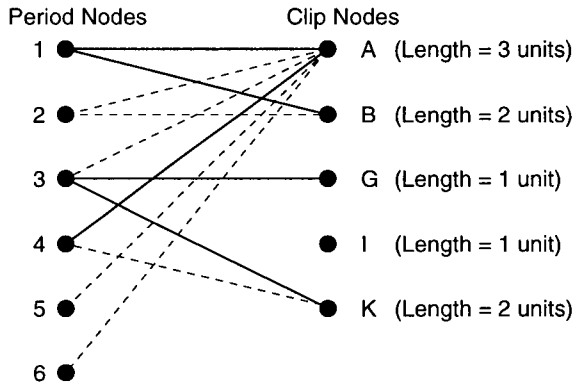

Figure 7.2: Insufficiency of Bipartite Matching for Variable Length Video Clips

information that clip A is related to periods 1, 2 and 3, the bipartite matching only deals with one edge per each node. There are no way to represent the information that they have to be chosen together. For example, if the edge (1, A) is chosen, the supplementary edges, (1, B) and (1, C), have to be chosen together. In addition, the supplementary edge, for example, (2, A), cannot be chosen alone without choosing (1, A) and (3, A) because it is only a pseudo-edge. □

In general, the bipartite matching only deals with one choice per node. It cannot represent the relationship that if an edge is chosen, there are some edges which cannot be chosen. Nor it cannot represent the case to choose a set of node associated with only one edge. Therefore, a new model will be discussed in the following section to find out the maximum number of piggybacked periods.

7.2 Finding the Maximum Number of Piggybacked Periods

To find a feasible display sequence with the maximum number of piggybacked periods requires two steps. First, a set of the piggybacked clips which has the maximum number of periods should be identified. Second, with the piggybacked periods, a feasible display sequence should be built which will be discussed in Section 7.3.

This section begins with the description of how to find the maximum number of piggybacked periods by iterating all edge combinations. It then further refines the iteration method such that the total number of iterations can be reduced. Based on Example 7.1, the total number of iterations can be reduced from 32 iterations to 6 iterations.

7.2.1 Iterating All Edge Combinations

One way to find a display sequence with maximum piggybacked periods is to iterate all edge combinations in the bipartite graph. Although the computational time is $\mathcal{O}(2^{|E|})$ where $|E|$ is the total number of edges in the bipartite graph, this strategy is still useful as long as the actual computational time is not significant. In Example 7.1, because the total number of edges is 5, the total number of iterations required is $2^5 = 32$.

7.2.2 Iterating All Possible Video Clip Combinations

Since each video clip can only be chosen once, some of the above combinations are irrelevant. For example, if there are three edges connected to the same video clip, the combination of any two of these edges is invalid because the video clip can only be displayed in a period. Therefore, the computational time of iterating only the possible combination is better than $\mathcal{O}(2^{|E|})$. Assume that there are n periods with e total number of edges and m number of video clips. Let the total number of edges connected to each video clip $c_1, \ldots, c_m$ be $e_1, \ldots, e_m$. Then, the possible combination for clip c_i is $e_i + 1$ because c_i may not be chosen at all for the final matching. Therefore, the computational time of iterating the only possible video clip combinations is $\mathcal{O}(\prod_{i=1}^{m}(e_i + 1))$. The worst case is $\mathcal{O}((\frac{e}{m} + 1)^m)$ when $e_1, \ldots, e_m = \frac{e}{m}$. In Example 7.1, the total number of edges connected to clips A, B, G, K are 2, 1, 1, and 1 respectively. Therefore, the total number of iterations is equal to $3 \times 2 \times 2 \times 2 = 24$. As all the edge combinations removed from the iteration are invalid edge combinations, the optimal edge combinations is not discarded.

