ON-DEMAND DATA BROADCASTING

A Thesis

by

KANNAN KOTHANDARAMAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 1998

Major Subject: Computer Science

ON-DEMAND DATA BROADCASTING

A Thesis

by

KANNAN KOTHANDARAMAN

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Nitin H. Vaidya
(Chair of Committee)

Riccardo Bettati
(Member)

A. L. Narasimha Reddy
(Member)

Wei Zhao
(Head of Department)

August 1998

Major Subject: Computer Science

ABSTRACT

On-Demand Data Broadcasting. (August 1998)
Kannan Kothandaraman, B.S., Louisiana State University
Chair of Advisory Committee: Dr. Nitin H. Vaidya

This thesis considers environments wherein the bandwidth available to a server is insufficient to serve the clients one at a time. In such environments, broadcasting data to all clients is efficient. This thesis deals with issues related to on-demand data broadcasting. We look at the problem of data broadcasting in an environment where clients make explicit requests to the server. The server broadcasts requested data items to all the clients, including those who have not requested the item. This thesis evaluates two new broadcast scheduling algorithms for such an on-demand model with the objective of minimizing the waiting time at clients. A new caching scheme for clients is also proposed. This caching scheme uses information from the server to make caching decisions. A hierarchical model for data broadcasting, using proxy servers between the clients and the main server, is also evaluated. Finally, we address the issue of scheduling broadcasts such that the variance of the waiting time is reduced. The proposed algorithm tries to trade-off the variance with the mean access time.

To my parents and my sister

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Vaidya for all his help, guidance and understanding. He introduced me to this topic and has been a great mentor ever since I met him before coming here. My research with him and his mobile computing class have greatly enhanced my knowledge and helped land my dream job.

I would like to thank Dr. Bettati for being on my committee. I would like to thank Dr. Reddy for his time and consideration.

I would like to thank Mr. Peterson and Barbara for all that they have done. I am thankful to Dr. Vaidya, the Computer Science department and the library for funding me.

I would like to thank all my friends here and at LSU for the great times.

I would like to thank my parents and my sister for all the support they have given me. Their constant encouragement, and belief in my abilities has been invaluable. I would not be where I am without them.

TABLE OF CONTENTS

CHAPTER		Page
I	INTRODUCTION	1
II	TERMS AND NOTATIONS	5
III	RELATED WORK	7
	A. Broadcast Scheduling	7
	B. Caching Schemes	9
	C. Indexing the Broadcast	10
IV	BROADCAST SCHEDULING	12
	A. SR Scheduling Algorithm	12
	B. Total Wait Time Scheduling Algorithm	14
V	CACHING SCHEME	17
	A. Prefix Cache : A Server Initiated Caching Policy	18
	B. Generalization of the Prefix Caching Algorithm	22
VI	HIERARCHICAL BROADCAST MODEL	25
VII	REDUCING THE VARIANCE	29
	A. Variance	29
	B. Reducing the Variance With the SR Scheduling Algorithm	30
VIII	PERFORMANCE EVALUATION	32
	A. Performance Evaluation of Scheduling Algorithms	33
	B. Performance Evaluation of Prefix Caching Scheme	36
	1. The Mature Cache Problem	48
	C. Performance Evaluation of Hierarchical Model	49
	D. Performance Evaluation of the α-SR Scheduling Algorithm	51

CHAPTER Page

IX FUTURE WORK AND CONCLUSIONS 53
A. Future Work 53
B. Conclusions 54

REFERENCES 56

APPENDIX A 60

VITA 66

LIST OF TABLES

TABLE		Page
I	Standard Deviation of α-SR and RxW Scheduling Algorithms	51
II	Overall Mean Access Time of α-SR and RxW Scheduling Algorithms	52
III	Data for Fig. 8 in Tabular Form	60
IV	Data for Fig. 9 in Tabular Form	60
V	Data for Fig. 10 in Tabular Form	61
VI	Data for Fig. 11 in Tabular Form	61
VII	Data for Fig. 12 in Tabular Form	61
VIII	Data for Fig. 13 in Tabular Form	62
IX	Data for Fig. 14 in Tabular Form	62
X	Data for Fig. 15 in Tabular Form	62
XI	Data for Fig. 20 in Tabular Form	63
XII	Data for Fig. 16 in Tabular Form	63
XIII	Data for Fig. 17 in Tabular Form	64
XIV	Data for Fig. 18 in Tabular Form	64
XV	Data for Fig. 19 in Tabular Form	65

LIST OF FIGURES

FIGURE		Page
1	Client-Server Model	2
2	Broadcast Model	3
3	Sample Broadcast System	14
4	100-Prefix Array	19
5	60-Prefix Array	19
6	Hierarchical Broadcast Model	26
7	Data Delivery Options for a Hierarchical Model	27
8	Overall Mean Access Time Versus Access Skew for Items of Unequal Lengths Obtained by Simulation of Scheduling Algorithms Given in Chapter III	34
9	Overall Mean Access Time Versus Access Skew for Items of Equal Lengths Obtained by Simulation of Scheduling Algorithms Given in Chapter III	35
10	Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV	38
11	Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV	39
12	Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV	40
13	Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV	41
14	Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV	42

FIGURE Page

15 Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV 43

16 Cache Hit Ratios Versus Cache Size Obtained by Simulation of Caching Policy Given in Chapter IV 44

17 Overall Mean Access Time Versus Cache Size Obtained by Simulation of Caching Policy Given in Chapter IV 45

18 Cache Hit Ratios Versus Prefix Percentage Obtained by Simulation of Caching Policy Given in Chapter IV 46

19 Overall Mean Access Time Versus Prefix Percentage Obtained by Simulation of Caching Policy Given in Chapter IV 47

20 Overall Mean Access Time Versus Access Skew Obtained by Simulation of Hierarchical Broadcast Model Given in Chapter V 50

CHAPTER I

INTRODUCTION

The recent explosion of interest in the Internet has created a great demand on scarce network resources. It has also given rise to a number of interesting and challenging network problems. In the future, tens of millions of people will carry a portable computer with a wireless connection to a worldwide information network [1]. Data management in this paradigm poses many challenging problems. As the number of users increases exponentially, new ways of delivering data to them have to be found.

Many networked applications currently use the client-server model. This model is illustrated in Fig. 1. In the network, some computers are designated as service providers, or *servers*. There are other computers that request services or data from these computers. These make up the *clients*. In the client-server approach, a client requests data from a server and the server responds individually to each client. With a dramatic increase in the number of clients requesting data from the server, the performance degrades.

We consider environments wherein the bandwidth available to the server is insufficient to serve the clients one at a time. In such environments, broadcasting data to all the clients is efficient. This model is illustrated in Fig. 2. Data broadcasting is ideally suited for several applications, such as traffic information dissemination [2], video on-demand [3], airline information, weather information, stock quotes, and emergency services information [4]. Pointcast [5] and Airmedia [6] are two of the many commercial companies that are using data broadcasting as their data delivery model. Recently, data broadcasting is being used as the data delivery tool for

---

The journal model is *IEEE Transactions on Automatic Control.*

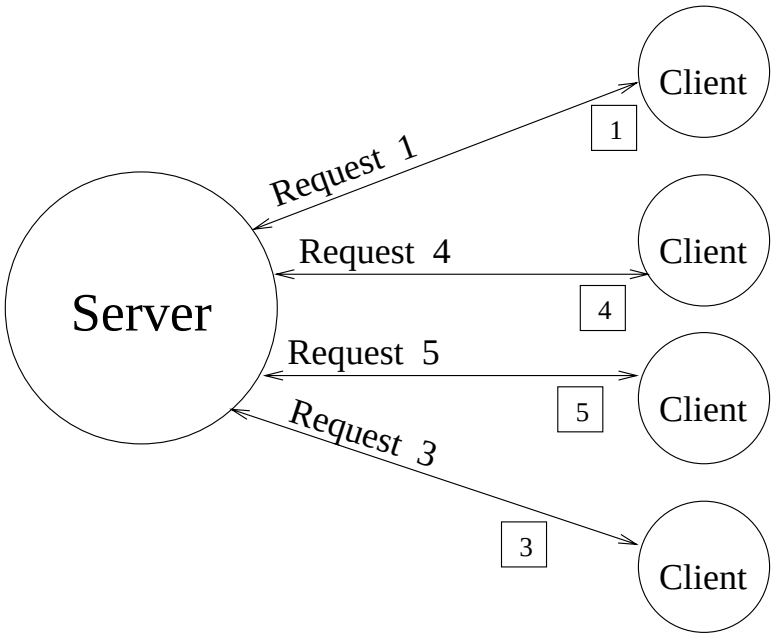

Fig. 1. Client-Server Model

message-oriented middleware [7].

In this thesis, we assume that all information is in the form of data items. The first issue that this thesis deals with is scheduling data broadcasts. Scheduling algorithms for data broadcasting determine which item the server broadcasts at a particular instance of time. There has been a lot of research [4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] in broadcast scheduling. Most of these schemes assume that the client plays no part in the scheduling decisions made by the server. In some cases [10, 17, 20], including this thesis, the schedule is based on explicit requests made by the clients. This is the on-demand or pull-based data broadcast model. *Access time* can be defined as the time between a client request and the time when it has finished receiving the requested item. We present algorithms that attempt to minimize the

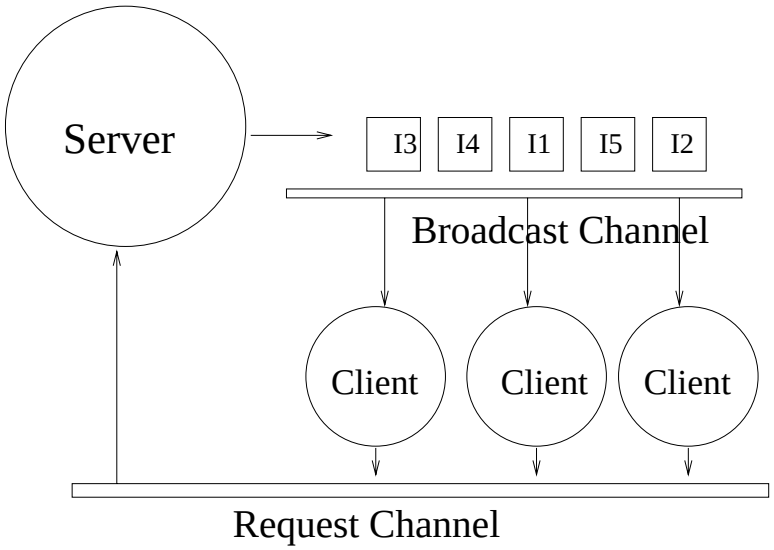

Fig. 2. Broadcast Model

access time.

The second issue that this thesis deals with is caching at the client. Caching is a technique used to store items in the client's local memory to improve the responsiveness of an application [8]. Caching schemes have to work with the scheduling algorithms to further reduce the access time for the client. As the client cache becomes full, a policy is needed to decide which item(s) to delete, when subsequent items cannot fit in the cache. Several caching policies [8, 11, 21, 22] for data broadcasting have been proposed. This thesis proposes a caching policy for an on-demand broadcast model. The caching decisions made by the clients are completely based upon information received from the server.

Since the server is one of the main bottlenecks, a model where the load on the server is distributed to other servers is bound to perform well. Here clients make requests to a *proxy server* instead of the main server. Only requests that cannot

be satisfied by the proxy server are sent to the main server. We evaluate such a hierarchical broadcast model.

Jiang and Vaidya [16] have proposed scheduling algorithms that aim to minimize the variance of the access times in a push-based broadcast environment. We look at how we can try to balance the performance of a pull-based system, so that the variance of the access times is reduced.

The rest of the thesis is organized as follows. Chapter II describes some of the terms and notations used in this thesis. Chapter III presents some of the related work in this area. Chapter IV describes two new scheduling algorithms for an on-demand broadcast model. Chapter V presents a new caching policy. Chapter VI describes a hierarchical broadcast model. Chapter VII discusses the use of the standard deviation as a performance metric for the scheduling algorithms, and presents a modification to one of the algorithms proposed in Chapter IV. Chapter VIII presents performance evaluation of the scheduling algorithms, the caching polices and the hierarchical model. Chapter IX concludes the thesis.

CHAPTER II

TERMS AND NOTATIONS

In this chapter, some of the terms and notations to be used in the rest of the thesis are introduced.

- The information requested by the client is made up of data items. The length of these data items may not be identical.
- The clients send requests for the data items to the server.
- Clients use a separate channel to send requests to the server. This channel is different from the broadcast channel. Hence, the bandwidth available to the server is not affected by the client requests. This assumption is similar to that made in [8, 20].
- Every item, that is broadcast by the server, is received by all the clients.
- *Scheduling Algorithm* [13]: It is an algorithm that is used by the server to determine which item to broadcast.
- *Caching Scheme*: A caching scheme is used by the clients to determine how best to make use of local cache. In general, the caching scheme tells the clients whether to cache the received item and which current item in cache to replace if space is needed.
- M denotes the total number of items in the server database.
- l_i denotes the length of item i, $1 \leq i \leq$ M. We assume that the length of item i is the time taken to transmit item i, to all the clients.

- *Spacing*: Spacing is the amount of time between successive broadcasts of an item. Intuitively, more popular items should have low spacing and less popular ones, higher. The spacing between consecutive instances of an item may change over time.

- *Demand Probability*: We model popularity of item i using demand probability p_i. An item that is popular has a higher demand probability than a less popular one. Hence if item i, with demand probability p_i, is more popular than item j, with demand probability p_j, then, $p_i > p_j$.

- *Mean Access Time*: Access time is the duration, between the time a client makes a request, and the time it finishes receiving the requested item. Mean access time is simply the average over all the requests made by the clients. It is used as a performance measure throughout this thesis. Other researchers [8, 10, 11, 13, 14, 17, 20] have used this as their performance measure as well.

- *Weight*: Weight of an item is the "cost" associated with *not* transmitting the item. Hence, the item with the highest weight is the one chosen by the scheduling algorithm to be broadcast next, so that the "cost" associated with not sending the item can be minimized. The choice of the right parameters to use, to calculate the weight, is critical.

- R_i, $1 \leq i \leq$ M, is the number of client requests that is pending for item i.

- $(Total\_Wait\_Time)_i$, $1 \leq i \leq$ M, for item i, is the sum of the wait times for all pending requests for item i.

- *Proxy Server*: A Proxy server is one that acts as the intermediary between the client and the main server. Clients send requests to a designated proxy server, instead of the main server.

