AN EFFICIENT BROADCAST SCHEDULING ALGORITHM FOR PULL-BASED MOBILE ENVIRONMENTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

By

K. Murat Karakaya

August, 2000

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Özgür Ulusoy (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Tuğrul Dayar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

An Efficient Broadcast Scheduling Algorithm for Pull-Based Mobile Environments

K. Murat Karakaya
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Özgür Ulusoy
August 2000

Thanks to the advances in telecommunications and computers, today mobile computing becomes a significant means in every pace of life. Many people are now carrying portable devices such as laptop computers, Personal Digital Assistants (PDAs), and cellular phones. These mobile computing devices are supported by rapidly expanding telecommunication technology. Cellular communication, wireless LAN and WAN, and satellite services are available for daily life applications, and portable devices make use of these wireless connections to contact with the information providers. Thus, a user does not need to maintain a fixed connection in the network and may enjoy almost unrestricted user mobility.

As the new and various mobile infrastructures emerge, users demand a new class of applications running in this environment. However, the narrow bandwidth of the wireless communication channels, the relatively short active life of the power supplies of mobile units, and the mobility of clients make the problem of data retrieval more difficult than that in wired networks. Therefore, mechanisms to efficiently transmit information to vast numbers of mobile users are of significant interest. *Data broadcasting* has been considered one of the most promising ways of data dissemination in mobile environments. There are two basic data broadcasting approaches available: *push* and *pull*. In push-based broadcasting approach, data is broadcast to mobile users according to users' profiles or subscriptions, whereas in pull-based broadcasting approach, transmission of data is initiated by the explicit request of users.

In this thesis, we have focused on the problem of scheduling data items

to broadcast in a pull-based environment. We have developed an efficient broadcast scheduling algorithm, and comparing its performance against several well-known algorithms, we have observed that our algorithm appears to be one of the best algorithms proposed so far.

Key words: Mobile computing, mobile database, data broadcast, broadcast delivery, pull-based, scheduling algorithm.

ÖZET

Çekmeye Dayalı Mobil Ortamlarda Etkin Bir Yayım Programlama Algoritması

K. Murat Karakaya
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Agustos 2000

Haberleşme ve bilgisayar teknolojilerindeki gelişmeler sayesinde bugün mobil sistemler hayatın her alanında önemli bir araç olmuştur. Şimdilerde, bir çok insan dizüstü bilgisayar, kişisel sayısal yardımcı ve cep telefonu gibi taşınabilir araçlar kullanmaktadır. Bu tür mobil araçlar hızla gelişen haberleşme teknolojisi tarafından desteklenmektedir. Hücresel haberleşme, kablosuz Yerel Alan Şebekeleri (YAŞ) ve Geniş Alan Şebekeleri (GAŞ) ile uydu hizmetleri günlük hayatta kullanılan uygulamalar için mevcuttur ve taşınabilir araçlar bilgi sağlayıcılara bu gibi kablosuz bağlantıları kullanarak ulaşabilmektedir. Böylece, bir kullanıcının ağ üzerinde sabit bir bağlantı sürdürmesine gerek kalmamıştır. Artık kullanıcılar hemen hemen sınırsız hareketlilik imkanına kavuşmuşlardır.

Yeni ve değişik mobil altyapılar oluştukça, kullanıcılar bu ortamda çalışacak yeni uygulamalar talep etmektedirler. Ancak, telsiz kanallarının dar olan bant genişliği, taşınabilir araçların batarya sürelerinin göreceli olarak az olması ve kullanıcıların hareket halinde bulunmaları, kablolu ağlara göre veri iletişimini daha zor bir hale getirmektedir. Bu nedenle, bilgiyi çok geniş mobil kitlelere etkin olarak ulaştırabilecek mekanizmalar büyük önem arz etmektedir. *Veri yayımı*, mobil ortamlarda veri iletişimini sağlamakta en çok gelecek vaad eden yöntemlerden birisi olarak değerlendirilmektedir. Genel olarak, veri yayımında iki ana yaklaşım mevcuttur: *itme* ve *çekme*. *İtmeye* dayalı veri yayımında, veri mobil kullanıcılara, kullanıcıların profiline veya üyelik bilgilerine göre yayımlanmaktadır. Diğer taraftan, *çekmeye* dayalı veri yayımında ise verilerin aktarımı kullanıcıların açıkça talep etmesi suretiyle gerçekleşmektedir.

Bu tezde, *çekmeye* dayalı ortamlarda, veri yayımının programlanması problemi üzerinde yoğunlaşılmış ve veri yayımını programlayan etkin bir algoritma geliştirilmiştir. Diğer bilinen algoritmalarla başarım sonuçları karşılaştırıldığında, algoritmanın, şimdiye kadar önerilen en iyi algoritmalardan biri olduğuna kanaat getirilmiştir.

Anahtar sözcükler: Mobil sistemler, mobil veritabanı, veri yayımı, çekmeye dayalı, programlama algoritmaları.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest thanks and gratitude to my advisor Assoc. Prof. Özgür Ulusoy for his invaluable supervision and motivating support during this research. I would also like to thank İlker Cengiz and Arzu Özdemir who listened to my long discussions patiently and shared their ideas with me. I am grateful to Assist. Prof. Tuğrul Dayar and Assist. Prof. Uğur Güdükbay for reading the thesis and for their instructive comments. Finally, my thanks go to my mother for her encouragement and support on my continuing in the path of research.

Biricik Anneme.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last two decades, advances in telecommunication and mobile computing has enabled the increasing use of portable wireless computing devices. The rapidly expanding telecommunication technology has made cellular communication, wireless LAN and WAN, and satellite services available for daily life applications. Many people are now carrying portable computing devices at the spectrum from a laptop computer to a Personal Digital Assistant (PDA). All these devices are equipped with an interface for wireless connection to information providers. The resulting *mobile environment* does not require a user to maintain a fixed connection in the network and enables almost unrestricted user mobility.

As the new and various types of mobile infrastructures emerge, users demand new classes of applications running in this environment. Mobile and wireless applications aim to provide the users the required data and services at *any time and anywhere* manner. The first class of mobile applications and services includes *mail enabled* applications and information services to mobile users [22]. Users can receive and send electronic mail from any location at any time. Using user profiles, electronic news services can be delivered to users via e-mail. Information services provide the user with many diverging data sources such as airline schedules, movie programs, weather information, etc. Furthermore, the World Wide Web has emerged as a universal platform for developing and deploying *dissemination-based* applications. Users begin to feel the requirement

of connection to Internet at every pace of life.

The fundamental usage of mobile environment is to have online access to a large number of databases via wireless networks. However, the narrow bandwidth of the wireless communication channels, the relatively short active life of the power supplies of mobile units, and the mobility of clients make the problem of data retrieval more difficult than that in wired networks. Therefore, mechanisms to efficiently transmit information to vast numbers of mobile users are of significant interest. *Data broadcasting* has been considered as one of the most promising ways of data dissemination in mobile environments. There are two basic data broadcasting approaches available: *push* and *pull*. In *push-based* broadcasting approach, data is broadcast to mobile users according to users' profiles or subscriptions, whereas in *pull-based* broadcasting approach, transmission of data is initiated by the explicit request of users.

In each type of the broadcasting approaches, a fundamental design issue is the *scheduling* of the broadcast of data items to the requesting mobile users. Broadcast scheduling algorithms have been developed to determine *which* data item should be broadcast and *when*. The main performance evaluation criterion of such scheduling algorithms is the responsiveness to the client requests. A good scheduling algorithm should deliver the requested data items to the clients as soon as possible. Although a number of methods which aim to minimize the average delay for retrieving data items have been proposed, most of those methods are based on the push-based data delivery which use the access probabilities of data items.

In this thesis, we have focused on the problem of data broadcast scheduling in a pull-based environment and developed an efficient, easy-to-implement scheduling algorithm. Our algorithm utilizes an approximate version of the *Longest Wait First* (LWF) heuristic and implements it efficiently by a bucketing scheme.

The remainder of the thesis is organized as follows. In Chapter 2, we first present the main idea behind data broadcasting and introduce the mobile computing environment that we assume. Then, two main approaches of data broadcasting are discussed. The current research on the scheduling algorithms is

summarized in the last section of the same chapter. In Chapter 3, well-known data scheduling algorithms are examined in more detail. In Chapter 4, after discussing the intuition behind the LWF heuristic, we describe our ATWT heuristic and its implementation, the *Bucketing* scheduling algorithm. Performance evaluation results of the proposed Bucketing scheduling algorithm are provided and compared with the results of other broadcast scheduling algorithms in Chapter 5. Finally, in Chapter 6, concluding remarks are provided.

Chapter 2

Background and Related Work

y

In this chapter, we first present the fundamental idea behind broadcast delivery and introduce the mobile computing environment that we have used in our work. Then, two main approaches of data delivery by broadcast are discussed. In the last section of the chapter, current research on scheduling algorithms of broadcast delivery is summarized.

2.1 Background

Broadcast technology has long been used to deliver information to a large number of users. The early practice of broadcast technology can be exemplified by the radio and television systems [36]. Having known the time schedule, the users can watch and/or listen to the programs. In this way, many users can receive information in a simple and effective way, which emphasizes the main concept of the data dissemination by broadcast delivery.

2.1.1 Data Delivery by Broadcasting

Data broadcast can potentially satisfy the requests of many clients in a single transmission [35]. When some information is broadcast, all pending requests of the multiple users for that information are satisfied at the same time. Generally speaking, the broadcast approach can better utilize the limited bandwidth of a channel and it is particularly suitable for dissemination of large volume of data to a large number of mobile clients at the same time, when compared to *unicast* systems [25]. Traditional *unicast* (*point-to-point*) data services are unpractical because the existing infrastructures are far to meet the demand in both network bandwidth and server capacity, when the system load is high. Furthermore, if any infrastructure was deployed to satisfy the requirements, most of it would be underutilized and wasted during non-peak periods.

Broadcast-based delivery provides an important means for a wide range of applications which are related to dissemination of information to a large population of users. Such dissemination-based applications may be used in electronic commerce, electronic newsletters, mailing lists, road traffic management systems, cable TV, and information feeds such as stock quotes and sports tickets [1, 3, 4]. Data delivery by broadcasting is supported by many new emerging infrastructures provided by satellite and cable technologies [20]. There are many mobile systems deployed overall the world (e.g., GSM [27], CDPD, CDMA, PDC, PHS, TDMA, FLEX, ReFLEX, iDEN, TETRA, DECT, DataTAC, Mobitex [20]). Information dissemination on the Internet has gained significant attention lately. Many commercial services have been developed and operated which enable the wireless dissemination of information available on the Internet (e.g., AirMedia [26], DirecPC [15], etc.). Furthermore, in order to enable mobile users to easily access and interact with information providers and mobile services, there are some protocols developed and deployed. One of the most important and widespread protocols is the *Wireless Application Protocol* (WAP) [28]. WAP has been designed aiming to provide easy and secure access to relevant Internet/intranet information and other services via mobile phones, pagers, or other wireless devices.

A significant facet of broadcast-based systems is their inherent *communication asymmetry* [2, 3]: the volume of data transmitted from server to user is much greater than the volume transmitted in the reverse direction. The communication asymmetry can result from several factors [2]. Generally speaking, in wireless computing, a stationary server is often provided with a relative high-bandwidth channel which supports broadcast delivery to all mobile users located inside the geographical region it covers. The number of users with respect to the number of broadcasting server(s) is much more. The size of data requests is very small compared to the size of responses from the server. The requirement of updating and delivering new information can also cause the communication asymmetry. All the facts mentioned above impose constraints on the design of broadcast systems.

2.1.2 Mobile Environment

The architectural model of a mobile computing environment is presented in Figure 2.1 [12, 13, 22].

In mobile computing, the geographical area is usually divided into regions, called *cells*, each of which is covered and serviced by a stationary controller. A mobile computer system consists of *mobile units* (computers) (MUs) and *stationary computers* (SCs). SCs are connected together via a fixed network. Some of SCs are equipped with wireless interfaces to communicate with the MUs and are called *base* (radio) *stations* or *mobile support stations* (MSSs). MSSs behave as entry points from MUs to the fixed network. SCs communicate over the fixed network, while MUs communicate with other hosts (mobile or fixed) via a wireless channel. MUs can consume and also produce information by querying and updating the online database stored on SCs. MSSs can be proxy servers on behalf of the other SCs or they can themselves be information servers.

It is usually assumed in a mobile environment that there is a single broadcast channel dedicated to data broadcast. Users monitor this channel continuously to get the data item they require. If available, there can be a *backchannel* which enables MUs to send data requests to MSSs.

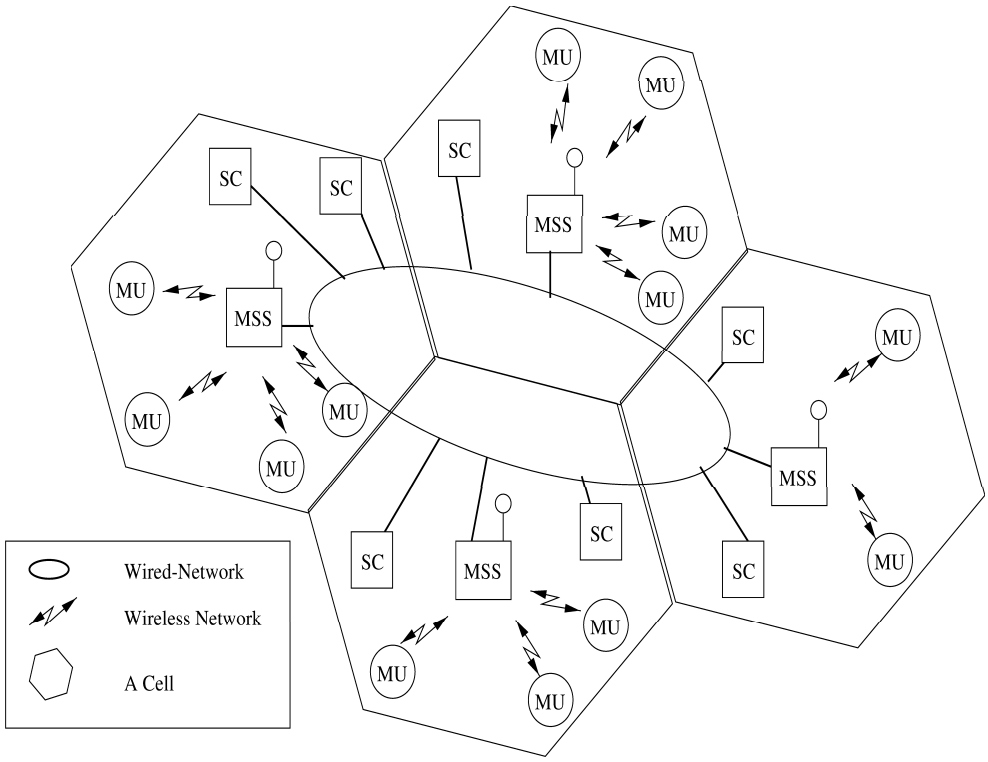

Figure 2.1: Mobile computing environment.

The mobile environment has its inherited limitations and challenges such as [20, 22]:

- frequent disconnections due to some problems (e.g., short battery life, being out of coverage area) or demand of the mobile user,
- limited communication bandwidth,
- security and anonymity,
- mobility management and scalability.

The mobile computing research and development efforts consider all these limitations. For instance, since the bandwidth and the power of the portable devices are scarce resources and need to be carefully managed, efficient design of the query and update operations is of special importance. As a result of these limitations and characteristics of the mobile environment, the techniques required for the design of efficient and cost effective mobile systems are quite different from those developed for systems based on wired networks [22].

One of the most important topics in data dissemination is *efficient* broadcasting of data. Generally speaking, the efficiency is determined mainly by the *mean access time* for data items. In this thesis, the main performance metrics aimed to minimize are *overall mean waiting time*, *variance of waiting time* and *decision overhead* [1]. In Section 5.2, the importance and impact of the performance metrics are discussed in detail.

2.1.3 Push vs. Pull Based Systems

There are two main approaches for data dissemination in broadcast systems; *push-based* and *pull-based* dissemination [1, 2, 22, 36].

In broadcast systems that make use of push-based data delivery, the information server tries to predict the data needs using the knowledge provided by user profiles or subscriptions. The server constructs a broadcast schedule in which initiation of the data transmission does not require an explicit request from MUs. The server repetitively transmits the content of the broadcast schedule to the user population. MUs monitor the broadcast channel and retrieve the items they require as they arrive. That is, accessing the data does not require MUs to use the *backchannel* and is "*listen only*" [22].

Push-based systems have the advantage of their inherent scalability. It is because the performance of any user receiving data is not directly affected by the other users that are also monitoring the broadcast channel. The main idea is that the information servers exploit their advantage in bandwidth by broadcasting the same data to multiple users simultaneously. Using this approach, the cost of data dissemination is independent of the number of users and their requests. The result is scalability and more efficient utilization of the bandwidth. Because of these benefits, many of the previous work on data dissemination in mobile systems are based on the push-based broadcast approach.

[1] There are also some other metrics proposed recently such as the *stretch* metric in [6]. This metric is proposed to evaluate the potential benefits of *preemptive* scheduling. The stretch of a request is the ratio of the waiting time of a request to its service time, where the service time is the time to complete the request if it were the only job in the system. Preemptive scheduling involves interrupting a broadcast to service other requests before resuming the remainder of the original broadcast. The algorithm we propose is not a preemptive algorithm, so we do not use the stretch metric.

One drawback of push-based systems is that, users receiving information from a broadcast channel are *passive*, in the sense that they do not communicate with the server to inform it about their data needs. Therefore, the server lacks valuable information about actual data needs.