7.2.3 Iterating All Possible Edge Combinations

In Example 7.1, when the edge (1, A) is chosen, three edges, (1, B), (3, G) and (3, K) become invalid because periods 1, 2 and 3 are now allocated for displaying video clip A. Instead of iterating all combinations of edges per

video clip, only the possible edge combinations are tested. This technique can prune away some of the invalid combinations across different clips because each period in the display sequence can only assigned to a video clip. Thus, hopefully, the computational time can be further reduced. In this case, a feasibility matrix is set up to represent the possible edge combinations.

The *feasibility matrix* shows the possible edge combinations associated with a particular edge. Each row or column in the matrix represents the possible edges which can be chosen together. The entries in the matrix have a value of either 1 or 0. If the value is 1, it means that these two edges can be chosen together. If the value is 0, these two edges cannot be chosen together. The matrix itself is a symmetric matrix as shown in Example 7.3.

Example 7.3 The feasibility matrix of the Example 7.1 is shown as follow:

$$F = \begin{array}{c} \\ 1A \\ 1B \\ 3G \\ 3K \\ 4A \end{array} \begin{array}{c} \begin{array}{ccccc} 1A & 1B & 3G & 3K & 4A \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \end{array}$$

Each row represents the possible edge combinations of the edge. For example, the fifth row in the feasibility matrix shows the possible edge combinations of the edge (4, A). The edge (4, A) can be chosen with either (1, B) or (3, G). The edge (4, A) cannot be chosen with the edge (1, A) because the video clip A can only be displayed at either period 1 or 4. The edge (4, A) also cannot be

chosen with the edge (3, K) because the video clip K requires period 3 and 4 to display. □

The feasibility matrix only represents the possible combinations of two edges. But, it also shows all possible combinations of edges using the logical AND of two rows instead of the transitive closure of the matrix. The result obtained by doing a logical AND of two rows represents the possible edge combinations with the two edges. In Example 7.3, the result of logical AND the (1, A) and (1, B) row is [0 0 0 0 0] which means that (1, A) and (1, B) cannot be chosen together. However, the result of logical AND the (3, G) row and (4, A) row is [0 1 1 0 1]. Besides (3, G) and (4, A) columns have a value of one, the (1, B) column also has a value of one which means that (1, B), (3, G) and (4, A) can be chosen together.

The possible edge combinations can be obtained by iterating from the first row to the last row in the feasibility matrix. As the edge combination is a set of edges, it is not necessary to count the edges which has already been considered.

Example 7.4 Considering the feasibility matrix in Example 7.1, the first row of the matrix is (1, A). There is no other edge which can be selected with (1, A). Therefore, the first edge combination is only (1, A). When the second row of the matrix is considered, (1, B) is one of the edge combinations. As (1, B) can be selected together with (3, G), (3, K) or (4, A), there are three more edge combinations which are {(1, B), (3, G)}, {(1, B), (3, K)}, and {(1, B), (4, A)}. However, the logical AND result of (1, B) and (3, G) is [0 1 1 0

1] which means that there is another additional edge combination. That is {(1, B), (3, G), (4, A)}. When (3, G) is considered, the edge (1, B) does not need to be included as it has been considered when the edge (1, B) is iterated. Therefore, only the edge (4, A) is left which leads to two edge combinations, {(3, G)} and {(3, G), (4, A)}. The process continues until all edges have been considered. □

The optimal edge combinations is not discarded because the edge combinations removed are those which are invalid. Using the feasibility matrix to exclude the infeasible edge combinations can reduce the total number of iterations. In Example 7.1, only 10 iterations is required.

7.2.4 Iterating All Possible Edge Combinations Leading to Maximum Piggybacked Periods

As the goal of iterating all the possible combinations is to find the maximum number of piggybacked periods, there are also cases which can be discarded earlier. This can also reduce the total number of iterations required.

