Performance Evaluation of a Wireless Hierarchical Data Dissemination System

Qinglong Hu*& Dik Lun Lee
Department of Computer Science
University of Science and Technology
Clear Water Bay, Hong Kong
qinglong@cs.ust.hk dlee@cs.ust.hk
FAX: (852) 2358-1477

Wang-Chien Lee
GTE Laboratories Incorporated
40 Sylvan Road
Waltham, MA 02451, USA
wlee@gte.com
FAX: (781) 466-4387

Abstract

Various techniques have been developed to improve the performance of wireless information services. Techniques such as information broadcasting, caching of frequently accessed data, and point-to-point channels for pull-based data requests are often used to reduce data access time. To efficiently utilize information broadcast, indexing and scheduling schemes are employed for the organization of data broadcast. Most of the studies in the literature focused either on individual technique or a combination of them with some restrictive assumptions. There is no study considering these techniques working together in an integrated manner. In this paper, we propose a dynamic data delivery model for wireless communication environments. An important feature of our model is that data are disseminated through various storage mediums according to the dynamically collected data access patterns. Various results are presented in a set of simulation studies, which give some of the intuitions behind the design of a wireless data delivery system.

1 Introduction

In a mobile computing environment, limited battery power, scarce wireless bandwidth, and asymmetric communication[1] impose a challenge to the design of a mobile system. A criterion often used to measure the data access efficiency of a mobile system is *access time*[2]. To reduce the access time, three major classes of techniques, namely, data caching, information broadcast (push-based), point-to-point data delivery (pulled-based), were investigated in [1, 9, 12]. Moreover, research on balancing the push-based information broadcast and pulled-based data delivery methods was conducted in [2, 11, 16].

Broadcast schedule and cache management policies are important for efficient utilization of broadcast channels on the air and cache memory in mobile clients [1, 3, 13]. With static data access patterns, it has been shown in [1] that the broadcast disks outperformed the flat broadcast and the *LIX* cache management policy did better than the *LRU* strategy. In a previous paper [5], the broadcast disks were also shown to have better performance than the flat broadcast when the content of the broadcast channel and the client population were not fixed.

When the database is very large, it is necessary to identify an appropriate subset of the database for broadcast. Stathatos et al. investigated the dynamic adjustment of the hot-spot for the broadcast program by monitoring "broadcast misses" [16]. Leong et al. investigated several mechanisms in selecting the proper database items for broadcast and called these mechanisms air-storage management [14]. However, they did not consider the access probability collection mechanism together with the air-storage management. The studies made in [14, 16] were for flat broadcast only and data caching was not considered.

When data can be delivered by both broadcasting and point-to-point services, the clients can select a delivery method that will result in better access time. This requires the server to inform the clients how the data items are scheduled for broadcast. One way to accomplish this is by broadcasting an index of the data together with the data themselves [2, 9, 16, 12, 13]. However, the index methods proposed are mainly for power conservation based on the idea of selective tune-in. Furthermore, many of the above studies assumed *one* shared back channel for all users. As such, responses for client requests are sent back to the clients on the broadcast channels, thus requiring the clients to monitor the broadcast channels for the results.

In this paper, we study a hierarchical data delivery (HDD) model which integrates data caching, information

*The author is now with Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[1]The downlink channels from a server to its clients have far more bandwidth capacity than the uplink channels from clients to the server.

[2]It is the time elapsed from the moment a mobile computer requests a data item until that item is received.

broadcasting, and pull-based point-to-point data delivering. Compared to the previous studies, the focus of this paper is how to achieve low *access time* by a combination of these techniques. In particular, we assume that point-to-point channels are available not only for sending queries from the clients to the server but also for returning results from the server back to the clients.

In HDD, the client cache and the broadcast schedule are dynamically adjusted based on the clients' access patterns to minimize access time. For example, according to the access frequencies, data can be stored in cache, broadcast on broadcast channel, or passively stored on the server waiting for pull requests to arrive. The hot spot of the uncached data can be obtained by asking the clients to monitor the *cache misses* and then piggyback the information to the server on some subsequent pull requests.

The performance of HDD is evaluated by simulation. The effects of various techniques on the overall performance (e.g., the performance is compared under different broadcast scheduling techniques and index techniques) are carefully studied with our experiments. In summary, the main contributions of our research includes:

- a hierarchical data delivery model integrating various wireless data access techniques such as point-to-point channels, broadcast channels and data caching;
- new data dissemination methods are defined for this model;
- indexing methods are explored in this integrated environment;
- both broadcast disks and flat broadcast scheduling methods are studied in this integrated environment[3] with various client population and dynamically changing access patterns.

The rest of the paper is organized as follows. Section 2 gives an overview of the individual techniques employed in the hierarchical data dissemination system. Section 3 describes the basic ideas of hierarchical data dissemination. Section 4 describes the simulation model developed for the evaluation of the hierarchical data delivery system. Section 5 presents a set of experiments and the simulation results obtained from the model. Finally, Section 6 concludes the paper.

2 Background

In this section, we provide an overview of the individual techniques employed in the hierarchical data dissemination system.

[3]In most of the previous work, broadcast disks and flat broadcast methods were compared within a pure broadcast environment or a non-integrated environment consisted of some data access methods we consider in this paper.

2.1 Channel Allocation

Three channel allocation methods, namely, exclusive broadcast, exclusive on-demand, and hybrid allocation (hereafter, denoted as *Pure_Push*, *Pure_PULL*, and *HDD*, respectively) were explored in [11]. Exclusive broadcast and exclusive on-demand methods use all channels for either broadcast or on-demand services, respectively. For *HDD*, both broadcast channels and on-demand channels[4] are used. The clients may obtain data from either broadcast channels or on-demand channels but not both. That is, at any time, a client either monitors the broadcast channels for the desired data to appear or makes a pull request to the server for an on-demand delivery of the data.

