UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

PUSH LESS AND PULL THE CURRENT HIGHEST DEMANDED DATA ITEM TO DECREASE THE WAITING TIME IN ASYMMETRIC COMMUNICATION ENVIRONMENTS

Cristina Maria Pinotti and Navrati Saxena

October 2002

Technical Report # DIT-02-0076

Push less and pull the current highest demanded data item to decrease the waiting time in asymmetric communication environments

Cristina M. Pinotti Navrati Saxena
Dept. of Computer Science and Telecommunications
University of Trento, Italy
E-mail: pinotti@science.unitn.it, navrati@science.unitn.it

Abstract

In these days, the world experiences an unprecedented demand for data and data services, driven mainly by the popularity of the Web services and by the evolution of the Internet towards an information super-highway. We introduce a hybrid scheduling that effectively combines broadcasting for very popular data (push data) and dissemination upon-request for less popular data (pull data) in asymmetric communication environments. In our solution, the server continuously broadcasts one push item and disseminates one pull item. The clients send their requests to the server, which queues-up them for the pull items. At any instant of time, the item to be broadcast is designated applying a pure-push scheduling, while the item to be pulled is the one stored in the pull-queue, which has accumulated, so far, the highest number of pending requests. The value of the average expected waiting time spent by a client in the hybrid system, denoted by $T_{exp\text{-}hyb}$, is evaluated analytically, and the cut-off point between push and pull items is chosen in such a way that $T_{exp\text{-}hyb}$ is minimized. We find out that by doing so we can drop the cut off point to a value, which is much less than the total number of items present in the system, improving upon the average waiting time spent by a client in a pure push system and also on that spent in some of the hybrid systems already proposed in literature.

1. Introduction

Day by day the ability to interconnect computers through cable, satellite and wireless networks is increasing and proportionately to this is also increasing a new application based on *data dissemination*. This application focuses on delivering data to a large population of clients. Often in dissemination-based systems, there exists communications' asymmetry. This asymmetry may arise due to several factors like:

- The downstream communication capacity (bandwidth from server to client) may be much greater than the upstream communication capacity (bandwidth from client to server);
- In Information Retrieval Applications, the clients make requests to the server through small request messages that result in the transfer of much larger objects;
- Systems with small number of servers an d large number of clients also result in such asymmetry.

Basically, there are two approaches to spread data items in such systems: the Push-based data scheduling, and the pull-based data scheduling.

A system where the client simply grabs the data being broadcast without *making any requests* is an example of *push-based system*. In such systems, the clients continuously monitor the broadcast process and retrieve the data items they require. The server, on the other hand, broadcasts data items on scheduled time no matter whether the particular item is being required at that time or not. On the contrary, *pull-based systems* are o*n demand* traditional client server systems where clients and server have a request/response style of relationship. In such systems, the clients initiate the data transfer by sending requests and the server then makes a schedule to satisfy the clients' requests.

Both push- and pull- based scheduling have their own advantages and disadvantages; as shown in [4, 17] neither push nor pull based scheduling alone can achieve the optimal performance. A better performance is achieved when the two scheduling approaches are used in a combined manner.

In this paper, we divide the data items in two disjoint sets – *push-set* and *pull-set*. A push-based scheduling is then used to broadcast the items in the push-set, whereas a pull-based scheduling is used to disseminate those in the pull-set. The system performance metric for our system is the average expected access time experimented by the clients, which depends upon the push scheduling, the pull scheduling and heavily upon the criteria used to partition the items in the push- and pull-sets.

The rest of the paper is organized as follows: Section 2 reviews the past-related work that has been done in this area. Section 3, after introducing some preliminaries, offers motivations behind using our new hybrid system. Section 4 describes the behavior of the server and of the clients in the new hybrid scheduling. Besides, it devises the analytic evaluation of the average expected waiting time based on the push-based and pull-based scheduling adopted. In Section 5, the experimental results are reported. Finally, conclusions are offered, along with some discussions on future work, in Section 6.

2. State of the art: past related work

Scheduling theory has been studied since decades. Among the research on broadcast scheduling, Acharya et al. [1-3] came up with the idea of Broadcast Disks. In such an approach, data items are assigned to several disks of different sizes and speeds, and are then multiplexed on a single broadcast channel. From this channel, the clients can retrieve data items based on their probability of access. In effect, items of higher access probability are assigned to faster disks than the items of low access probability, which are assigned to slow revolving disks.