CHAPTER III

RELATED WORK

This chapter summarizes past work on data broadcasting. Issues in data broadcasting can be classified as

1. Scheduling broadcasts
2. Caching policies for clients
3. Indexing the broadcast

A. Broadcast Scheduling

Most of the research on data broadcasting has been on scheduling the broadcast. Since the amount of time a client waits to receive an item is of vital importance, scheduling is critical to the performance of a broadcast system. As mentioned in Chapter I, minimizing the mean access time has been the primary performance objective in designing scheduling algorithms.

Some of the early research in broadcast scheduling was performed by Ammar and Wong [11, 12, 20]. [20] studied the problem of on-demand broadcast scheduling and proposed the First-Come First-Serve algorithm. Here, the server broadcasts the item with the earliest request arrival time. However, instead of just responding to the client that made the request, the item is sent out to all the clients waiting for this item. This is done by adding subsequent requests for an item, in the same position in the request queue as the first one. However, no consideration is given to the popularity of the items. [23] proposed the Most Request First(MRF) algorithm, which chooses the item with the highest number of pending requests as the next item

to be broadcast. Since no consideration is given to the waiting times, this policy could lead to starvation for certain items.

Acharya et. al. [8, 9, 21] proposed interesting ways of solving the broadcast scheduling problem. They use a combination of server scheduling and client caching, as does this thesis. Their schemes simply divide up the bandwidth based on the item demand probabilities, and determine the schedule a priori. This scheme however cannot be implemented in an on-demand environment, since it assumes that the server has a priori knowledge of client request probabilities. Also, their algorithm can leave *holes* (periods of time for which the channel is unused), since it strives to maintain constant spacing between broadcast instances of an item. They also studied the issue of dividing up the bandwidth between client-server and broadcast based data delivery.

Su and Tassiulas [17] have also studied the problem of broadcasting in an on-demand environment. They propose using $\lambda_i^{-\gamma} R_i$, $1 \leq i \leq$ M, as the item weight, where λ_i is the request arrival rate for item i. They calculate λ_i as λp_i, where λ is the overall request arrival rate. The best mean access time is obtained for $\gamma = 0.5$.

Vaidya and Hameed [18] proposed an O(M)-time algorithm that uses $s_i^2 p_i / l_i$, $1 \leq i \leq M$, as the weight of each item, and transmits an item with the largest weight. [15] also presents two O(logM) algorithms, based on *packet fair queueing*, for single and multiple channel broadcast scheduling. For the O(logM) algorithm, two variables - the earliest start time for the next broadcast and the broadcast after that (C_i) - are maintained for every item. Whenever the server becomes idle, from amongst all items whose next broadcast time is less than or equal to the current time, the item with the lowest C_i is chosen as the next item to be broadcast. This algorithm is also extended to a multi-channel broadcast model. [13] also considers scheduling broadcasts in the presence of transmission errors.

Recently, [16] proposed using the variance of the access times as a performance goal of the scheduling algorithm. They proposed scheduling algorithms that aim to minimize the variance. Their α-algorithm is a variation of the O(M)-time algorithm, from [18], in that it varies α, between 2 and 3, in $s_i^{\alpha} p_i / l_i$. They also proposed a *Variance Optimal Algorithm* that uses

$$\frac{p_i s_i}{l_i}(\frac{2}{3}s_i - \frac{1}{2}\sum_{i=1}^{M} p_i s_i), \qquad 1 \leq i \leq M \tag{3.1}$$

as the weight to make the scheduling decision.

B. Caching Schemes

Some of the proposed schemes for caching are:

- LRU [24]: LRU is the simplest of the schemes used as the cache replacement policy. In response to a cache miss, LRU chooses as the victim, the item which has the lowest last use time. Therefore, every time an item is needed by the client, if it is in cache, the request is served immediately, and the item's last use time is set to current time. In case of a cache miss the item with the lowest last use time is replaced. As evident, LRU is not a broadcast specific scheme and hence does not make use of the knowledge the server has about client demands.

- P [8]: P is a demand-driven caching policy which keeps items with the highest probability of access in the memory. So, in response to a cache-miss, P chooses as the victim, the cache-resident item with the lowest probability of access. P requires perfect knowledge of the probability of access of the items. Therefore, it cannot be used in an on-demand environment.

- PIX [8]: PIX is a cost based heuristic for demand driven caching. In PIX, the cost of replacement of an item already in memory with the newly fetched one is

considered to be the ratio of the access probability of item (P) and its broadcast frequency. The replacement algorithm ejects any item in memory which has the lowest pix value or has value lower than the current item.

- PT [8]: PT is a pre-fetching heuristic. It exploits the dissemination-based nature of the broadcast, which are particularly conducive to the user's prefetching. PT computes the value of an item by taking the product of the access probability of the item (P) with the time (T) that will elapse before that item appears again on the broadcast. This is called the item's pt value. PT finds the item in memory with the lowest pt value and replaces it with the item being broadcast, provided the item (being broadcast) has a higher pt value. Since this scheme requires a priori knowledge of the popularity of the items and the broadcast schedule, it cannot be used in an on-demand environment.

- Ammar's scheme [11]: In this scheme each item in the broadcast has a control information associated with it. The control information in item i is a list of items that are most likely to be requested next by the user. After a request for an item i is satisfied, the client prefetches D most likely items associated with item i, where D is the size of the local memory.

C. Indexing the Broadcast

In mobile computing systems, the amount of time the clients spend listening to the broadcast channel can be crucial. This is because the clients in this case may be low-powered devices, and the more time spent listening to the broadcast, the higher the amount of energy spent.

Viswanathan et. al. [1, 4, 25] have proposed solutions that address this issue. They call the time a client spends listening to the broadcast as the *tuning* time. They

propose indexing the broadcast so that the client knows exactly when to listen to the broadcast to get the required item. Tuning time is not the same as access time (i.e., waiting time). A client can obtain the index information from the server, then decide to do other things in the meanwhile, and finally "tune" in only at the appropriate time. Access time includes the tuning time. They have proposed different indexing schemes that will enable a client to minimize its tuning time, and hence its energy use. The simplest scheme, called (1,m) indexing, simply broadcasts the entire index m times during the broadcast cycle [1]. They calculate the optimum value of m as $\sqrt{N/n}$, where N is the size of the broadcast cycle and n is the size of the index. They also present other distributed indexing schemes that break up the index and minimize replication of information.

[26] also proposes indexing schemes for data broadcasting. They take into consideration the item demand probabilities. They propose constructing index trees such that items with high demand probabilities can be found faster, than those with lower probabilities, in the index.

Note that to index the broadcast, schedules have to be determined a priori.

[1]For cases where the schedule is determined a priori, we can define the broadcast cycle as a period of time that meets a certain criteria. A cycle could be one that contains all the items in the database, or a certain subet of the items, or one that lasts for a specified amount of time, etc.

CHAPTER IV

BROADCAST SCHEDULING

In this chapter we present two on-line algorithms to schedule broadcasts in an on-demand environment. On-line algorithms are those that do not determine the schedule ahead of time. They can adapt to changing conditions and provide the best performance based on the current state. In an on-demand environment, clients make explicit requests for items to the server and the server makes scheduling decisions based on the currently pending requests.

A. SR Scheduling Algorithm

This section describes an on-line algorithm which is used by the server to decide which item to broadcast next, as soon as it becomes idle. The performance goal of the algorithm is to minimize the mean access time for the clients. Every time the server becomes idle, which happens immediately after it has finished broadcasting an item, the server uses this algorithm to decide the next item to broadcast [1]. [18] proposed an algorithm which uses a *decision* rule to determine the next broadcast item. Every time the server needs to make a decision, the value

$$\frac{s_i^2 p_i}{l_i} \qquad 1 \leq i \leq M \tag{4.1}$$

is calculated, where s_i is the time since the last broadcast of item i, p_i the demand probability, and l_i the length of item i. However, [18] assumes that the server has a priori knowledge of the demand probabilities of all items, which is not valid for an on-demand model. Hence, we modify the value in 4.1 to suit our on-demand model,

[1] Alternatively, the algorithm could be called sometime during the current item transmission

so that no a priori knowledge of client behavior is required. Assuming that the rate at which requests are made, λ, is the same for the entire client population, 4.1 is modified as

$$\frac{s_i(s_i p_i \lambda)}{l_i} \qquad 1 \leq i \leq M \tag{4.2}$$

We replace $s_i p_i \lambda$ by the number of requests, R_i, for item i since its last broadcast. Hence, 4.2 is modified as

$$\frac{s_i R_i}{l_i}, \qquad 1 \leq i \leq M \tag{4.3}$$

We use this as the weight of every item in the database at the time the server has to make a scheduling decision. We calculate the weight for all items, and an item with the largest weight is chosen as the next item to be broadcast.

We can now formally specify the algorithm to be used to make scheduling decisions. Let Q denote the current time, and L(i) the last time item i was broadcast. We define the weight of item i, W(i) as

$$W(i) = (Q - L(i))R_i/l_i \qquad 1 \leq i \leq M$$

Note that W(i) is $s_i R_i / l_i$.

SR Scheduling Algorithm

1. Calculate W(i), $1 \leq i \leq M$.

   Let W_{max} be the maximum weight.

2. Choose item j such that W(j) = W_{max}.

3. Set L(j) equal to Q, and broadcast item j.

□

The algorithm is used whenever the server becomes idle.

Example III.1: Consider a 5 item database which is being broadcast to a client population. Let the lengths of the items be $l_1 = l_2 = l_3 = l_4 = l_5 = 5$. Consider

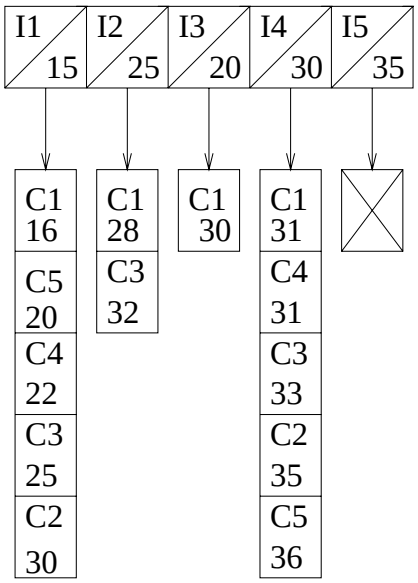

Fig. 3. Sample Broadcast System

the system at time 40, as shown in Fig. 3. The top row shows the items and their last broadcast times. Below each item is a list of pending requests for that item at time 40. The client that made the request and the time of the request is shown. The server has just broadcast item 5 and has to make a scheduling decision. Based on the SR scheduling algorithm, $W_1 = 25$, $W_2 = 6$, $W_3 = 4$, $W_4 = 10$, and $W_5 = 0$. Since I1 has the largest weight, the server broadcasts I1. □

B. Total Wait Time Scheduling Algorithm

In the previous section, a cost based scheduling algorithm was presented. Every time the server has to make a scheduling decision, the value in 4.3 is calculated for all items and the item with the maximum value is broadcast. On closer examination of 4.3, we can discern the fact that the spacing between successive broadcasts of an

item is an estimate of the average time clients have waited for that item since its last broadcast.

In an on-demand model, the server is not under the constraint of having to estimate the average waiting time, since it can calculate the precise value of the average waiting time, by simply looking at the request backlog for every item. Hence, we can modify 4.3, as

$$\frac{t_i R_i}{l_i} \qquad 1 \leq i \leq M \tag{4.4}$$

where t_i is the average waiting time for pending requests for item i. Since the average waiting time for item i is the ratio of the total waiting time for all pending requests and the number of requests, we can rewrite the value in 4.4 as

$$\frac{(Total\_Wait\_Time_i)R_i}{R_i l_i} \qquad 1 \leq i \leq M \tag{4.5}$$

or

$$\frac{Total\_Wait\_Time_i}{l_i} \qquad 1 \leq i \leq M \tag{4.6}$$

We can now formally specify the algorithm.

Total Wait Time Scheduling Algorithm

1. Calculate TWT(i) = $(Total\_Wait\_Time_i)/l_i$, $1 \leq i \leq M$.
   Let TWT_{max} be the maximum weight.

2. Choose item j such that TWT(j) = TWT_{max}, and broadcast item j.

□

The algorithm can be used whenever the server becomes idle.

Example III.2: Consider again, the 5 item database, with $l_1 = l_2 = l_3 = l_4 = l_5 = 5$. The state of the system, at time 40 is shown in Fig. 3. Using the Total Wait Time scheduling algorithm, we obtain $TWT_1 = 87$, $TWT_2 = 30$, $TWT_3 = 10$, TWT_4

$= 34$, and $TWT_5 = 0$. Since I1 has the maximum total waiting time, the server will broadcast I1. □

Performance measurements for the two algorithms are presented in Chapter VIII.

CHAPTER V

CACHING SCHEME

In Chapter IV, we presented two scheduling algorithms that aim to minimize the mean access time for the clients. No part is played by the clients in trying to improve the performance of the system. Caching items, sent by the server, is another way to improve system performance. Here, clients store items sent to them by the server.

A caching policy is needed to decide the best way to make use of the cache available at the client. This is because a client, in most cases, cannot store every item that the server broadcasts. The caching policy is needed to decide:

- Whether the current broadcast item is worthy of being stored in cache at the expense of other item(s).
- Which item(s) is to be removed from cache to make place for the current item.

The caching policy has to make sure that any caching decision that it makes will tend to minimize the mean access time for the client. In this chapter, we propose a cost based caching policy. We assign a weight to every item. Items with a lower weight are the ones chosen to be replaced by items with a higher weight.

Caching policies have to make use of the differences in item popularity (called the *access skew*). A system where all items are equally popular, makes it difficult to improve performance using caching, since all items have the same cost associated with their deletion or addition. Therefore, our caching policy works very well in the presence of high access skew.

Previous caching policies [8, 24] require the clients to store information about the items, such as demand probability, in order to make caching decisions. However, client request patterns can be completely unpredictable. They can vary with time

and hence the information stored at the client may actually prove detrimental to the caching decision. Our approach is meaningful if the request patterns of different clients have similar demand distribution (or if they are correlated), even if they are time-variant. We let the server make, or help make, caching decisions for the clients. In a broadcast system, the server is aware of the current popularity of the items. We can make use of this knowledge at the server to help the clients determine which items to keep in cache.

The caching policy that we propose is used for both demand-driven and prefetch caching. Demand-driven caching is where a client makes a caching decision only in response to a cache miss (hence, an explicit request to the server). Prefetch caching is where a client decides to cache an item, regardless of whether a cache miss occurred (hence, without a request from the client). This way, the server can make use of potential correlation between client requests to judge item popularity and improve the cache effectiveness.

A. Prefix Cache : A Server Initiated Caching Policy

In an on-demand broadcast system, since the server has precise knowledge of client requests, the server can make an estimate of the popularity of the items and convey it to all the clients. Since the server calculates the weight of every item, before every broadcast, the server can precisely gauge the popularity of the items.