2.2 Broadcast Scheduling

To efficiently deliver data on broadcast channels, the content organization and data broadcast scheduling could be determined based on client access patterns. A simple broadcast scheduling method, called *flat broadcast* (denoted as *FLAT*), broadcasts each data item only once in a broadcast cycle. However, with a skewed access pattern, flat broadcast may not perform well. In this situation, *broadcast disks* (denoted as *B_DISK*) [1] is an excellent broadcast scheduling method. With broadcast disks scheduling, the server divides a broadcast cycle into groups called *disks*. Each disk is broadcast with a different frequency in a broadcast cycle in order to imitate multiple disks spinning at different speeds. The broadcast frequencies of disks are in proportion to the probability of access in order to gain the optimal access time of the system. In this paper, broadcast disks and flat broadcast schedules are referred to as *broadcast programs*.

2.3 Client Access Statistics Collection

To guarantee the success of the broadcast scheduling, a server has to *know* the data access patterns in its user population, which may change with time. For example, during rush hours, users normally have more interest in traffic information. At noon, some people may like to take a break to check their stock portfolio. In the evening, some shoppers may be engaged in an auction event. To obtain the dynamically changing data access patterns of users, a bit vector mechanism was proposed in [5]. By receiving piggyback client access information along with the pull requests from the clients, a server can record the number of pull requests for each data item and the number of hits produced by its broadcast programs.

In the following, we briefly describe the method we used to capture data access patterns[5]. In this method, each data item i is associated with a score, $\delta_{i,j}$, where j represents j-th broadcast evaluation cycle. An evaluation cycle is a broadcast cycle where the scores of data items are computed. As

[4]On-demand services are implemented with point-to-point channels. Thus, we use on-demand channels to refer to the point-to-point channels where it's appropriate.

[5][14] proposed a similar method called EWMA.

a result, broadcast programs are adjusted based on the item scores. The data items with higher scores are more likely to be broadcast. The period of an evaluation cycle is called the *evaluation period* (denoted as ϵ). Let $\beta_{i,j}$ denote the client access statistics accumulated for item i between *(j-1)*-th and j-th evaluation cycles. The score is computed as, $\delta_{i,j} = \lambda\delta_{i,j-1} + \beta_{i,j}$, where $\lambda < 1$ is a decaying factor and $\epsilon \geq 1$. The λ guarantees that the impact of an old access frequency decreases with time.

2.4 Indexing and Schedule Caching

There are a few indexing methods for wireless data broadcast appeared or to appear in the literature [7, 8, 9, 12, 13]. The general idea is to interleave index information with data on the broadcast channels such that the mobile clients, by first retrieving the index information, are able to obtain the arrival time of the desired data items on the broadcast channels . Based on the index information, the client may select between broadcast and on-demand channels to obtain data with a smaller access time.

The integrated signature method was discussed in details in [12, 13]. In this method, a signature is broadcast before a group of consecutive items from which that signature is constructed. However the signature methods developed in [12] are for energy efficient data retrieval. An *access efficient* signature method is discussed in this paper.

To provide accurate broadcast schedule, the server interleaves integrated signatures with data for broadcast. In this paper, we propose to use a signature for indexing $SigGroup$ of the up-coming data frames. The number of items between two signatures, $SigInterval$, may be less or equal to $SigGroup$. By selecting the appropriate $SigInterval$, we may decide the degree of overlap between two consecutive signature groups. On the client side, a query signature is generated in a similar way based on the query specified by the user. By checking the signatures broadcast, the mobile computer can decide whether the forthcoming item group contains the desired data or not. Although signatures incur index overhead, such an overhead is low since the signature size is usually very small.

An alternative approach for predicting the arrival of data items is the *cached schedule* method. Instead of broadcasting indexing information with data frames, the complete broadcast schedule is broadcast at the beginning of each cycle. The clients monitor the broadcast channels to retrieves schedules into their caches. Based on the schedules, a client can make selection between broadcast and on-demand services to minimize access time. A major drawback of this approach is the update of cache due to disconnected operations by mobile clients [4].

Contrast to the indexing approaches, a server may choose to broadcast only the data on the broadcast channels. We call this approach *non-index*. Since there is no index overhead for the broadcast channel, the data management for broadcast is simple. Since the mobile clients do not know when the requested data frames will appear on the broadcast channels, it may be difficult for them to determine a better data access method between broadcast mode and on-demand mode.

3 Hierarchical Data Dissemination

In this section, we introduce a hierarchical data dissemination system for wireless data access. In this system, caching, broadcasting and pull-based delivery are used together to minimize the access time. Logically speaking, data are stored in a hierarchy of media where the most frequently accessed data are cached in the client, the commonly requested data subset is temporarily stored on broadcast channels, and the rest of data must be pulled from server via explicit client requests. Data caching and push-based data dissemination alleviate pull-based request considerably, as most frequently access data are retrieved either from the client's cache or the broadcast channel. On the other hand, requests for infrequently accessed data can always be served on point-to-point channels.

When a user issues a query to the mobile client, the client first searches its cache. If there is a valid copy in the cache, an answer is replied immediately. Otherwise, the client attempts to obtain the data item from the server site. The data access operations at the client is dependent on the indexing methods supported by the server. When the non-index method is used, the client first monitors the next $Threshold$ number of data frames on the broadcast channels. If the desired data items are found, then they are retrieved into the client cache. If they are not found after monitoring $Threshold$ data frames, the client turns to on-demand data service and issues a pull request to the server. When the cached schedule approach is adopted by the server, the client checks the broadcast schedule in its cache. If the desired data items will appear within the next $Threshold$ number of data frames on the broadcast channels, the client monitors the broadcast channels. Otherwise, the client issues a pull request to the server immediately. Finally, if the integrated signature approach is adopted by the server, the client first monitors the data items and the signatures on the broadcast channels. If the signatures indicate that the desired data item will be broadcast in the cycle and that the number of data frames before the desired data item appears in the broadcasting cycle is less than $Threshold$, then the client continues to monitor the broadcast channels and retrieves the item into the cache when it arrives. Otherwise, the client issues a pull request to the server.