Example 7.5 In Example 7.3, the possible maximum piggybacked periods for each row is shown under the column PM below:

$$F = \begin{array}{c} \\ 1A \\ 1B \\ 3G \\ 3K \\ 4A \end{array} \begin{array}{c} \begin{array}{ccccc} 1A & 1B & 3G & 3K & 4A \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} PM \\ \\ 3 \\ 8 \\ 6 \\ 4 \\ 6 \end{array}$$

Example 7.4 shows an order of iterating different edge combinations. In this example, the same order is being considered. When the edge (1, A) has been chosen, the total number of piggybacked periods is three. When the maximum possible piggybacked periods of the second row is greater than the edge combination (1, A), there is a possibility to obtain an edge combination with higher number of piggybacked periods. Therefore, the iteration of the row (1, B) is continued. The total number of piggybacked periods obtained in iterating the row (1, B) is six which is generated by the edge combination {(1, B), (3, G), (4, A)}. As the maximum possible piggybacked periods of the following rows are less than or equal to six, no more iteration is required because no edge combination in the following rows will generate a higher number of piggybacked periods. Therefore, the iteration can be terminated. □

As the edge combinations removed by comparing the possible maximum cannot generate any higher number of possible piggybacked periods, the optimal edge combinations are not pruned away. By removing the edge com-

binations which cannot obtain the maximum number of piggybacked periods, further iteratioins can be reduced. In Example 7.5, only six iterations are required.

7.3 Building the Final Display Sequence

When a list of piggybacked periods and clips is generated, the unassigned periods are required to be filled up to complete the final display sequence. However, this time the task is more complicated than for constant length video clips because some video clips may not be able to fill in the *holes*, which are defined as the contiguous unassigned periods after filling in all the piggybacked periods.

Example 7.6 There are six video clips, A, B, C, D, E, and F which have lengths of 2, 1, 3, 2, 1, and 3 respectively. The video clips A and B can be piggybacked at period 5 and period 12 respectively. There are two holes which are from period 1 to period 4 and from period 7 to period 11. The partly filled display sequence is shown as follows:

Period:	1	2	3	4	5	6	7	8	9	10	11	12
Video Clip:					A	A						B

The unassigned clips cannot be put in the display sequence randomly as described in Section 3.2.3 because some of the choices may violate the non-overlapping and continuity conditions stated in Section 1.1. In Example 7.6,

the video clips D and E cannot be displayed at period 2 and at period 4 respectively because there is not enough space to fit in both clip C and clip F from the period 7 to period 11. Therefore, this is more complicated than building a display sequence for constant length video clips. □

To assign the remaining clips to the unassigned periods for variable length video clips, the longest video clip is chosen to fit into the holes first. The hole, which is large enough to fit the longest clip, can fit shorter clips but not vice versa. Therefore, it is important to give higher priority for longer video clips. However, it is also important to keep the holes as large as possible to serve other longer video clips. Therefore, the remaining video clips are assigned in descending order to the ascending order of the holes. In that case, it can generate a feasible display sequence if there is any.

Example 7.7 In Example 7.6, there are four unassigned clips, namely C, D, E and F. The longest clips are C and F. There are two separate holes as mentioned in Example 7.6. One hole has a length of four periods and another one has a length of five periods. In order to best fit the longest clip, say clip C, the hole of four periods should be used. Therefore, clip C should begin at period 1. After clip C has been assigned, the display sequence still has two holes; one of them is five periods in length and another one is only one period in length. To best fit the second longest clip which is clip F, the periods 7, 8 and 9 are assigned to clip F. There are still two holes in the display sequence. The assignment continues on until all the clips have been assigned. The steps of filling the display sequence are shown in Table 7.1. □

Period	1	2	3	4	5	6	7	8	9	10	11	12
Step 1					A	A						B
Step 2	C	C	C		A	A						B
Step 3	C	C	C		A	A	F	F	F			B
Step 4	C	C	C		A	A	F	F	F	D	D	B
Step 5	C	C	C	E	A	A	F	F	F	D	D	B

Table 7.1: Steps to Fill in the Holes

7.4 MaxPP Algorithms

Section 7.2 describes how to find the maximum number of piggybacked periods by iterating all the possible edge combinations. Section 7.3 shows how to generate a final display sequence. This section describes the MaxPP algorithm in detail.