We call the information, that the server sends to the clients with every broadcast, the *prefix array*. The prefix array contains the numbers (or any other identification) of the items in the database. Let $Weight_i$ and $Weight_j$ be the weights of items that are in position i and j respectively, in the prefix array. The prefix array is simply an array of items numbers, such that $Weight_i < Weight_j$, if $i < j$. Since the prefix

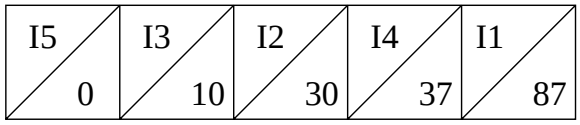

Fig. 4. 100-Prefix Array

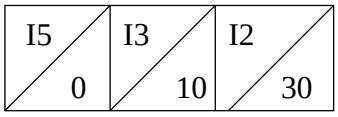

Fig. 5. 60-Prefix Array

array takes up bandwidth, the server might decide to transmit only a fraction of it. The prefix array is updated at every broadcast. If the server sends the entire prefix array, we call it *100-prefix* (or 100% of the prefix array), if it does not send any part of the prefix array, we call it *0-prefix* (or 0% of the prefix array). In general, *x-prefix* means that $x\%$ of the prefix array ($x\%$ of the lowest weights) is sent by the server with each broadcast. The weight of the current broadcast item is always sent in addition to the prefix array.

Consider once again, Fig. 3. Assuming that we are using the *Total Wait Time Algorithm*, the server would transmit the prefix arrays shown in Figures 4 and 5. Fig. 4 gives the prefix array for *100-prefix* and Fig. 5 gives the prefix array for *60-prefix*. Note that the weights are not sent. They are included in the figure only to illustrate the idea.

Effort is made to cache every item that is being broadcast, since it is presumed that this item is the most popular item at the time. Since the prefix array provides

an estimate of the popularity of the items in increasing order (at the time of the broadcast), the clients delete the items, starting with the first one in the prefix array, until the item that is being broadcast can fit in the cache. Note that multiple items in cache may need to be removed to make room for the new item, if the items are not all of the same length.

It is possible that removing cached items, that are also in the prefix array, is not sufficient to make space for the current item. This could happen when the server decides to send less than 100% of the prefix array. To handle such cases, the client stores the following information for every item i that is stored in cache:

- $cache\_weight_i$, the weight of item i in cache.
- $cache\_time_i$, the time when item i was last broadcast or the last time item i in cache was used by the client, whichever is latest.

At every broadcast, all the clients update the *cache_weight* and the *cache_time* of the item being broadcast with the current weight of the item sent by the server, and the current time respectively. For every item, remaining in cache after the above deletion, the ratio of the item's weight and the time since its last broadcast is calculated. The item with the lowest ratio is chosen as the victim, if its *weight* is lower than that of the current item, and deleted. This process is repeated until there is enough space to store the current item, or the weight of the chosen item is higher than that of the current item. In the latter case, the current item is not cached, and the deleted items are restored to cache.

The rationale for this can be explained as follows. We want to delete the item in cache, that has the lowest weight. However, since a client only knows the weight of an item when it is broadcast, we want to make sure that the weight is not very old. Hence, we use the ratio of the weight of the item and the time since it was cached. We

also make a weight comparison to make sure that the deleted item was less popular than the current item (we do not want to store "streak" items).

We now formally specify the Prefix Caching Policy. Let *Pre* be an *x-prefix* array, for some x, $0 \leq x \leq 100$, with the item numbers sorted in increasing order of their weights (obtained from the scheduling algorithm). The caching policy has to choose items that have to be removed from cache to make room for the current item k. We define a few more terms before specifying the algorithm. Let

- N be the number of items in the prefix array.
- *incache(i)* be a boolean such that, *incache(i)* is 1, if item i is in cache, and 0 if it is not.
- *curr_weight* be the weight of the item being broadcast.
- Q be the current time.

Prefix Caching Algorithm

1. Set $cache\_weight_k = curr\_weight$
2. Set $cache\_time_k$ = Q
3. for $i = 1$ to N {

   if Pre_i is in cache, then delete it, set *incache(i) = 0*

   if there is enough space for item k, cache it and *EXIT*

   }
4. Find item j, such that,

   $cache\_weight_j/(Q - cache\_time_j) \leq cache\_weight_i/(Q - cache\_time_i)$,

   $\forall i, j, incache(i) > 0, incache(j) > 0$

5. if $cache\_weight_j \leq cache\_weight_k$ {

   delete item j, set $incache(j) = 0$

   if there is enough space for item k, cache it and $EXIT$

   else go to Step 4

   }

   else {

   restore deleted items

   $item_k$ cannot be cached, $EXIT$

   }

□

The algorithm is used by all the clients, when

1. the server broadcasts an item, and
2. the item being broadcast is not already in cache, and
3. the item cannot fit in cache without any deletions of existing items.

Chapter VIII presents performance results for the Prefix Caching algorithm.

B. Generalization of the Prefix Caching Algorithm

In this section we present a generalization of the Prefix caching algorithm.

Let $costf()$ be a cost function for every data item in cache. Choices for this function are listed later in this section.

Generalized **Prefix Caching Algorithm**

1. Set $cache\_weight_k = curr\_weight$
2. Set $cache\_time_k$ = Q

3. for $i = 1$ to N {

   if Pre_i is in cache, then delete it, set *incache(i) = 0*

   if there is enough space for item k, cache it and *EXIT*

   }

4. Find item j, such that,

   $cache\_weight_j / costf(j) \leq cache\_weight_i / costf(i)$,

   $\forall i, j, incache(i) > 0, incache(j) > 0$

5. if $cache\_weight_j / cache\_time_j \leq cache\_weight_i / last\_broadcast\_time_i$ {

   delete item j, set *incache(j) = 0*

   if there is enough space for item k, cache it and *EXIT*

   else go to Step 4.

   }

   else {

   restore deleted items

   $item_k$ cannot be cached, *EXIT*

   }

□

The comparison in Step 5 is for cases when the demand distribution of the clients is highly time-variant. It is optional, if that is not the case.

There are several choices possible for *costf()* in Step 4 of the generalized algorithm. For instance:

1. The time since the item was last used by the client, similar to LRU [24].

2. The time since the last use of the item by the client or the last broadcast time, whichever is latest (as in our algorithm).

3. The product of the length of the item and the time since the last use of the item by the client or the last broadcast time, whichever is latest. This is based on a caching scheme that was proposed in [27]. The scheme in [27] was proposed for an Internet server, not a broadcast system.

We did not evaluate the algorithm for all the choices of $cost f()$. Only choice 2 was evaluated.

CHAPTER VI

HIERARCHICAL BROADCAST MODEL

In this chapter, we evaluate a *hierarchical* broadcast model. Here, clients send their requests to a *proxy* server, instead of the main server. A proxy server could be located close to the main server, or we can can have proxy servers that are geographically closer to some portion of the client population. We assume that some re-routing mechanism [28, 29] exists, that diverts requests from the clients to the appropriate proxy server. An example of such an hierarchical system is shown in Fig. 6.

In the model that we evaluate, the behavior of the main server is unchanged. Here, only the proxy servers form its client population. It responds to requests only from the proxy servers.

There is no difference in the way the clients behave. Clients simply make their requests to the server. As mentioned earlier, we assume that a re-routing mechanism exists to make sure that this request is delivered to the appropriate proxy server.

All the changes that we have to deal with are in the proxy server, which acts as both a client and a server. When the proxy server receives a request for an item, it first checks its local cache. If the item requested is in its cache, the item is broadcast to that proxy's client population. If not, the proxy server sends a request for this item to the main server. Hence, caching at the proxy server is crucial, and is the main source of performance improvement. We now specify the algorithm, that is used by the proxy server, whenever it has to broadcast an item. Let $sorted\_items$ be an array containing the items numbers, such that $item\_weight[sorted\_items[i]] < item\_weight[sorted\_items[j]]$, if $i < j$, and P be the number of items for which there are pending requests, at the time the algorithm is run.

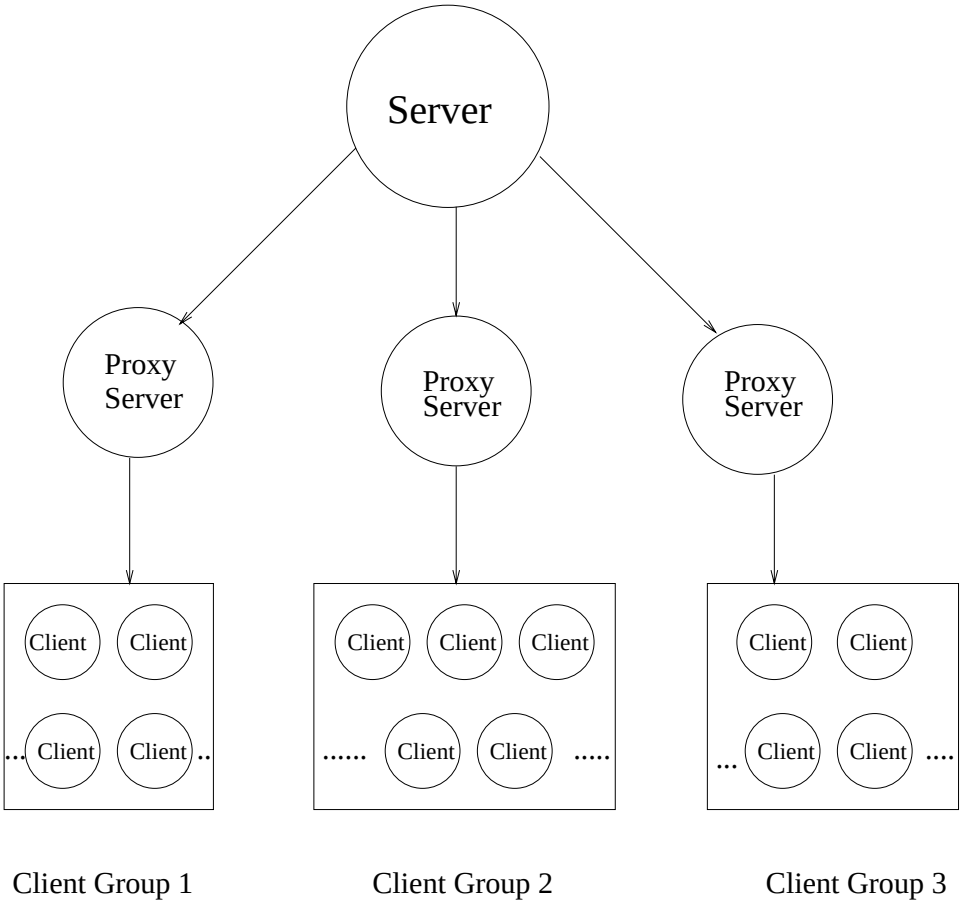

Fig. 6. Hierarchical Broadcast Model

Scheduling Algorithm at a Proxy Server

1. Calculate the weights for all items that have pending requests, for instance using 4.3 or 4.6, and determine the array *sorted_items*.

2. for $i = 1$ to P

   if *sorted_items[i]* is in cache

   Transmit item *sorted_items[i]* to the proxy server's clients

   and *EXIT*.

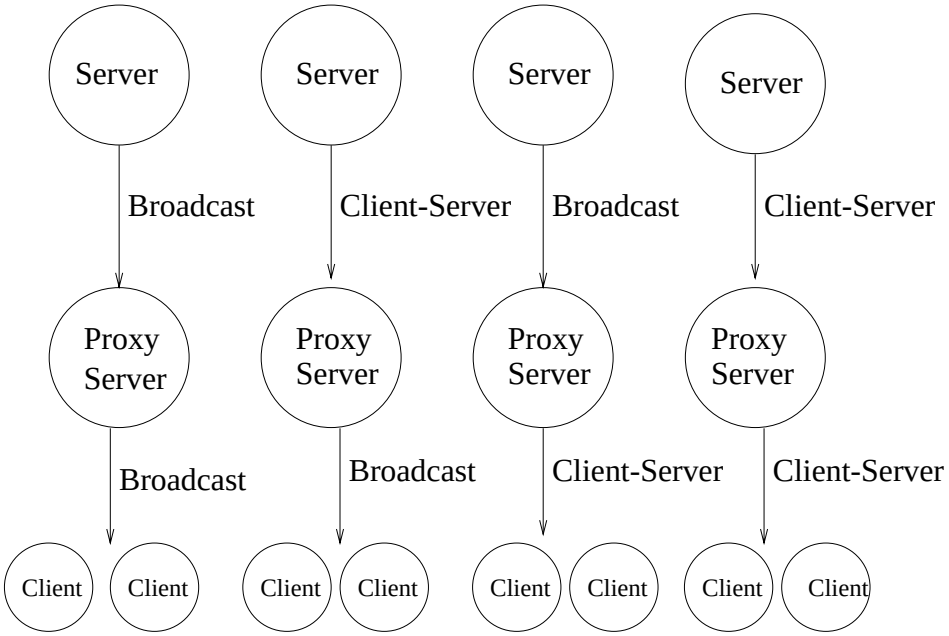

Fig. 7. Data Delivery Options for a Hierarchical Model

else

Send a request for item *sorted_items[i]* to the main server. □

In our simulation, the main servers and the proxy servers use the *Total Wait Time* algorithm (4.6) to determine the weights. Any other algorithm, such as *SR*, may also be used.

The caching algorithm used at the proxy server is the Prefix caching algorithm. However, we have to make some modifications to the caching policy at the proxy server for this model. Since the aim of the proxy server is to serve its client population most effectively, the caching decisions made by the proxy server have to be based on the requests from its clients. Therefore, the prefix array that is used by the proxy server is not sent by the main server. Rather, the proxy server uses the weights that it

calculates (to make its own scheduling decisions), as the basis for caching decisions. Therefore, every time the proxy server makes a scheduling decision, it updates its prefix array with the weights that were calculated. Therefore, to make space for a new item, item(s) in cache, starting with the one with the lowest weight in the prefix array is deleted. This difference is critical to the performance of the system. The clients can base their caching decisions on the prefix array received from their proxy server. In our simulation of the hierarchical system, however, the client cache size is zero.

The model that we evaluate in this thesis, uses data broadcasting between the proxy server and the client, as well as between the main server and the proxy servers. There are other models that could be used depending upon the application needs. These models are illustrated in Fig. 7. Note that the "broadcast" may be implemented using IP multicasting [30] or any other multicasting tool.

