Available online at www.sciencedirect.com

SCIENCE DIRECT®

Performance Evaluation 60 (2005) 201–221

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Performance analysis of a novel hybrid push–pull algorithm with QoS adaptations in wireless networks

Azzedine Boukerche[a,*], Trivikram Dash[b], Cristina M. Pinotti[c]

[a] SITE, University of Ottawa, Ottawa, Ont., Canada K1N 6N5
[b] Department of Computer Science, Texas A&M, College Station, TX 77843, USA
[c] University of Perugia, I-06100 Perugia, Italy

Abstract

In this paper, we present a novel hybrid push–pull algorithm which combines broadcasting of push data items, with dissemination upon request of pull items in asymmetric communication environments. These environments are made up only of one database server and many clients. Requests made by the clients are queued up for the pull items. The (pull) item with the number of pending requests is the one selected to be pulled. We present a performance analysis of our scheme, and determine the individual response time for each item disseminated and the overall time for the pull queue to be flushed. Next, we extend our algorithm by incorporating quality of service (QoS) factors, and then, study its performance analytically.
© 2004 Published by Elsevier B.V.

Keywords: QoS; Push–pull algorithm; Probability

1. Introduction

In this world of information on demand at your fingertips whether it be traffic updates, stock quotes, horoscopes, sports results, or weather information where the clients far outnumber the servers, the traditional client–server approach by itself is no longer efficient. Popular data must be broadcasted while those infrequently needed can be pulled. Developing efficient schemes to do this is a challenge which

* Corresponding author.
E-mail addresses: boukerch@site.ottawa.ca (A. Boukerche), pinotti@unipg.it (C.M. Pinotti)

0166-5316/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.peva.2004.10.020

has to be met. It is especially important in wireless networks, where we have to deal with issues such as limited bandwidth, power consumption, and reliable connections.

An asymmetric communication environment is made up of one server and many clients. There are two ways for a client to receive data. The data could either be given without the client asking for it or it could be explicitly requested by the client. When the data is sent to all the clients at the same time regardless of whether any of the clients had requested that data item or not, this is known as broadcasting. When the data is given to a particular client only if it is requested then that data item is referred to as being pulled. Our communication environment can either be *push-based* or *pull-based*. A push-based environment is one in which the server sends out items to the client without them requesting it. A pull-based environment is one in which all items sent are those explicitly requested by the client. A hybrid system combines these two approaches.

There have been many algorithms in the literature which combine the two approaches. The goal is to minimize both the individual access and response times and overall access and response times or strike some kind of a balance between the two. Access time is how long a client has to wait for a data item to arrive after it starts listening to the broadcast schedule. Response time is how long a client has to wait for a data item to arrive after it explicitly requests it. A broadcast schedule is a specified sequence or order for data delivery of the most popular items, such as push data. A hybrid scheme is the best way to approach the problem. A pure broadcast system will broadcast not only frequently requested items, but also those ones that are rarely needed. This will unnecessarily increase the average access time of a data item. A pure pull system on the other hand is also not efficient as well.

The rest of this paper is organized as follows. Section 2 reviews previous and related work. Section 3 discusses our hybrid push–pull algorithm, followed by its description by a mean of pseudo-code. In Section 4, we present the performance analysis of our algorithm. Section 5 presents an enhanced hybrid push–pull algorithm. Section 6 will give its pseudo-code. Section 6 will analyze the performance of the enhanced hybrid push–pull algorithm. Finally, Section 7 will discuss our conclusions.

2. Related work

There are many schemes [1–9,11–14,16] which have been developed to partition the data items into either the push-based set or the pull-based set. However, none exist so far which have studied at all these four factors in combination: multiple data set sizes, quality of service (QoS), spurious cases leading to inaccurate portrayals of the system dynamics, and doing implementation without previous knowledge of data access probabilities. Although there have been papers which have examined some of these factors, though independently, to our knowledge, none so far has studied all of these four factors together. For example, the R$\times$W algorithm [4] examines at a broadcast system, but looks only at data items of uniform sizes. It has also only been applied to a pure broadcasting system. The stretch-optimal scheduling algorithm [16] does look at non-uniform data item sizes, however it is only intended for a pure broadcast system. While some authors [1–3,5,8,11,12,14] have assumed a priori knowledge of the data item access probabilities, others [6,7,9] have developed algorithms based on static analysis and the fact that there is only downward communication from the server to the clients.

Quality of service is another important component that have been investigated in [13], though, their scheme has only been applied to the pull-based case, and has considered only two levels of priority. To the best of our knowledge, spurious cases and anomalies have not been investigated fully at in the literature.

Suppose, we have one client requesting a particular data item 2000 times in the last 30 min and we have no other client requesting that particular data item, previous algorithms would place that particular data item in the broadcast cycle due to that one client making the probability of access for that data item high. The outlier creates unfairness. Our proposed algorithm detects the outliers and creates a fair situation when compared to previous schemes.

Our enhanced algorithm (see Section 5) will broadcast the most data items (i.e., pull data) during a cycle. After broadcasting a single item, the pull queue will be examined and the item which maximizes the product of the overall stretch and time of first request for that item will be selected for dissemination.

The concept of broadcast disks has been discussed extensively in [1,2,5–7,9]. The high bandwidth channel which the server uses to push data can be thought of as disks for the clients. Disks of different sizes and speeds are used for pushing items to clients. A type of memory hierarchy is created. Higher speed disks are used for data items that are more frequently accessed. Caching techniques in conjunction with different broadcast scheduling algorithms have been studied in connection with broadcast disks. Packet fair queueing is another technique [11,12] used in broadcast scheduling. A log-time algorithm [11] for scheduling has been developed. Broadcast scheduling techniques were first studied for teletext systems before being studied for wireless systems. All of the different techniques and algorithms mentioned in this paragraph have assumed previous knowledge of the data item probabilities.