In contrast to push-based systems, in a pull-based environment, a system similar to traditional client-server is used. Clients explicitly request data items by sending messages to the server. The requests are compiled in a service queue, and a scheduling algorithm decides which data item is broadcast. After broadcasting, the requests for that page are removed from the queue.

Pull-based systems have the advantage of enabling MUs to be active in obtaining the data they require, rather than only monitoring the broadcast of a push-based server. On the other hand, pull-based systems may have two obvious drawbacks [2]. First, the usage of a backchannel to send the requests of MUs to MSSs can cause additional expense. Second, the server interrupted by continuous client requests can become a scalability bottleneck if the number of clients is large enough.

Besides these two main approaches, some researchers (e.g., [2, 16, 29, 31, 36]) have also proposed *hybrid* approaches in which push-based broadcast of data is combined with support for explicit user requests. The performance of a hybrid approach depends mostly on the allocation of bandwidth between the two types of data dissemination.

The focus of this thesis is a pull-based (*on-demand*) system. Perhaps the most important design issue in this approach is the method used for selecting data items from the user requests to broadcast. Although a number of methods which aim to minimize the average delay for retrieving data items have been proposed, most of them are based on the push-based data delivery which use the access frequency probabilities of data items.

2.2 Related Work

2.2.1 Early Work

With the increasing popularity of wireless computers, services, and underlying technologies, *data broadcasting* and *scheduling algorithms* have attracted the attention of researchers during recent years. Early studies on the problem of schedule design for broadcast information are performed by Ammar and Wong in the context of *teletext* and *videotext* systems [10, 11, 36].

In [36], Wong proposes three alternative architectures for broadcast information delivery systems: *one-way broadcast (push)*, *two-way interaction (pull)*, and *one-way broadcast/two-way interaction (hybrid)*. These alternatives are compared with others using the performance metric of mean response time. For two-way interaction (pull) *First-Come First-Served* (FCFS) scheduling algorithm and some heuristic scheduling algorithms are discussed. *Longest Wait First* (LWF) heuristic has the best performance among all the alternatives tried, including the *Most Requested First* (MRF), *MRF-Low* (MRFL) and FCFS. In recent works [9, 32], the LWF heuristic is shown to be a good choice for pull-based systems as well. Unfortunately, the straightforward implementation of LWF incurs more scheduling overhead than that of other algorithms. In this thesis, we propose an approximation algorithm for LWF by an alternative implementation.

2.2.2 Datacycle and Broadcast Disks Projects

The *Datacycle Project* [17, 21] uses the broadcast paradigm to achieve high throughput database systems. In the proposed architecture, the contents of the entire database are broadcast repeatedly over a high-bandwidth channel (such as an optical system) to *data filters*. Data filters listen to the channel and perform complex associative search operations requested by clients. Compared to Datacycle which broadcasts data using a flat disk to the clients, the *Broadcast Disks* project [1, 3, 37] suggests to employ a multi-level disk. Also, in Datacycle project the clients are allowed to communicate with the host using

"*upstream network*". In Broadcast Disks environment, there is no backchannel. As the broadcast medium, the Broadcast Disks project uses wireless connection.

The Broadcast Disks project integrates the use of bandwidth and client storage to improve response time. The main parts of the proposed system are multiple broadcast programs and client cache management schemes. Prefetching techniques [4] are also employed to exploit broadcast programs more efficiently. In [2], Acharya et al. also study on integrating a backchannel to the system which enables the clients to explicitly request data items from the server. Therefore, the Broadcast Disks environment is transformed to be a hybrid system to investigate the effects of both the pull-based and push-based approaches.

In our work, we assume a pull-based system in developing a scheduling algorithm. In the Broadcast Disks approach for push-based systems, besides broadcast scheduling, cache management of the clients is also taken into consideration. In our system model, we focus our attention to the scheduling algorithm and its performance. For cache management and prefetching, the static broadcast program information is exploited in the Broadcast Disks project [5, 3]. However, the system we use for the broadcast program is dynamic, and the data item to be broadcast next is selected according to actual client requests rather than access probabilities as in Broadcast Disks. The clients are assumed to use any cache management scheme to improve the performance. The data item requests are filtered by the client cache and only the misses are directed to the server as requests. We use an approach like the one in [2] to measure the performance of scheduling algorithms over a specific client whose request pattern is different than that of others.

2.2.3 Work done by Vaidya et al.

Vaidya et al. have worked on data broadcast scheduling algorithms extensively and proposed several scheduling algorithms [18, 19, 23, 24, 33, 34, 35]. In their work, the authors try to minimize the average waiting time and analyze the

scheduling algorithms in the presence of communication errors, and a multi-channel. One of the most important characteristics of the proposed algorithms is that the size of data items is taken into consideration. For various distributions of item size, the overall mean waiting time is observed. In their work, the mobile environment is supposed to be *asymmetric* in that the server has relatively much more bandwidth available than that of clients. In the environment under consideration, the authors suggest that the transmission of information to the mobile clients can be performed efficiently by broadcasting the information periodically. Therefore, their approach is based on *push-based* systems. On the other hand, the authors state that the scheduling algorithm [33] can be applied to a pull-based broadcast environment by replacing access probabilities with the number of pending requests for a data item. Taking this remark of the authors into account, we have modified and tested the algorithm proposed in [18, 33] as to schedule the broadcast in on-demand environments.

The same authors investigate how to achieve an optimal overall mean access time using the underlying assumptions such as arrival of client requests, demand probability and *equal spacing* [33, 34, 35]. Equal spacing means that the time between two instances of an item on the broadcast is always equal. However, each data item can have different spacing. Assuming that instances of each item are equally spaced, they conclude in the theorem called *Square-root Rule*:

> **Square-root Rule:** The access time is minimized when the frequency of an item (in the broadcast schedule) is proportional to the *square root* of its demand (characterized as demand probability) and inversely proportional to the *square root* of its length.

This theorem is accepted as a generalized version of a result presented in [36]. As a result of the theorem, a lower bound on overall mean access time is formulated. However, broadcasting equally spaced data items is not always possible and is hard to accomplish through all the cycles of broadcast. Nevertheless, the theorem can be used to derive a heuristic to select which item to broadcast.

Two different algorithms depending on the square-root rule are proposed: *on-line* and *off-line* algorithms [34]. The on-line algorithm determines which

data item to transmit next in the broadcast. On the other hand, the off-line algorithm selects the data items and prepares the broadcast program for a given cycle size. The difference between the two algorithms is that in the former changing the demand for data items can affect the scheduling easily, but in the latter, entire schedule is decided a priori and broadcast repeatedly. The algorithms are evaluated in the presence of transmission errors and multiple broadcast channels. When transmission errors are considered, it is suggested that large data items should be broadcast in smaller chunks. On the other hand, clients should manage the order of these packages, which is an overhead.

A large broadcast bandwidth can be divided into multiple channels to use the bandwidth efficiently. Clients can listen to one or more of these channels depending on their capacity and/or interests. The algorithms are also modified to use a number of channels [18, 19, 34].

The performance of the proposed algorithms is compared against the optimal overall access time derived from mathematical analysis. Simulation results of the proposed algorithm are close to optimum values. It is also shown that the proposed algorithm is robust in the presence of transmission errors and multiple broadcast channels [33].

In [23], Jiang and Vaidya investigate how the variance of response time can be minimized. They point out that other researchers have usually focused on the overall mean response time of scheduling algorithms as the main performance metric. The variance of response time affects the Quality of Service. Since not all the clients follow the same request pattern with the whole client population, the perceived waiting time can change for different request patterns. Therefore, it is asserted in [23] that if a client's pattern is much different than the average pattern, its own waiting time may be greater than the overall mean waiting time. The authors derive a new relation to minimize the variance using the assumptions stated above. The algorithm presented in [33] is modified to exploit the new relation. The result of the work shows that there is a trade off between mean access time and variance of response time. When one of these two metrics is minimized, the other tends to be maximized. The authors claim that the algorithm, which minimizes the variance of waiting time can be adapted to pull-based systems as well.

In a more recent work, Jiang and Vaidya focus more on the individual client's waiting time [24]. In previous works, the clients are supposed to request a data item and wait until it is broadcast. However, the situation can be different in practical applications. After a certain amount of time, the client can give up requesting the data item. Such "*impatient*" clients require the scheduling algorithm to take into account the *service ratio*. Service ratio is formulated as the fraction of requests that are satisfied before the client withdraws. A cost model is used to maximize the service ratio. The cost in the model is the waiting time. As an analytical result, there is a trade off between mean waiting time and service ratio. The new algorithm considering the service ratio is compared with the previous algorithm proposed in [33], and observed to have better service ratio. Nevertheless, the overall mean waiting time experienced with the new algorithm is worse.

Hameed and Vaidya have developed another broadcast scheduling algorithm [18, 19] based on the *fair queuing* algorithm. The authors relate the scheduling algorithm with the packet fair queuing algorithms. In the fair queuing problem, there are many input channels or queues that use one output channel. The problem is to select from which channel (queue) the item should be transmitted on the output channel. There are two conditions that need to be satisfied [19]:

- Each input queue should use at least a predetermined amount of the output channel bandwidth.
- Bandwidth allocation between the input queues should be evenly distributed, rather than being bursty.

Vaidya et al. point out in the works [33, 34, 35] that they derive a lower bound on the overall mean access time under the assumption of equal spacing of data items. The similarity between the scheduling problem and the fair queuing problem is the usage of the output channel (broadcast bandwidth) and spacing the items evenly. The basic algorithm adapts the solution presented in [14] for the fair queuing problem to the broadcast scheduling algorithm. The algorithm first determines the spacing for each data item. Then, the item with the nearest *broadcast time* is selected. The broadcast time is calculated by

adding the spacing to the last broadcast time. The algorithm's average time complexity is shown to be $O(\log M)$, where M is the number of data items in the database. This algorithm significantly improves the time-complexity over the previously proposed broadcast scheduling algorithms [34, 35] of the authors. By using the base algorithm for environments that are prone to transmission errors and have multiple broadcast channels, new near optimal algorithms are proposed.

2.2.4 A Recent Work: RxW

The work which is most related to our thesis is the one performed by Aksoy and Franklin [8, 9]. Aksoy and Franklin focus on pull-based systems and propose a scheduling algorithm which improves and unifies FCFS and MRF heuristics. The algorithm, called *RxW*, selects the data item with the maximal product of the pending requests number and the waiting time of the first request for that item. RxW is purely an on-demand scheduling algorithm. The authors investigate the algorithm's performance under the criterion they define in their work. Noticing implementation issues, the authors also propose variants of the main algorithm.

In our work, we mainly focus on the same problems. The system model and assumptions used in [8, 9] look like ours, but there are some different parameters and performance criteria considered in our work. We summarize and discuss RxW algorithm in more detail in Chapter 3.

Chapter 3

Previous Broadcast Scheduling Algorithms

3.1 Main Heuristics: FCFS, MRF, MRFL, LWF

There are not many scheduling algorithms proposed for *pull-based* broadcast systems. In the first work in that direction, the *pull-based* approach is called as *two-way interaction* and the performance of a number of scheduling algorithms is discussed [36]. The heuristics used in the *two-way interaction* scheduling algorithms are as follows [36]:

- The well-known *First-Come First-Served* (FCFS) heuristic has been modified such that if a page was requested and placed in the service queue, a new request for that page is ignored. In other words, new requests to a page which has been already queued are placed in the same queue position with the first request. In this way, redundant broadcasts of the same page are avoided [9].
- Another heuristic proposed to be used in broadcast scheduling algorithms is *Most Requested First* (MRF). As the name of the heuristic implies, the page with the largest number of pending requests is selected to broadcast.

In MRF, it is required to record the number of pending requests to a page, whereas in FCFS it is not.

- The MRF heuristic is configured to break ties in favor of the page with the lowest request probability if the request probabilities of the pages are available to the scheduling algorithm. This version of the heuristic is termed *Most Request First Lowest* (MRFL).

- The heuristic which selects the page with the largest total waiting time of all pending requests is the *Longest Wait First* (LWF) heuristic.

These main heuristics for pull-down systems are evaluated in [36] and it is concluded that when the system load is light, the mean response time is not sensitive to the scheduling algorithm used. This is due to the fact that in light loads, few scheduling decisions need to be made. On the other hand, when the system load is high and the page request probabilities follow *Zipf's Law* [38], LWF has the best performance, whereas FCFS has the worst.

3.2 RxW Algorithm

Aksoy and Franklin have also focused on pull-based systems and proposed a scheduling algorithm which improves and unifies FCFS and MRF heuristics [8, 9].

In their work, the authors first discuss the criteria for evaluating scheduling algorithms. They propose to use some performance metrics under three main topics, which are *responsiveness*, *scalability* and *robustness*:

- The *responsiveness criterion* consists of metrics related to the satisfaction of user requests such as *average wait time*, *worst case wait time* and *decision overhead*. The authors define average wait time as the average time from the instant that a client request arrives at the server, to the time that the item is broadcast. The worst case wait time is defined as the maximum amount of time that a user request waits before being satisfied. The

reason to use this criterion is to check if the algorithm causes starvation of some requests, which is an important property for interactive applications. The decision overhead is the time taken to make a new scheduling decision. Broadcast of requests should follow each other without any time gap to make full use of the broadcast bandwidth. Therefore, the decision must be done in an amount of time that is less than the broadcast time of a data item.

- The *scalability* of the proposed scheduling algorithms is tested when the request *arrival rates*, *database size* and *broadcast rate* are varied. These parameters are important characteristics of the mobile environment and combination of them can construct a new environment in which the scheduling algorithm should operate efficiently.

- The effectiveness of approximations and heuristics used in the scheduling algorithms should not be lost when the environment parameters change. This property is called the *robustness* of the algorithm.

The authors present a summary of the scheduling algorithms proposed in [36] and discuss the advantage and disadvantage of them in the view of the criteria given above. They conclude that the LWF heuristic has the best performance according to overall mean waiting time. However, the authors also point out that the straightforward implementation of the heuristic is not practical. On the other hand, other heuristics have their own drawbacks. The FCFS heuristic serves the page according to its arrival time without considering the popularity of the page. The MRF heuristic serves the *hottest* (i.e., most requested) page first, and can cause starvation of *cold* (unpopular) pages. LWF balances the service to hot and cold pages according to their total waiting time in the service queue. A page is broadcast either if it is requested by many clients or it is awaited for a long time. This balance causes the LWF heuristic to have a good overall mean waiting time. After observing this fact, they suggest to integrate the heuristics FCFS and MRF in a practical way to combine their advantages and eliminate the disadvantages.

As a result, the authors propose the RxW heuristic which balances the

selection criterion between the number of pending requests and the first request arrival time of a data item. Basically, RxW would select a data item to broadcast either if it is a hot item or it has at least one long awaited request. Therefore, the RxW heuristic computes the product of the total number of pending requests (R) and waiting time of the first request (W) of that data item, and selects the data item with the maximum RxW value.

The direct implementation of the RxW heuristic is the *exhaustive RxW algorithm.* In the exhaustive RxW algorithm, each requested page has an entry in a hash table with two data fields which are the total number of pending requests (R) and waiting time of the first request (W) of that data item. When a request arrives, if it is the first request for that page, a new entry for the page is created. The page's R field is set to one, and W field is assigned the current time. Otherwise, if the entry for that page is already created, only the R field is incremented by one without modifying the W field. In order to select a page to broadcast, all the entries are examined and their RxW values are computed. The page with the maximum RxW value is broadcast.

It is clear that the exhaustive RxW algorithm runs in $O(N)$ time, where N is the number of the items in the hash table. To improve the running time, a pruning technique is employed by changing the implementation of the heuristic. In the new algorithm, called *maximal RxW*, the authors apply two lists, W and R lists; one is increasingly ordered according to arriving time of requests (as in FCFS), and the other is decreasingly ordered according to the pending request numbers of each page (as in MRF). The maximal RxW algorithm scans these two lists in turn to find the entry with the maximal RxW value. The pruning technique prevents the algorithm searching all the entries of the R and W lists by setting limits on the values of entries in both lists. If the maximal RxW algorithm encounters an entry with less or equal value to the limit value on a list, then it ceases to search and returns the data item with the largest RxW value computed so far to broadcast.

The maximal RxW algorithm begins to search with the R list. After computing the RxW value of the first entry in the R list, the MAX variable is set to that value. In order to determine the limit value for the W list, the MAX value is divided by the next entry of the R list. Since the R list is sorted in

descending order, it is known that for any unexamined entry to have an RxW value greater than MAX, it must have a W value greater than MAX/R. Therefore, the entries with the W value less than the limit are not to be considered in searching the largest RxW value, and this truncates the W list. The limit on the R list is set by a similar reasoning. The maximal RxW algorithm searches the lists until it hits a limit or it reaches half of the lists. Then, it is known that MAX holds the maximal RxW value. In the experimental results provided in the work, it is observed that the pruning technique checks only %27 of the entries to find the entry with the maximal RxW value.

Furthermore, the authors develop a *parameterized* version of the algorithm in which the percentage α of the entries to be compared can be identified. For instance, if the parameter α is set to 0, then only the first entries of both lists are considered in the comparison and the one with the largest RxW value is broadcast.

The results of simulation experiments are discussed in the view of the criteria given above. It is observed that the less entries the algorithm compares, the more the mean waiting time worsens. That is, the parameterized version of the algorithm with low values of α causes worse mean waiting time than the maximal RxW algorithm. On the other hand, the comparative performance results observed with the worst waiting time is the opposite; i.e., the parameterized version of the algorithm causes smaller worst waiting time compared to the maximal RxW algorithm. Another benefit of the parameterized version of the algorithm is the low scheduling decision overhead compared to the maximal RxW algorithm.