CHAPTER VII

REDUCING THE VARIANCE

In this chapter, we discuss using the standard deviation of the access times - in addition to the mean access time - as a performance goal of the scheduling algorithm. Standard deviation is the square root of the variance. We try to balance the mean access time with the standard deviation, so as to provide individual users better performance, even though their requests may not be for the popular items.

A. Variance

In Chapter IV, scheduling algorithms, whose main performance objective was to minimize the mean access time for all the clients, were presented. The access time that is minimized, is averaged over all the users. However, an individual user's experience may be vastly different from the overall average. Therefore we have to take into account the variance, and hence the standard deviation, of the access time. Minimizing the access time can result in a high standard deviation.

In this chapter, we use a modified version of the SR scheduling algorithm to study the trade-off between the standard deviation and the mean access time.

We first introduce a few terms. Let

- μ be the overall mean access time.
- σ^2 be the variance.
- σ be the standard deviation.
- Y be the total number of requests.
- RWT_j be the wait time for request j, $1 \leq j \leq$ Y.

By estimating the standard deviation, we can be assured that the access times will very likely be in the range $(\mu - \sigma, \mu + \sigma)$. The standard deviation is given by

$$\sigma = \sqrt{\frac{\sum_{j=1}^{Y}(RWT_j - \mu)^2}{Y-1}} \tag{7.1}$$

We can calculate it using the formula [31]

$$\sigma = \sqrt{\frac{Y\sum_{j=1}^{Y} RWT_j^2 - (\sum_{j=1}^{Y} RWT_j)^2}{Y(Y-1)}} \tag{7.2}$$

B. Reducing the Variance With the SR Scheduling Algorithm

In this section, we discuss how the *SR* scheduling algorithm can be used to find a trade-off between the mean access time and its standard deviation.

The *SR* algorithm uses the product of the number of requests pending for an item (R_i) and the time elapsed since its last broadcast (s_i), divided by the length of the item (l_i), as the weight to decide the item to broadcast next. The item with the highest weight is chosen. On closer examination of the parameters, we can see that unpopular items, which obviously will have a low value for R_i, can be forced out of contention by popular items. This is acceptable (in fact, it may be necessary), if our only objective is to minimize the mean access times. However, this can cause the waiting time for the unpopular items to be very high. As a result, we have to give more importance to s_i, if we are to alleviate some of the high waiting times for the unpopular items.

[16] proposed scheduling algorithms that aim to minimize the variance by giving more importance to the time since the last broadcast for every item. Based on the work done in [16], we conducted various experiments that used a different weight

measure for the *SR* scheduling algorithm. We used

$$\frac{s_i^{\alpha} R_i^{(1-\alpha)}}{l_i} \qquad \forall i, 1 \leq i \leq M \tag{7.3}$$

as the weight to make the scheduling decision. The modified algorithm is referred to as α-*SR*. By varying α appropriately, we can give more importance to the waiting time, than the number of requests. We assumed equal lengths for all the items in our measurements, and hence we get the original SR algorithm for $\alpha = 0.5$. Simulation results for different values of α are presented in Chapter VIII.

This algorithm is a specific case of a more general algorithm, which uses $s_i^{\alpha} R_i^{\beta} / l_i$ as the weight to make the scheduling decisions. When $\beta = (1 - \alpha)$, we get the α-SR algorithm, and when $\alpha = \beta = 1$, we get the original SR algorithm. An alternative is using $s_i^{\alpha} R_i / l_i$, similar to the push algorithm in [16].

CHAPTER VIII

PERFORMANCE EVALUATION

In this chapter we present the performance measurements. An event-driven simulator was used to measure the performance. A database of 1000 items (M) was used. Other parameters are described below.

Demand Probability: We use the Zipf distribution to model item popularity. Other researchers [8, 11, 12, 13, 17, 18, 19, 20], have also made this assumption. The Zipf distribution gives probability values for the items as follows:

$$p_i = \frac{(1/i)^\theta}{\sum_{1=1}^{M}(1/i)^\theta} \qquad 1 \leq i \leq M \tag{8.1}$$

where θ is the *access skew coefficient*. The higher the value of θ, the greater the disparity in popularity among the items.

Item Lengths: We use items of varying lengths for the simulations. The items have lengths uniformly distributed between 1 and 10. However, since [10] assumed equal lengths for their items, we also produce results with equal lengths (10) for all the items, so as to provide a fair comparison with [10].

Request Generation: The inter-request time for the clients was obtained according to a Poisson process. We set the mean inter-request time of the Poisson process to be 2 in all cases, except the simulation for the hierarchical model. A mean of 15 was used there, in order to slow the build up of requests. However, this change should not affect the results.

A. Performance Evaluation of Scheduling Algorithms

In this section, we present the performance evaluation of the *SR* and *Total Wait Time* (Total_Wait) scheduling algorithms presented in Chapter IV. All simulations were performed for 8 million client requests.

We compared the performance of both the algorithms, with the Most Request First (MRF) [23] algorithm, the S2P [18] algorithm, and the RxW [10] algorithm. Chapter III provides a brief description of these algorithms. Fig. 8 plots the performance evaluation results for these algorithms. Since the RxW algorithm was proposed for items of equal lengths only in [10], we plot simulation results for the algorithm, where the items are of equal length in Fig. 9. To simulate unequal lengths in Fig. 8, we modified the RxW algorithm to include the lengths of the items by dividing the product of R_i and the maximum waiting time of all pending requests for item i, by the length of item i. Each of these figures plot the *Overall Mean Access Time* versus the access skew coefficient θ.

From the simulation, we observe that the SR and the Total Wait Time scheduling algorithms perform better than S2P, MRF and RxW. The Total Wait Time algorithm performed better than the SR algorithm. Although, the performance results were much better than the MRF algorithm, we observed only modest gains over the S2P algorithm and RxW algorithm. Note that the S2P algorithm cannot be used for an on-demand system, since it requires the server to know the demand distribution. The results for S2P were obtained assuming that the server knows the Zipf demand distribution.

There is very little difference between the RxW, the SR and the Total Wait Time algorithms. All three aim to make an estimate of the total wait time of the requests for each item. Since Total Wait Time algorithm uses the exact measure, we

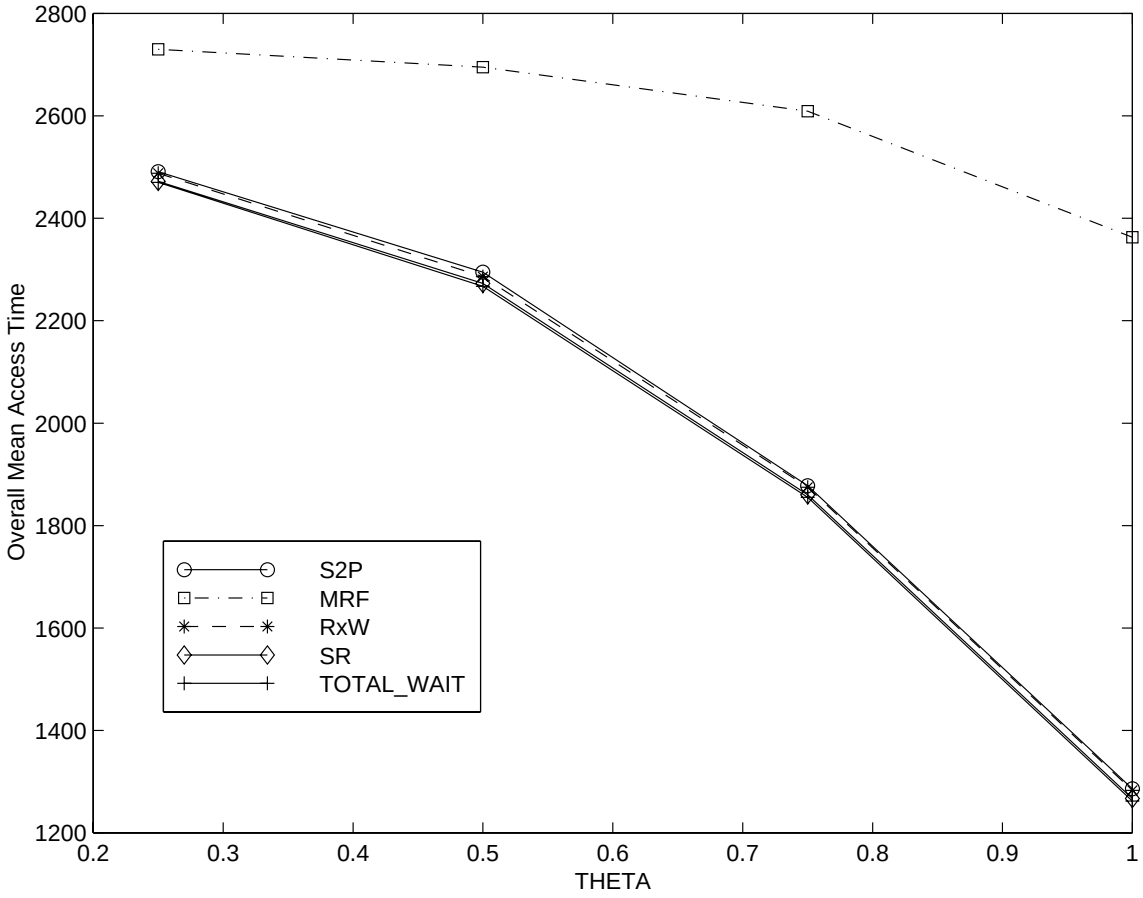

Fig. 8. Overall Mean Access Time Versus Access Skew for Items of Unequal Lengths Obtained by Simulation of Scheduling Algorithms Given in Chapter III

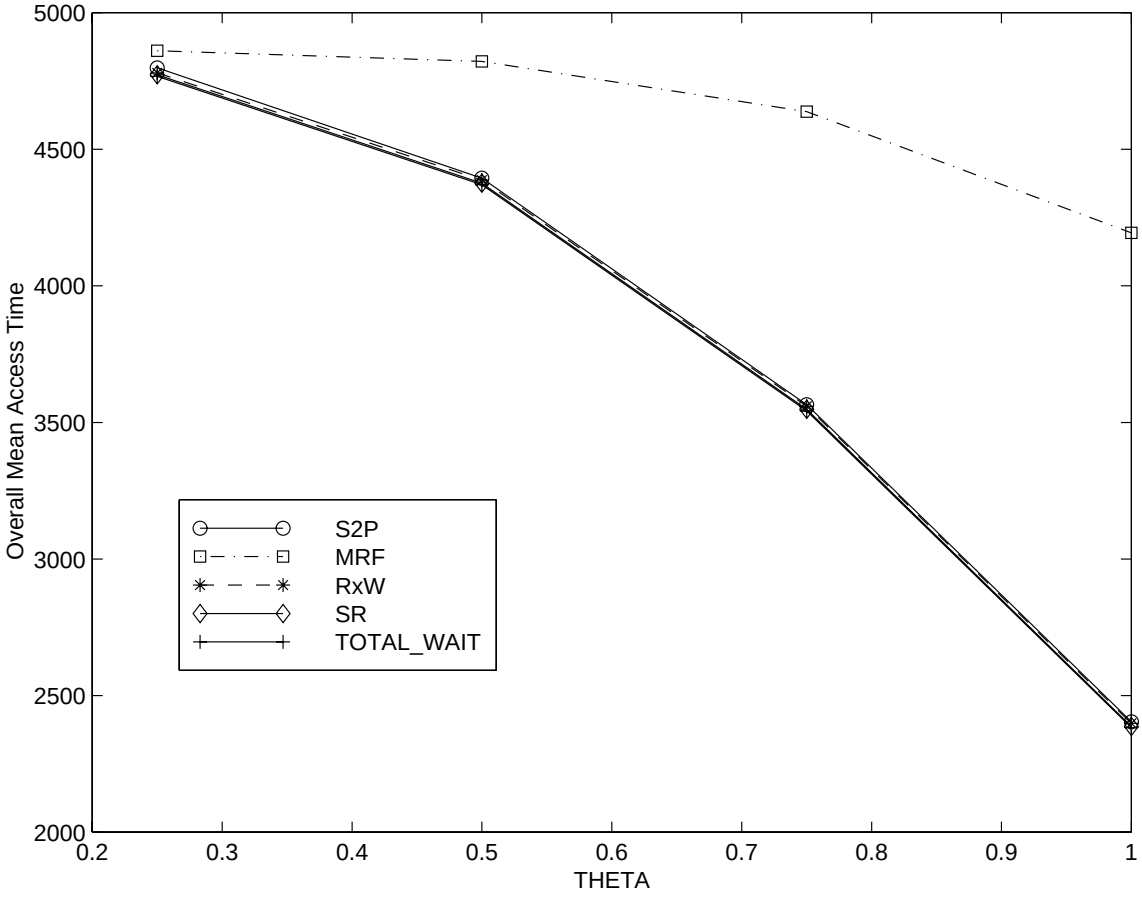

Fig. 9. Overall Mean Access Time Versus Access Skew for Items of Equal Lengths Obtained by Simulation of Scheduling Algorithms Given in Chapter III

see marginal performance improvements.

B. Performance Evaluation of Prefix Caching Scheme

This section presents the performance evaluation for the *Prefix* caching policy that was described in Chapter V. All simulations were performed for 8 million client requests.

We compared the performance of the Prefix caching policy to the Least Recently Used (LRU) caching policy and the PT [8] caching policy. Chapter III provides a description of these policies. All simulations for the caching policies used the Total Wait Time algorithm, since it performed the best. The additional bandwidth consumed by the prefix array was ignored for the simulations, since it was assumed to be negligible when compared to the items.

Since the PT caching policy requires the clients to know the demand distribution and the time till next broadcast of every item, we obtained results for it assuming that all the clients know the demand probabilities of the items. Also, PT assumes that the broadcast schedule is determined a priori, and hence the time till the next broadcast of every item is known at any time. For an on-demand system, we cannot make this assumption. Hence, we make an estimate of the time till the next broadcast for every item. This was done using the prefix array. The prefix array contains the items in increasing order of weights. The item that was just broadcast is the last one in the array. Hence, the *100-prefix* array can be used as an estimate of the broadcast schedule, and hence the time till the next broadcast of every item. Therefore, the time till the next broadcast of item i, which is in position k in the prefix array, is $\sum_{j=k+1}^{M} l_j$. Note that this is just an approximate estimate. Requests that come in during and after the current item transmission can alter the estimate. Only a schedule

determined a priori can give precise values for time till next broadcast of all the items.