4 Performance Model

In this section, we describe a simulation model used for evaluation of the *HDD* system. The simulation model consists of

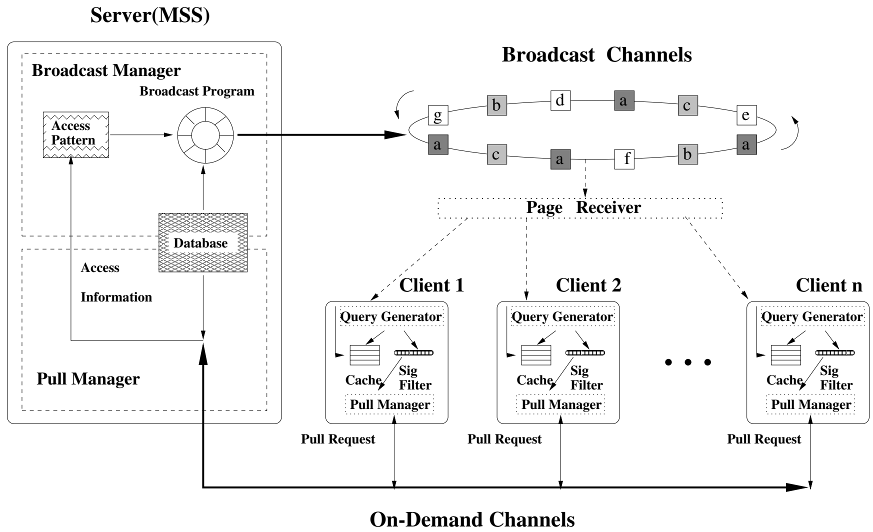

Figure 1: Simulation Model.

$DataItemSize$	Size of a data item in bytes
$NumClient$	Number of clients in a cell
$DatabaseSize$	Number of data items in the database
$NumChannel$	Number of channels in a cell
$pullVSpush$	Ratio between the broadcast and the on-demand channels
$DownlinkBW$	Channel bandwidth from the server to the client
$UplinkBW$	Channel bandwidth from the client to the server

Table 1: System Parameter Description.

a single server, a set of clients, and a fixed number of channels (Figure 1). Some of the channels work in on-demand mode, while the rest is in broadcast mode[6].

Table 1 describes the configuration and the physical resource parameters of a cell. The database is modeled as a collection of $DatabaseSize$ data items. Each item has $DataItemSize$ bytes. There are $NumChannel$ channels in a cell. For the exclusive channel allocation approach, all $NumChannel$ channels work in either on-demand mode or broadcast mode. For *HDD* approach, the channel allocation made between broadcast and on-demand modes is fixed such that the ratio between the number of broadcast channels and the number of on-demand channels is $pullVSpush$. The downlink bandwidth $DownlinkBW$ is greater than uplink bandwidth client $UplinkBW$.

4.1 The Client Model

Each client is modeled by a process[7], which consists of a sub-process called *query generator*. After the current query request is finished, *query generator* waits for a period of $ThinkTime$ and then makes the next query request. $ThinkTime$ is a parameter used to model workload processing as well as the relative speeds of the CPU. Table 4.1 presents the parameters used to model the resource and the data access pattern of each client.

Query generator runs a continuous loop that requests a data item according to a specific distribution (*Zipf* or *Gaussian*). Each client has $CacheSize$ cache. Sub-process *page receiver* keeps on monitoring the broadcast channels and becomes active when a data item arrives. For each client which is monitoring the broadcast channels, it checks whether the forthcoming item is the desired one for that client. If the data item is the requested one, it is brought into the client's cache.

When the client finds that the desired items cannot be obtained within the next $Threshold$ items on the broadcast channels, it stops monitoring the broadcast channels and turns to the on-demand service by activating a sub-process called *pull manager*. Once activated *pull manager* issues a pull request (size of $PullReqSize$ bytes) for a point-to-point connection with the server. Upon receipt of the pull request, the server tries to establish a connection with the client and returns the data item to the client when the connection is established.

Two different distributions, *Zipf* (also used in [10]) and

[6] The models for exclusive channel allocation are special cases of the *HDD* model.

[7] [2, 16] modeled a large client population by one client process. Since the client-server model is not memoryless due to data caching, the interleaving of push and pull slots, and the bounded server process capability, a single client process may not exactly reflect the true contention on the uplink channels among the clients.

$CacheSize$	Client cache size (in data items)
$PullReqSize$	Size of a pull request in bytes
$ThinkTime$	Mean think time (in seconds) between queries for each client
$Threshold$	Threshold for the client to select between broadcast and on-demand
$AccessRange$	Queries access range
θ	Zipf distribution parameter
σ	Width of hot-spot for Normal distribution
μ	Center of hot-spot for Normal distribution

Table 2: Client Parameter Description.

$ProgramSize$	Number of distinct data items in broadcast program
$DiskNum$	Number of disks for B_DISK
$DiskSize_i$	Size of disk i (in data items)
$DiskFreq_i$	Relative broadcast frequency of disk i
$SigSize$	Integrated signature size (in bytes)
$SigGroup$	Number of data items integrated in a signature
$SigInterval$	Interval between two consecutive signatures
ϵ	evaluation period
λ	decaying factor for the computation of item scores

Table 3: Server Parameter Description.

Gaussian (used in [16] as well), are used to model the data access patterns in the experiments. The *Zipf* distribution (with parameter θ) is frequently used to model skewed access patterns where θ is a parameter named *access skewness coefficient* and can be varied from zero to one. The distribution becomes increasingly "skewed" as θ increases. The *Gaussian* distribution, $Normal(\mu, \sigma)$, is used to model the dynamic changes in the access patterns with the center of hot-spot μ and the width of hot-spot σ. During the experiments, the value of μ can be varied to create the effect of dynamic workload and the value of σ reflects the skewness of client access patterns. Only data items within $1 \sim AccessRange$ are accessed.

4.2 The Server Model

The server performance is modeled by the process *broadcast manager*. Table 4.1 gives the parameters used to describe the resource of the server, the structure of the broadcast program, and the index built on the broadcast data. For B_DISK, only the content of the broadcast program is tailored to meet the access patterns collected. The structure of the program is determined by several parameters: $ProgramSize$ is the number of data items scheduled to be broadcast in the program, $DiskNum$ is the number of disks in the program, and $DiskSize_i, i \in [1, \ldots, DiskNum]$, is the number of data items assigned to disk i which is broadcast at a speed of $DiskFreq_i$. In addition to the broadcast schedule, parameters for the estimation of data access patterns and for the signatures are also described in the table.