Algorithm 7.1 (MaxPP Algorithm with No-Retry)

1. All new queries are appended to a queue.

2. Find the candidate set of each period for the query at the head of the queue.

3. Check the candidate set in each period. If the candidate set is empty and all the buffer spaces have been allocated in that period, the query is put back to the head of the queue. The algorithm will be invoked again after the last period which the candidate set is empty and all the buffer spaces have been allocated.

4. Construct the feasibility matrix as described in Section 7.2.3.

5. Iterate all feasible choices of edge combinations to find the maximum number of piggybacked periods as described in Section 7.2.

6. Build the final display sequence as described in Section 7.3. If the construction of the final display sequence fails, the query will be put back to the queue and the algorithm will be invoked in the next period.

7. Check the display sequence to see whether it will exceed the maximum buffer space allowed in each period. If any one period exceeds the buffer space limit, the query will be put back to the head of the queue. The algorithm will be invoked again in the following period. Otherwise, the query will be activated and buffer space will be allocated for this query whenever the buffer space is required.

□

The MaxPP (No-Retry) algorithm generates only one feasible display sequence to test for the admission of the query. However, there are many possible alternatives generated when the algorithm iterates all possible edge combinations. The MaxPP (Retry) algorithm is developed in the hope that other alternatives may be able to generate a feasible display sequence which allows the query to be admitted into the system.

Algorithm 7.2 (MaxPP Algorithm with Retry)

The MaxPP (Retry) algorithm is similar with the MaxPP (No-Retry) algorithm except that it saves all the inspected edge combinations during the

iteration in Step 5 and it checks other possible display sequence using the alternative edge combinations saved when it fails in Step 7. These two steps are rewritten as follow:

5′. Iterate all feasible choices of edge combinations to find the maximum number of piggybacked periods. During each iteration, all the inspected edge combinations are stored in a list.

7′. Check the display sequence to see whether it will exceed the maximum buffer space allowed in each period. If any one period exceeds the buffer space limit, repeat step 6 with another edge combination chosen in the stored list in step 5′ with higher number of piggybacked periods until the stored list is exhausted. If no display sequence can be used, then the query will be put back to the head of the queue. The algorithm will be invoked again in the following period. Otherwise, the query will be activated and buffer space will be allocated for this query whenever it requires.

□

Example 7.8 This example is used to demonstrate how Step 7′ works. Consider the Example 7.5 with the six iterations which are shown in Table 7.2. If the edge combinations {(1, B), (3, G), (4, A)} cannot formulate a feasible display sequence, the edge combination {(1, B), (4, A)} will be used to formulate an alternative display sequence even though it has less number of piggybacked periods. Each edge combination is tested in the descending order of the total

Edge Combinations	Total Piggybacked Periods
{(1, A)}	3
{(1, B)}	2
{(1, B), (3, G)}	3
{(1, B), (3, G), (4, A)}	6
{(1, B), (3, K)}	4
{(1, B), (4, A)}	5

Table 7.2: Possible Edge Combinations

number of piggybacked periods. If there is no feasible display sequence, the query is waited. Otherwise, the query will be activated. □

7.5 Experimental Result

An experiment has been carried out to investigate the effectiveness of the MaxPP algorithms. The experiment runs for 5 trials with 5 different seeds with the random number generator. The length of video clips ranges from 1 to 3 units in a random distribution. The maximum available buffer is 12 units. Queries request 5 to 15 video clips from a random distribution. The total number of video clips available ranges from 75 to 1000. The result is shown in Figure 7.3. In general, the throughput of using MaxPP is over 20% when the total number of video clips available is 150. When the total number of video clips available is 75, the MaxPP (No-Retry) algorithm only has over 30% increase in throughput while the MaxPP (Retry) algorithm has more than 35% increase in throughput. As the total number of video clips available