Figures 10 and 11 plot the cache hit ratio and the overall mean access time respectively, versus the access skew coefficient. The client cache size is 10% of the database size, and the server sends 100% of the prefix. Figures 12 and 13 plot the results, for the case where the cache size of the clients is 20% of the database size and the server sends 100% of the Prefix. Figures 14 and 15 plot the performance for client cache size of 10% of the database size and *0-prefix* (our implementation of the PT algorithm always uses *100-prefix*). Figures 16 and 17 plot the performance of the Prefix caching policy, with $\theta = 0.50$ and *100-prefix*, for various cache sizes.

Figures 18 and 19 plot the performance, with $\theta = 0.50$ and cache size = 10% of the database size, for various prefix percentages. Both the cache hit ratio and the mean access time remain constant when the prefix percentage is greater than or equal to 4%. This is because the server does not receive requests for items already in cache. As a result, cached items, which have low weight, are at the beginning of the prefix array. The clients are able to find enough items to delete (to make space for the current broadcast item), within the first 4% of the prefix array, at all times. Although this may seem to be a desirable feature, this result is caused by the *Mature Cache Problem*, which is explained later in this section. Obviously, this "cutoff" percentage may vary for different cache sizes and different item demand distributions.

Any caching scheme that can accurately estimate the popularity of the items is bound to perform well. Since it very difficult to get a 100% hit ratio without cache space for all the items, the caching policy has to strive to keep the most popular items in the cache. Cache misses for infrequently requested items can be tolerated. Both the PT and Prefix caching algorithms do exactly this.

By multiplying the time to next broadcast with the demand probability of the item, PT manages to do this very well. However, PT is not a practical algorithm for