5 Experiments and Results

Table 5 defines the system parameter settings used in the experiments. We assume that there are 20 channels in a cell. Therefore, there are 20 channels for on-demand service or for broadcast service when exclusive channel allocation methods are used. For *HDD*, the number of broadcast channels is equal to the number of on-demand channels (i.e., $pullVSpush$ is 1:1).

The client population in a cell varies from 10 to 1000. The server maintains a database of 3000 self-identifying data items. All data items have the same size (1000 bytes). A pull request size is 10 bytes. The downlink bandwidth is 10000 bytes/second and the uplink bandwidth is 100 bytes/second (1% of the downlink bandwidth). The client cache size is 50 data items. LRU-K scheme is used to manage cache replacement [3]. In LRU-K scheme[8], the cache management keeps track of the times of the last K references to popular data items and selection is based on the past K hits history. The mean think time between two consecutive queries is 10 seconds.

The number of items grouped in a signature is 500. The signature has the smallest size which still guarantees low false drop probability of the signature. Through experiments, 256 bytes were found to be the appropriate signature size for 500 data items. The number of distinct data items in the broadcast program (for both B_DISK and FLAT) is 600, while the client access range is $1 \sim 1000$. Two disks are

[8] In [6], LRU-K police is shown to outperform LIX policy proposed in [1] when dynamic broadcast programs are used in the system.

Parameters	Values	Parameters	Values
$DataItemSize$	1000 bytes	$NumClient$	$10, \cdots, 1000$
$DatabaseSize$	3000 data items	$SigSize$	256 bytes
$AccessRange$	1000 data items	$SigInterval$	100 data items
$PullReqSize$	10 bytes	$GroupSize$	500 data items
$NumChannel$	20	$DownlinkBW$	10000 bps
$pullVSpush$	1:1	$UplinkBW$	1 % of downlink bandwidth
$ThinkTime$	10 seconds	$Threshold$	600 data items
$ProgramSize$	600 data items	$DiskSize_i$	$DiskSize_1$=200,$DiskSize_2$=400
$DiskNum$	2	$DiskFreq_i$	$DiskFreq_1$=2,$DiskFreq_2$=1
$CacheSize$	50 data items	θ	0.95
σ	100 data items	μ	randomly selected winthin $AccessRange$
ϵ	10 cycles	λ	0.9

Table 4: Parameter Settings for the Experiment.

used in *B_DISK* where the fast disk size is 200 data items, while the slow disk size is 400 items, with the relative spin speeds being two and one, respectively.

We examine the performance of the exclusive channel allocation approaches and *HDD* for the dynamic workload environments. There are two circumstances when the dynamic workload happens. The first case is when the clients join the system with empty cache and the server does not know the client access patterns. In the second case, the existing clients change their access hot-spots with time. For *HDD*, the server first randomly selects a data set for the broadcast program. Obviously, for the the exclusive channel allocation approach, dynamic workload only affects the client cache; for *HDD*, dynamic workload has influences on not only the client cache but the content of the broadcast program as well.

We define a client to be in the *stable stage* if the client has performed at least 4000 accesses after the client cache is filled. A cell is at stable stage when all clients in the cell are in the stable stage. Otherwise the cell is in the initial stage. Unless otherwise specified, the access time results are obtained when the cell reaches a steady state such that the warm-up effects in the client cache and the broadcast program are eliminated. In the experiments, the primary performance metric employed is the average access time, measured in seconds. The simulation is implemented using CSIM [15].

In what follows, we investigate the system performance subjected to client populations, index methods, threshold values, hybrid channel allocation schemes, and data access patterns. The purpose is to examine the adaptiveness of the system for different workloads and system parameter settings. The client access patterns are assumed to follow the *Zipf* distribution with the default Zipf parameter set to 0.95. Finally, we evaluate the influence of dynamically changed client access spots to the system performance. The client access patterns are assumed to follow the *Gaussian* distribution.

5.1 Client Performance

In this set of experiments, the adaptiveness of *HDD* for firstly joined clients is evaluated. To provide comparison baselines, we introduce the ideal *HDD*: *ideal Flat* and *ideal broadcast disks* with respect to *FLAT* and *B_Disk*. For the ideal ones, the system (both the clients and the server) knows the exact client access patterns and *HDD* disseminates information according to the optimal access distribution such that the most frequently accessed data subset is cached in client, the less frequently accessed data subset is provided by broadcast, and the rest is pulled from the server by explicit requests. The broadcast program for broadcast disks is also constructed according to the access probability of the data subset. The ideal *HDD* is the ultimate performance goal of our system.

5.1.1 Adaptiveness under Different Index Schemes

To study the influence of client population on performance, the average access time is evaluated as a function of the client population. Figure 2 compares B_DISK and FLAT using different index methods (i.e., cached schedule, signature, and non-index) with the ideal *HDD* (labeled as OPT_BD and OPT_FLAT). For clarity and brevity, only the system performance in the stable stage is shown. As an aside, experiments showed that the system performs better in the stable stage than in the initial stage because of better cache performance and more accurate estimate of access statistics when the system is operating beyond the initial stage.