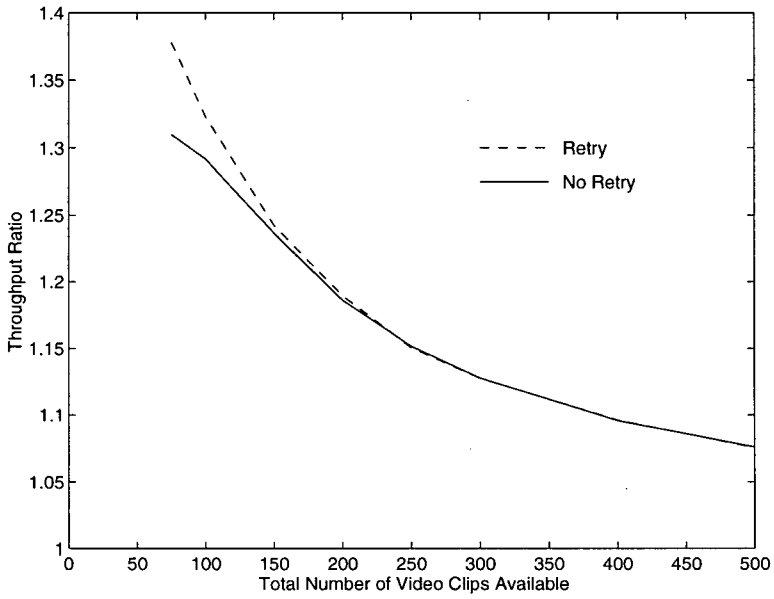

Figure 7.3: Throughput Ratio for Variable Length Video Clips

increases, the difference between these two algorithms is negligible.

The computational time of the MaxPP algorithm in an experiment is about 24 milliseconds per query depending on the total number of video clips available.

7.6 Result Analysis

The throughput improvement for the variable length video clips is not as significant as the one for constant length video clips. One of the reasons may be the change of the maximum number of clips chosen per query. Due to the implementation constraint, the maximum number of clips chosen per query is reduced from 20 to 15. This leads to the reduction of the total number

of possible piggybacked periods. Therefore, the algorithm seems to be less effective than applying to constant length video clips. Another reason is the complication of generating a feasible display sequence. Even though a list of piggybacked periods is found, a feasible display sequence may be unable to formulate. Therefore, the performance of the algorithm also decreases.

The major difference between the MaxPP (No-Retry) algorithm and the MaxPP (Retry) algorithm is the retry mechanism. When one display sequence fails to be admitted into the system, other possible display sequences are generated until none can be admitted. When the total number of video clips available is small, many alternative display sequences can be generated because of more edge combinations. This increases the possibility of admitting the query. As the number of video clips available increases, the number of edge combinations decreases. The possibility of admitting the query decreases. Thus, the effectiveness of the retry mechanism decreases.

The computational time per query is in the order of tens of milliseconds depending on the total number of video clip available. With this small amount of processing time, the MaxPP algorithm is useful to improve the throughput of the multimedia server.

Chapter 8

Conclusion

To improve the performance of a multimedia system which serves multi-clip queries, the "piggybacking" technique is used. This technique re-arranges the display sequence of flexible or prefix order queries. In this thesis, three different algorithms are proposed to improve the performance of the system.

The MaxPVC algorithm is developed to generate a display sequence for multi-clip queries with constant buffer and constant length video clips. The display sequence obtained has the maximum number of piggybacked video clips such that less buffer resource is required to deliver the queries. The MaxPVC-Bipartite theorem shows that the maximum number of piggybacked video clips can be obtained by running the bipartite graph algorithm.

However, when the buffer requirement is considered, the MaxPVC seems to be insufficient. The MaxIBS algorithm is developed to generate a display sequence for multi-clip queries with constant buffer and constant length video

clips. The display sequence generated maximizes the impact on buffer space in each period. The MaxIBS-Maximum Weight Bipartite theorem shows that the piggybacked video clips can be obtained by running the maximum weight bipartite graph algorithm. In addition, both algorithms can be used to generate a display sequence for multi-clip queries with variable buffer with some modifications.