Later, Jain and Werth [18] proved that the optimal expected access time results when the instances of each item to be pushed are equally spaced, while Bennett and Zhang [9] determined which packet from many input queue should be transmitted next in the output channel so that the channel is used in a fair way. The fact that the push broadcast scheduling problem was related to Packet Fair Queuing was brought up by Vaidya and Hameed in [12, 14], who also studied scheduling for multiple broadcast channels and the impact of the transmission error on scheduling.

For broadcast disks with polynomial cost functions, Bar-Noy et al. [6] presented an asymptotically optimal algorithm for a fluid model, where the bandwidth may be divided to allow for fractional concurrent broadcasting. They also give a greedy algorithm reaching the best performance in most cases.

Mostly all of the above scheduling algorithms assume that the server knew the access probability of all the items in advance, however, in real time this is not the case. This problem was tackled in [13, 15]. Precisely, [13] proposed to use broadcast misses to understand the access patterns and Yu, Sakata and Tan [15] presented a statistical estimation model to estimate access probability.

Besides, deadline constraints have been integrated into the Broadcast Disks model in [7, 8, 10], where, the server tries to compute a periodic schedule that provides worst case guarantees, even in the event of failures and data updates. However, this model is not bidirectional, that is, there is no uplink channel and consequently the server periodically broadcasts items based on a static estimation of the potential user population, not on the actual load.

Although the push-based systems have attracted for their high capability to scale up, those systems may sacrifice some users' needs to the public interest. Based on that, more recently, on-demand dissemination scheduling have been pursued (see [2] for a complete review) which propose advanced scheduling policies allowing, for example, preemption and using sophisticated performance metrics, like data item stretch.

In addition, many researchers, including [3, 5, 19], realized that caching and prefecthing can save much of the expected access time and thus are important for both push and pull based data dissemination scheduling. In particular, in [19], an efficient gain-based cache replacement policy, called SAIU, is designed for on-demand scheduling that balance individual and overall system needs. SAIU integrates in its performance measure the influence of the data retrieval delays, of the data sizes, of the data access probabilities as well as of the data update frequencies.

Clearly, hybrid approaches, that use both the push-based and the pull-based scheduling algorithms in one system, appear to be attractive to meet both the massive data dissemination and the upon-request data delivery. Acharya, Franklin and Zdonik [4] present an asymmetric system with multiple clients and a single server to control the broadcast. In this system, the server pushes all the data items according to some push-based scheduling, but simultaneously the clients are provided with a limited back channel capacity to make requests for the items, which are missing for a time interval greater than a given threshold. In this way, some of the items are both disseminated and also broadcasted, increasing the average waiting time. This happens, however, more frequently in low or medium loaded system since in heavily loaded systems clients' requests on-demand are likely to be dropped/ignored by the server.

In [13], a model for assigning the bandwidth to the push- and pull-scheduling in an adaptive way is proposed. To minimize the number of requests arriving at the server, as soon as an item becomes popular it

is inserted in the push scheduling. Hence, the cut-off point between the push and the pull items is highly dynamic. Clients listen to the broadcast first, and make a request only if the requested item is not in the push broadcast program. To cope with the dynamism of the push-set, just the flat push scheduling, which sends in round-robin fashion all the items, is adopted.
Finally, a hybrid approach is discussed in [17], which divides the data items in two disjoint sets: one for push and one for pull according to their degree of access probability. Roughly speaking, the set of the pull items contains those items which are so rarely requested in the system that no more than one or two requests for all of them are sent by the clients in a single unit of time (i.e., the time necessary to broadcast/disseminate a single data item). All the remaining items belong to the push-set. Repeatedly, the server broadcasts the push item precomputed by the packet fair queuing scheduling applied to the push-set, and after broadcasting, it serves, in first-come-first-serve order, the pending requests arrived for the pull items. The reason behind the good performance of such a hybrid scheduling is that the pull-set is selected in such a way that no more than one item can be disseminate between two broadcast items. However such a condition of the pull-set, when the system is highly loaded or all the items have more or less the same degree of access probability, leads to the selection of an empty pull-set, reducing the hybrid scheduling to a pure-push scheduling.