For different index methods, the cached schedule always does the best while non-index always does the worst for both FLAT and B_DISK. This is because the cached schedule can provide precise broadcast schedule and as such allows the clients to immediately decide whether to monitor the broadcast channel or go uplink for point-to-pint connection without monitoring the broadcast channels. At the other extreme, the non-index approach provides no broadcast schedule in-

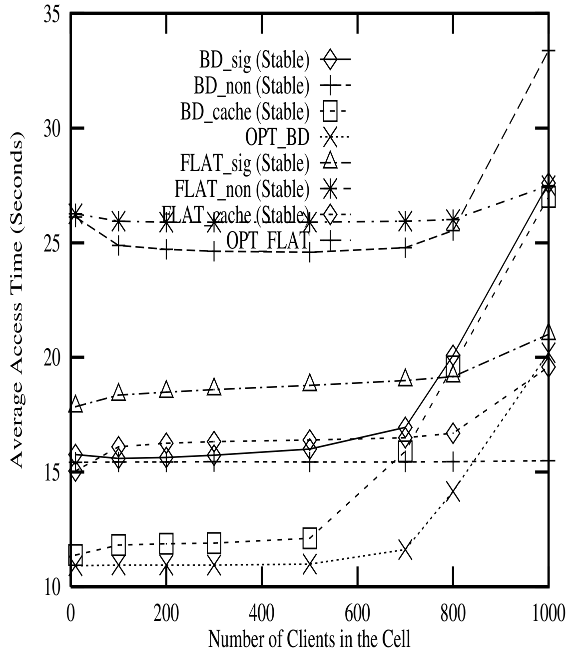

Figure 2: Access Time vs Client Population.

formation to the clients and the clients have to monitor the broadcast channels first. The signature provides index information to the clients and hence, by retrieving the signatures, the clients are able to make selection between broadcast and on-demand services ahead of time. Thus, the performance for signature index does significantly better than non-index approach.

It is obvious that both FLAT and B_DISK manage to follow the ideal ones very well. There is a certain gap between the real approaches with the ideal approaches due to the minor imprecision of the access patterns collected by the server comparing to the ideal access patterns. Moreover, the non-ideal LRU_K page replacement strategy may introduce some hit-misses for cache management. For the signature and the non-index methods, the need to monitor the broadcast channels for the desired pages increases the gap. Although FLAT and B_DISK may never achieve the ideal performance, the dynamic broadcast program is still an effective approach because of its adaptiveness to the system.

When the cell is not under heavy-loaded (i.e., clients population less than 700 in Figure 2), B_DISK always gives a better performance than FLAT approach. This is consistent with the results obtained for static broadcast program [2]. However, for heavy-loaded cells, the performance of B_DISK is worse than that of FLAT. In Figure 2, we can see that FLAT outperforms B_DISK when the client population is greater than 800 for the non-index and the signature methods and 700 for the cached schedule method. There are two possible reasons for that. First, the accuracy of client access patterns is more important to B_DISK than to FLAT. As mentioned above, the access information collected becomes less precise when the cell is heavy-loaded. Second, within the same $Threshold$ slots of the broadcast channels, FLAT provides more distinct data items than B_DISK. Thus, there are more query requests turning to the pull based services in B_DISK than in FLAT.

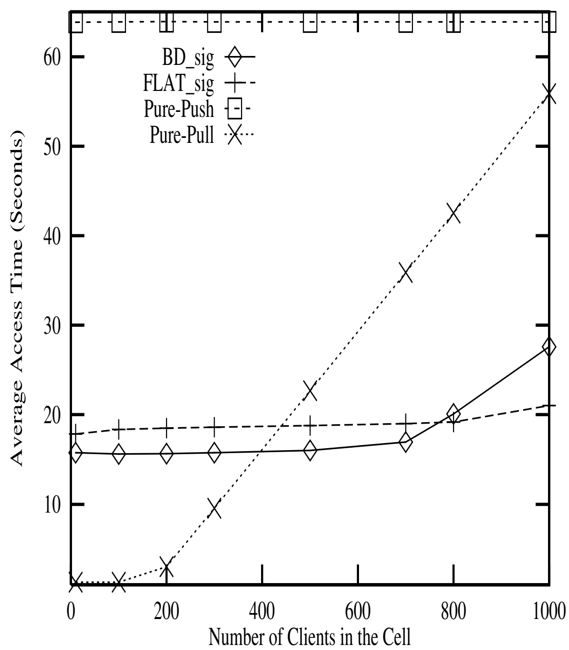

Figure 3: Access Time vs Client Population.

Figure 3 contrasts the performance of the index methods with the exclusive broadcast (Pure-Push) and exclusive on-demand methods (Pure-Pull). For clarity, we only include the performance of the signature method under FLAT and B_DISK since its performance is in between the non-index and cached schedule methods. As expected, the *Pure-Pull* approach is good only when the system workload is light (left side of the figure). The access time for this method degrades rapidly as the client population grows. When the number of clients is beyond 600, *Pure-Pull* performs worse than B_DISK and FLAT for any index methods. Since the page replacement algorithm for client caching is the same for *HDD* and *Pure-Pull*, the improvement of the performance for *HDD* over *Pure-Pull* is a result of the tailored broadcast program. The performance of *Pure-Push* is not affected by client population and always does the worst. Compared with *Pure-Push*, *HDD* manages to broadcast only the hot-spot on the channel and hence avoids wasting broadcast bandwidth.

5.1.2 Evaluation of Signature Schemes

To find out the influence of the number of items grouped in a signature, we vary the number of items in the group from 100 to 600 while the signature period is fixed to 100 (see Figure 4). Three different client populations, light-loaded cells (i.e., 100 clients), medium-loaded cells (i.e. 500 clients), and heavy-loaded cells (i.e., 1000 clients), are evaluated. For any client population, the greater the number of items in a sig-

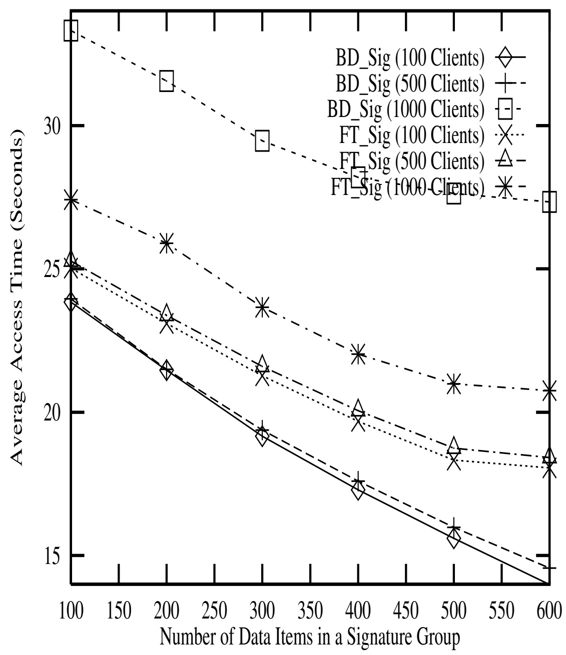

Figure 4: Signature Group Size vs Access Time.

nature group the better the system performance. However, when the number of items in a group reaches 500, the improvement becomes flat. This is because when the broadcast program consists of 600 items most of items are already included in the group.