As a result, the authors conclude that the proposed RxW algorithm provides good performance across all of the criteria considered and can be tuned to trade off the average and worst case waiting times by using the parameterized version of the RxW algorithm.

Chapter 4

Bucketing Algorithm

We have aimed to develop a scheduling algorithm which can minimize both the mean waiting time and its variance, as well as is robust and easy to implement. The related work has shown that the Longest Wait First (LWF) heuristic is the best compared to other proposed algorithms with respect to the main performance criterion, *mean waiting time*. Therefore, in this chapter, we first try to identify the reasoning behind the good performance of the LWF heuristic. Then, on the basis of our observation, we describe a new heuristic which we name *Approximate Total Waiting Time* (ATWT). The proposed ATWT heuristic is implemented using a bucketing scheme and the resulting algorithm is termed the *Bucketing Algorithm*. We also present a variant of the heuristic to handle unequal page sizes, as well as a modified version of it that can be tuned to trade off performance with the mean waiting time and the variance of the waiting time.

4.1 Intuition Behind the Longest Wait First Heuristic

In [9] and [36], it is stated that the LWF heuristic has better performance than other proposed heuristics when the overall mean waiting time is considered. However, the implementation of LWF is subject to more scheduling overhead

than that of other competing heuristics. This is because, the LWF heuristic should calculate the total waiting time of each requested page to decide which page to broadcast next. The computation can easily become a bottleneck for a system in which the broadcasting of a page takes less time than the time required for the calculation of the pending requests' waiting time. For example, if the system is characterized by a large database, a high-bandwidth, relatively small page sizes, and almost uniformly distributed requests, then the computation can lead to cease the broadcasting. Since time is precious, this would not be acceptable.

Other heuristics have their own drawbacks. The First-Come First-Served (FCFS) heuristic serves the page according to its arrival time without considering the popularity of the page. The Most Requested First (MRF) heuristic serves the hottest (i.e., most requested) page first, and can cause starvation of unpopular pages. LWF balances the service to hot and cold pages according to their total waiting time in the service queue. A page is broadcast either if it is requested by many clients or it is awaited for a long time. This balance causes the LWF heuristic to have a good overall mean waiting time. This conclusion is reasonable and stated by some other researchers as well [9, 36]. In the rest of this section, we present a different view in proving the effectiveness of the LWF heuristic.

As we focus on minimizing the overall mean waiting time of the pending requests, we may try to keep the overall mean waiting time as low as possible. For this reason, the page which contributes to the overall waiting time most may be selected to broadcast at each broadcast cycle. If a page has a negative effect on system's waiting time, by selecting and broadcasting it, we may prevent it to worsen the situation. Because, if the pending requests of that page wait longer, then they can contribute more to the overall waiting time. Therefore, we may propose a method to determine the effect of each requested page on the overall mean waiting time.

It is assumed that the time (t) is measured by a specific unit called *tick* and each page takes one tick to broadcast. Therefore, each cycle of broadcast can be enumerated by the successive values of ticks such that while time t shows the current time, the next cycle of the broadcast takes place at time $t+1$. Let

the overall mean waiting time of the system at time t be *M(t)*, the satisfied requests' total waiting time until time t be *W(t)*, and the number of satisfied requests until time t be *R(t)*. Then,

$$M(t) = \frac{W(t)}{R(t)} \tag{1}$$

Furthermore, let an individual requested page be associated with similar parameters such that $M_p(t)$ is the mean waiting time for page p, $W_p(t)$ is the total waiting time of the pending requests and $R_p(t)$ is the number of the pending requests for that page at time t. That is, for a requested page p, the mean waiting time is formulated as

$$M_p(t) = \frac{W_p(t)}{R_p(t)} \tag{2}$$

Now we can relate the overall mean waiting time of the system with an individual page's effect on it. If we select and broadcast page p, then the mean waiting time of the system at the next broadcast cycle will be

$$M(t+1) = \frac{W(t) + W_p(t)}{R(t) + R_p(t)} \tag{3}$$

The total number of satisfied requests for all pages will be sufficiently larger than the number of pending requests to a single page after broadcasting a considerable number of pages. Therefore, we may ignore $R_p(t)$ in the summation $R(t)+R_p(t)$ of Formula 3. Then, the formula becomes

$$M(t+1) = \frac{W(t) + W_p(t)}{R(t)} \qquad where \quad R(t) >> R_p(t) \tag{4}$$

$$M(t+1) = \frac{W(t)}{R(t)} + \frac{W_p(t)}{R(t)} \tag{5}$$

$$M(t+1) = M(t) + \frac{W_p(t)}{R(t)} \tag{6}$$

Formula 6 implies that the page with the largest total waiting time would have the largest effect on the system's overall mean waiting time. So, if we do

not select this page to broadcast, its waiting time can get larger worsening the overall performance. If we want to minimize the overall mean waiting time, we should select a page with the largest value of $W_p(t)$. This observation could be another way of confirming the fact that the LWF heuristic is the best among all heuristics proposed for broadcast scheduling.

However, as mentioned before the direct implementation of the LWF heuristic can cause a considerable overhead. Therefore, we try to devise an approximation for the implementation of LWF.

The problem with the implementation of the LWF heuristic is the requirement to compute the total waiting time for each data item with pending requests. At each time click, this value of a data item changes even if there is not any new request for the item. We try to find an approximate formula to calculate the total waiting time on a data item. Normally, we should record the arrival time of each request to a page. Whenever the total waiting time of a page is needed, we subtract the requests' arrival times from the current time to find the waiting times of each request. Then, these waiting times are summed up to find the total waiting time for that page. It is clear that this computation is not a negligible overhead for the system.

4.2 Approximate Total Waiting Time

In order to decrease the amount of computation, we assume that all requests for a page come at the same time as the first one. That is, we only keep the arrival time of the first request for each page. When we need to compute the total waiting time of a page, we can simply multiply the number of pending requests with the elapsed time since the first request arrived. This approximation gives us the upper bound of the total waiting time of a page. If the clients' requests' arrival is governed by the Poisson process, then it follows that if a page is broadcast τ time units after the arrival of the first request to it, the mean waiting time for pending requests for this page is $\frac{\tau}{2}$ [33, 10]. This fact gives an approximation to compute the total waiting for a page as follows:

$$W_p(t) = \frac{t - A_p}{2} * R_p(t) \tag{7}$$

where t is the current time, A_p is the first request arrival time, and $R_p(t)$ is the total number of pending requests for page p at time t. $W_p(t)$ is the approximate total waiting time of page p. The LWF heuristic needs to compute the total waiting time for every page to select the one with the largest value. We can drop the division by 2 in Formulation 7 to simplify the calculation since $W_p(t)$ of each page will be compared. This finalizes the basic formulation, that we call *Approximate Total Waiting Time* (ATWT), to select the data item which has the most adverse effect on the overall mean waiting time of the system. ATWT enables us to record less information and do less computation. Through simulation experiments, we show in Section 5.3 that this approximation works well.

The approximation we provide for total waiting time has also been suggested by Aksoy and Franklin but through completely different reasoning and observations from ATWT [9]. In deriving their approximation, they point out the drawbacks of the FCFS and MRF heuristics, and the implementation overhead of the LWF heuristic. They suggest to integrate the heuristics FCFS and MRF in a practical way to combine their advantages and prevent the disadvantages. As a result, they propose the RxW heuristic which balances the selection criterion between the number of requests and the arrival times. In implementing the heuristic, they apply two lists; one is ordered according to arriving time (FCFS) of requests and the other is ordered according to the pending request number (MRF) of each page. This data structure is also different from ours that is discussed in Section 4.4.

4.3 Finding the Maximal ATWT

Even if we simplify the calculation of total waiting times of pages, we still suffer from the evaluation of this formula for each page being requested to find the page with the maximum ATWT value. Therefore, the direct implementation of the heuristic we propose above has a time complexity of $O(N)$, where N is

the total number of requested pages.

In order to avoid the calculation of each requested page's ATWT, we use a method which selects a few pages and calculates only their ATWTs to select the page with *maximal ATWT* value. Our implementation is based on a bucketing technique [1].

We classify the pages according to the number of pending requests associated with them. All the pages that lie in bucket i will have pending request numbers ranging between 2^{i-1} and 2^i-1. The number of buckets is limited by the number of pending requests for distinct pages. There will be $\lceil log(R+1) \rceil$ buckets of pages, where R is the number of pending requests of the most requested page in the system. In each bucket, the pages are ordered according to their first request arrival time. The first page of each bucket is the first requested page within that bucket.

Whenever we need to find the page with the maximal ATWT, we compare only the ATWT values of the first pages of each group. Since the number of buckets is logarithmic with respect to the most requested page's request number, we would examine very few candidates. The page with the largest ATWT value is selected among the first entries of all buckets.

It can be shown that the bucketing scheme results in selecting a page with an ATWT value which is at least half of the maximum ATWT value. In bucket i, it is given that the total number of requests of pages is ranging between 2^{i-1} and 2^i-1. That is, the total request number of a page is at most two times larger than that of any other page that lies in this bucket. Let page p_1 be the first entry in the bucket. Page p_1 has the arrival time A_{p_1} which is less than or equal to the arrival times of other entries of bucket i. Furthermore, assume that page p_m has the maximum ATWT value in this bucket. In the worst case, page p_m can have a total number of requests twice larger than that of page p_1 and the first request arrival time of page p_m can be equal to that of page p_1.

[1] There exist some similar bucketing schemes proposed in different contexts in the literature (e.g., [7, 33]). As an example, in [7], the bucketing technique is used in the context of Web object caching. The authors propose to extend the *Least Recently Used* (LRU) policy to handle varying size objects. They group the Web pages according to their size and each bucket of pages is treated as a separate LRU list. The bucketing scheme used in that work is called *The Pyramidal Selection Scheme.*

That is,

$$R_{p_m} < 2 * R_{p_1} \tag{8}$$

$$A_{p_m} >= A_{p_1} \tag{9}$$

The maximum ATWT value of bucket i, which is the ATWT of page p_m is:

$$(t - A_{p_m}) * R_{p_m} \tag{10}$$

Using inequalities 8 and 9, we obtain:

$$(t - A_{p_m}) * R_{p_m} < (t - A_{p_m}) * (2 * R_{p_1}) \tag{11}$$

and thus

$$(t - A_{p_m}) * \frac{R_{p_m}}{2} < (t - A_{p_1}) * R_{p_1} \tag{12}$$

This concludes that the first entry of each bucket has an ATWT value which is at worst half of the maximum ATWT value in that bucket.

In practice, the maximal ATWT value found by the bucketing method and the maximum ATWT value are close to each other, so that the performance is not affected much by this selection. We prove this fact through simulation experiments in Chapter 5. The results show that the bucketing technique is a good way to decrease computation considerably without degrading the performance much.

4.4 Bucketing Algorithm

The data structure used for each requested page in the Bucketing algorithm is illustrated in Figure 4.1. The field A holds the arrival time of the first

request, and the field R holds the number of pending requests for that page. Each bucket is a linked list of requested pages. Pages are ordered in the linked lists according to the first request arrival time. The fields *Prev* and *Next* are pointers to the previous and next pages in the linked list [2].

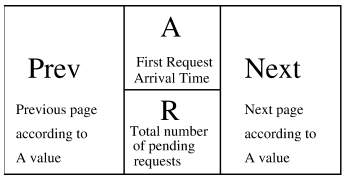

Figure 4.1: Page data structure.

Entries for pages are placed in the buckets by mapping the total number of requests to the bucket number. A page with a total request number i will be placed to bucket $\lfloor log(i) \rfloor + 1$. For example, while a page with three pending requests is placed to the second bucket, a page with 32 pending requests lies in the sixth bucket. In other words, bucket i contains entries for the pages with the pending request numbers varying between 2^{i-1} and 2^i-1. A snapshot of the Bucketing algorithm's data structure is provided in Figure 4.2.

The Bucketing algorithm works as follows: When a request arrives to the server, if it is the first request for the page, its arrival time is recorded to the A field of the page data structure and the number of pending requests (R) is set to one. The page is placed at the end of the linked list in the first bucket since its R value is 1.

Otherwise, if the page was requested and not yet broadcast, the number of pending requests (R) is incremented by one. Then, if the page does not belong to the existing bucket anymore, it is moved to the appropriate bucket according to its R value. The page is then inserted in the linked list of this bucket with respect to its A value.

In the selection of the page to broadcast, only the first page of each bucket is examined. The page with the largest ATWT is broadcast and removed from the bucket. The bucketing scheme reduces the decision overhead considerably

[2]For performance concerns, instead of using a linked list data structure, a heap data structure can also be implemented to store the items in each bucket. However, for the sake of simplicity we prefer to implement linked list data structure in the simulation.

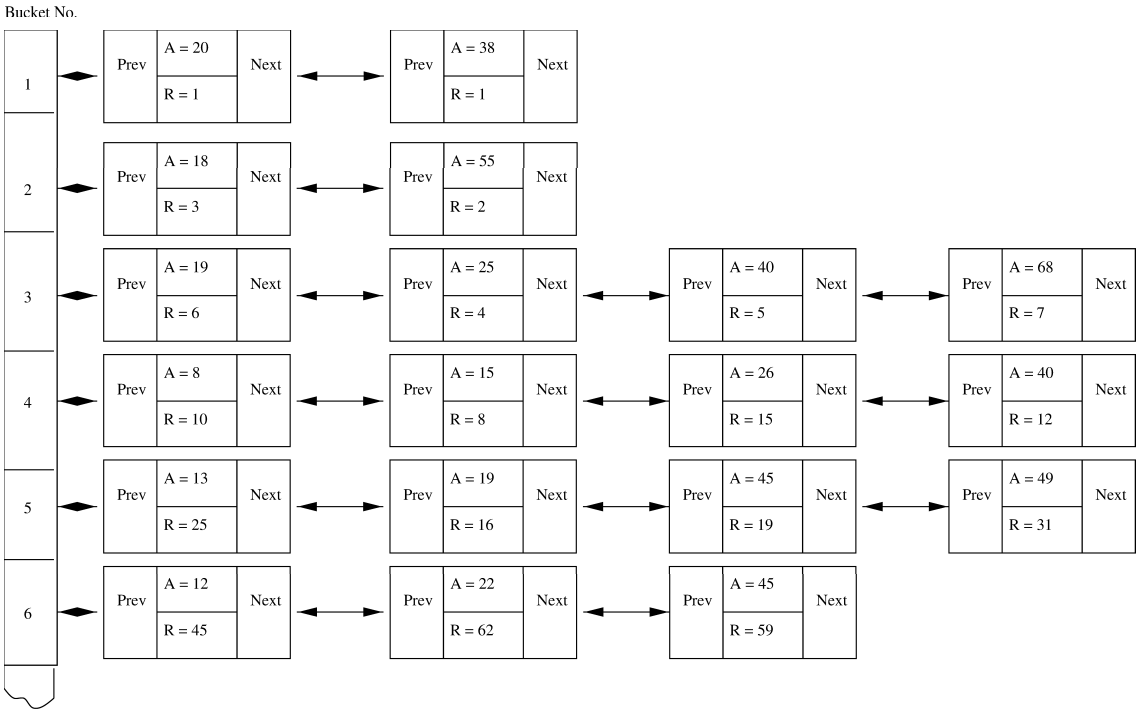

Figure 4.2: The pages are in buckets of exponentially increasing total request number.

without deteriorating the quality of the produced broadcast.

We have also implemented a variant of the Bucketing algorithm, called *k-depth* Bucketing algorithm, in which we examine the first k entries of each bucket. By comparing more entries, it is expected to have more accurate ATWT. The results obtained from the simulation of k-depth Bucketing algorithm in Section 5.8 indicate that when we increase the depth, the overall mean waiting time is improved, however, the decision overhead increases as well. This trade-off can be managed by setting k to different values.

4.5 Handling Pages with Unequal Sizes

As discussed in Chapter 3, it is assumed that the sizes of data pages are uniform. In this section, we examine the situation in which the pages do not have to have the same size.

Vaidya et al. proposed several scheduling algorithms to broadcast data

items with unequal size [19, 33, 34, 35]. After mathematical derivations, the authors suggest heuristics in which the data item size is inversely proportional to the selection criterion.

In our simulation results, we observed that the overall mean waiting time is approximated well when the ATWT value of a page is divided by the size of that page. In this variant of the heuristic, the modified ATWT value of a page p is computed as:

$$\frac{(t - A_p) * R_p(t)}{s_p} \tag{13}$$

where s_p is the size of page p. The intuition behind (13) is to give precedence to small sized pages in selecting the page to broadcast. This was observed to reduce the total waiting time of other requests [19].

4.6 Minimizing the Variance of Waiting Time

Like many other researchers, we have so far focused on the overall mean waiting time achieved by the proposed heuristic. However, there is another important performance metric that should be considered. *Variance of the waiting time* is the second criterion we would like to minimize. This performance metric also affects the Quality of Service. Since not all clients follow the same request pattern, the waiting time perceived by an individual client might be different from the overall mean waiting time.