3. Preliminaries and motivation behind our work

Before discussing the motivation behind our work, which improves on [17], let us introduce some assumptions and terminologies.
First of all, we assume a system with a single server and multiple clients thereby imposing an asymmetry. The database at the server is assumed to be composed of D total number of distinct data items, each of unit length. The *access probability* P_i of item *i* is a measure of how worth is having an item access/request that is a measure of its degree of popularity. It is assumed that the server knows the access probability of each item in advance. The items are numbered from 1 to D in decreasing order of their access probability, thus $P_1 \geq P_2 \geq ... \geq P_D$. Clearly, from time to time, the server recomputed the access probability of the items, renumber them as necessary and eventually make available to all clients the new numbering of the items.
It is assumed that one unit of time is the time required to spread an item of unity length.
We say that the client accesses an item if that item is *pushed*, i.e. broadcasted by the server, while that an item is *requested* if the item is *pulled*, i.e. it is disseminated on air on demand. Moreover, let the *load N* of the system be the number of requests/access in the system for unit of time.
Let the *access time*, $T_{acc,I}$ be the amount of time that a client waits for a data item *i* to be broadcast after it begins to listen. Moreover, let the *response time*, $T_{res,I}$ be the amount of time between the client request of item *i* and the data transmission.
Clearly, the aim of the *push scheduling* is to keep the access time for each push item *i* as small as possible, while that of the *pull scheduling* is to minimize the response time for each pull item *i*.
Recalling that the pure push-based systems repeat the same schedule cyclically, let a single repetition of the schedule be termed a *broadcast cycle*. During a broadcast cycle, some items may appear several times. Each appearance is referred to as an *instance* of the item. Indicated with *s* the space between two consecutive instances of an item, if all instances of item *i* are equally spaced, then the space between any two instances of item *i* will be denoted as s_i.
In a push-based system, one of the overall measures of the scheduling performance is called *average expected access time*, $T_{exp\text{-}acc}$, which is defined as

$$T_{\exp-acc} = \sum_{i=1}^{D} P_i \bullet \overline{T_{acc,i}}$$

where $\overline{T_{acc,i}}$ is the average expected access time for item *i*. If instances are equally spaced in the broadcast cycle, then $\overline{T_{acc,i}} = s_i / 2$.
Many scheduling algorithms have been designed to minimize $T_{exp\text{-}acc}$. One of them, the *Packet Fair Scheduling* has been widely studied and its performance is well modeled analytically [11]. Such a push scheduling, which was used in [17], is also adopted in our hybrid scheduling as the push scheduling.

Therefore, from now on, in this paper, the term *push scheduling* indicates the *cyclic scheduling* derived by the *packet fair scheduling algorithm* applied to the push-set.
Similarly, it can be defined the *average expected response time,* denoted $T_{exp\text{-}res}$, for the pull scheduling.

In order to explain the rational behind our approach, let us first describe in details the intuition behind the hybrid scheduling in [17] and let us point out some of its drawbacks.
Recall that in purely push-based systems, the server alone decides which data items have to be transmitted without interacting with the clients, while in purely pull-based systems; the server is totally guided by the clients' requests. To make the average expected access time of the system smaller, the solution in [17] sends on-demand the less popular items immediately after having broadcasted the most popular items. Indeed, let the *push-set* consist of the data items numbered from 1 up to *K,* termed from now on the *cut-off point*, and let the remaining items from K+*1*up to *D* form the *pull-set*, the average expected waiting time for the hybrid scheduling is defined as:

$$T_{\exp-hyb} = T_{\exp-acc} + T_{\exp-res} = \sum_{i=1}^{K} P_i \bullet \overline{T_{acc,i}} + \sum_{i=K+1}^{D} P_i \bullet \overline{T_{res,i}}$$

Clearly, as the push-set becomes smaller, the average expected access time $T_{exp\text{-}acc}$ becomes shorter. However, the pull-set size becomes larger, leading to a longer expected response time $T_{exp\text{-}res}$. The size of the pull-set might also increase the average access time $\overline{T_{acc,i}}$, for every push item. In fact, if the hybrid scheduling serves, between any two items of the cyclic push scheduling, all the pending requests for pull items in First-Come-First-Served order, it holds for the average expected access time for item *i*:

$$\overline{T_{acc,i}} = (s_i + s_i \bullet q)/2,$$

where *q* is the average number of distinct pull items for which, arrives, at least one pending request in the pull-queue for unit of time. From now on, we refer to *q* as the *dilation* factor of the push scheduling.