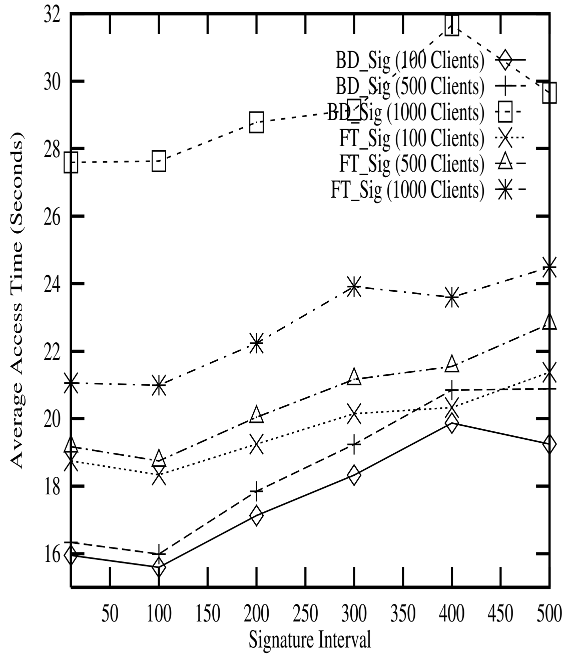

Figure 5: Signature Interval vs Access Time.

In Figure 5, the number of items in a group is fixed to 500 while the signature broadcast period is varied. Obviously, period 100 is the best choice. In these two sets of experiments, except heavy-loaded cells, B_DISK always outperforms FLAT.

5.1.3 Impact of Threshold

Since the frequently accessed data set is cached in clients and the rarely accessed part of the database is explicitly pulled from server database, the broadcast program can be made very compact. For example, in our model, 600 data items are broadcast. Compared with broadcasting the entire database approach (i.e., *Pure-Push*), this effectively avoids wasting scarce broadcast bandwidth. Furthermore, it imposes an upper bound to the waiting time for data retrieval from the broadcast. For FLAT, the average waiting time is half of the entire broadcast program, while for B_DISK it is less than half of the entire broadcast program.

In addition to tailored broadcast programs and index methods, a proper threshold mechanism is needed for the effectiveness of the broadcast channels. While the broadcast channels are the second hot-spot (next to the cache) where mobile clients may receive the desired data items, the threshold mechanism allows a client to determine whether it should stay with the broadcast channels patiently or switch to on-demand service. If the desired data usually appears on the broadcast channel within the preset threshold period, a considerable number of explicit pull requests can be reduced because of high *air hits*. Consequently the workload of the server is alleviated and on-demand channel congestion is avoided. Figures 6 and 7 give the experimental results related to the threshold mechanism. In this set of experiments, the signature index is used.

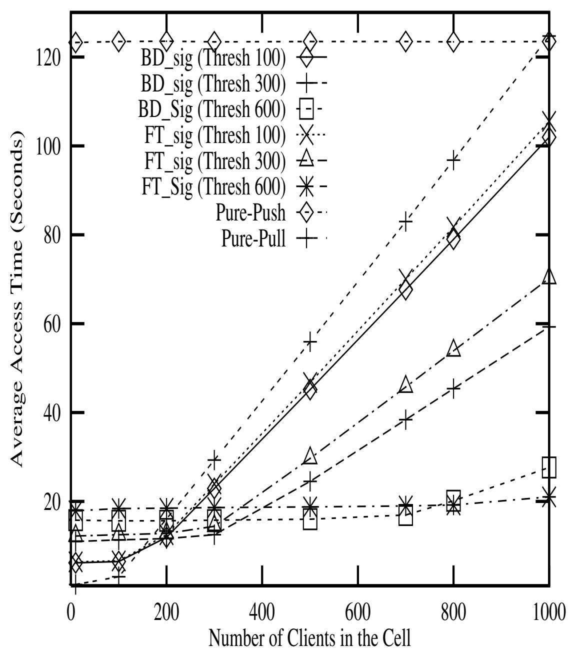

Figure 6: Access Time vs Client Population with Various Threshold Values.

Figure 6 shows the impacts of client population on the access time for three threshold values: 100, 300, and 600, which control the utilization of the air cache from low to high level. The thresholds are given as the number of data

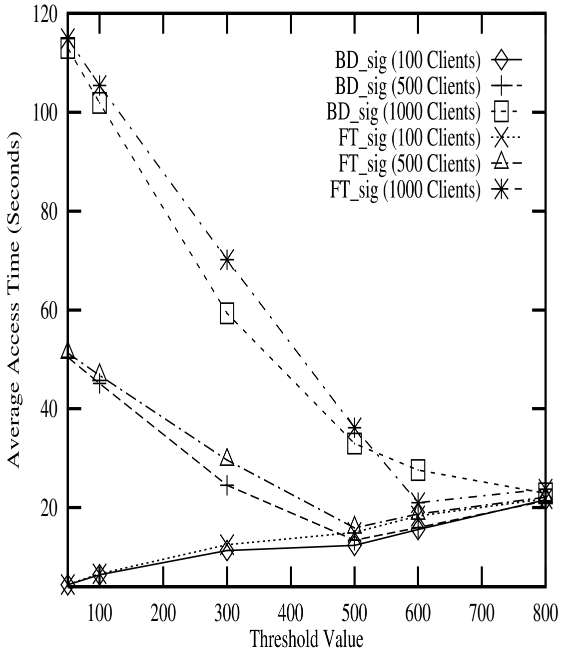

Figure 7: Threshold Value vs Access Time.

items in a broadcast cycle. For example, a threshold 100 would restrict the client from sending pull requests for any items that would appear within the next 100 data items of the broadcast program. For small threshold (i.e., 100), the performance is similar to *Pure-Pull* approach and is very bad. The reason is that only a small percentage of cache misses can get their answers within the next 100 pages of the broadcast cycle and most of the cache misses goes uplink to the server. Due to the low utilization of the broadcast data service, *HDD* has a similar performance as the *Pure-Pull* approach. In general, the larger is the threshold value, the shorter is the access time. The only exception is when the cell is light-loaded where *HDD* with small threshold value tends to yield a good performance as in *Pure-Pull*. This is because there is plenty of on-demand bandwidth to satisfy the requests of a small client population and small threshold lets the client spends less time in estimating the arrival time of the desired data.