The MaxPP algorithm is developed to generate display sequences for multi-clip queries with variable length. Instead of iterating all possible combinations to find a display sequence which piggybacks the maximum number of periods, it discards many infeasible combinations in order to reduce the number of iterations. Furthermore, it also needs to put the unassigned clips into the display sequence in an intelligent way such that a feasible display sequence can be obtained.

Finally, the MaxPVC, MaxIBS and MaxPP algorithms are very useful in increasing the throughput of a multimedia system as shown in the experimental results. The throughput of the system increases significantly in using the MaxPVC, MaxIBS and MaxPP algorithms. Experimental results show that MaxPVC algorithm and MaxIBS algorithm can increase the throughput of a system from over 350% to 20% when the total number of available video clips is from 50 to 450. The MaxPP algorithm increases the throughput of the system by 20% to 30% depending on the total number of available video clips. Although there are significant increases in the throughput of the system, the computational time of these algorithms is negligible.

8.1 Future Work

To further improve the performance of a multimedia system, further investigation in the following areas may have a positive outcome.

1. All the display sequences of queries are assumed to be fixed once the queries have been admitted to the system. However, as the users do not care about the order of the display sequence, the display sequence can be changed even after it has been admitted into the system. In this case, when a new query arrives, all the display sequences of active queries can also be changed in such a way that it benefits the system the most. The complication of dynamically changing the display sequence is to maintain the non-overlapping and continuity conditions for each query.

2. As all queries are admitted based on first come first serve discipline, the earlier the query can be admitted into the system, the better the throughput is. Since all queries are lined up in a queue, if the information of several queries can be obtained and display sequences can be generated together, this may increase the throughput of the system because there may be a possibility that the second query in the queue can be accepted to the system while the first query cannot. In this case, the complication of this "pre-planning" is to maintain the fairness for every query.

3. All queries are considered for admission when they arrive. Although a query can be admitted at period 1, it may generate a better display

sequence at period 2 instead of period 1 such that the number of piggy-backed video clips can be increased. Therefore, the delay of admission may actually share more buffer resources and hopefully, may increase the overall throughput of the system.

Bibliography

[AOG92] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transaction on Computer Systems*, 10(4), November 1992.

[Ber57] C. Berge. Two Theorems in Graph Theory. In *Proceedings of the National Academy of Sciences*, volume 43, pages 842 –844, 1957.

[BGMJ94] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, and Xiangyu Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 79–90, Minneapolis, Minnesota, May 1994.

[CKY94] Mon-Song Chen, Dilip D. Kandlur, and Philip S. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. *ACM SIGMOD*, May 1994.

[CLR90] Thomas H. Cormen, Charles Eric Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[CY89] D. Cornell and P. Yu. Integration of Buffer Management and Query Optimization in Relational Database Environment. In *Proc. 15th International Conference on Very Large Data Bases*, 1989.

[FNS95] Christos Faloutsos, Raymond Ng, and Timos Sellis. Flexible and Adaptable Buffer Management Techniques for Database Management Systems. *IEEE Transactions on Computers*, 44(4):546–560, April 1995.

[FR94] Craig Federighi and Lawrence A. Rowe. A Distributed Hierarchical Storage Manager for a Video-On-Demand System. In *Storage and*

Retrieval for Image and Video Databases, San Jose, CA, February 1994. Society of Photo-optical Instrumentation Engineers.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[GBT94] Simon Gibbs, Christian Breiteneder, and Dennis Tsichritzis. Data Modeling of Time-Based Media. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 91–102, Minneapolis, Minnesota, May 1994.

[GC92] Jim Gemmell and Stavros Christodoulakis. Principles of Delay-Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[Gem93] D. James Gemmell. Multimedia Network File Servers: Multi-channel Delay Sensitive Data Retrieval. *ACM Multimedia*, June 1993.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Company, New York, 1979.