To limit the growth of the $\overline{T_{acc,i}}$, and therefore that of the $T_{\exp-acc}$, the push-set is taken in [17] enough large that, in average, no more than 1 request for all together the pull items arrives from all the clients during a single unit time. To guarantee a dilation factor *q* equal to 1 when the system load is equal to *N*, [17] introduces the concept of the *build-up point B*. B is the minimum index between 1 and D for which it holds $N\left(1-\sum_{i=1}^{B} p_i\right) \le 1$, where *N* is the average access/requests for unit of time. In other words, [17] pushes all the items from 1 up to B to guarantee that no more than 1 item is waiting to be disseminate, and therefore to achieve a dilation factor *q* equal to 1.

After having bounded the dilation factor to 1, [17] chooses as the cut-off point between the push and pull items the value *K*, with *K* > *B,* such that *K* minimizes the *average* expected waiting time for the hybrid system.
Intuitively, the partition between push and pull items found out in [17] is meaningful only when the system load N is small and the access probabilities are much skewed. Under these conditions, indeed, the build-up point B is low. Hence, there may be a cut-off *K,* such that B < K < D, which improves on the average expected access time of the pure-push system. However, when either the system has a high load N and/or all items have almost the same degree of probability, the distinction between the high and low demand items becomes vague, artificial, hence the value of build-up point B increases, finally leading to the maximum number D of items in the system. Thus, in those cases, the solution proposed in [17] almost always behaves as a pure push-based system.

To corroborate what discussed so far, in Table 1, the relation of the value of the load N (taken in the columns); of the distribution of the access probabilities (θ in rows) with the value of the build up point B is illustrated, when the total number of distinct items D is 20.

According to the previous literature, we assume that the access probabilities P_i follow the Zipf's distribution with access *skew coefficient* θ :

$$P_i = \frac{(1/i)^\theta}{\sum_{j=1}^{n} (1/j)^\theta}$$

The access probabilities P_1 … P_D are well balanced for small values of θ , while they become skewed for increasing values of θ . Table 1B shows the relation of access probability of 10 items with θ .

	2	4	6	8	10	12	14	16	18	20
0.5	8	14	16	17	18	19	19	19	20	20
0.6	7	13	16	17	18	18	19	19	19	20
0.7	6	12	15	16	17	18	18	19	19	19
0.8	6	11	14	16	17	17	18	18	19	19
0.9	5	10	13	15	16	17	17	18	18	19
1	4	9	12	14	15	16	17	17	18	18
1.1	4	8	11	13	14	15	16	17	17	18
1.2	3	7	10	12	13	14	15	16	16	17
1.3	3	7	9	11	12	13	14	15	16	16

Table 1(a)

	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3
1	0.199	0.225	0.252	0.28	0.31	0.341	0.373	0.4052	0.437
2	0.141	0.148	0.155	0.161	0.166	0.171	0.174	0.1764	0.178
3	0.115	0.116	0.117	0.116	0.116	0.114	0.111	0.1084	0.105
4	0.1	0.098	0.095	0.093	0.089	0.085	0.081	0.0768	0.072
5	0.089	0.086	0.082	0.077	0.073	0.068	0.064	0.0587	0.054
6	0.081	0.077	0.072	0.067	0.062	0.057	0.052	0.0472	0.043
7	0.075	0.07	0.064	0.059	0.054	0.049	0.044	0.0392	0.035
8	0.07	0.065	0.059	0.053	0.048	0.043	0.038	0.0334	0.029
9	0.066	0.06	0.054	0.048	0.043	0.038	0.033	0.029	0.025
10	0.063	0.056	0.05	0.044	0.039	0.034	0.03	0.0256	0.022

Table 1(b)

Table 1: (a) Build-up point B for several values of N (taken in the columns) and θ (in rows) when D = 20; (b) Access Probability of 10 items (in rows) with varying θ (in Columns)