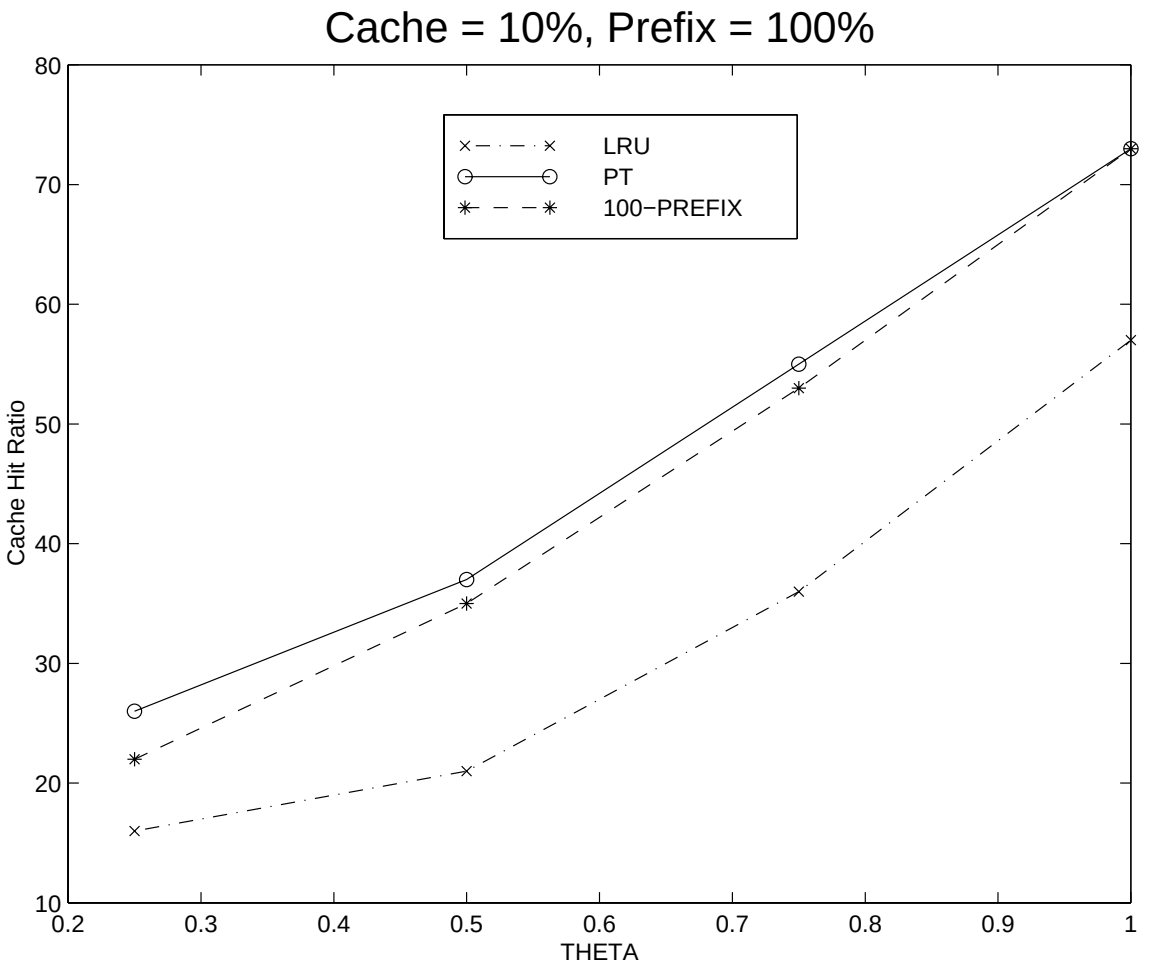

Fig. 10. Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

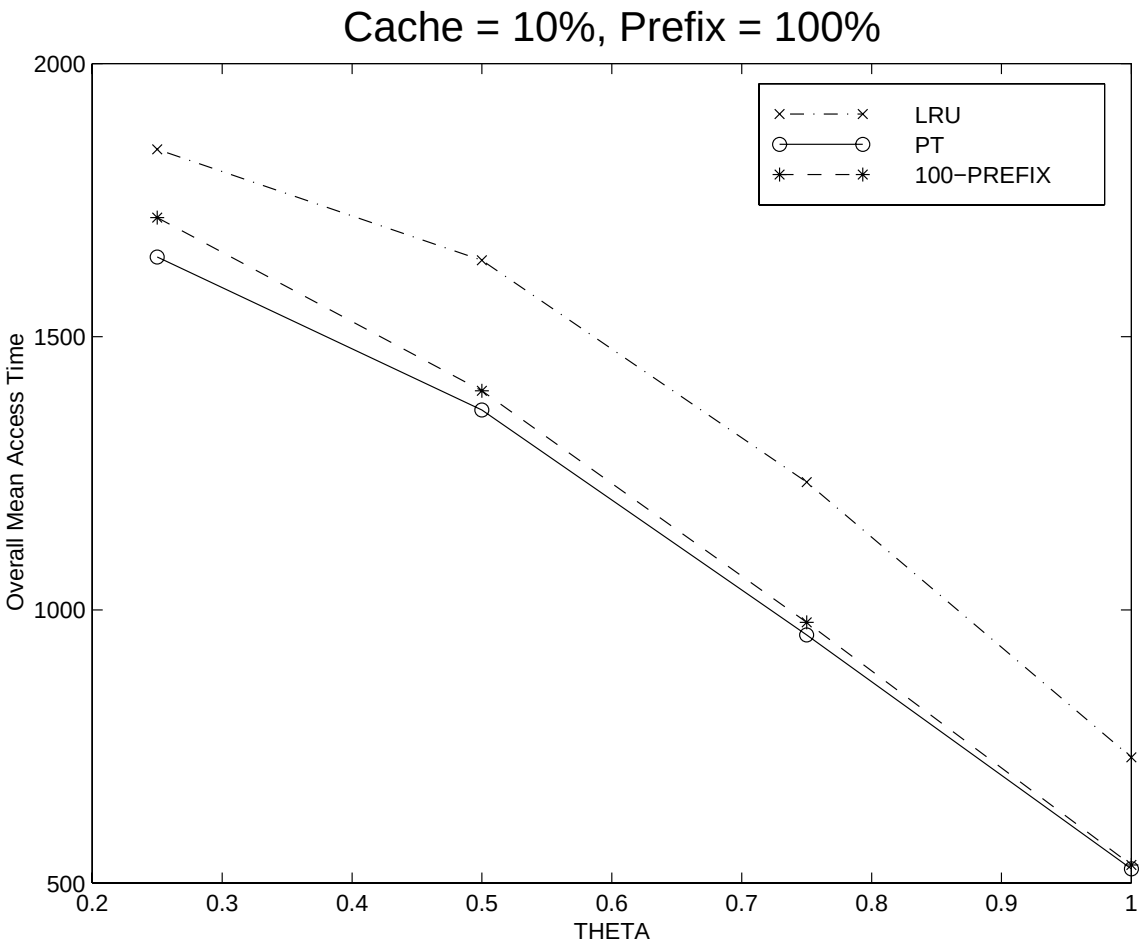

Fig. 11. Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

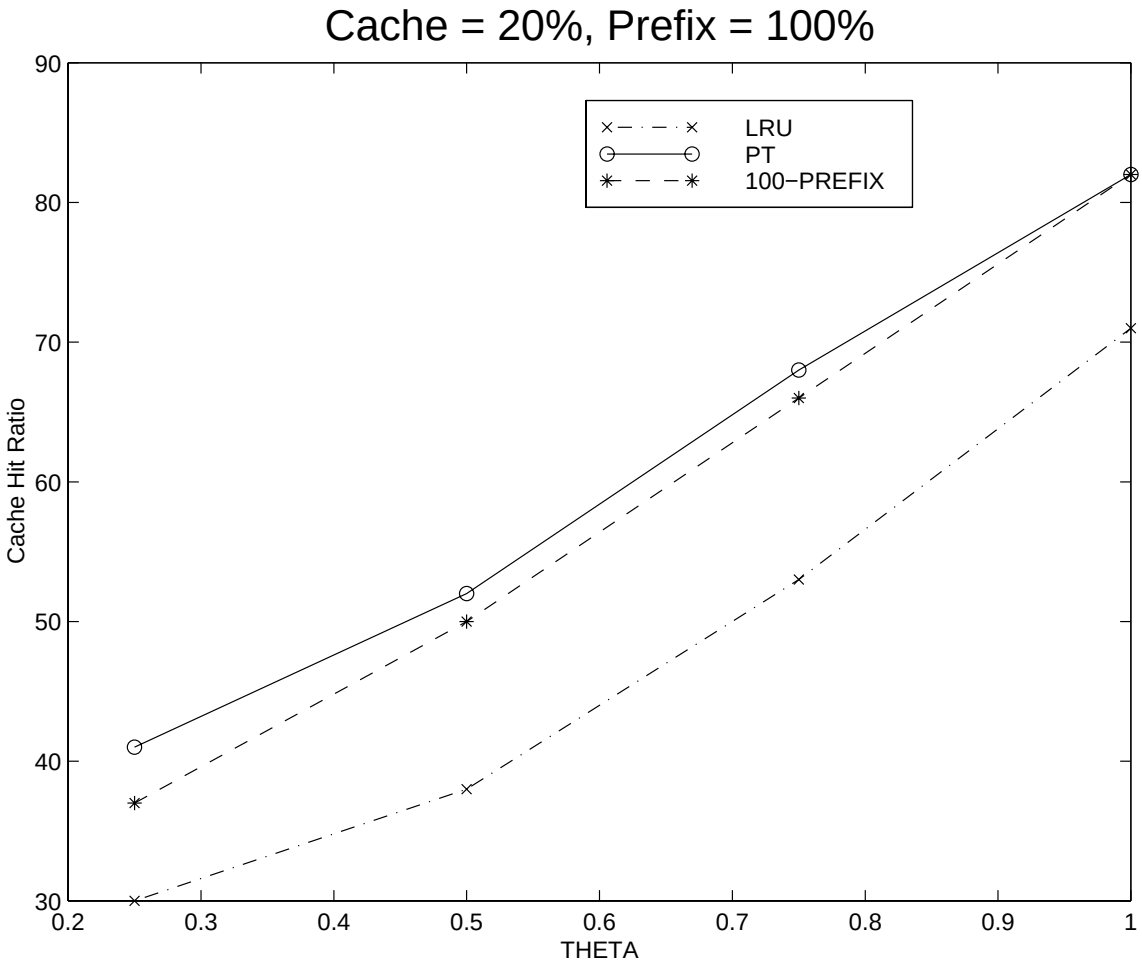

Fig. 12. Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

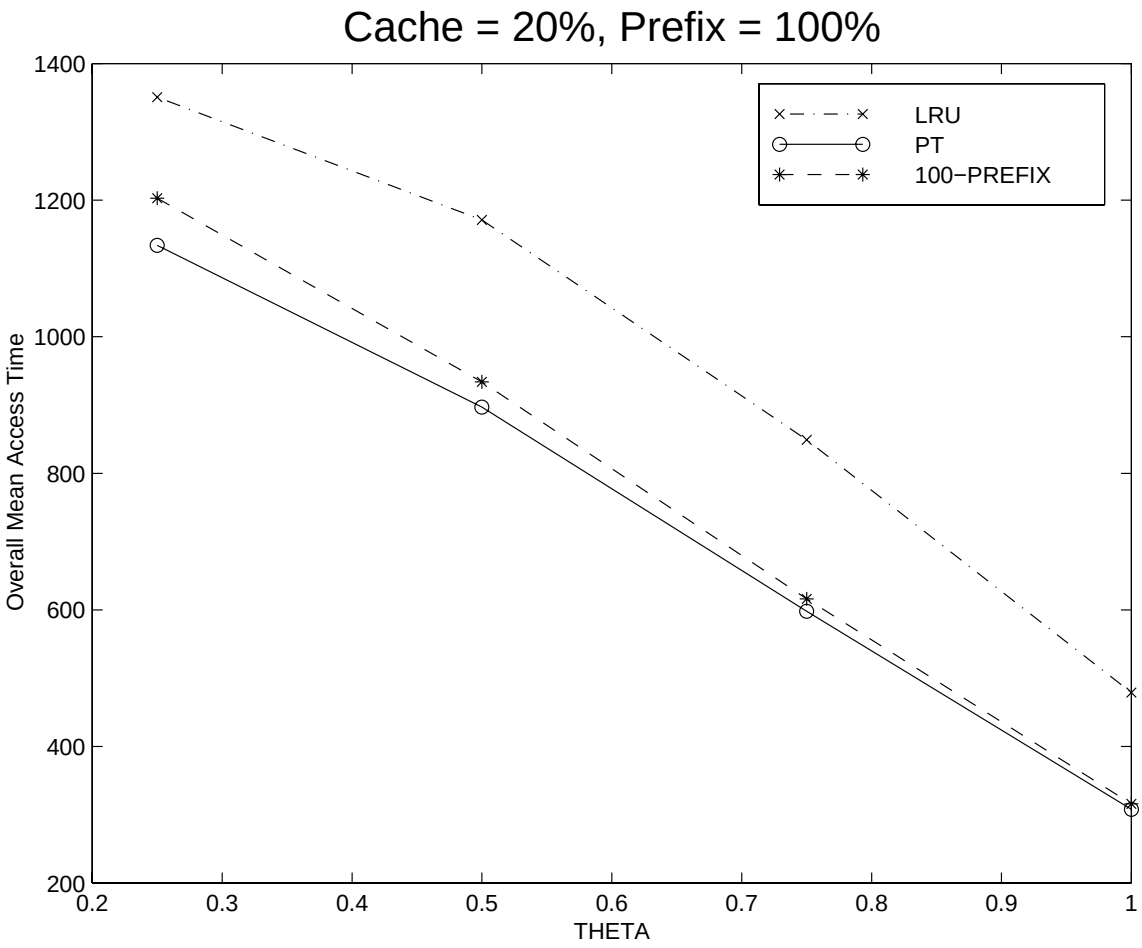

Fig. 13. Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

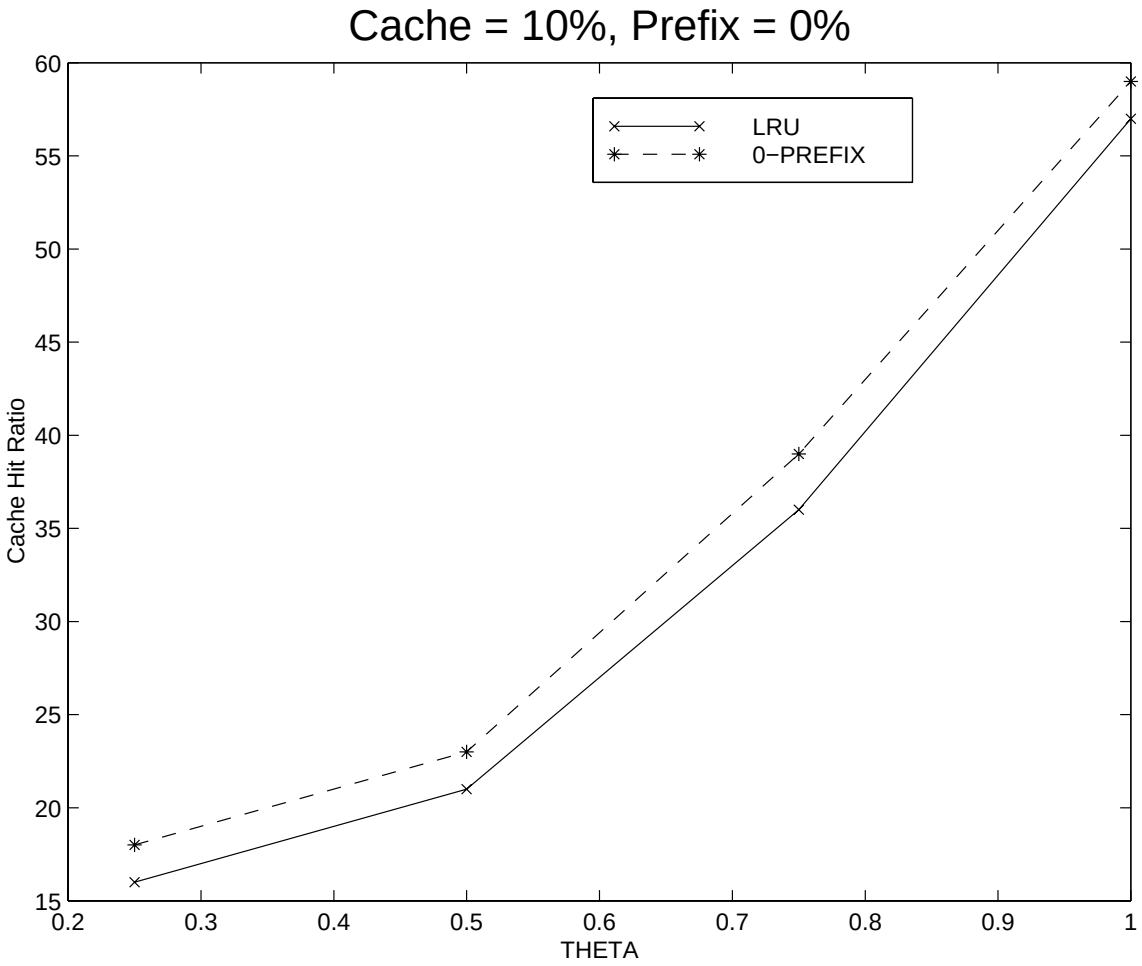

Fig. 14. Cache Hit Ratios Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

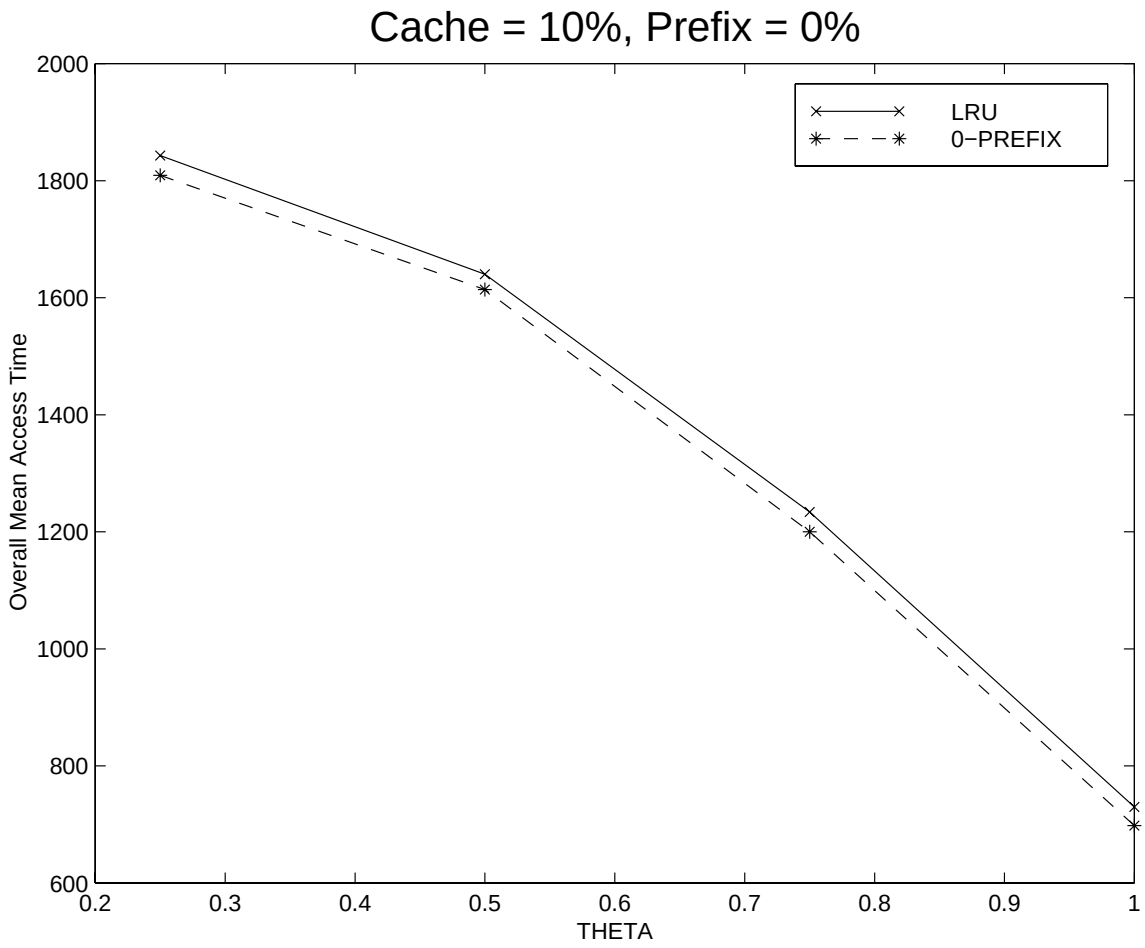

Fig. 15. Overall Mean Access Time Versus Access Skew Obtained by Simulation of Caching Policy Given in Chapter IV

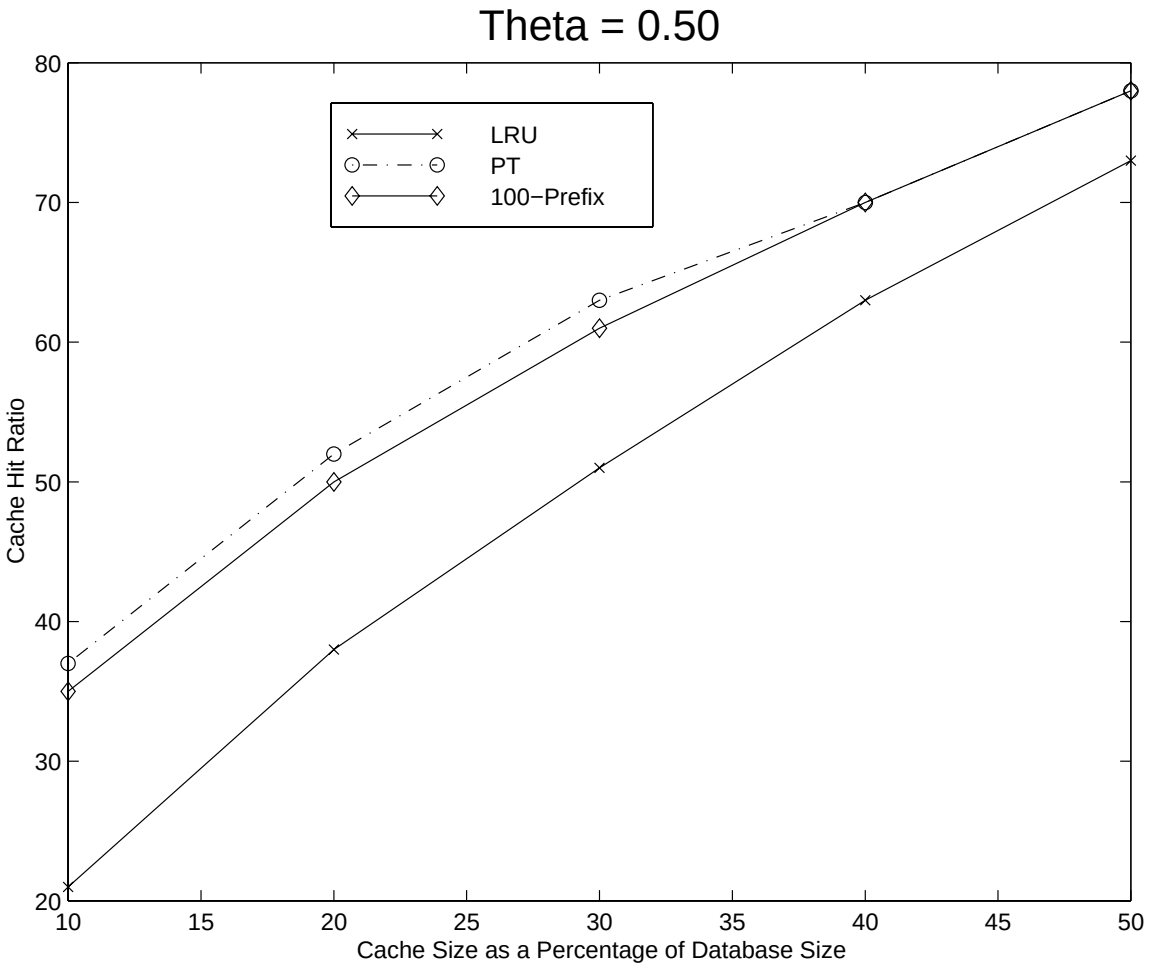

Fig. 16. Cache Hit Ratios Versus Cache Size Obtained by Simulation of Caching Policy Given in Chapter IV

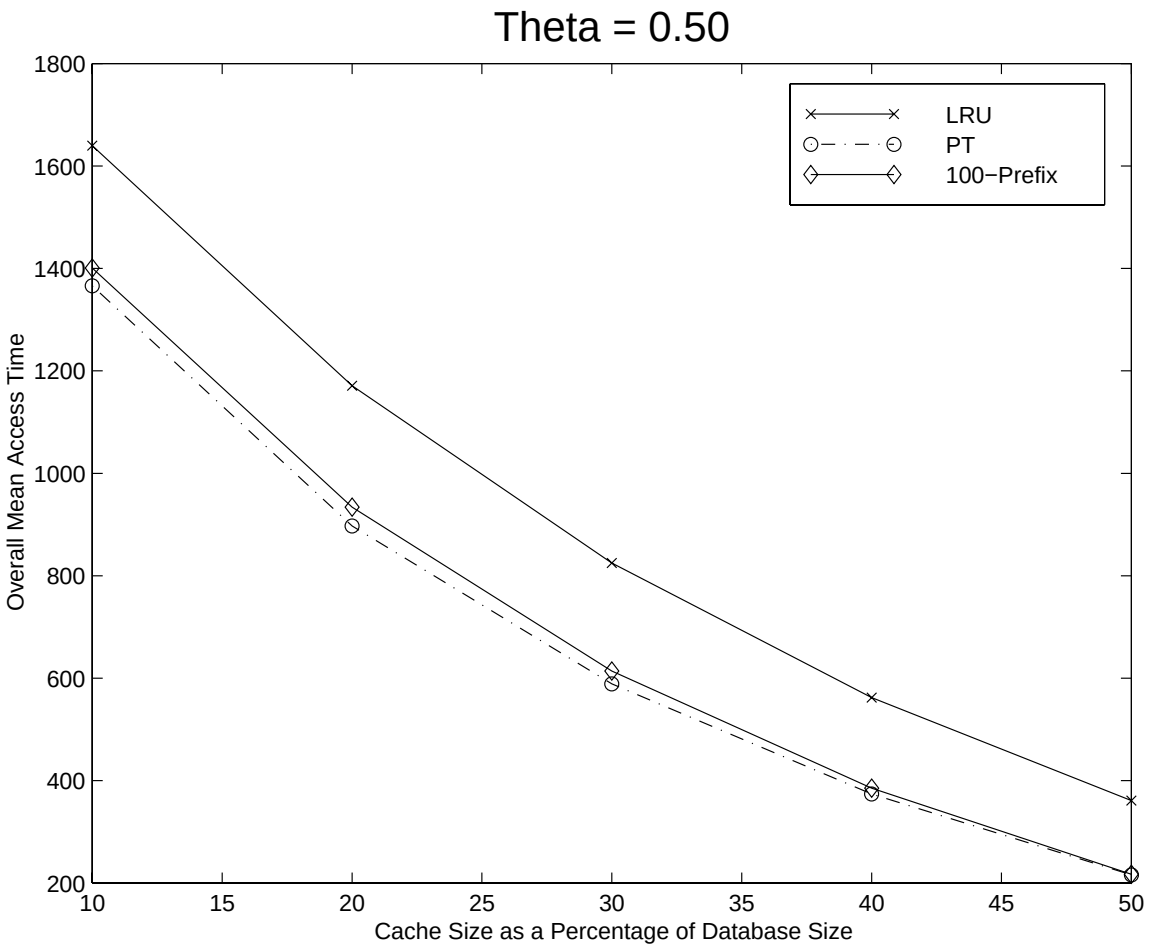

Fig. 17. Overall Mean Access Time Versus Cache Size Obtained by Simulation of Caching Policy Given in Chapter IV