Figure 7 shows the results of the threshold values as a function of the access time. For medium-loaded cells (500 clients) and heavy-loaded cells (1000 clients), the access time decreases as the threshold values are increased. However, after the threshold values reach a certain value (around 600, the whole broadcast cycle), further increase would degrade the performance. For light-loaded cells (100 clients), the access time always increases as the threshold values are increased. However, the amount of increment is small. Hence, it can be concluded that a large threshold value should be used in *HDD* to efficiently utilize the data on the broadcast channel.

5.1.4 The Influence of Channel Allocation

Intuitively the percentage of the channels allocated between broadcast and on-demand data services has direct impact on the relative performance of *HDD*. In the previous experiments, the percentage of on-demand channels (or broadcast channel percentage) is set to 50% of the total number of channels in a cell. Figure 8 compares the performance of *HDD* with different channel allocation schemes. For example, on-demand bandwidth takes up 30%, 50%, and 70% of the total channel bandwidth. In the experiments, the same signature index is used (i.e., item group size is 500 and signature interval is 100) and the threshold is set to 600 for both B_DISK and FLAT.

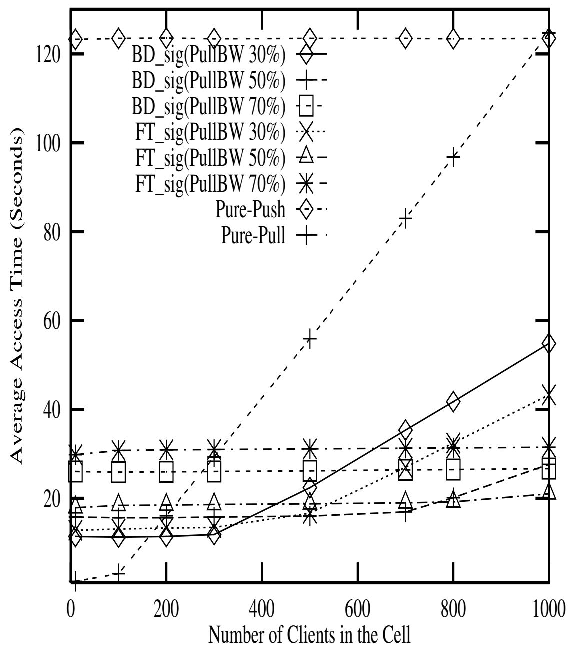

Figure 8: Impacts of Bandwidth Allocation.

When most channels work in on-demand mode (i.e., curves labeled with pullBW 70%), the system performance is similar to pure-push method. For example, both B_DISK and FLAT give similar performance regardless to the client population, because the majority of the query requests is answered by data cache and data broadcast and there are enough channels for pull-based data delivery. In contrast, when most of channels are in broadcast mode (i.e., curves labeled with pullBW 30%), the system has good performance for light-loaded cells but poor performance for heavy-loaded cells, because the small number of on-demand channels is not able to satisfy the pull-based data delivery for heavy-loaded cells. When the number of channels in broadcast and on-demand modes is the same (i.e., curves labeled with pullBW 50%), in general, the system performance is better than the other two allocation schemes.

5.1.5 Influence of Access Patterns

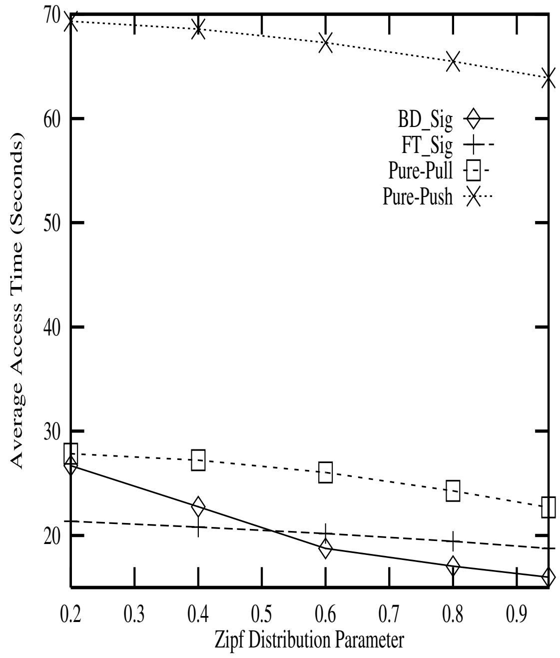

Figure 9: Influence of Client Access Pattern.

Figure 9 shows the impact of client access patterns on the performance of *HDD*. In the experiments, the Zipf distribution parameter θ is varied while the number of clients is set to 500. To make comparisons, *Pure-Push* and *Pure-Pull* are also included in the figure. As the access pattern becomes increasingly skewed (right part of the figure), all methods give better performance. However, the amount of improvement is different. For example, B_DISK has much greater improvement than the other methods. When the data access patterns are flat (i.e., at the left part of the figure), FLAT has a better performance than B_DISK and the situation is reversed when data access patterns become increasingly skewed (i.e., the right part of the figure). This is consistent with the intuition that B_DISK does better than FLAT only for skewed data access patterns.

5.2 Performance for Changing Hot-Spot

To evaluate the performance of *HDD* in dynamic workloads, a set of experiments is conducted under the assumption that the client access patterns follow the *Gaussian* distribution (refer to Figure 10). The effect of dynamic workload is created by varying the focus of the client access patterns. This is achieved by eliminating a hot-spot μ and randomly generating a new hot-spot in the server database. To interpret the impact of hot-spot migration on the system performance, we assumed that all clients have the similar hot-spots and they change access demand at the same time. Clients stay with each hot-spot for *Duration* periods of simulation time.