In the remaining of this paper, we present a hybrid scheduling that improves on [17] when the load is high or when the access probabilities are balanced, that is, when the scheduling in [17] reduces to the pure-push scheduling. The solution proposed in this paper again partitions the data items in the push-set and the pull-set, but it chooses the value of the cut-off point K between those two sets independent of the build-up point. Indeed, we let the pull-queue grow in size, and the push-set can contain any number of data items. After each single broadcast, we do not flush out the pull-queue, which may contain several different pending requests. In contrast, we just pull one single item: the item, which has the largest number of pending, requested in the pull-queue. Observe that simultaneously with every push and pull, N more access / requests arrive to the server, thus the pull-queue grows up drastically at the beginning. In particular, if the pull-set consists of the items from K+1 up to D, at most $N * \sum_{i=K+1}^{D} P_i$ requests can be inserted in the pull-queue at every instance of time, out of which, only one, the pull item that has accumulated the largest number of requests, is extracted from the queue to be pulled.

We are sure, however, that the number of distinct items in pull-queue cannot grow uncontrolled since the pull-queue can store at most as many distinct items as those in the pull-set, that is no more than $D-K$ items**.** So, after a while, the new arriving requests will only increase the number of clients waiting in the queue for some item, leaving unchanged the queue length. From this moment, we say that the system has reached a *steady state*. In other words, the pending requests will start to accumulate behind each pull-item without increasing anymore the queue length. Hence, just pulling the high demanded pull item, the system will not serve just one client but many. Our intuition is that a pull item cannot be stuck in the pull-queue for more than as many unit of time as the length of the queue. Indeed, in the worst case, when all the pull items have more or less the same access probability, the number of pending requests will be in average the same for all the pull items. Then, the system serves the pull-queue in a *round-robin* manner, and each pull item waits in average half of the length the pull-queue before being pulled. Besides, when the access probability

of the pull items vary a lot (i.e., θ larger than or equal to 1), the expected average response time can only decrease for the high demanded pull items, possibly improving the average expected response time.

In conclusion, the main contribution of our approach is to show that to preserve a constant dilation of the push scheduling is not necessary to avoid that the pull-queue starts to build-up. That is, it is not necessary to choose the cut-off point *K* larger than the build-up point *B.* In fact, our approach guarantees a dilation factor *q* equal to 1 just pulling a single item, and it shows that when K is chosen independent of B, a better tradeoff between the average expected access time and the average expected response time can be found.

4. The new hybrid algorithm

We are now in position to describe the behavior of our asymmetric system of communication.

As said, we assume that the database of the server contains the D items, indexed from 1 to D according to their decreasing access probabilities. That is, for the items 1 … D, it holds $P_1 \geq P_2 \geq ... \geq P_D$. Moreover, the clients to designate the data items they are interested in use the same indexes.

The server performs several actions simultaneously. From one side, it monitors the access probabilities of the data items and the system load. When those parameters diverge significantly from the assumptions previously made by the system, the server renumber the data items, and recalculates the cut-off point *K* to separate the push-set from the pull-set, as illustrated in Figure 1. Note that *K* is selected in such a way that the average expected waiting time of the hybrid scheduling $T_{exp\text{-}hyb}$ is minimized. In order to evaluate the cut-off point, recall that it holds:

$$T_{\exp-hyb} = T_{\exp-acc} + T_{\exp-res} = \sum_{i=1}^{K} P_i \bullet \overline{T_{acc,i}} + \sum_{i=K+1}^{D} P_i \bullet \overline{T_{res,i}}$$

Substituting for $\overline{T_{acc,i}}$ the optimal instance space $S_i = \frac{\sum_{j=1}^{K} \sqrt{\hat{P}_j}}{\sqrt{\hat{P}_i}}$ for item *i* in the pure-push packet fair scheduling obtained by normalizing the access probabilities $\hat{P}_i = \frac{P_i}{\sum_{j=1}^{K} P_j}$ of the push items (see [11, 18]) and for $\overline{T_{res,i}}$ the maximum length D –K of the pull-queue, *K* is selected in such a way that

$$T_{\exp-hyp}(K) = \sum_{i=1}^{K} S_i P_i + \sum_{i=K+1}^{D} P_i * (D-K)$$

is minimized.