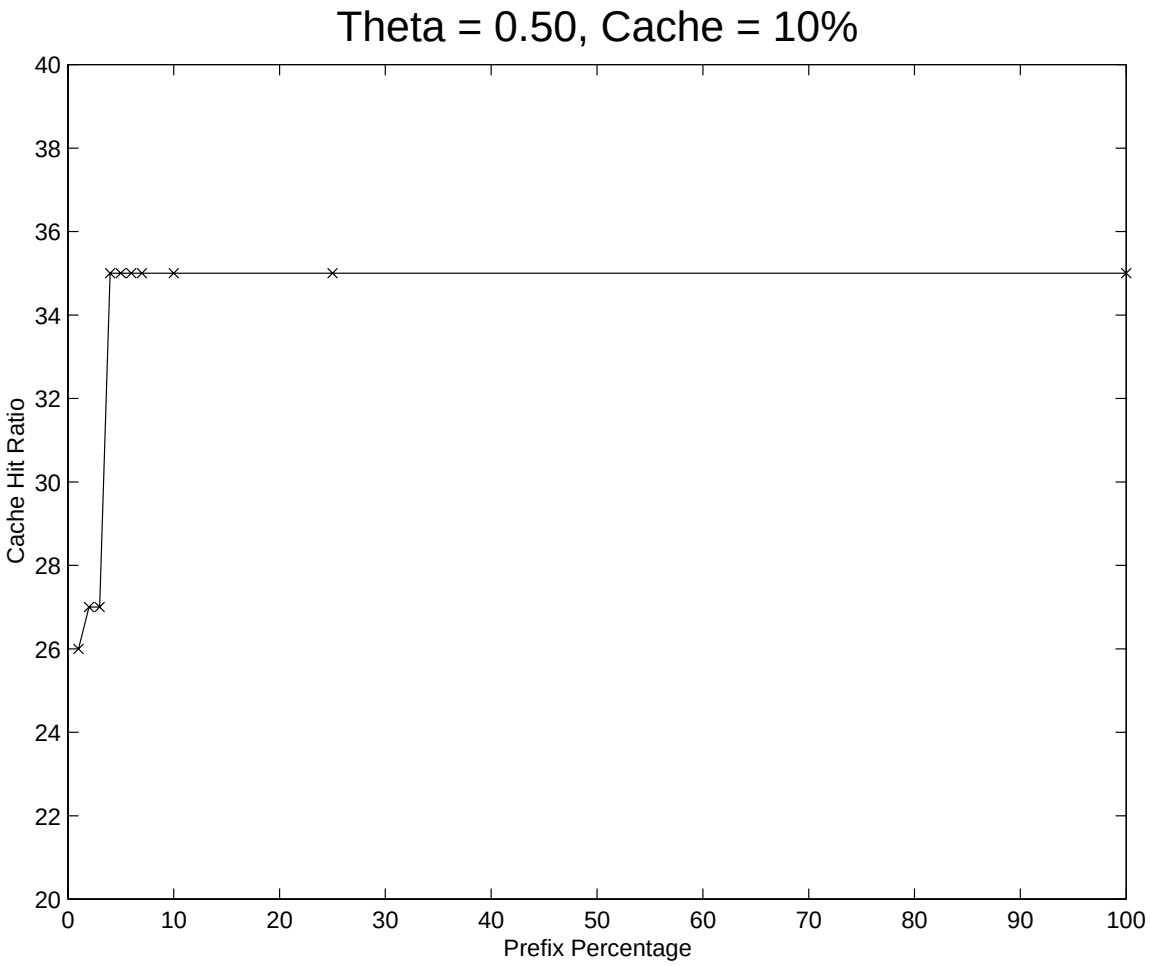

Fig. 18. Cache Hit Ratios Versus Prefix Percentage Obtained by Simulation of Caching Policy Given in Chapter IV

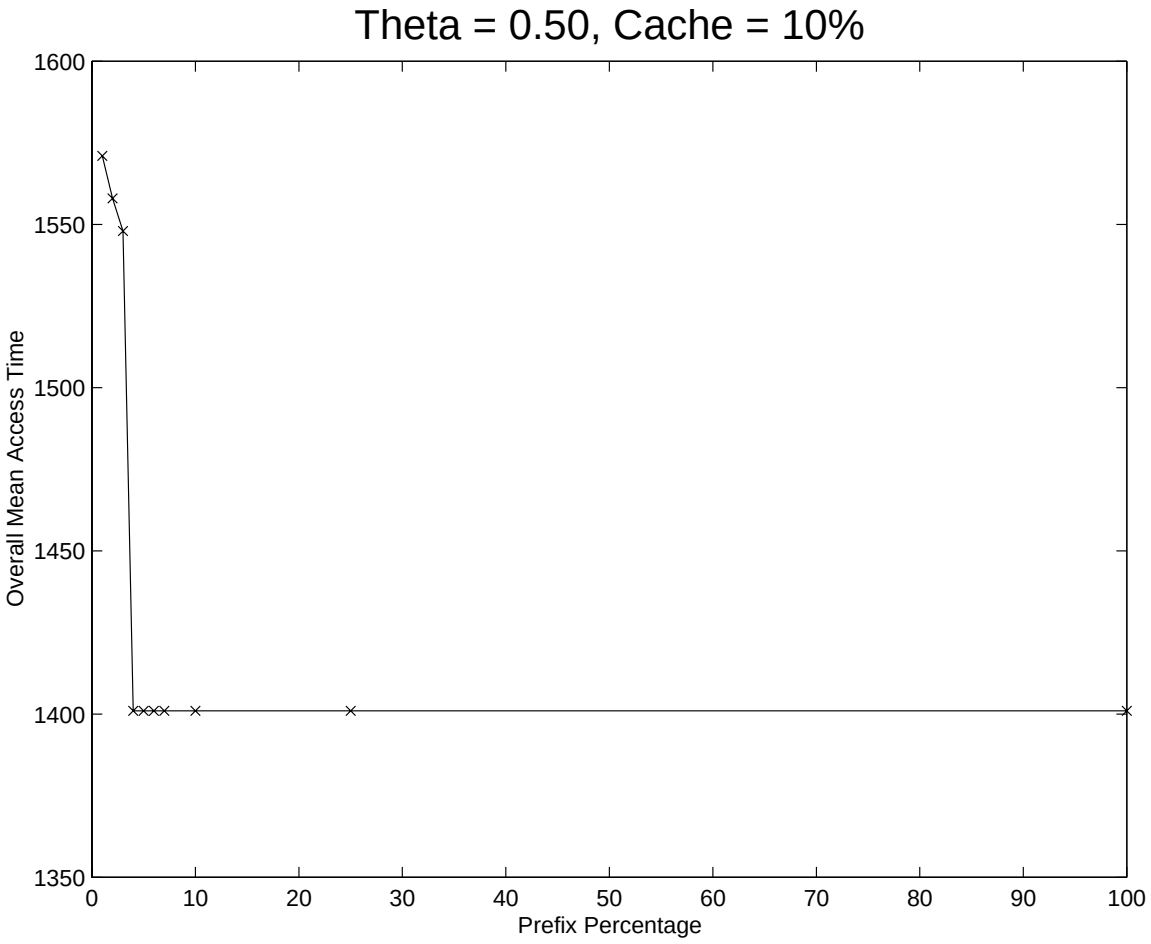

Fig. 19. Overall Mean Access Time Versus Prefix Percentage Obtained by Simulation of Caching Policy Given in Chapter IV

an on-demand model. In our model, we have to estimate, not assume, the popularity of the items. This is what the Prefix caching policy does. From the results, we can see that the Prefix caching policy is much better than LRU. However, it performs marginally lower than PT in some cases (of course, PT cache as such cannot be implemented for an on-demand system). This is due to the *Mature Cache Problem*, which is explained next.

1. The Mature Cache Problem

Although prefix based caching performs well, its performance is not equal to the PT caching scheme. This is due to what we call the *mature cache* problem.

The server makes an estimate of the popularity of the items based on the requests it receives from the clients. In the initial stages (before the cache becomes full), the server accurately predicts the popularity, since all the requests are for popular items. However, as the cache matures, the clients send requests only for those items that are not in cache. Obviously, these are for the unpopular items, since the popular items are already in cache. Therefore, the server is fooled into thinking that the unpopular items are the popular ones. Therefore, for a short period of time the server instructs the clients to cache the unpopular items. Fortunately, this mistake is quickly rectified, as new requests for the popular items come in. However, this problem repeats itself. Hence, we can see that the server goes through cycles of correct and incorrect caching hints, and this manifests in the less-than-optimal performance.

For the actual performance numbers, which provide a better view of the performance gains, please see Appendix A.

C. Performance Evaluation of Hierarchical Model

We compared the performance of the hierarchical model, described in Chapter VI, with that of a model with no proxy servers. All simulations were performed for 4 million client requests.

Both the proxy server and the main server use the Total Wait Time scheduling algorithm. In these simulations, the clients had a cache size of zero. The cache size at the proxies was equal to the size of the database at the server. Hence, any item transmitted by the server can fit in the cache, without any deletions.

Fig. 20 plots the overall mean access time versus the access skew coefficient for the hierarchical model with four proxy servers, and the model with just one main server (Regular) and no proxy servers. Note that the model used for the simulation uses broadcasting from the main server to the proxy servers, as well as from the proxy servers to the clients.

There is a definite performance improvement with the hierarchical scheme. This is in spite of the fact that we assumed it takes the same time for the items to reach the client from the proxy server, as it does from the server to the clients in the regular case. Also, we did not model any performance improvement at the server, even though it has to handle a lower number of requests. The reason for the improvement is that there is less contention for the broadcast channel. Since, each proxy server has a lower number of clients to serve, the number of requests at any instant of time at the proxy server is lower than the number of requests for the model with just one server and no proxies. Hence, the number of *different* items for which there are requests is lower. Since the number of requests, and hence the variety of items requested is lower for the proxy, it can serve more requests with fewer transmissions.

From these results, there is a definite merit in investigating this model further.

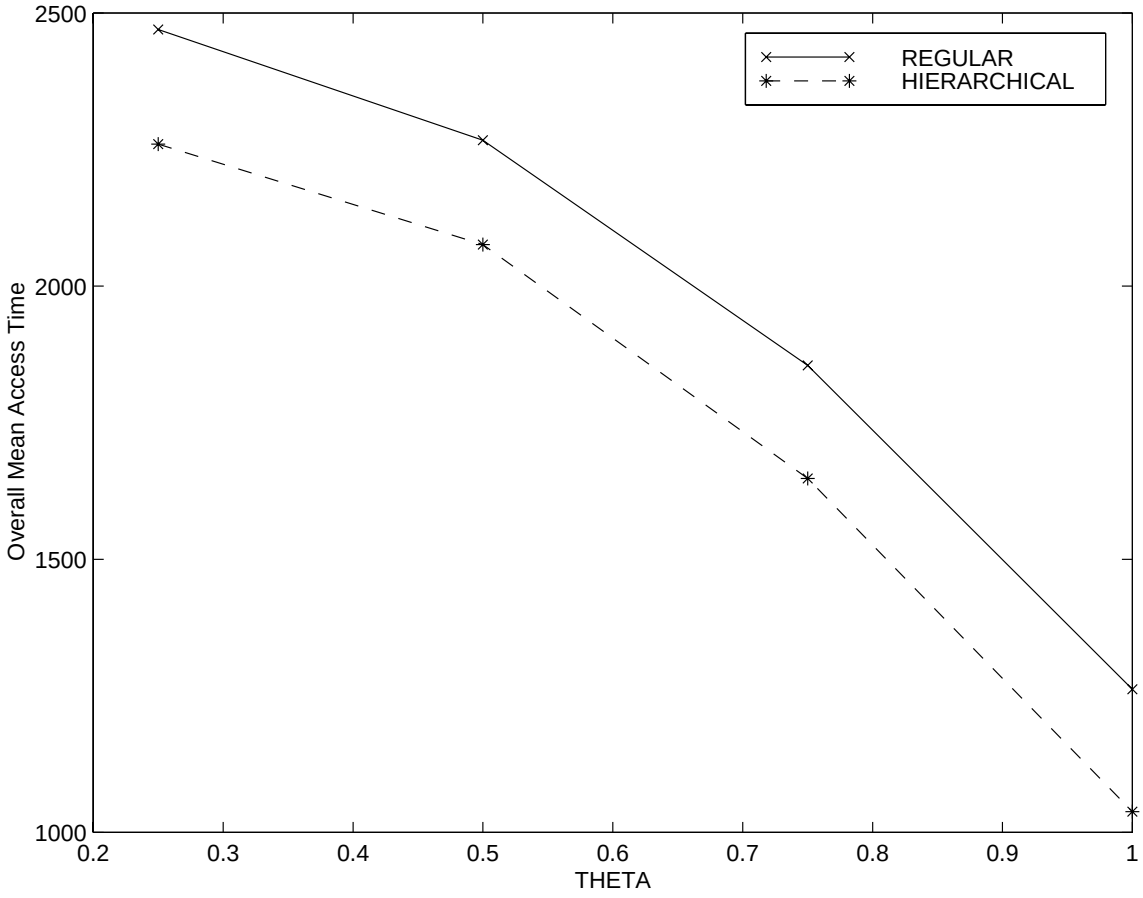

Fig. 20. Overall Mean Access Time Versus Access Skew Obtained by Simulation of Hierarchical Broadcast Model Given in Chapter V

Appendix A gives the the precise values from the simulation results.

D. Performance Evaluation of the α-SR Scheduling Algorithm

This section gives the performance results for the α-SR algorithm that was described in Chapter VII. All simulations were performed for 8 million client requests. All items are assumed to be of length 10.

Table I gives the standard deviation, and Table II gives the mean access time, for the α-SR algorithm (for different values of α) and the RxW algorithm. For $\alpha = 0.2$, the α-SR algorithm performs poorly, with very high values for both the mean and the standard deviation. This is because the number of requests R_i, is given a disproportionate amount of emphasis. However, as we increase α, we can see that the mean access time first decreases (till $\alpha = 0.5$) and then increases. The standard deviation decreases as we increase α, since we are giving more importance to s_i, and hence the waiting time of the requests.