[GLM95] Leana Golubchik, John Lui, and Richard Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. In *ACM Sigmetrics/Performance '95 Conference*, pages 25 – 36, May 1995.

[GR93] Shahram Ghandeharizadeh and Luis Ramos. Continuous Retrieval of Multimedia Data Using Parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, August 1993.

[GVKR95] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, and P. Venkat Rangan. Multimedia Storage Servers: A Tutorial and Survey. *IEEE Computer*, 28(5):40–49, May 1995.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal of Computing*, 2(4):225–231, December 1973.

[LL95] S. W. Lau and John Lui. A Novel Video-On-Demand Storage Architecture for Supporting Constant Frame Rate with Variable bit Rate Retrieval. In *International Conference on Operating System Support for Digital Audio and Video*, April 1995.

[LLW95] S. W. Lau, John Lui, and P. C. Wong. A Cost-effective Near-line Storage Server for Multimedia System. In *11th International Conference on Data Engineering*, March 1995.

[LS93] P. Lougher and D. Shepherd. The Design of a Storage Server for Continuous Media. *The Computer Journal*, 36(1), 1993.

[MN95] D. Makaroff and R. Ng. Schemes for Implementing Buffer Sharing in Continuous-Media Systems. *Information Systems - Special Issue on Multimedia Database Systems*, 20(6):445 – 464, 1995.

[MS91] Bernard M. E. Moret and Henry D. Shapiro. *Algorithms from P to NP*, volume 1. The Benjamin/Cummings Publishing Company, Inc., 1991.

[NMH96] Gerald W. Neufeld, Dwight J. Makaroff, and Norman C. Hutchinson. Design of a Variable Bit-Rate Continuous Media Server for a n ATM Network. In *IS&T/SPIE Multimedia Computing and Networking*, San Jose, CA, January 1996.

[NU] Stefan Näher and Christian Uhrig. *The LEDA User Manual Version R 3.3.*

[NY96] Raymond T. Ng and Jinhai Yang. An Analysis of Buffer Sharing and Prefetching Techniques for Multimedia Systems. *ACM-Springer Journal of Multimedia Systems*, 4(2):55 – 69, April 1996.

[RBE94] Lawrence A. Rowe, John S. Boreczky, and Charles A. Eads. Indexes for User Access to Large Video Databases. In *Storage and Retrieval for Image and Video Databases*, San Jose, CA, February 1994. Society of Photo-optical Instrumentation Engineers.

[RS91] P. Venkat Rangan and Daniel C. Swinehart. Software Architecture for Integration of Video Services in the Etherphone System.

IEEE Journal on Selected Areas in Communications, 9(9):1395–1404, December 1991.

[RV91] P. Venkat Rangan and Harrick M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the 13th ACM Symposium on Operation Systems Principles (SOSP '91)*, pages 69–79, Monterey, California, October 1991.

[RV93] P. V. Rangan and H. M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *Transactions on Knowledge and Data Engineering*, August 1993.

[RW93] A. L. Narasimha Reddy and Jim Wyllie. Disk Scheduling in a multimedia I/O system. *ACM Multimedia*, June 1993.

[SS82] G. Sacca and M. Schkolnick. A Mechanism for Managing the Buffer Pool in a Relational Database System using the Hot Set Model. In *Proc. 8th International Conference on Very Large Data Bases*, 1982.

[Vin94] Harrick M. Vin. Multimedia System Architecture. In Stephen F. Lundstrom, editor, *Defining the Global Information Infrastructure: Infrastructure, Systems, and Services*, pages 287–296. SPIE Press, November 1994.

[VRG95] Harrick M. Vin, Sriram S. Rao, and Pawan Goyal. Optimizing the Placement of Multimedia Objects on Disk Arrays. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '95)*, pages 158–165, Washington, D. C., May 1995.

[Yao77] S. Yao. Approximating Block Accesses in Database Organizations. *Communications of the ACM*, 20(4), 1977.