Integer function CUT OFF POINT (D, P = {P_1, P_2 ...P_D}): K
/* D: Total No. Of items in the Database of the server
P: Sorted vector of access probability of items in decreasing order
K: Optimal Cut off Point */

K: = 1; T_{exp_hyb} (0): = T_{exp_hyb} (1): = D;

while K < = D **and** T_{exp_hyb} (K-1) > = $Texp_{\_hyp}$ (K) **do**
begin

Set $S_i = \frac{\sum_{j=1}^{K} \sqrt{\hat{P}_j}}{\sqrt{\hat{P}_i}}$, where $\hat{P}_i = \frac{P_i}{\sum_{j=1}^{K} P_j}$;

$$T_{\exp-hyp}(K) = \sum_{i=1}^{K} S_i P_i + \sum_{i=K+1}^{D} P_i * (D-K);$$

K: = K + 1;
end
return (K-1)

Figure 1: Algorithm to set the optimal cut-off point K between the push and pull items.

In addition, the server listens to all the requests of the clients and manages the pull-queue. The pull-queue, implemented by a max-heap, keeps in its root, at any instant, the item with the highest number of pending requests. For any request *i*, if *i* is larger than the current cut-off point *K*, i > *K*, *i* is inserted in the pull-queue, the number of the pending requests for i increased by one, and the heap information updates accordingly. Vice versa, if *i* is smaller than or equal to K, *i* <= *K,* the server simply drops the request because that item will be broadcast by the push-scheduling sooner or later.

Finally, the server is in charge of deciding at each instant of time which item must be spread. The scheduling is derived as explained in Figure 2, where the details for obtaining the push scheduling are omitted. The interested reader can found them, for example, in [11].

Procedure HYBRID SCHEDULING;
while true **do**
begin
compute an item from the push scheduling and broadcast it;
if the pull-queue is not empty **then**
extract the most requested item from the pull-queue,
clear the number of pending requests for that item, and pull-it
end;

Figure 2: Algorithm at the server that produces the hybrid scheduling

To retrieve a data item, a client performs the following actions:

Procedure CLIENT-REQUEST (i):
/* i **:** the item the client is interested in */

begin
send to the server the request for item i;
wait until listen for *i* on the channel
end

Figure 3: The algorithm that runs at the client site

Note that the behavior of client is independent of the fact that the requested item belongs to the push-set or to the pull-set.

5. Experimental Results

It remains now to evaluate the performance of the new proposed algorithm.
First of all, we compare the simulation results of the new algorithm with those of the hybrid scheduling in [17], with the results of the pure-push scheduling and with the analytic expression used to derive the optimal cut-off point. We run experiments for D=100, for the total number of access / requests in the system M = 25.000 and for N=10 or N=20. The results are reported in Table 2 and 3, respectively for N= 10 and N= 20.

	0.50	*0.60*	*0.70*	*0.80*	*0.90*	*1.00*	*1.10*	*1.20*	*1.30*
New	40.30	37.78	35.23	32.36	29.38	25.95	22.95	19.90	17.04
[17]	44.54	42.35	40.01	37.31	34.12	29.93	24.38	20.61	17.04
Push	45.03	43.01	40.50	37.47	34.30	30.90	27.75	24.50	20.86
Analytical	36.01	36.21	35.04	33.56	29.94	29.73	27.09	25.19	22.51

Table 2: Expected hybrid access time for different values of θ (taken in the columns) and different algorithms (taken in the rows) when N = 10.

	0.50	*0.60*	*0.70*	*0.80*	*0.90*	*1.00*	*1.10*	*1.20*	*1.30*
New	41.96	39.39	36.59	33.49	30.39	27.20	23.88	21.14	18.26
[17]	44.44	42.45	40.10	37.39	33.78	30.69	27.54	23.23	19.49
Push	44.70	42.61	40.30	37.65	34.12	30.78	27.71	23.94	21.07
Analytical	37.59	35.68	34.53	33.06	29.94	29.73	27.09	24.43	24.24

Table 3: Expected hybrid access time for different values of θ (taken in the columns) and different algorithms (taken in the rows) when N = 20.