Table I. Standard Deviation of α-SR and RxW Scheduling Algorithms

θ	α-SR							RxW
	α							
	0.2	0.4	0.5	0.6	0.7	0.8	0.9	
0.25	3139	3011	2962	2923	2898	2883	2882	2947
0.50	3737	3302	3134	2995	2894	2832	2827	3120
0.75	4585	3631	3300	3039	2841	2715	2705	3285
1.0	5534	3663	3167	2820	2582	2433	2471	3147

For cases when the item demand distribution has a low skew, the increase in

Table II. Overall Mean Access Time of α-SR and RxW Scheduling Algorithms

θ	α-SR							RxW
	α							
	0.2	0.4	0.5	0.6	0.7	0.8	0.9	
0.25	4805	4775	4774	4777	4793	4818	4849	4784
0.50	4516	4389	4377	4403	4459	4563	4707	4387
0.75	3892	3578	3548	3602	3741	3985	4538	3559
1.0	2910	2437	2391	2469	2692	3087	3771	2398

mean access time, with increase in α (for $\alpha > 0.5$) is not very high. However, this increase is more pronounced as the skew increases. The reason for this is that, as the skew increases, the number of requests for the most popular items increase, and is much larger than that for the less popular items. For lower access skew, the increased importance to s_i is averaged over all the items, and hence the mean access time remains stable. However, as the access skew increases, the increased importance to s_i is not averaged among all the items. The popular items feel the negative impact of this, and hence the mean access time suffers.

The RxW algorithm performs marginally better, with a lower standard deviation than the α-SR algorithm for $\alpha = 0.5$. This is because, the RxW algorithm uses the product of maximum waiting time and the number of requests as its weight, and hence the unpopular items get a marginally fairer treatment. But as the simulation results show, the performance of the two algorithms is almost identical, with only marginal differences.

CHAPTER IX

FUTURE WORK AND CONCLUSIONS

A. Future Work

Data broadcasting is a nascent field and more research has to be done to make it a viable alternative to the client-server approach. There are several issues that have to be tackled in this new paradigm. We list a few in this section.

1. Although we looked at the issue of reducing the variance of the access times, more work has to be done in this area. As mentioned in Chapter III, [16] proposed some solutions. But they do not take the on-demand model into consideration. If any quality of service guarantees are to made, then the variance is as important as the overall mean access time.

2. The caching scheme that we proposed aims to further reduce the overall mean access time. However, another aim of the caching scheme could be to reduce the variance as well. Hence, the caching scheme would have to find a balance between the variance and the mean access time. It would have to store some of the unpopular items as well, so that the clients do not have very large waiting times for them.

3. More work needs to be done to evaluate the generalization of the prefix caching policy that was proposed in Chapter V.

4. We did not take the additional bandwidth that the prefix array consumes while measuring the performance of the Prefix caching policy (thus, the prefix array size was assumed to be negligible when compared to the item size). Further evaluation is needed if the item sizes are small and hence, the prefix size cannot

be ignored.

5. We have only done preliminary work on the hierarchical broadcast model. We did not take various parameters (for instance, lower transmission time from the proxy to its clients than from the main server, proxy's cache size) into consideration, while measuring the performance gain.

6. The hierarchical broadcast model maps very well to the wireless PCS model, with base-stations and mobile hosts. This could be a very interesting and practical application. With servers and networks strained with ever-growing demand, more work, with better models, needs to done in this important area.

7. Data broadcasting is not going to replace the client-server approach. However, as explained in Chapter I, for selected environments, it is better suited. Hence, more work is needed to develop applications that take advantage of data broadcasting.

B. Conclusions

This thesis deals with data broadcasting. Data broadcasting is ideally suited for environments wherein the bandwidth available to the server is insufficient to serve the clients one at a time. In such environments broadcasting is efficient. In this thesis, we considered an on-demand broadcast environment. Here, the clients send requests to the server, and the server makes scheduling decisions based on these requests.

This thesis evaluates two scheduling algorithms for the on-demand broadcast environment. The *SR* algorithm and the *Total Wait Time* algorithm make scheduling decisions on-line, based on the current pending client requests. Both these algorithms aim to minimize the overall mean access time for the clients. Performance results

showed that the *Total Wait Time* algorithm performed the best.

Another way to improve the mean access time is for clients to cache the items that are received from the server. In an on-demand broadcast environment, the server has precise knowledge of client request patterns. This thesis proposed a new caching scheme, called the *Prefix* caching policy, where the server helps the client make caching decisions. Simulation results show that the *Prefix* caching policy performs very well. However, the *Mature Cache Problem* limits the performance gain somewhat.

We evaluated a *Hierarchical* broadcast model. Here, the clients send their requests to a designated proxy server, instead of the main server. By reducing the number of clients that each server has to serve, we achieved improvements in the mean access time. However, there is a lot more work to be done in this area.

Finally, the *SR* scheduling algorithm was modified, to reduce the variance, in addition to the mean access time. α-*SR*, the modified algorithm, was evaluated, and the results show that for low skew among the item demands, the variance can be significantly reduced without drastically increasing the mean access time. For high access skew, the reduction in the variance comes at the expense of a large increase in the mean access time.

REFERENCES

[1] B. R. Badrinath and T. Imielinski, "Data management for mobile computing," *Communications of the ACM*, vol. 37, no. 10, pp. 19–28, March 1994.

[2] S. Shekar, A. Fetterer, and D.-R. Liu, "Genesis: an approach to data dissemination in advanced traveller information systems," *Bulletin of the Technical Committee on Data Engineering*, vol. 19, no. 3, pp. 40–47, September 1996.

[3] T. Imielinski and S. Viswanathan, "Pyramid broadcasting for video on demand," in *Proceedings of ACM/IEEE Multimedia Conference*, February 1995. http://www.cs.rutgers.edu/ imielins/index.html, accessed on September 25, 1997.

[4] S. Viswanathan, "Publishing in wireless and wireline environments," Ph.D. dissertation, Rutgers University, New Brunswick, New Jersey, November 1994.

[5] Pointcast, Inc., http://www.pointcast.com, accessed on January 10, 1997.

[6] Airmedia, Inc., http://www.airmedia.com, accessed on January 10, 1998.

[7] IBM, "MQSeries: message oriented middleware," http://www.software.ibm.com /ts/mqseries/library/whitepapers/mqover /, accessed on May 28, 1998.

[8] S. Acharya, "Broadcast disks - data management for asymmetric communications environments," Ph.D. dissertation, Brown University, Providence, Rhode Island, May 1998.

[9] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based data delivery using broadcast disks," *IEEE Personal Communications*, vol. 2, no. 6, pp. 50–60, December 1995.

[10] D. Aksoy and M. Franklin, "Scheduling for large-scale on-demand data broadcasting," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, April 1998, pp. 652–659.

[11] M. H. Ammar, "Response time in a teletext system: An individual user's perspective," *IEEE Transactions on Communications*, vol. COM-35, no. 11, pp. 1159–1170, November 1987.

[12] M. H. Ammar and J. W. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Transactions on Communications*, vol. COM-35, no. 1, pp. 68–73, January 1987.

[13] S. Hameed, "Scheduling information broadcast in asymmetric environment," Master's thesis, Texas A&M University, College Station, Texas, May 1997.

[14] S. Hameed and N. H. Vaidya , "Efficient algorithms for scheduling data broadcast," *ACM/Baltzer Wireless Networks Journal*, to appear. http://www.cs.tamu.edu/faculty/vaidya/mobile.html, accessed on June 16, 1998.

[15] S. Hameed and N. H. Vaidya , "Log-time Algorithms for Scheduling Single and Multiple Channel Data Broadcast," *ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, September 1997. http://www.cs.tamu.edu/faculty/vaidya/mobile.html, accessed on December 1, 1997.

[16] S. Jiang and N. H. Vaidya, "Scheduling algorithms for a data broadcast system: minimizing variance of the response time," Tech. Rep. 98-005, Computer Science Department, Texas A&M University, College Station, February 1998.

[17] C. J. Su and L. Tassiulas, "Broadcast scheduling for information distribution," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, 1997. http://www.ee.umd.edu/ leandros/infocom97.ps, accessed on June 20, 1998.

[18] N. H. Vaidya and S. Hameed , "Data broadcast in asymmetric wireless environments," *Workshop on Satellite Based Information Services (WOSBIS)*, pp. 38–52, November 1996.

[19] N. H. Vaidya and S. Hameed, "Scheduling data broadcast in asymmetric communication environments," *ACM/Baltzer Wireless Networks Journal*, to appear. http://www.cs.tamu.edu/faculty/vaidya/mobile.html, accessed on June 16, 1998.

[20] J. W. Wong, "Broadcast delivery," in *Proceedings of IEEE*, December 1998, pp. 1566–1577.

[21] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a broadcast disk," *12th International Conference on Data Engineering*, February 1996. http://www.cs.umd.edu/users/franklin/papers/icde96.ps.gz, accessed on June 20, 1998.

[22] C. J. Su and L. Tassiulas, "Optimal memory management strategies for a mobile user in a broadcast delivery system," *IEEE Journal on Selected Areas in Communications*, 1999, to appear. http://www.ee.umd.edu/ leandros/tass1.htm, accessed on June 1, 1998.

[23] H. D. Dykeman, M. H. Ammar, and J. W. Wong, "Scheduling algorithms for videotex systems under broadcast delivery," in *Proceedings of the International Conference on Communications*, 1986, pp. 1847–1851.

[24] A. Tanenbaum, *Modern Operating Systems*. Upper Saddle River, New Jersey: Prentice Hall, 1992.

[25] B. R. Badrinath, T. Imielinski and S. Viswanathan, "Energy efficient indexing on air," in *Proceedings of ACM Special Interest Group on Management of Data (SIGMOD)*, May 1994, pp. 25–37.

[26] M.-S. Chen, P. S. Yu, and K.-L. Wu, "Indexed sequential data broadcasting in wireless mobile computing," in *Proceedings of 17th IEEE International Conference on Distributed Computing Systems*, May 1997, pp. 124–131.

[27] A. L. N. Reddy, "Caching strategies for a multimedia server," in *Proceedings of IEEE Conference on Multimedia Computing and Systems*, June 1997. http://ee.tamu.edu/ reddy/papers/ieeemm97.ps, accessed on June 25, 1998.

[28] T. Brisco, "RFC 1794: DNS support for load balancing," April 1995. Status: Informational. ftp://ftp.internic.net/rfc/rfc1794.txt, accessed on June 10, 1998.

[29] H. Chawla and R. Bettati, "Replicating IP services," Tech. Rep. 97-008, Department of Computer Science, Texas A&M University, September 1997.

[30] S. Deering, "RFC 1112: Host extensions for IP multicasting," August 1989. Status: Standard. ftp://ftp.internic.net/rfc/rfc1112.txt, accessed on June 24, 1998.

[31] J. E. Freund and F. J. Williams, *Modern Business Statistics*. Englewood Cliffs, New Jersey: Prentice Hall, 1958.

APPENDIX A

TABULAR REPRESENTATION OF SIMULATION DATA

Table III. Data for Fig. 8 in Tabular Form

θ	S2P	MRF	RxW	SR	TOTAL_WAIT
0.25	4798	4861	4784	4774	4767
0.50	4394	4822	4387	4376	4370
0.75	3565	4638	3559	3548	3542
1.0	2404	4194	2398	2387	2383

Table IV. Data for Fig. 9 in Tabular Form

θ	S2P	MRF	RxW	SR	TOTAL_WAIT
0.25	2491	2730	2488	2472	2470
0.50	2295	2695	2286	2273	2267
0.75	1878	2609	1875	1860	1855
1.0	1286	2363	1283	1267	1262

Table V. Data for Fig. 10 in Tabular Form

θ	LRU	PT	100-Prefix
0.25	16%	26%	22%
0.50	21%	37%	35%
0.75	36%	55%	53%
1.0	57%	73%	73%

Table VI. Data for Fig. 11 in Tabular Form

θ	LRU	PT	100-Prefix
0.25	1843	1646	1718
0.50	1640	1366	1401
0.75	1234	954	977
1.0	730	526	533

Table VII. Data for Fig. 12 in Tabular Form

θ	LRU	PT	100-Prefix
0.25	30%	41%	37%
0.50	38%	52%	50%
0.75	53%	68%	66%
1.0	71%	82%	82%

Table VIII. Data for Fig. 13 in Tabular Form

θ	LRU	PT	100-Prefix
0.25	1351	1134	1203
0.50	1171	897	934
0.75	849	598	616
1.0	479	308	316

Table IX. Data for Fig. 14 in Tabular Form

θ	LRU	0-Prefix
0.25	16%	17%
0.50	21%	23%
0.75	36%	39%
1.0	57%	59%

Table X. Data for Fig. 15 in Tabular Form

θ	LRU	0-Prefix
0.25	1843	1809
0.50	1640	1614
0.75	1234	1200
1.0	730	698

Table XI. Data for Fig. 20 in Tabular Form

θ	Regular	Proxy
0.25	2470	2260
0.50	2267	2076
0.75	1855	1648
1.0	1262	1038

Table XII. Data for Fig. 16 in Tabular Form

Cache Size	LRU	PT	100-Prefix
10%	21%	37%	35%
20%	38%	52%	50%
30%	51%	63%	61%
40%	63%	70%	69%
50%	73%	78%	78%

Table XIII. Data for Fig. 17 in Tabular Form

Cache Size	LRU	PT	100-Prefix
10%	1640	1366	1401
20%	1171	897	934
30%	825	589	614
40%	562	374	385
50%	361	216	217

Table XIV. Data for Fig. 18 in Tabular Form

Prefix Percentage	Cache Hit Ratio
1%	26%
2%	27%
3%	27%
4%	35%
5%	35%
6%	35%
7%	35%
10%	35%
25%	35%
100%	35%

Table XV. Data for Fig. 19 in Tabular Form

Prefix Percentage	Overall Mean Access Time
1%	1571
2%	1558
3%	1548
4%	1401
5%	1401
6%	1401
7%	1401
10%	1401
25%	1401
100%	1401

VITA

Kannan Kothandaraman was born on September 2, 1974 in Madras, India. He received his Bachelor of Science degree in Computer Science from Louisiana State University, Baton Rouge in May 1996. Subsequently, he joined the graduate program in Computer Science at Texas A&M University. His research has been in mobile computing and data broadcasting. His address is Department of Computer Science, Texas A&M University, College Station, TX 77843.

The typist for this thesis was Kannan Kothandaraman.