We have investigated the variance of waiting time caused by ATWT and several other heuristics, and the effect of variance on the individual waiting times through simulation. One of the most important results we obtained is the inverse relation between the overall mean waiting time and the variance of waiting time in such a way that, if a heuristic is good at one of the two metrics, it is not good at the other. For example, FCFS has better performance than most of the heuristics we examined according to the variance of the waiting time metric, however, it has worse performance when the overall mean waiting

time is considered. A similar observation for push-based systems is declared by Jiang and Vaidya as well. In [23], the authors investigate how the variance of response time can be minimized. The algorithm presented in [33] is reformulated considering the variance metric and a new algorithm called *α-algorithm* is proposed. It is shown in that work that there is a trade-off between the mean waiting time and the variance of waiting time. When one of these two metrics is minimized, the other tends to be maximized. The authors claim that the algorithm can be adapted to pull-based systems as well. We modify the computation of ATWT in our heuristic in a way similar to that suggested in [23] as follows:

$$(t - A_p)^{\alpha} * R_p(t) \tag{14}$$

where α can be assigned different values in order to tune the variance of waiting time and the mean waiting time. For instance, when α is set to 1, the original ATWT heuristic is obtained and the overall mean waiting time is attempted to be minimized. On the other hand, when α is set to higher values than 1, the heuristic attaches more importance to the first request arrival time than the total pending request number as in the FCFS heuristic, and the variance of the waiting time is attempted to be minimized. As a result, there are two extreme cases in which one of the two metrics is the best.

Through simulation experiments, we observe that although the α parameter values used in [23] is different from that in our work, the results and implications are very similar. By setting α to different values, the system performance can be optimized according to the chosen performance metric.

In order to see the impact of the variance in the waiting time over an individual client's waiting time, we have also simulated a client whose access pattern deviates from the the rest of the client population. We have observed that if the variance of mean waiting time is minimized, then the waiting time perceived by the client can be improved.

Chapter 5

Performance Evaluation

We have simulated the mobile environment introduced in Section 2.1.2 and performed extensive experiments in order to evaluate the performance of the Bucketing algorithm relative to other scheduling algorithms. The simulation program was written in CSIM [30]. The simulation model and the performance results are provided in the following sections.

5.1 Simulation Model

Our simulation model consists of three main components: a *mobile support station* (MSS), a population of *mobile units* (MUs), and *communication channels* as depicted in Figure 5.1. MSS, which is a pull-based server, contains the necessary parts and mechanisms to simulate the broadcast delivery. Published requests are kept in the *service queue*. The *online database* stores the shared data items. The decision process is performed by a *scheduling algorithm*. Client population represents MUs within the cell. The communication channel is a two-way medium. In the *broadcast channel*, selected data items are delivered to MUs, whereas the *backchannel* is used to send data requests of MUs to MSS.

Simulation parameters and their values are summarized in Table 5.1. We assume that MSSs are information servers and they serve the demands of MUs from the database located on themselves. Let *dbSize* be the total number of

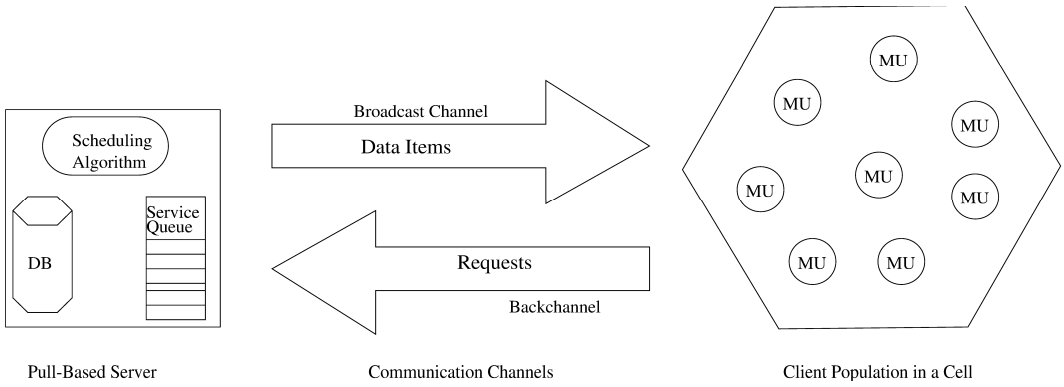

Figure 5.1: Simulation system model.

Symbol	Description	Default	Range	Unit
λ	Mean Req.Arrival Rate	10	[10-100]	req./tick
Θ	Request Pattern Skewness	1.0	[0.1-1.0]	-
dbSize	Database Size	1,000	[1,000-10,000]	pages
pSize	Page Size	1	-	tick

Table 5.1: Simulation parameters

available data items at the MSS. Data items are numbered from 1 to dbSize, where a data item is, for example, a web page or a file. We use the terms *data item* and *page* interchangeably, since the information server can be a database or web server.

The requests of MUs are represented by a single request stream. The request arrivals are assumed to be *Poisson* with a mean value of λ. By increasing λ, we can simulate a higher system load. MUs may exhibit data locality, querying a particular subset of the database repeatedly [13, 22]. This subset is a *hot spot* for an MU. In general, a user may request multiple items simultaneously and would expect to receive mutually consistent versions of the requested items. In this thesis, similar to many of the past work, we consider the case where a user demands only one item per request, and unless the user gets the item, a new request is not initiated. Furthermore, we assume that MUs can cache the requested data items. Caching of frequently accessed data items is essential to reduce the contention on the narrow bandwidth wireless channel between a user and a server. In our work, the effect of transmission errors is not considered.

We assume that when a data item is broadcast, all the users requesting that item receive it completely.

It is assumed that access probabilities follow the *Zipf* [38] distribution over the database items as in many other related work (e.g., [3, 9, 36]). Data items are supposed to be ordered in the database according to their access probabilities in decreasing order, i.e., the most favorite data item is in the first place in the database. Zipf's law states that the relative probability of a request for the i'th most popular data item is proportional to $\frac{1}{i}$, where i is between 1 and the total number of data items in the database ($dbSize$). The Zipf distribution can be formulated to show the demand probability of each data item as below:

$$p_i = \frac{(1/i)^{\Theta}}{\Sigma_{i=1}^{dbSize}(1/i)^{\Theta}} \tag{1}$$

where Θ is a parameter termed *access skew coefficient* [35]. By changing the value of access skew coefficient Θ, different Zipf distributions can be obtained. For instance, if Θ is set to zero, Zipf distribution turns out to be a uniform distribution with $p_i = 1/dbSize$.

The time to broadcast a data item is calculated by a specific time unit called *tick*. Instead of specifying the bandwidth of the broadcast channel, we assume that page sizes ($pSize$) are equal and each page can be transmitted in a specified number of ticks. The use of tick as a time unit enables us to compare easily the results of systems with different properties such as bandwidth and data item size. We assume that each page is broadcast in one tick, when the size of all the pages is the same. Mean waiting time is measured by the number of ticks.

MSS selects the data item to broadcast under the control of a scheduling algorithm and then initiates the broadcast. Since the model is a pull-based system, after completing the broadcast, the scheduling algorithm removes the request for the broadcast data item from the queue. Then, the requests in the service queue are processed to select another data item to be delivered.

For evaluating a broadcast scheduling algorithm for a particular set of parameters, the broadcast schedule is produced for at least 30,000 cycles [1]. Furthermore, we run each configuration ten times and use the averages as final estimates.

Database (dbSize) is assumed to consist of 1,000 data items. However, in order to examine the effects of database size on performance, dbSize is increased up to 10,000.

5.2 Performance Criteria

We have evaluated our Bucketing algorithm and compared it with other scheduling algorithms with respect to the following performance criteria:

- *Waiting time of a request* is defined as the duration of time from when the request is made until the desired data item begins to be transmitted on the channel. The primary performance metric used in evaluating the quality of the broadcast scheduling algorithms has long been the overall mean waiting time. It is because the minimization of the waiting time reduces the idle time of MUs. Since portable devices have scarce resources and the connection cost is high, users should get the response as soon as possible.

- *Variance of waiting time* can also be taken into consideration to evaluate the Quality of Service experienced by any user, where the overall mean waiting time is an indication of the idle time for the whole user population. An individual user whose request pattern is different from the overall pattern may be associated with a different mean waiting time than the overall mean waiting time [23].

- *Worst waiting time* is defined as the maximum amount of time that any user request waits before being satisfied. The reason to use this criterion

[1] We have observed the mean waiting times and found out that for all the scheduling algorithms the mean waiting time is stabilized at most after the first 5,000 broadcast ticks for the given values of simulation parameters. In order to record more accurate values of the worst waiting time and variance of waiting time, and to check if starvation occurs, we have run the simulation for a considerably larger time period, i.e., for 30,000 broadcast ticks.

is to check if the algorithm causes starvation of some requests, which is an important property for interactive applications [9].

- Scheduling algorithms examine the requested data items to select one of them to broadcast. In this process, decision making may take an important amount of time of the scheduling algorithms and might cause time gaps between broadcasts, which leads to an inefficient use of the broadcast channel. *Decision overhead* is the time taken by the computation which should be done for selecting a data item to broadcast next. The decision overhead of a good scheduling algorithm should not be high.

5.3 Evaluation of Approximations used in the Bucketing Algorithm

As discussed in Chapter 4, the ATWT heuristic and its implementation through the Bucketing algorithm, are based on two approximations. We have suggested to use Approximate Total Waiting Time (ATWT) instead of computing the actual total waiting time in order to decrease the computation overhead. Even if we simplify the calculation of total waiting time, we still need to evaluate the ATWT value for each data item being requested to find the page with the maximum ATWT value. In order to avoid the calculation of each requested page's ATWT, we use the bucketing scheme which selects a few pages and calculates only their ATWTs to select a page with *maximal ATWT* value. In our first experiment, we evaluate the impact of these approximations on the system performance.

First, we compare the ATWT value with the actual total waiting time. For the parameter values given in Table 5.1, the corresponding values obtained for the ratio of ATWT to actual waiting time are displayed in Figure 5.2.

As seen in Figure 5.2, the ATWT value is at least 73% of the actual total waiting time. If the system workload is high, the approximation of ATWT to the actual value is even better. As discussed in the derivation of ATWT in Section 4.2, we suppose that the clients' requests' arrival is governed by

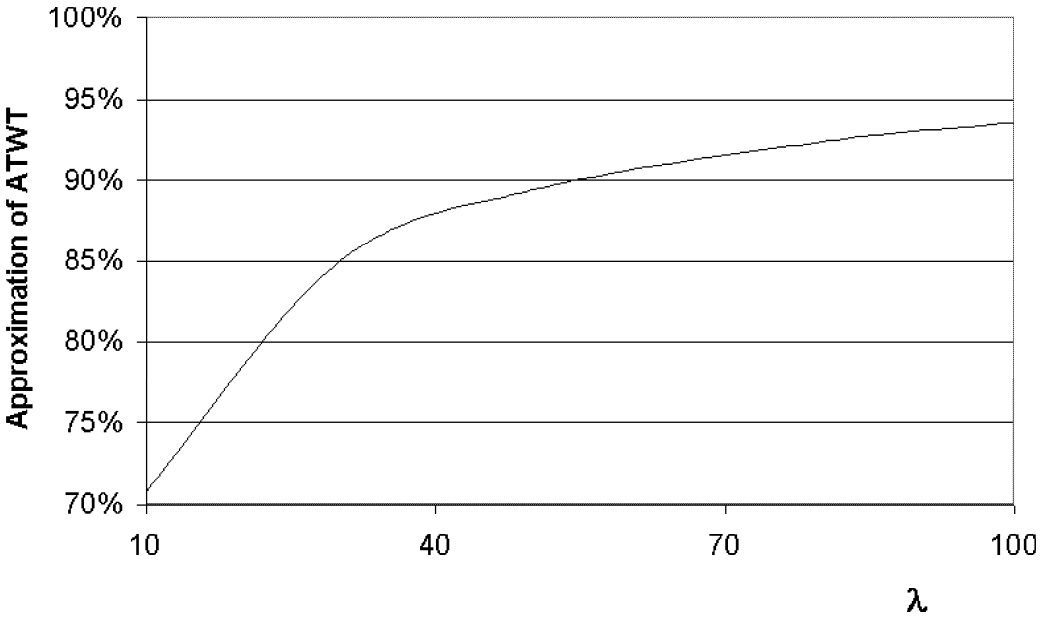

Figure 5.2: Ratio of the ATWT value to the actual total waiting time.

the *Poisson process*. However, the distribution of the requests over the data items is not uniform, and we assume that it is a *Zipf distribution* [38]. In light workloads, as the distribution is not uniform, some data items are more frequently requested, compared to the others. This leads the approximation to predict less accurate values. On the other hand, when the system load is high, i.e., there are more requests in a given time period, even cold data items get frequent requests, which enables the approximation to display more accurate results.

We have used the second approximation in implementing the ATWT heuristic, which is the bucketing scheme. Instead of searching all the data items' ATWT values to find an item with the maximum value, we group in the buckets the data items according to their pending total request numbers, and compare only ATWT values of the first entries in the buckets. Therefore, the bucketing scheme selects a data item with a maximal [2] ATWT value.

We have also compared the maximal ATWT found by the bucketing scheme

[2]The ATWT value found by the bucketing scheme may not always be the maximum. On the other hand, it is shown in Section 4.3 that the maximal ATWT value is close to the maximum ATWT value.

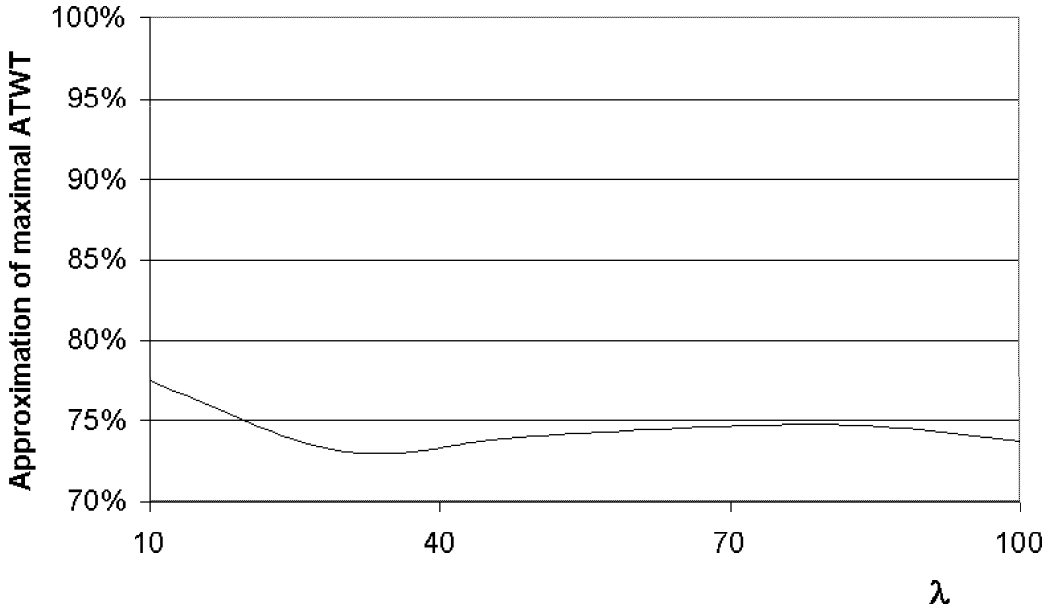

Figure 5.3: Approximation of maximal ATWT to maximum ATWT.

and the maximum ATWT existing in the system. Furthermore, we have evaluated the effect of using more accurate values of ATWT on the overall mean waiting time of the system. Using the same parameter values provided in Table 5.1, we have observed that the ratio of the maximal ATWT to the maximum ATWT is around 0.75 (see Figure 5.3).

The result obtained for this ratio seems to be unrelated with the workload of the system. It might be asserted that the approximation is low, but the effect of the selection of a data item with a smaller ATWT value than the maximum value is not much influential on the observed overall mean waiting time. In order to see this effect, we have increased the *depth* property of the bucketing scheme and observed that the maximal values are getting very close to the maximum values (see Table 5.2).

Even if the approximation of the maximal ATWT to the maximum ATWT is not very accurate, the overall mean waiting time does not worsen much. In other words, using the maximal ATWT instead of the maximum ATWT does not deteriorate the performance of the scheduling algorithm deadly. Therefore, the approximation is practical to be used, considering the decision overhead [3].

[3]It could be argued that if the higher approximation of maximal ATWT does not yield

Depth	Approximation of Maximal ATWT	Mean Waiting Time
50	98.6%	214.1
25	87.1%	215.6
10	79.9%	216.3
1	77.4%	217.5

Table 5.2: Effect of using maximal ATWT on overall mean waiting time.

5.4 Mean Waiting Time

In this section, we examine one of the most important performance criteria, waiting time of a request which is defined as the duration of time from when the request is made until the desired data item begins to be transmitted on the channel. The importance of this metric for the scheduling algorithms has been discussed in Section 5.2.

In the first experiment, we have implemented our Bucketing algorithm as well as four other scheduling algorithms which are discussed in detail in Chapter 3. Performance results of the scheduling algorithms in terms of the mean waiting time are displayed in Figure 5.4 for the simulation parameter values given in Table 5.1.

In this experiment, the mean request arrival rate (λ) is varied from 10 requests per tick to 100 requests per tick. The database size is 1,000 pages. In this figure, it can be seen that the mean waiting time for all the algorithms is increasing while the request arrival rate is getting higher. However, after a certain rate, it levels off. This result was also observed in [9].

As can be observed in Figure 5.4, the Most Requested First (MRF) and

a significant improvement on the mean waiting time, then it is questionable to try to select an entry with the maximum ATWT. It is reminded that ATWT itself is an approximation of LWF. It does not always mean that when a data item with the maximum ATWT is selected, the data item should have the maximum total waiting time as LWF requires. This observation is due to the approximation made in the computation of ATWT. The improvement observed in the mean waiting time seems to be small, but it does not mean it is not significant. Because, even the depth is one, and even the approximation of maximal ATWT to maximum ATWT is not at higher level, we have the mean waiting time very close to the original LWF algorithm (see Figure 5.4) .