For both Tables 2 and 3, the value of θ is varied from 0.50 to 1.30, so as to have the access probabilities of the items initially from similar to very skewed. Note that for θ no larger than 1, the analytic average expected access time is close to that measured with the experiments. This confirms that, when the access probabilities are similar, the pull items remain in the pull-queue for a time no larger than to total number of pull items that is D-K. For larger values of θ, the experimental measure of the expected response time is smaller that the analytic expected value because due to the fact that the access probabilities are very skew fewer than *D-K* items can be present simultaneously in the pull-queue. Therefore, the actual waiting time of the client is eventually shorter than D-K. Further experimental results have shown that when θ is varied from 0.90 to 1.30; the length of the pull-queue is approximated better by the value $D * \sum_{i=K+1}^{D} P_i$ than by D-*K*. Moreover, as earlier discussed, when the system is highly loaded, the scheduling algorithm in [17], whose cut-off point K must be larger than the build-up point B, almost reduces to the pure-push scheduling. Contradictory to [17], the new hybrid algorithm, even with very high loaded system, experiments better results than a pure-push based system as illustrated in Figure 4.

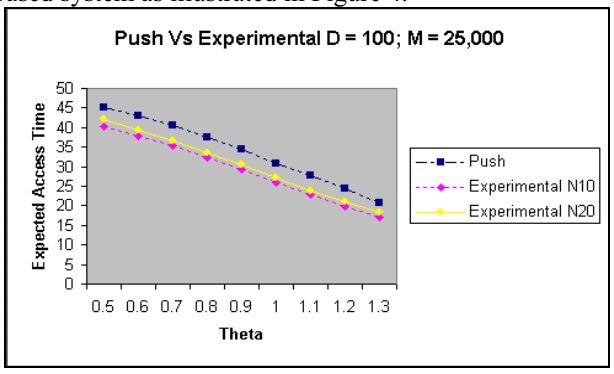

Figure 4: Pure-push scheduling Vs new algorithm

Besides, in Figure 5, the values of the cut-off point *K* for our solution, which takes K independent of B, and for the hybrid scheduling proposed in [17] are depicted for N=10 and N=20.

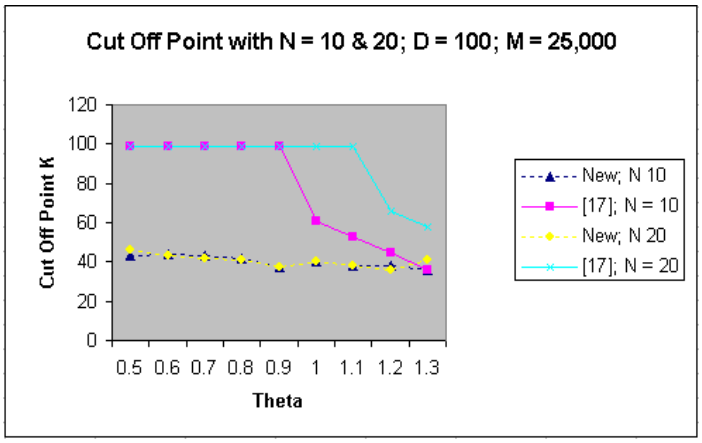

Figure 5: Cut-off point when N = 10; N = 20

6. Conclusions

We offer the following conclusions:

- As already pointed out in [17], by separating the most demanded items from the less demanded ones, broadcasting the former and disseminating the latter, the total overall expected waiting time of the system drops.
- If the cut off point between the push and the pull items is chosen in such a way that the no more than one pending request for pull item arrive in a unit of time, as in [17], for large number of requests / access arriving to the system, the system has a behavior similar to that of a total push system.
- To overcome such a drawback, we propose a new hybrid scheduling which continuously broadcasts one push item and disseminates one pull item. The clients send their requests to the server, which queues up only those for the pull items. At any instant of time, the item to be broadcast is designated applying the packet fair queue scheduling to the push items, while the item to be pulled is the one stored in the pull-queue which has accumulated, so far, the highest number of pending requests. The cut-off point between push and pull items is chosen in such a way that the analytic expected access time of the hybrid system is minimized, and is independent of the number of pending requests arriving in a unit of time. We found that by doing so we can drop the cut-off point to a value, which is much less than the total number of items present in the system, improving upon the expected access time of the hybrid scheduling in [17].
- As the experiments pointed out, our new hybrid approach is more effective than [17] when the access probabilities are similar than when they are skewed.