In Figure 10, we show the results obtained as a function of *Duration* minutes, where the client population is 500, the

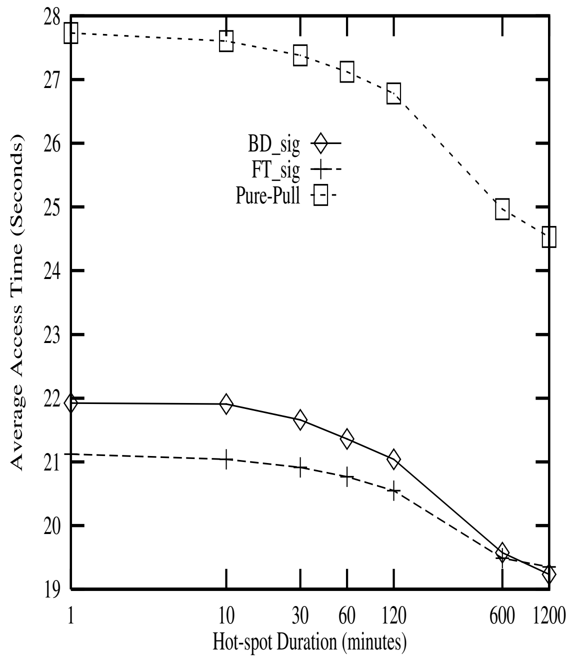

Figure 10: Migration of Hot-spot.

data access range is between $1 \sim 1000$, and σ is set to 100. To provide the comparison baseline, *Pure-Pull* is also included in the figure. Obviously for all three methods, the performances are improved when the *Duration* is increased. *Pure-Pull* performs the worst compared with *B_DISK* and *FLAT*.

If we focus on B_DISK and FLAT, FLAT does better than B_DISK for small $Duration$ such as $Duration < 600$ minutes, but B_DISK performs better when $Duration$ is greater than 600 minutes. This is because it takes time for the server to learn the correct data access patterns. Furthermore, B_DISK has a stricter requirement on the precision of the data access patterns than FLAT. With small $Duration$, the broadcast program of B_DISK may not reach steady state or the matured broadcast program is obtained shortly before the clients change access hot-spot to a new one. As a result, the benefit of B_DISK has no time to show. Only when the system reaches steady state and stay there long enough would B_DISK gives better performance than FLAT.

6 Conclusion

In this paper, we evaluate the performance of a hierarchical data delivery system for asymmetric communication environments, where information can be retrieved from different media (i.e., client cache, broadcast channels, and the server database by pull requests) to achieve the maximum access advantages. The way data are disseminated is determined dynamically by the access patterns of the clients, not by precompiled user profiles. The broadcast program is tailored according to access patterns so that only the most desired data subset is scheduled to broadcast. Pull requests are needed only when the desired data cannot be found in the client cache or from the broadcast program.

A simulation model was developed to evaluate the performance of the data delivery system. We found that the system adapts to the dynamically changing workload very well. We demonstrated that with the appropriate threshold, index methods and bandwidth allocation methods can improve the performance and scalability of the system. In general, broadcast disks and flat broadcast have better performance than pure pull approach when data access patterns are skewed. For skewed data access patterns and less dynamic workloads, broadcast disks outperform flat broadcast. However, pure pull technique is a better choice for random data access patterns or highly dynamic workloads.

As future work, we will investigate ways to dynamically adjust the ratio between pull and push bandwidth according to the server workload and the impact of different broadcast program structure (i.e., the size of broadcast program etc.) on the performance of data delivery. Moreover, we will include the dissemination of updates in the hierarchical data delivery system.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 199–210, San Jose, California, May 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 183–194, Tuscon, Arizona, May 1997.

[3] E.J.O'Neil, P.E.O'Neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 297–306, Washington, D.C, May 1993.

[4] A. K. Elmargarmid, J. Jing, and T. Furukawa. Wireless client-server computing for personal information services and applications. *ACM Sigmod Record*, 1995.

[5] Q. L. Hu, D. L. Lee, and W.-C. Lee. Dynamic data delivery in wireless communication environments. In *Workshop on Mobile Data Access*, pages 213–224, Singapore, November 1998.

[6] Q. L. Hu, D. L. Lee, and W.-C. Lee. Hierarchical data dissemination and access in asymmetric communication environments. *Submitted to Journal of Parallel and Distributed Computing: Special Issue on Wireless and Mobile Computing and Communications (JPDC)*, 1999.

[7] Q. L. Hu, W.-C. Lee, and D. L. Lee. A hybrid index technique for power efficient data broadcast. *Distributed and Parallel Databases Journal*, to appear.

[8] Q. L. Hu, W.-C. Lee, and D. L. Lee. Indexing techniques for power management in multi-attribute data broadcast. *ACM/Baltzer Mobile Networking and Applications (MONET)*, to appear.

[9] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficiency indexing on air. In *Proceedings of the International Conference on SIGMOD*, pages 25–36, 1994.

[10] D. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, CA, 1981.

[11] W.-C. Lee, Q. L. Hu, and D. L. Lee. A study of channel allocation methods for data dissemination in mobile computing environments. *Mobile Networking and Applications (MONET): Special Issue on Resource Management in Wireless Networks*, to appear.

[12] W.-C. Lee and D. L. Lee. Using signature techniques for information filtering in wireless and mobile environments. *Special Issue on Database and Mobile Computing,Journal on Distributed and Parallel Databases*, 4(3):205–227, July 1996.

[13] W.-C. Lee and D. L. Lee. Signature caching techniques for information filtering in mobile environments. *ACM/Baltzer Wireless Networks (WINET)*, 5(1):57–67, January 1999.

[14] H. V. Leong and A. Si. Database caching over the airstorage. *The Computer Journal*, 40(7):401–415, 1997.

[15] H. Schwetman. *Csim user's guide (version 17)*. MCC Corporation, 1992.

[16] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd VLDB Conference*, pages 326–335, Athens, Greece, August 1997.