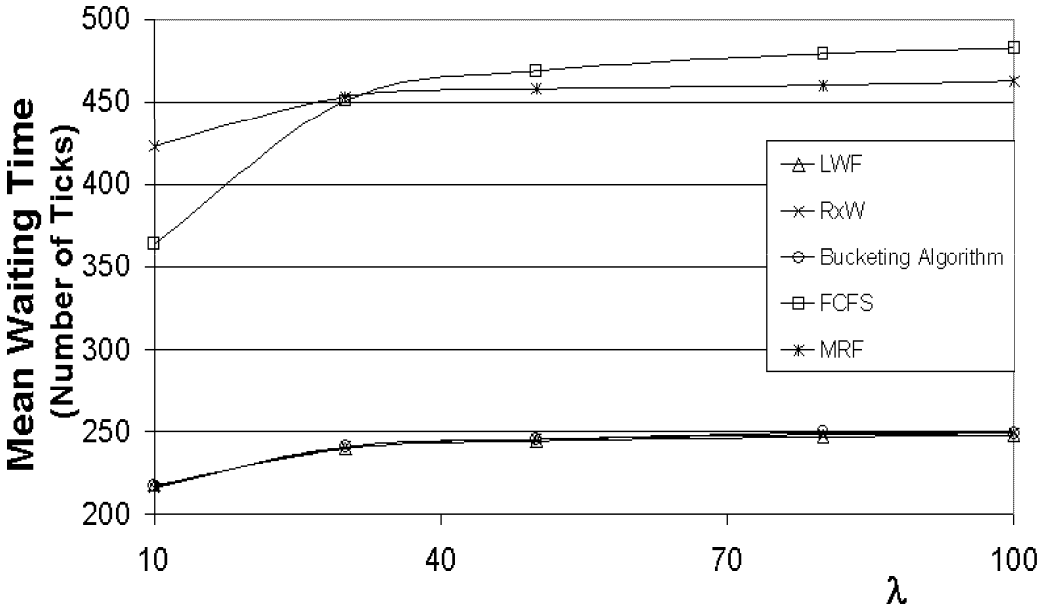

Figure 5.4: Mean waiting time of several algorithms.

First-Come First-Served (FCFS) algorithms have much larger mean waiting time than the other algorithms. The Longest Wait First (LWF), RxW, and Bucketing algorithms are characterized by almost the same mean waiting time. The largest difference between any two of these three algorithms is not more than 0.8%. In order to see the difference between these three algorithms, Figure 5.5 with a smaller scale of mean waiting time is provided.

Even if LWF is a good algorithm with respect to the mean waiting time metric, it has some drawbacks as discussed in Chapter 4. The straightforward implementation of the LWF heuristic is not practical for large databases and high-speed broadcast channels. We give the results of the LWF algorithm in the experiments for comparison with its approximation Bucketing algorithm and the other algorithms, i.e., FCFS, MRF, and RxW. The first observation about the algorithms shows that RxW and Bucketing algorithm are the only two algorithms which are practical to implement and satisfactory in terms of mean waiting time results. FCFS has the best values for some metrics, like variance of waiting time and worst waiting time, but we can not use it due to its unacceptable results for the main performance metric, the mean waiting time. Therefore, in evaluating the performance of the algorithms, we have concentrated on the results of RxW and Bucketing algorithms.

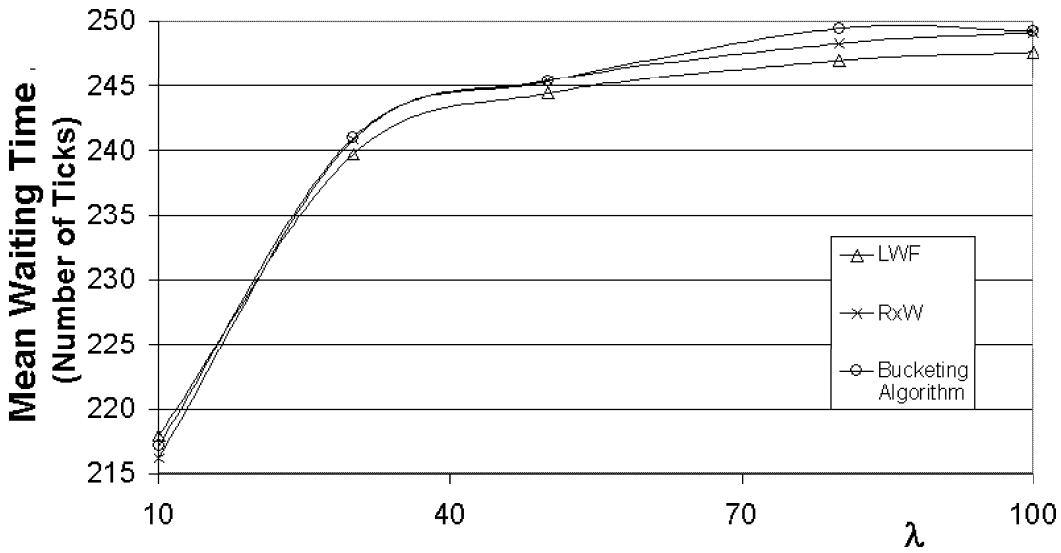

Figure 5.5: Mean waiting times of the LWF, RxW, and Bucketing algorithms.

5.4.1 Impact of Database Size

An important system parameter that could affect the performance is the number of data items in the database. A good scheduling algorithm should handle different database sizes without degrading the mean waiting time. In order to see the performance of scheduling algorithms in the case of a larger database, we conduct an experiment by increasing dbSize from 1,000 to 10,000 pages (see Figure 5.6).

In the experiment, we have fixed access skew coefficient (Θ) at 1.0 and mean request arrival rate (λ) at 10 requests per tick. As the database size increases, the mean waiting time increases as well. We have incremented dbSize parameter by ten times and observed that the mean waiting time of the algorithms rises only five or six times. The result shows that there is not much difference between the scalability of the algorithms with respect to database size, when the system workload is light. Similar results are obtained when the system load is high (i.e., λ=100) as depicted in Figure 5.7

In another experiment, we have fixed dbSize at 10,000 and changed the mean request arrival rate from 10 requests per tick to 100 requests per tick to

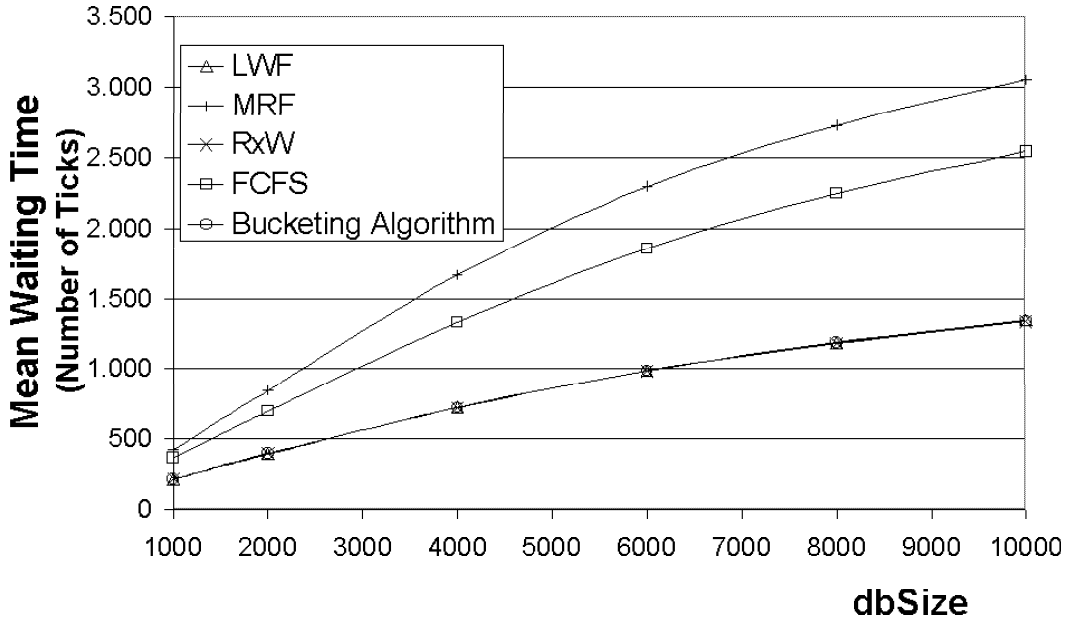

Figure 5.6: Mean waiting times when dbSize is incremented up to 10,000.

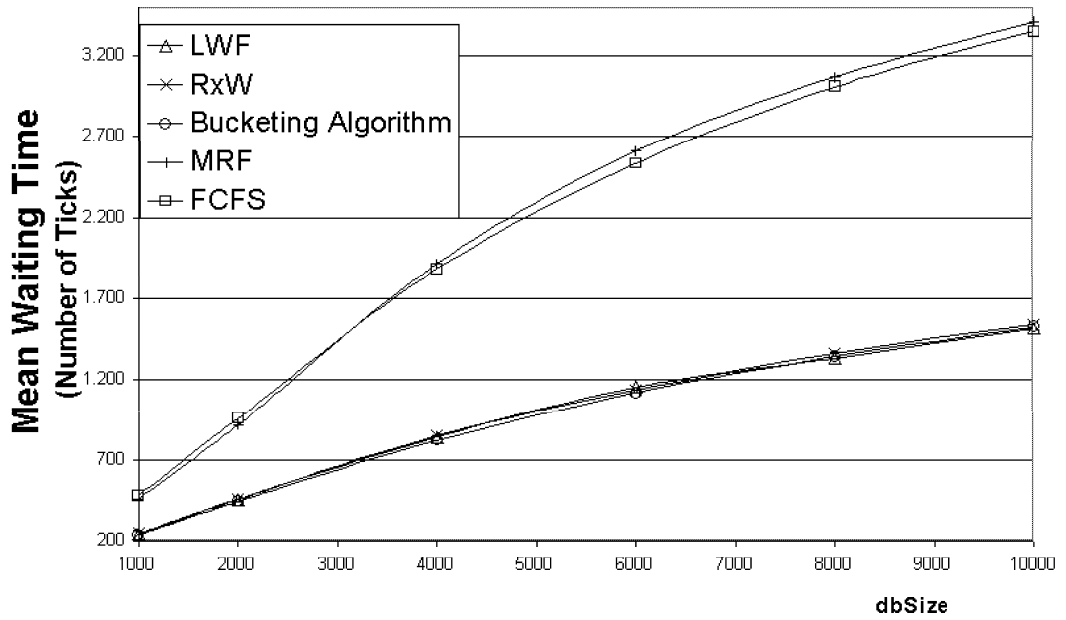

Figure 5.7: Mean waiting times when dbSize is incremented up to 10,000 with higher system workload.

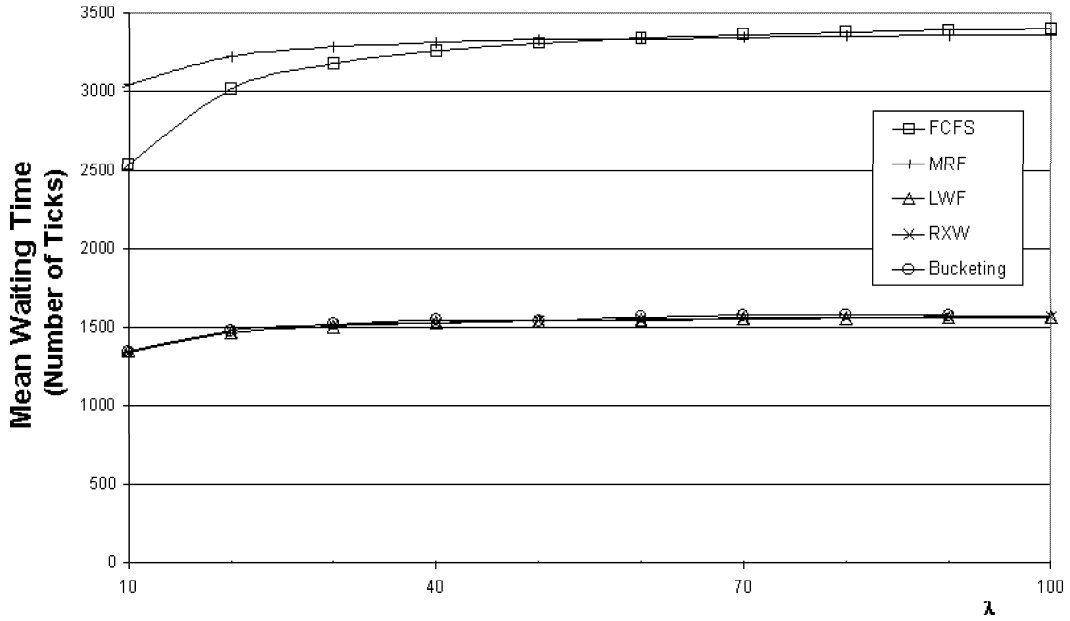

Figure 5.8: Mean waiting times when dbSize is 10,000.

examine if the scheduling algorithms yield different mean waiting times. As seen in Figure 5.8, the mean waiting time pattern shown in Figure 5.4 does not change much.

Concerning the time required to perform the simulation experiments, we have preferred to use a default database size of 1,000 pages for all other experiments without losing generality.

5.4.2 Impact of Access Skewness

We have also conducted several experiments to observe the effect of the access skewness parameter on the mean waiting time. In the experiment, we fix the request arrival rate at 10 requests per tick and vary the access skew coefficient (θ) from 0.1 to 1.0. As seen in Figure 5.9, except FCFS, mean waiting time values of the algorithms are getting considerably smaller as the skewness of the Zipf distribution is increased. This result is due to the fact that, the highly skewed request distribution (i.e., $\theta \geq 0.7$) leads to the existence of many pending requests to a few data items, and broadcasting one of the most requested

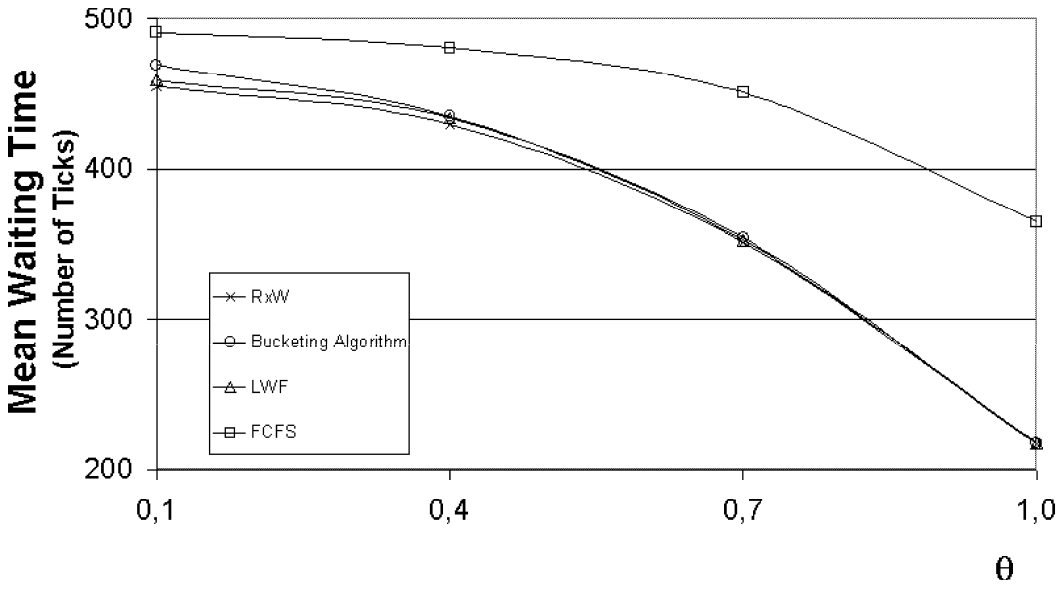

Figure 5.9: Changing skewness of request distribution over database.

data items satisfies many pending requests. RxW, LWF and Bucketing algorithms take the number of pending requests into account, and this property causes more efficient use of the broadcast channel. However, when θ is 0.1, the distribution reduces to almost uniform distribution, and each data item has almost the same pending requests. In that case, all the scheduling algorithms lead to almost the same mean waiting time.

The FCFS algorithm does not consider the pending request number in broadcast scheduling, so that the mean waiting time obtained with this algorithm does not improve much when the access skewness increases.

5.4.3 Impact of Page size

As mentioned in Chapter 2, when evaluating the performance of the scheduling algorithms, it is assumed that the size of each data item in the database is the same. However, this assumption may not be true for some applications. Therefore, we have conducted several experiments by relaxing this assumption to observe the effect of the database consisting different sizes of pages on the the scheduling algorithm's performance.

As discussed in Section 4.5, we have modeled the case of varying page size by adapting an approach by Vaidya et al. [19] into our heuristic, and modifying the ATWT formula as

$$\frac{(t - A_p) * R_p(t)}{s_p} \tag{2}$$

where s_p is the size of page p. The intuition behind this modification is to broadcast the small sized page first in order to reduce the total waiting time of other requests.

In order to simulate the mobile environment in which the size of data items in the database is not fixed, we have assigned different values to each data item size. In the experiments of this section, it is assumed that the page size (pSize) is varying between 20 KB and 400 KB, and the bandwidth of the broadcast channel is 2 Mb per second.