Finally, several further aspects remain to be investigated to make our system more suitable for real scenario. For example, in this paper, we assumed that the server knows the access probability of each item in advance, which may not be the case in the real scenario, as the demand for an item may change according to time. Thus our future work will might try to choose the cut off point in a dynamic way according to the changing access probability of the items present. We have also assumed that the server broadcast and disseminates the items on the same channel. A much better performance could be expected if we could use two channels one for broadcasting the data items and the other for disseminating the rest. Our future work could also investigate how the expected access time is related to the cut off point if the items in the database of the server are not of equal length, or if a deadline constraint is associated with each item

requested. Even more challenge is the perspective of supporting mobile clients, which pass from a base station to another (possibly broadcasting different flows of information) and which may experience poor transmission conditions.

References

[1] S. Acharya, M. Franklin and S. Zdonik, "Dissemination-based data delivery using broadcast disks", IEEE Personal Communications, pp. 50-60, December 1995.

[2] S. Acharya and S. Muthukrishnan, "Scheduling on-demand broadcasts: New metrics and algorithms", Proceedings of the Fourth Annual ACM/IEEE MobiCom, pp. 43-54, October 1998.

[3] S. Acharya, M. Franklin and S. Zdonik, "Prefetching from a broadcast disk", Proceedings of 12th International Conference on Data Engineering, pp. 276-285, February 1996.

[4] S. Acharya, M. Franklin and S. Zdonik, "Balancing push and pull for data broadcast". Proceedings of ACM SIGMOD Int. Conference on Management of Data, pp. 183-193, 1997.

[5] D. Barbara and T. Imielinski, "Sleepers and workaholics: Caching strategies in mobile environments", Proceedings of ACM SIGMOD Conference, pp. 1-12, May 1994.

[6] A. Bar-Noy, B. Patt-Shamir, I. Ziper, "Broadcast disks with polynomial cost functions", IEEE INFOCOM 2000, pp. 575 - 584.

[7] S. Baruah and A. Bestavros. "Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems", Proceedings of IEEE International Conference on Data Engineering, April 1997.

[8] S. Baruah and A. Bestavros. "Real-time mutable broadcast disks", Proceedings of Second International Workshop on Real-Time Databases, Burlington, VT, September 1997.

[9] J.C.R. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms", Proceedings of ACM SIGCOMM, pp. 43-56, 1996.

[10] A. Bestavros. "AIDA-based real-time fault-tolerant broadcast disks, Proceeding Second IEEE Real-Time Technology and Applications Symposium, June 1996.

[11] J. Gecsei, "The architecture of videotex systems", Englewood Cliffs, NJ: Prentice-Hall, 1983. 59

[12] S. Hameed, N.H. Vaidya, "Efficient algorithms for scheduling data broadcast", Wireless Networks 5, pp. 183-193, 1999.

[13] K. Stathatos, N. Roussopoulos, and J.S. Baras, "Adaptive data broadcast in hybrid networks", Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, pp. 326-335, 1997.

[14] N.H. Vaidya, S. Hameed, "Scheduling data broadcast in asymmetric communication environments", Wireless Networks 5, pp. 171-182, 1999.

[15] J.X. Yu, T. Sakata, and K. Tan, "Statistical estimation of access frequencies in data broadcasting environments", Wireless Networks 6, pp. 89-98, 2000.

[16] K. Stathatos, N. Roussopoulos and J. S. Baras. "Adaptive data broadcasting using air-cache", First International Workshop on Satellite-based Information Service, 1996.

[17] Y. Guo, S.K. Das & M.C. Pinotti, ``A new Hybrid Broadcast scheduling Algorithm for Asymmetric Communication Systems: Push and Pull Data based on Optimal Cut-Off Point", Mobile Computing and Communications Review (MC2R) 5, No. 4, 2001.

[18] R. Jain and J. Werth, "Airdisks and airraid: Modeling and scheduling periodic wireless data broadcast (extended abstract)", DIMACS Technical Report 95-11, Rutgers University, May 1995.

[19] J. Xu, Q. Hu, L. Lee, and W. C. Lee "SAIU: An efficient cache replacement policy for wireless on-demand broadcasts".