We have conducted two experiments to observe the impact of the size distribution on the overall mean waiting time. In both experiments, each item size is determined by a linear function. In the first experiment, the size of popular pages is smaller compared to the unpopular pages. It is reminded that we have assumed in Section 5.1 that data items are ordered according to their access probabilities in decreasing order, i.e., the most favorite data item is in the first place in the database. Therefore, in the first experiment, the size of the first page in the database is assigned the smallest value, whereas the size of the last page is assigned the largest value. The size of the pages (s_p) in between them is increased linearly using the following formula:

$$s_p = \begin{cases} size\_min + (\frac{size\_max - size\_min}{dbSize}) * (p - 1) & \text{if } 1 \leq p \leq dbSize - 1, \\ size\_max & \text{if } p = dbSize. \end{cases} \tag{3}$$

where p is the index of the page between 1 and the total number of data items (dbSize) in the database. *size_max* and *size_min* correspond to maximum and minimum page sizes, and they are set to 400 KB and 20 KB in the experiments, respectively. In other words, the size of data items in the database is associated with increasing values from 20 KB to 400 KB.

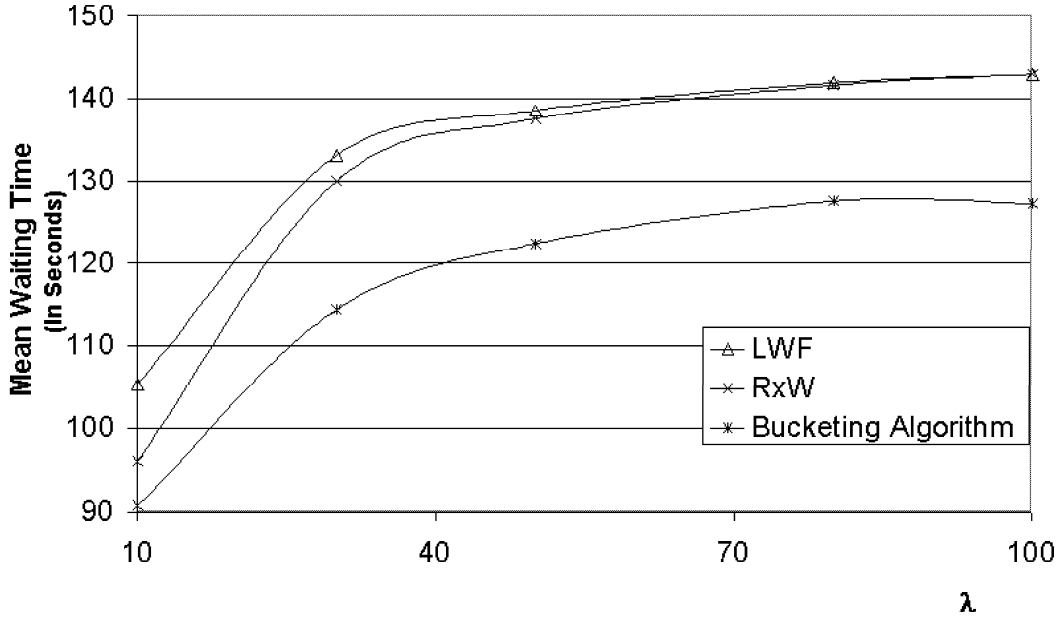

Figure 5.10: The page sizes are increasing linearly.

The mean waiting time results obtained for this configuration are provided in Figure 5.10.

Contrary to the first size distribution, in the second experiment the size of pages is linearly decreasing:

$$s_p = \begin{cases} size\_max - (\frac{size\_max - size\_min}{dbSize}) * (p-1) & \text{if } 1 \leq p \leq dbSize - 1, \\ size\_min & \text{if } p = dbSize. \end{cases} \quad (4)$$

In other words, the size of the most popular pages is larger than the size of unpopular pages. The experienced mean waiting time is depicted for the scheduling algorithms in Figure 5.11, when the size distribution is linearly decreasing.

In both experiments, the modified ATWT heuristic of the Bucketing scheduling algorithm provides the best mean waiting time results. The Bucketing algorithm produces less mean waiting time than that of the RxW algorithm about 5% in decreasing page size distribution, whereas this improvement is up to 13% when increasing page size distribution is employed.

As seen in Figure 5.11, when the size of hot pages is larger, the experienced mean waiting time becomes longer. On the average, when the decreasing size

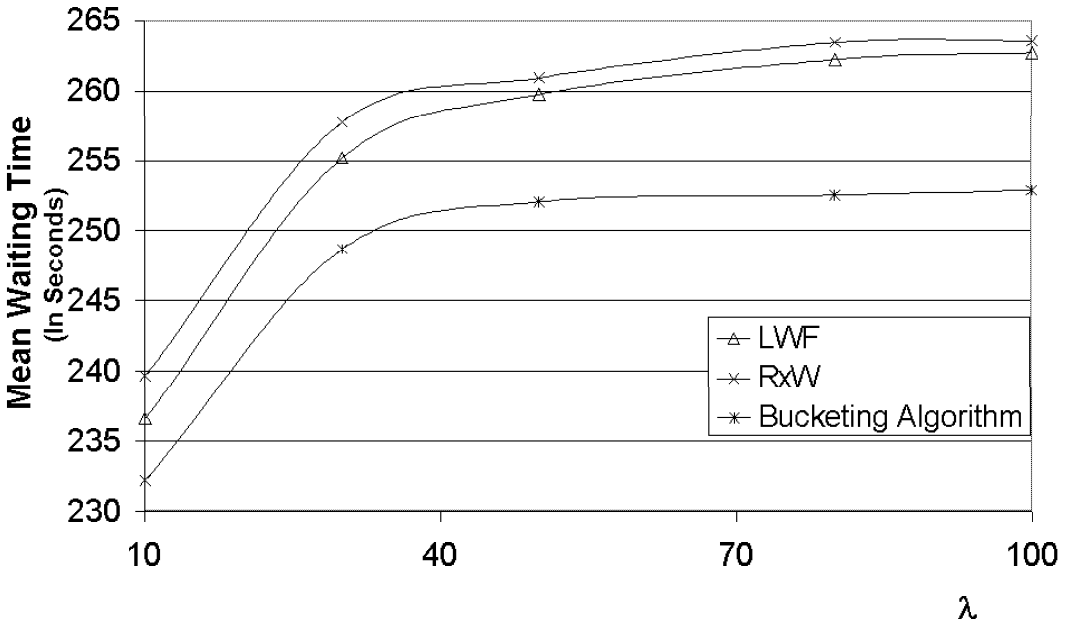

Figure 5.11: The page sizes are decreasing linearly.

distribution is employed, the mean waiting time is twice as much as the mean waiting time obtained with the increasing size distribution. This is due to the fact that as the popular pages are broadcast frequently, they make the requests to unpopular pages wait longer.

5.5 Variance of Waiting Time

In Section 4.6, we have discussed the importance of the variance of waiting time on the Quality of Service, and modified our algorithm to handle the trade-off between mean waiting time and variance of waiting time. The ATWT value used as a selection criterion in the Bucketing algorithm has been modified as follows:

$$(t - A_p)^{\alpha} * R_p(t) \tag{5}$$

where α can be set to different values to yield a good broadcast scheduling with respect to the desired criteria. For instance, if α is set to 1, the original ATWT heuristic is obtained and the overall mean waiting time is attempted

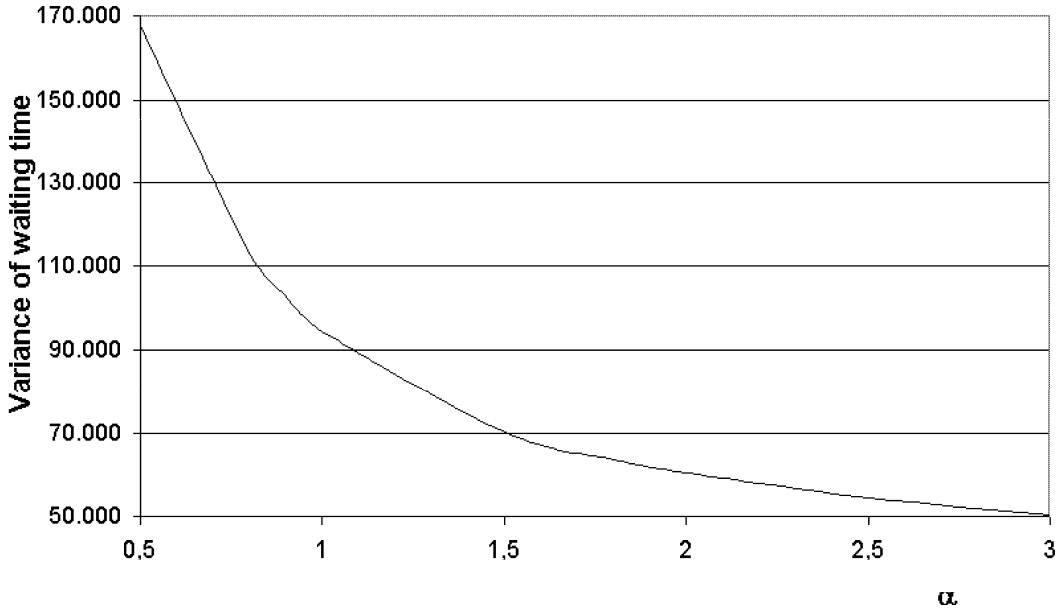

Figure 5.12: Impact of α parameter on variance of waiting time.

to be minimized. On the other hand, when α is assigned higher values than 1, the heuristic attaches more importance to the first request arrival time as in the FCFS heuristic and the variance of the waiting time is attempted to be minimized. As a result, there are two extreme cases in which one of the two metrics is the best.

In Figures 5.12 and 5.13 the results for the variance of waiting times and the mean waiting time of the Bucketing algorithm are presented, respectively. In this experiment, the α parameter is varied from 0.5 to 3.0, while using the other default parameter values given in Table 5.1. As seen in the figures, the trade-off between the mean waiting time and the variance of waiting time is clear. For higher values of α, the variance is improving, on the other hand, the mean waiting time of the algorithm is getting worse. The mean waiting time of the algorithm improves while the α parameter is increased from 0.5 to a certain value, which is 0.9 in our experiment. Then, for the values larger than this threshold, the mean waiting time begins to worsen again. This result is due to the fact that for the α values higher than 1, the modified ATWT heuristic in Formula 5 attaches more importance to the waiting time of the first request than the total number of pending requests of a page. Therefore, the heuristic selects the pages similar to those selected with the FCFS heuristic.

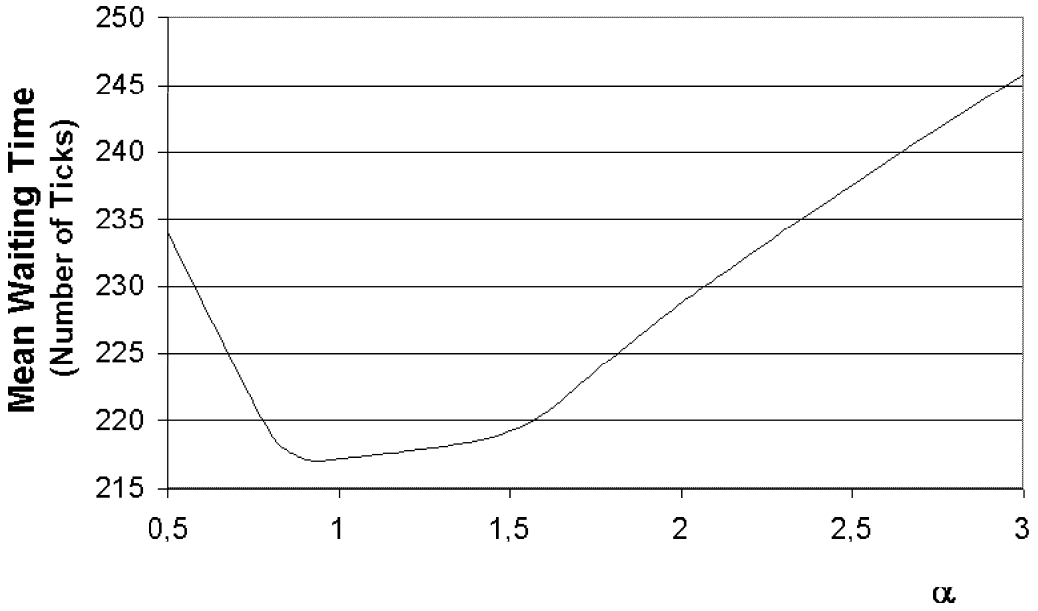

Figure 5.13: Impact of α parameter on mean waiting time.

On the other hand, when the α value is set to values lower than 1, the heuristic behaves in favor of the most requested pages like MRF. As a result, when the α value becomes farther than 1, the experienced mean waiting time becomes more similar to that of one of the two algorithms. We have seen in Figure 5.4 that for the mean request arrival rate (λ) of 10 requests per tick, MRF leads to worse mean waiting time than FCFS. As can be seen in Figure 5.13, when we set α to 0.5, we get worse mean waiting time compared to the value obtained by setting α to 1.5. This observation also supports the relationship between the α value and the resulting mean waiting time.

We have also conducted an experiment to observe the variance with the other algorithms and to compare the improvement gained by the modified ATWT in (5). The results obtained in the experiment is depicted in Figure 5.14. The algorithms, Bucketing, RxW, and LWF, which have the best mean waiting time results, have higher variance values. The FCFS algorithm has the lowest degree of variance observed, due to the fact that in the worst case, the algorithm broadcasts any requested data item after broadcasting the whole set of data items in the database. That is, for the waiting time of a request, there is an upper bound which is determined by the database and data item size. This upper bound also limits the variance of the waiting time. However, we can

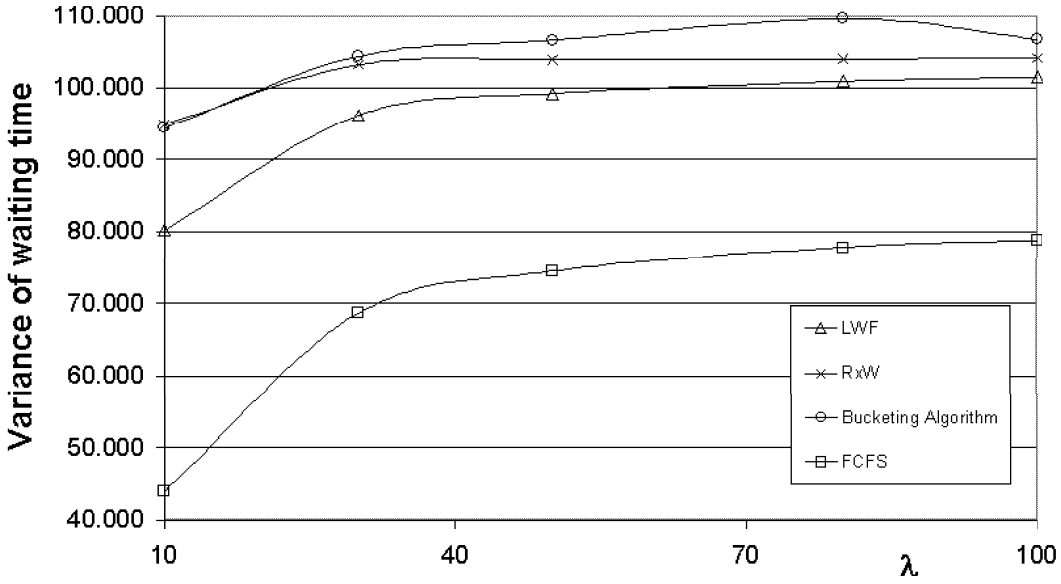

Figure 5.14: Variance of waiting time.

not claim this argument for the other algorithms in which, a data item can be broadcast many times before a data item is being broadcast.

After modifying our algorithm's heuristic as in (5), the new performance results obtained are presented in Figure 5.15. In this experiment, we have set the α value to 2. For higher workloads, the modified algorithm has even better variance of waiting time than that of FCFS.

However, as discussed above, the mean waiting time of the modified algorithm has become slightly worse (see Figure 5.16). With a greater value for α (e.g., 3), variance of the waiting time can be further decreased, however, the mean waiting time would become worse.

In order to see the impact of the variance of waiting time on the mean waiting time, we have conducted several experiments. As presented in Section 5.1, it is supposed that each client has an interest with all the data items in the database with different access probabilities determined by the access skew coefficient (Θ) parameter. However, in reality, it would be the case that, some clients are interested in only a portion of the database.

We have simulated a scenario such that there is a client who is requesting

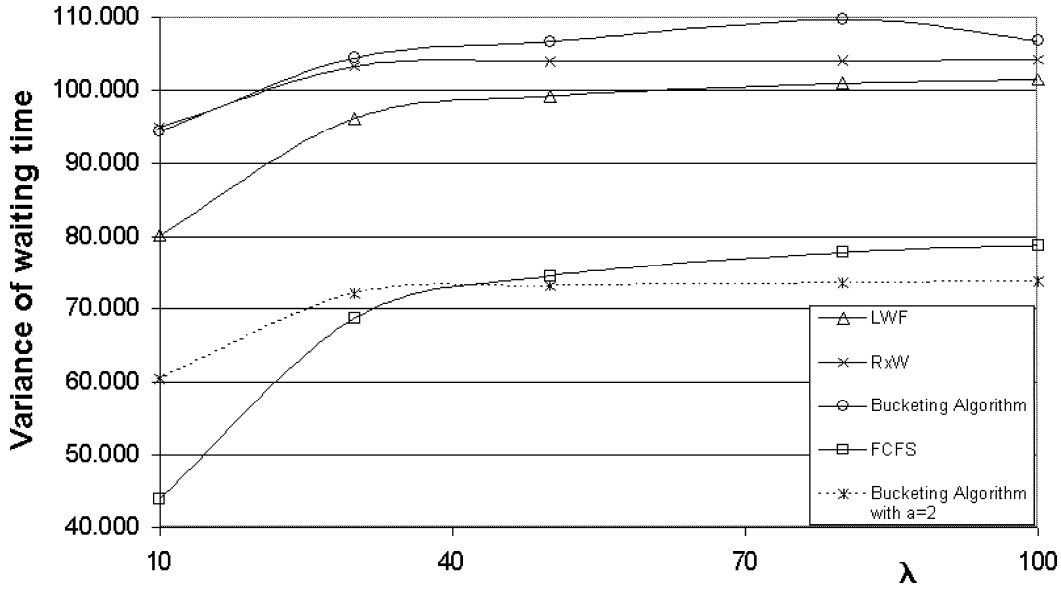

Figure 5.15: Comparing Bucketing algorithm when α=2.

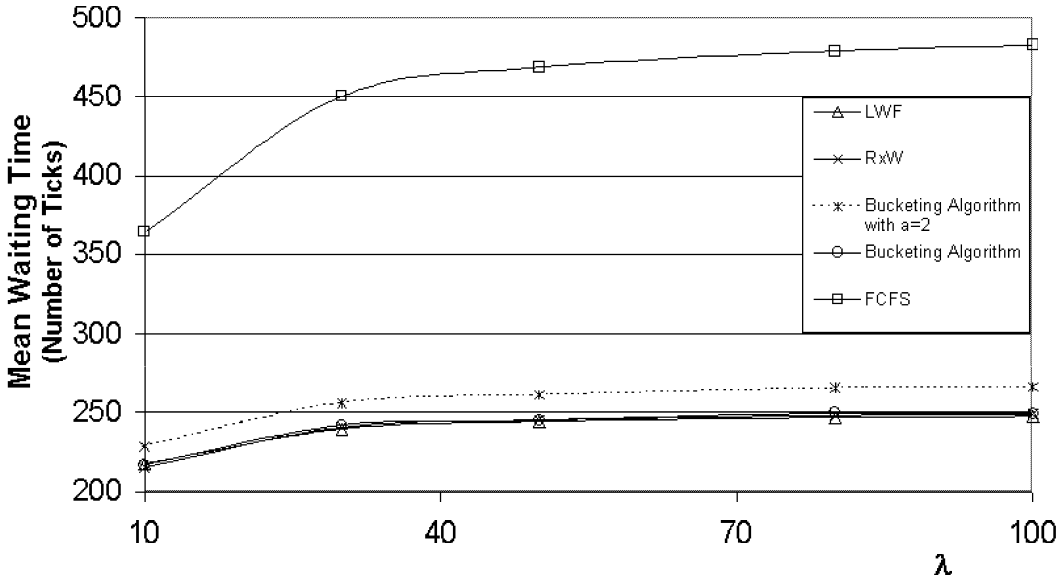

Figure 5.16: Mean waiting time of Bucketing algorithm with α=2.

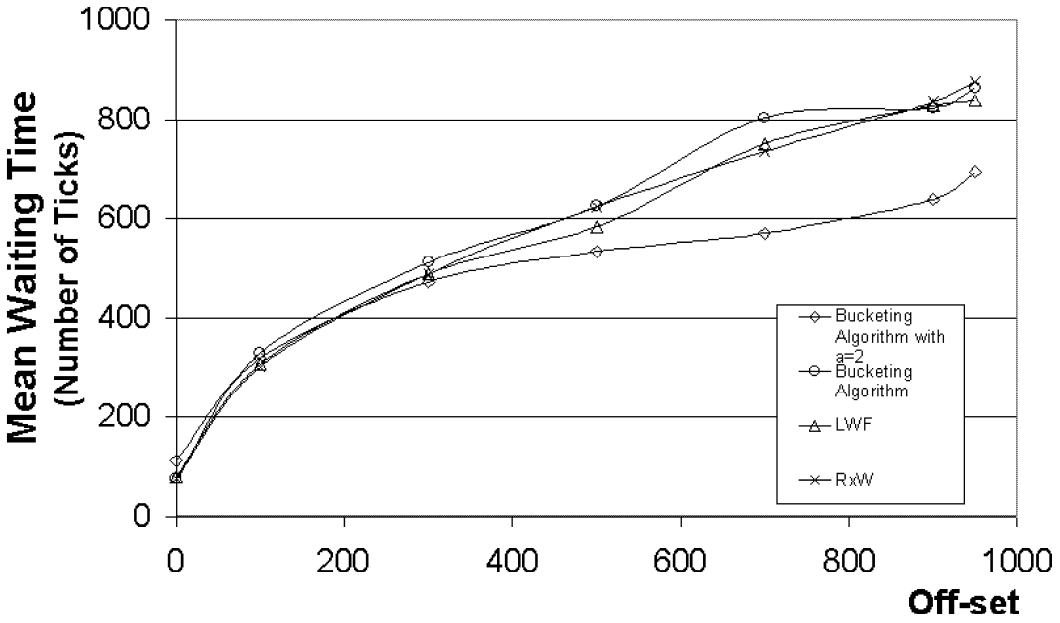

Figure 5.17: Mean waiting time of a client with different off-set values.

only a limited number of data items. We assume that these data items are located in the database consecutively. The number of data items requested by the client is termed as the *request range*. In previous experiments, we have supposed that each client's request range is between 1 and dbSize, where dbSize is the total number of data items in the database. For this experiment, the request range is specified as 50 data items. In Section 5.1, we have assumed that data items are ordered in the database according to their access probabilities such that the most probably requested data item is in the first entry, whereas the data item with lowest access probability is the last entry of the database. In the experiment, we select an *off-set* parameter which is the first data item of the client's request range. The client's requests are distributed over data items in the request range according to the *Zipf* distribution with Θ=1.0.

As seen in Figure 5.17, the client has different mean waiting time values from the rest of the client population. It is reminded that for the same parameter values, the mean waiting time of the client population experienced for the Bucketing, RxW, and LWF algorithms is about 217 ticks. As the client request range goes away from the *hot* data items, the mean waiting time increases. The three algorithms, Bucketing, RxW, and LWF have almost the same values, whereas the Bucketing algorithm with α=2 has the lowest values. The fact

is that the Bucketing algorithm with α=2 has the lowest variance of waiting time values among all the algorithms (see Figure 5.15), and this leads to the fact that the client's mean waiting time is not affected as much as the other algorithms. The difference between mean waiting time values obtained by different algorithms is small when the off-set value is chosen within the range of hot pages. On the other hand, when the client is requesting cold pages (e.g., for the off-set values greater than 500 pages), the difference is getting noticeable. For instance, the mean waiting time of the Bucketing algorithm with α=2 is about 30% less compared to other algorithms for the off-set value 900.

Another important result obtained from the experiment is that the mean waiting time experienced by the client is up to four times compared to the whole client population when the Bucketing, RxW, or LWF algorithm is implemented as the scheduling algorithm. However, if the Bucketing algorithm with α=2 is employed, the degeneration observed in the mean waiting time is limited to about 3 times of the original one.

5.6 Worst Waiting Time

The reason to evaluate this criterion is to check if the proposed scheduling algorithm causes starvation of any request, which is an important property that should be avoided in interactive applications as discussed in Section 5.2. Figure 5.18 displays the results for the longest waiting time experienced by any MU during the whole simulation time. For the default values of the simulation parameters presented in Table 5.1, FCFS has the lowest worst waiting time among all the algorithms. As discussed above, when FCFS is employed as the scheduling algorithm, any requested data item will be broadcast after the data items previously requested and the number of these data items is limited by the database size. In other words, the largest possible worst waiting time of a request is the time taken by broadcasting all the database items. However, for other algorithms, it could be possible that a request waits while some of the data items are broadcast multiple times. The Bucketing algorithm has lower worst waiting time than that of the RxW algorithm with changes between 3%

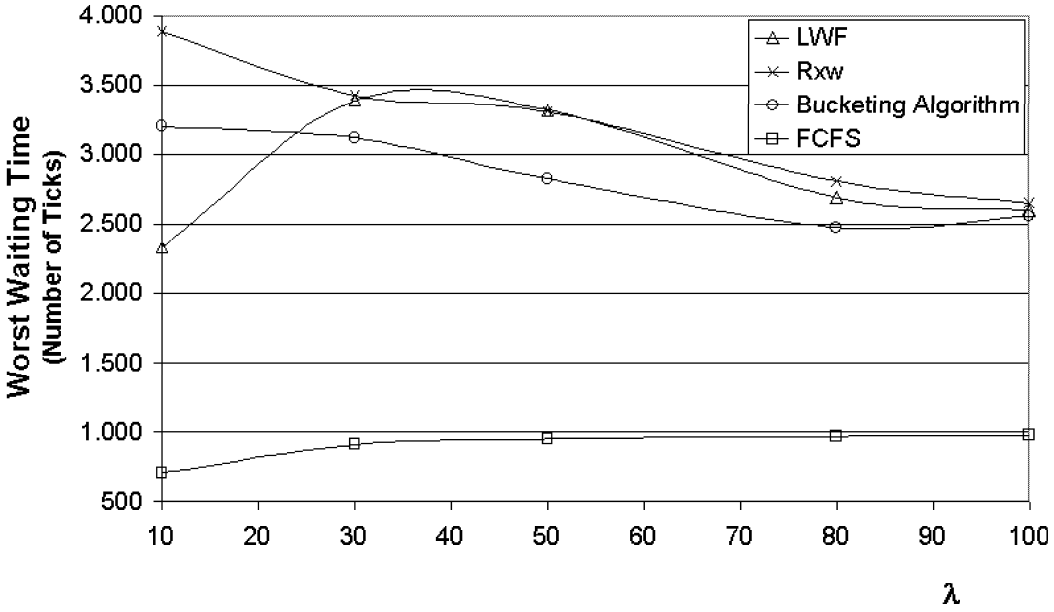

Figure 5.18: Worst waiting time.

and 21%.

The worst waiting time values observed with the algorithms are presented in Figure 5.19 as dbSize is varying between 1,000 and 10,000 with λ=50. In the figure, again FCFS has the lowest worst waiting time for all the values of the dbSize parameter. MRF leads to unfair usage of the broadcast band for some requests and makes them wait for unacceptable periods. Our algorithm produces better results compared to the RxW, LWF, and MRF algorithms. The worst waiting time of the Bucketing algorithm is less than those of the RxW and LWF algorithms up to 20%.

5.7 Scheduling Decision Overhead

As discussed previously, a good scheduling algorithm should not have much scheduling decision overhead. Otherwise, while the scheduling algorithm wastes its precious time to decide which data item to broadcast next, time gaps can occur between broadcasts, which leads to an inefficient use of the broadcast channel.

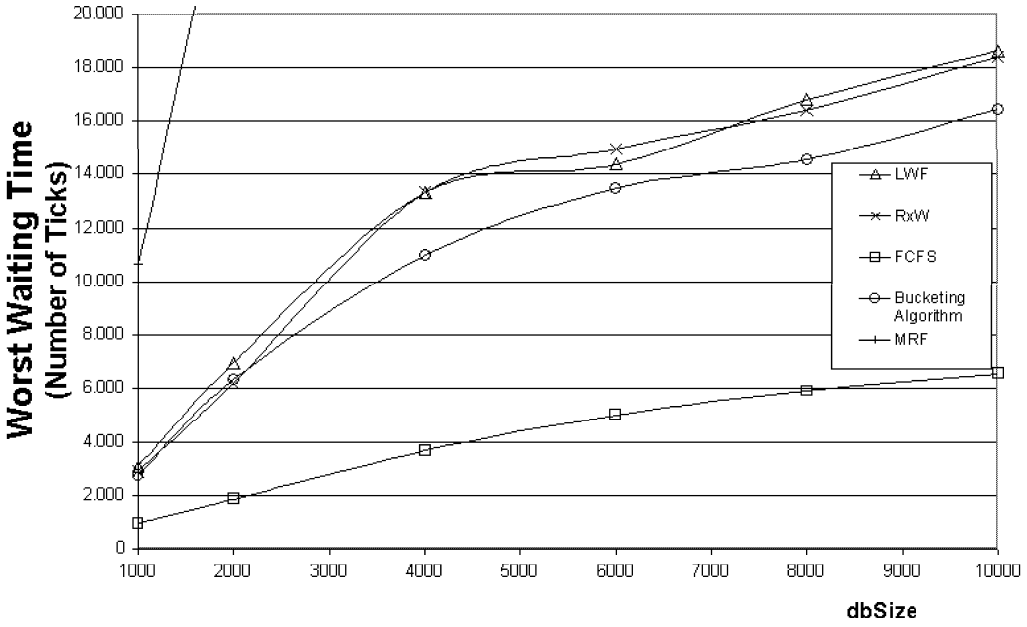

Figure 5.19: Worst waiting time for different dbSize values.

In order to evaluate the decision overhead of the scheduling algorithms, we have examined the number of requests scanned for selecting one of them to broadcast. If the number is large, the decision takes much time and may become a bottleneck.

We have compared the average number of data items scanned by three algorithms: LWF, RxW, and Bucketing, using the parameter values provided in Table 5.1. The system workload is increased by varying λ value from 10 to 100. FCFS is not included in this experiment. It only broadcasts the request that has arrived first, and does not need to compare any entry. On the other hand, its overall waiting time is so bad that it is not a competitive algorithm to be used.

As seen in Figure 5.20, LWF has the highest decision overhead, while the Bucketing algorithm has the lowest decision overhead. The overhead of the RxW algorithm is in between. Compared to other algorithms, the Bucketing algorithm examines significantly fewer number of requests at each scheduling decision. For a request rate of 10 requests per tick, the LWF algorithm compares 130 times more entries and the RxW algorithm compares 36 times more entries than that of the Bucketing algorithm. In [9], approximate versions of

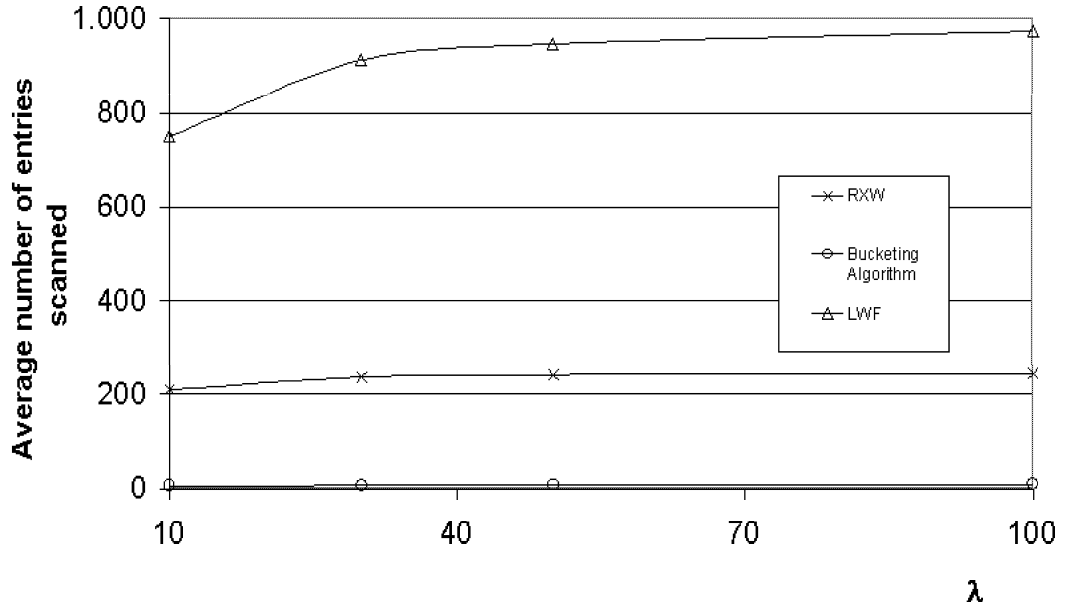

Figure 5.20: Decision overhead.

the RxW algorithm are proposed and they are shown to lead to less comparisons in deciding which data item to broadcast. However, these approximate versions have worse mean waiting times compared to RxW. The Bucketing algorithm, as discussed in Section 5.4, produces almost the same mean waiting time as RxW, while leading to less scheduling decision overhead.

5.8 Improving the Bucketing Algorithm

We have tried to improve the mean waiting time of our Bucketing algorithm and implemented the depth approach as presented in Section 4.4. There is a trade-off between the decision overhead and the mean waiting time in this approach. As we increase the search depth, we need to compare more entries of ATWT values, and we obtain lower mean waiting time (see Figure 5.21 and Table 5.2).

When we set the depth parameter to 50, the resulting mean waiting time of the Bucketing algorithm is smaller than to that of the RxW algorithm as seen in Figure 5.22.

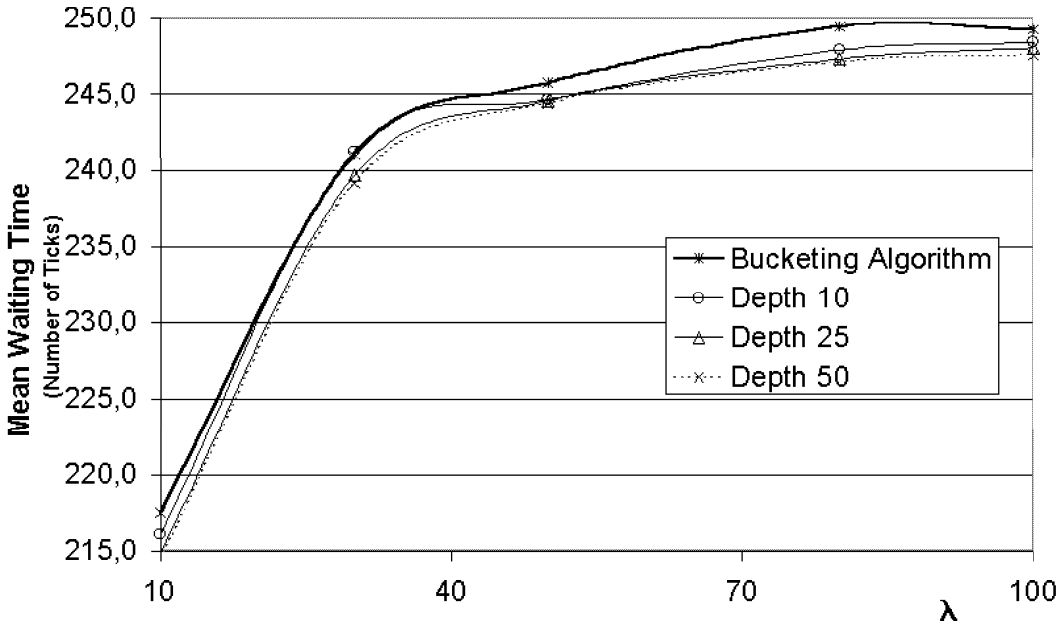

Figure 5.21: Increasing *depth* parameter of Bucketing algorithm.

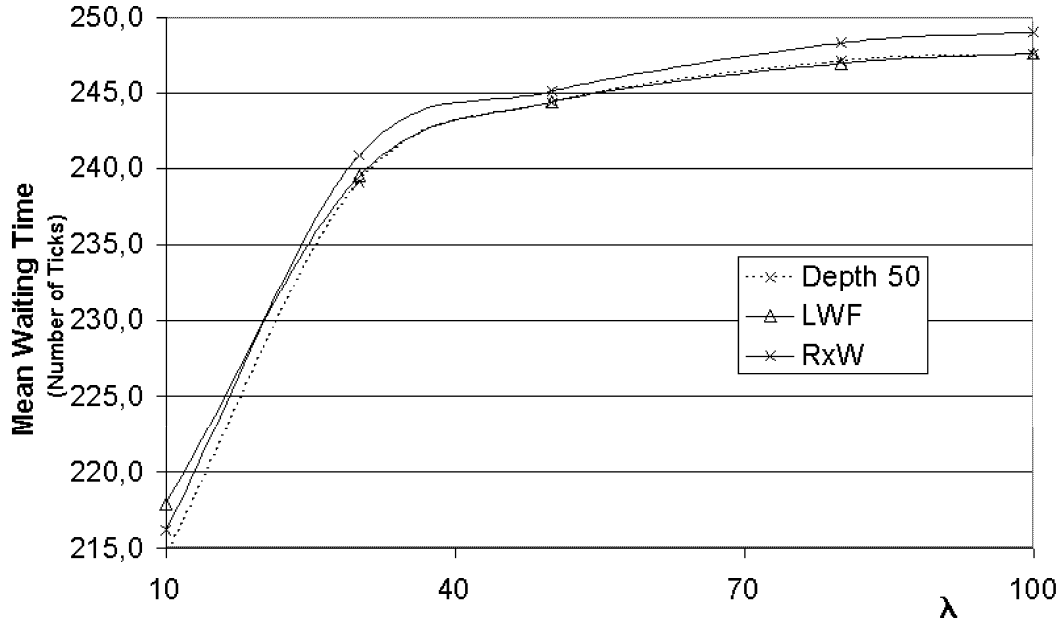

Figure 5.22: Bucketing algorithm with *depth*=50 and the other competitive algorithms.

5.9 Implementing A Push-Based Algorithm

We have summarized the related work done by Vaidya et al. in Section 2.2.3. The authors have worked on data broadcast scheduling algorithms extensively and proposed several scheduling algorithms [19, 35]. Vaidya et al. suggest that the transmission of information to the mobile clients can be performed efficiently by broadcasting the information periodically. Therefore, their scheduling algorithms are based on a push-based approach. On the other hand, the authors claim that the scheduling algorithm in [33] can be applied to a pull-based broadcast environment by replacing access probabilities with the number of pending requests for a data item. However, they do not implement the algorithm in a pull-based environment and evaluate it against the other algorithms. To observe the performance of the algorithm proposed by Vaidya et al. in [33, 18] for push-based systems, we have modified and implemented it as to schedule the broadcast in an on-demand environment.

The authors have offered to use a *decision rule* to select which data item to be broadcast next [33]. In the first algorithm they propose, which is called *Algorithm A*, the decision rule is determined as follows:

$$G(i) = \frac{(Q - R(i))^2 * p_i}{l_i} \tag{6}$$

where Q denotes the current time, $R(i)$ is the last broadcasting time, p_i is the access probability, and l_i is the size of item i. The data item with the maximum $G(i)$ value is selected to broadcast. It is assumed in the push-based systems that the access probabilities of each data item are known beforehand. However, in the pull-based systems, scheduling algorithms utilize the total number of pending requests for each data item, and access probabilities are assumed to be dynamic or unknown to the scheduling algorithms.

In the implementation of Algorithm A in a pull-based system, we have used the total pending request number (R_p) of a page instead of the access probability (p_i). Furthermore, as a default, we do not consider the page size and assume that the size of all pages are the same. Therefore, the l_i parameter

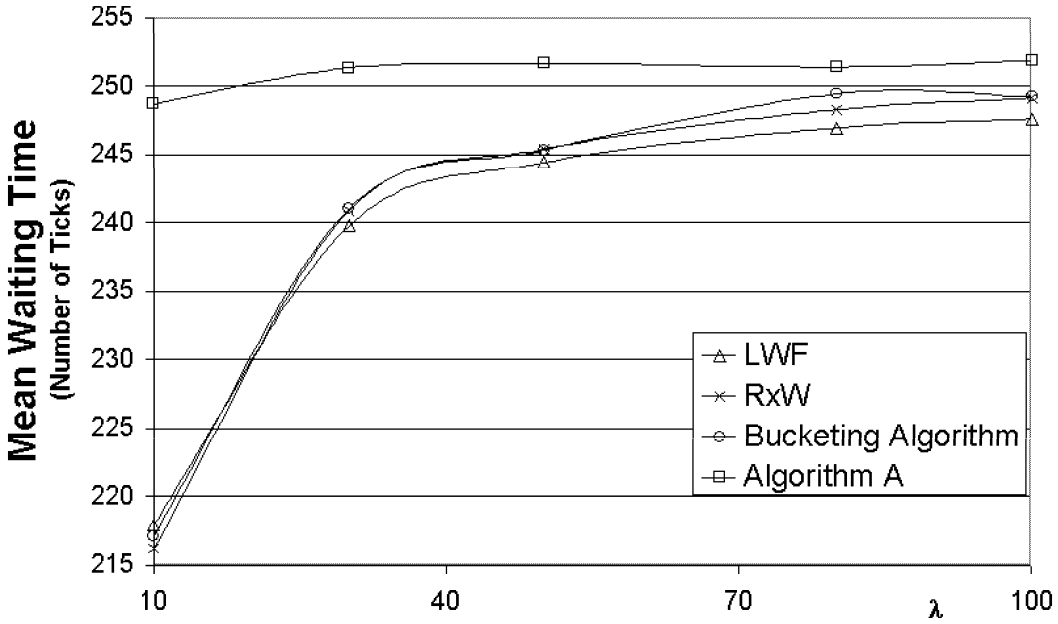

Figure 5.23: Comparing Algorithm A with pull-based algorithms.

of the decision rule in equation (6) is omitted.

The observed mean waiting time results are provided in Figure 5.23. The results show that the proposed algorithm does not provide better mean waiting time compared to the three algorithms we have experimented so far. An important point to be mentioned about is the fact that when the request rate is getting high, the difference between the performance of Algorithm A and the other algorithms decreases. However, even if we have increased the λ parameter value up to 250, the mean waiting time of Algorithm A is still worse.

We would like to make some comments on the decision rule in equation (6). After modifying the formula to work in pull-based environments, the decision rule attaches more importance to the duration than the total number of pending requests by taking its second power. However, the other algorithms, LWF, RxW, and Bucketing, do not give special importance to the duration.

Another comment that might be made is that, after modifying the formula to work in pull-based environments, if the second power of the subtraction is removed, then the formula becomes very similar to the one used in Bucketing and RxW scheduling algorithms. The only difference is that in the decision rule of Algorithm A, the duration used begins from the time that the data

item is last broadcast, whereas Bucketing and RxW scheduling algorithms are concerned with the time when the first request for that page arrives.

Chapter 6

Conclusion

In this thesis, the problem we attack is the design of a broadcast scheduling algorithm which efficiently meets the demands of a mobile computing environment and mobile users. We have first proposed a new broadcast scheduling heuristic, *Approximate Total Waiting Time* (ATWT), which is an approximate version of the *Longest Wait First* (LWF) heuristic [36]. Then, we have developed an algorithm called the *Bucketing* algorithm to implement the ATWT heuristic by using a bucketing scheme. Finally, we have done extensive simulation experiments to evaluate the performance of our algorithm, and to compare the performance results against those of previously proposed scheduling algorithms.

Considering the performance results, the first remark to be done is that the most competitive algorithm to our algorithm is RxW [9]. The other algorithms, except LWF, do not produce good results with respect to the main performance criterion, the *overall mean waiting time*. Although, the LWF algorithm has better performance than all the others, it has serious drawbacks which prevent its practical usage. However, as a yardstick we have also provided its performance results. In this chapter, we mainly focus on summarizing the comparative performance results of the Bucketing algorithm and RxW.

The Bucketing algorithm has a run time complexity of $O(\lceil log(R+1) \rceil)$, where R is the number of pending requests on the most requested page in

the system. The *exhaustive* RxW algorithm runs in $O(N)$ time, where N is the number of requested data items, whereas the *maximal* RxW algorithm which applies a pruning technique can eliminate %73 of the entries to find the entry with the maximal RxW value. Since the running time of the Bucketing algorithm is logarithmic with respect to the number of requests on the most requested page, it is faster than RxW.

Performance results of the Bucketing algorithm in terms of the main performance metric (i.e., overall mean waiting time), are very close to those of RxW. Furthermore, we can obtain better results by implementing the *k-depth* Bucketing algorithm. As presented in Section 5.8, when we set the *depth* parameter to 50 for the given simulation parameter values, the resulting mean waiting time of the Bucketing algorithm is less than that of the competing algorithm RxW.

We have also observed the impact of several different system parameters, such as the database size and access skewness parameter, on the overall mean waiting time achieved. The results obtained with the database size experiment have shown that there is not much difference between the scalability of the algorithms with respect to the database size. The effect of the *access skewness parameter* on the mean waiting time has been examined by varying the *access skew coefficient* (θ). It has been observed that the mean waiting time values of the RxW and Bucketing algorithms decrease as the skewness of the Zipf distribution is increased. This result is due to the fact that, the highly skewed request distribution leads to the existence of many pending requests to a few data items, and broadcasting one of the most requested data items satisfies many pending requests.

We have also conducted several experiments by setting the pages of the database to different sizes. The ATWT heuristic has been modified aiming to consider the size of individual pages in the computation of the ATWT value. The Bucketing algorithm with the modified ATWT heuristic has resulted in the best mean waiting time experienced for different size distributions. The improvement in the mean waiting time has been up to 13% when increasing page size distribution has been employed.

In order to handle the trade-off between the performance measures mean waiting time and variance of waiting time, we have modified the selection criterion of our algorithm. As a result of this modification, the system's performance can be adjusted to obtain required results for the given metrics. A parameter (α) is used by the algorithm for this adjustment. For instance, we can get very low values of the variance of waiting time metric by setting the α parameter of the algorithm to values larger than 1.

The *worst waiting time* performance metric has also been used in evaluations to check if any of the scheduling algorithms evaluated causes starvation of any request. The worst waiting time result obtained with the Bucketing algorithm has been observed to be less than that of the RxW and LWF algorithms up to 20%.

We have also implemented a *push-based* broadcast scheduling algorithm proposed by Vaidya et al. [33] by applying the required modifications to be used in an on-demand environment. The performance results obtained show that this algorithm does not provide better mean waiting time compared to other algorithms we have experimented with so far.

After evaluation of the simulation results obtained with the Bucketing scheduling algorithm, relative to the performance of the other well-known broadcast scheduling algorithms, we have concluded that the Bucketing algorithm is one of the best algorithms that can be used in pull-based broadcast environments.

One possible direction for future research is to extend the Bucketing algorithm to handle transmission errors and to support multiple broadcast channels. In order to reduce transmission errors, large data items can be broadcast in smaller packets. Multiple broadcast channels might provide better performance results. Database can be partitioned and each partition can be broadcast on a different channel.

Bibliography

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of ACM SIGMOD*, pages 199–210, May 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of ACM SIGMOD Conference*, Tuscon, Arizona, May 1997.

[3] S. Acharya, M. J. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6):50–60, December 1995.

[4] S. Acharya, M. J. Franklin, and S. Zdonik. Disseminating updates on broadcast disks. In *22nd International Conference on Very Large Data Bases (VLDB'96)*, Bombay, India, September 1996.

[5] S. Acharya, M. J. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *12th International Conference on Data Engineering (ICDE'96)*, New Orleans, LA, February 1996.

[6] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

[7] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, January/February 1999.

[8] D. Aksoy and M. Franklin. Scheduling for large-scale on-demand data broadcasting. In *Proceedings of the IEEE INFOCOM Conference*, pages 651–659, March 1998.

[9] D. Aksoy and M. Franklin. Rxw: A scheduling approach for large-scale on-demand data broadcast. *ACM/IEEE Transactions on Networking*, 7:846–860, 1999.

[10] M. H. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5:235–242, November 1985.

[11] M. H. Ammar and J. W. Wong. On the optimality of cyclic transmission in teletext systems. *IEEE Transactions on Communications*, 35:68–73, January 1987.

[12] D. Barbara. Mobile computing and database: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, January-February 1999.

[13] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environment. *ACM SIGMOD RECORD*, 23(2), May 1994.

[14] J. C. R. Bennett and H. Zhang. Wf2q: Worst-case fair weighted fair queueing. In *INFOCOM'96*, March 1996.

[15] Hughes Electronics (GMH) company DirecPc Home page. http://www.direcpc.com/, 2000.

[16] A. Datta, A. Celik, J. G. Kim, D. E. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservant retrieval by mobile users. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 124–133, Birmingham U.K, April 7-11 1997. IEEE Computer Society.

[17] T. F. Bowen et al. The datacycle architecture. *Communication of ACM*, 35(12):71–81, December 1992.

[18] S. Hameed and N. H. Vaidya. Log-time algorithms for scheduling single and multiple channel data broadcast. In *Proceedings of the Third Annual*

ACM/IEEE International Conference on Mobile Computing and Networking, pages 90–99, Budapest, September 1997.

[19] S. Hameed and N. H. Vaidya. Efficient algorithms for scheduling data broadcast. *Wireless Networks*, 5:183–193, 1999.

[20] A. Helal, B. Haskell, J. L. Carter, R. Brice, D. Woelk, and M. Rusinkiewicz. *Any Time, Anywhere Computing: Mobile Computing Concepts and Technology.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston/Dordrecht/London, 1999.

[21] G. Herman, K. C.Lee, and A. Weinrib. The datacycle architecture for very high throughput database systems. In *Proceedings of ACM SIGMOD*, pages 97–103, 1987.

[22] T. Imielinski and B.R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, pages 19–27, October 1994.

[23] S. Jiang and N. H. Vaidya. Response time in data broadcast systems: Mean, variance and trade-off. In *Proceedings of International Workshop on Satellite-based Information Services (WOSBIS)*, October 1998.

[24] S. Jiang and N. H. Vaidya. Scheduling data broadcasting to "impatient" users. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 52–59, Seattle, WA USA, August 1999.

[25] K. Lam, E. Chan, and J. C. Yuen. Approaches for broadcasting temporal data in mobile computing systems. *Journal of Systems and Software*, 51(3):175–189, May 2000.

[26] AirMedia Home Page. http://www.airmedia.com/, 1999.

[27] GSM Association Home Page. http://www.gsm.org/, 2000.

[28] WAP Forum Home Page. http://www.wapforum.org/, 2000.

[29] Y. Saygin and O. Ulusoy. Exploiting data mining techniques for broadcasting data in mobile computing environments. Technical Report BU-CEIS-9914, Department of Computer Engineering, Bilkent University, Ankara,Turkey, 1999.

[30] H. Schwetman. Csim18 - the simulation engine. In J. Charnes, D. Morrice, D. Brunner, and J. Swain, editors, *Proceedings of the 1996 Winter Simulation Conference*, pages 517–521, San Diego, CA, 1996.

[31] K. Stathatos, N. Roussopoulos, and J.S. Baras. Adaptive data broadcast in hybrid networks. In *23rd International Conference on Very Large Data Bases*, pages 326–335, Athens, Greece, September 1997.

[32] C. Su, L. Tassiulas, and V. J. Tsotras. Broadcast scheduling for information distribution. *Wireless Networks*, 5:137–147, 1999.

[33] N. H. Vaidya and S. Hameed. Scheduling data broadcast in asymmetric communication environments. *Wireless Networks*, 5:171–182, 1999.

[34] N. H. Vaidya and S. Hameed. Data broadcast scheduling: On-line and off-line algorithms. Technical report, Department of Computer Science, Texas A&M University, July 1996.

[35] N. H. Vaidya and S. Jiang. Data broadcast in asymmetric wireless environments. In *Proceedings of First International Workshop on Satellite-based Information Services (WOSBIS)*, NY, November 1996.

[36] J. W. Wong. Broadcast delivery. In *Proceedings of The IEEE*, volume 76, pages 1566–1577, December 1988.

[37] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya. Are 'disks in the air' just pie in the sky? In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[38] G. K. Zipf. *Relative frequency as a determinant of phonetic change*, volume XL. Reprinted from the Harvard Studies in Classical Philiology, 1929.