2.1. *Stretch-optimal scheduling algorithm*

The stretch-optimal scheduling algorithm [16] based on the LTSF algorithm developed in [3] is used for broadcasting data items of variable size. The enhanced hybrid push–pull algorithm will however with some modifications use it in the pull set. The modification we make is that we will take into consideration the time that the earliest request for a particular data item was made and not just the number of requests that have been made for that item. Another modification we will make is that we will look the effective number of requests (to be explained later) and not the actual number of requests.

Stretch is a metric which can be used to create fairness in systems which have items of non-uniform data sizes. The stretch is defined to be the response time for an item divided by the time it takes to transmit that item. The transmission time of an item is its size divided by the system bandwidth. In the LTSF algorithm, we pick the data item for which the sum of all the stretches for unfulfilled requests for that item is the greatest. The stretch optimal algorithm is based on LTSF and is used instead of it because LTSF does not give us the overall stretch value which is optimal. The stretch optimal algorithm also requires less overhead and is simpler to implement. The stretch optimal algorithm picks the data item for which the value of the ratio of the number of unfulfilled requests for it and its size is the greatest. Suppose, we have three data items whose sizes are $L_1 = 2$, $L_2 = 5$, and $L_3 = 7$, and the number of requests made for them are in respective order, $R_1 = 5$, $R_2 = 4$, and $R_3 = 6$. We will then pick data item 1 since its R/L^2 value, 1.25 is the most. This will be picked instead of another item, in order to minimize the overall stretch.

3. Hybrid push–pull algorithm

In this section, we will describe the hybrid push–pull algorithm. The main objective of this algorithm is to pick a cut-off point K which divides the push data items from the pull data items in such a way as to minimize the sum of the average access time and average response time. After describing the algorithm we will show its pseudo-code. The pseudo-code consists of three main parts: a function to determine the cut-off point, a procedure for the actual scheduling of the data items, and finally a procedure for requesting of data items which runs at the client end.

3.1. Description of hybrid push–pull algorithm

Our system is made up of one database server holding D data items and c clients. Fig. 1 shows what our system looks like. K of the D items will be broadcasted and the remaining $(D - K)$ items will form part of the pull set. The foundation of our hybrid algorithm comes from [10,12,15]. We will generalize this algorithm even further to consider the four factors mentioned earlier. Before we do that, let us briefly explain what has been done so far in [10,12,15].

Access probabilities are assigned based on the popularity of the items. Each item is assumed to be of uniform data size. The access time T_{acc} is the amount of time a client has to spend in waiting for an item to be broadcasted after listening. The response time T_{res} is the amount of time a client spends waiting after it has requested an item from the server. The idea for the push-based scheduling is to minimize the access time and the idea of the pull-based scheduling is to minimize the response time.

In the hybrid system, we propose to minimize the combined time. Let $\overline{T}_{\text{acc},i}$ be the average access time for data item i and $\overline{T}_{\text{res},i}$ be the average response time for data item i. Then, the average expected hybrid wait time is defined to be the sum of the average expected access time and average expected response time. In other words, $T_{\text{exp-hyb}} = T_{\text{exp-acc}} + T_{\text{exp-res}} = \sum_{i=1}^{K} P_i \times \overline{T}_{\text{acc},i} + \sum_{i=K+1}^{D} P_i \times \overline{T}_{\text{res},i}$. The P_i values refer to the access probabilities of the data items. The K most popular items are chosen to be broadcasted and the remaining $(D - K)$ items have to be explicitly requested by the client. The idea is to choose the value of K which will enable us to do things in an efficient manner.

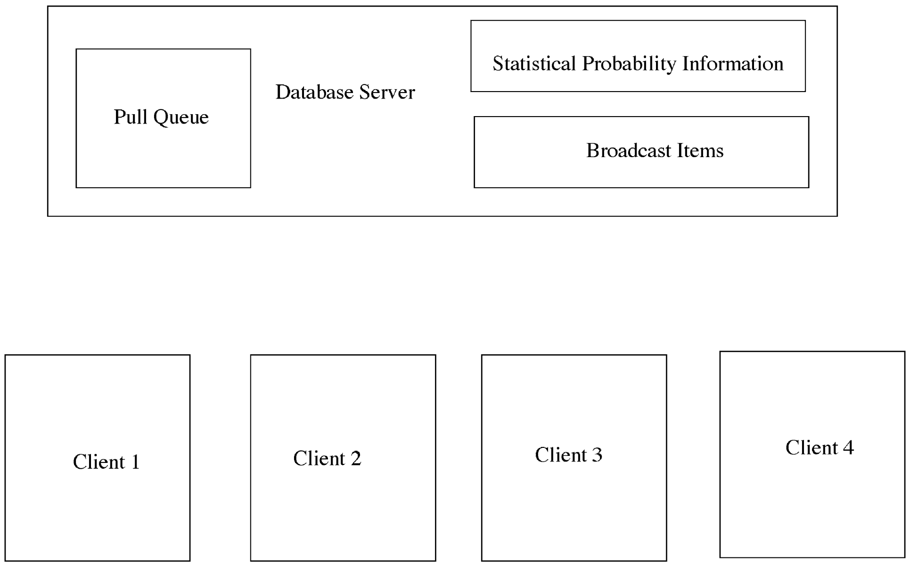

Fig. 1. Asymmetric client–server architecture.

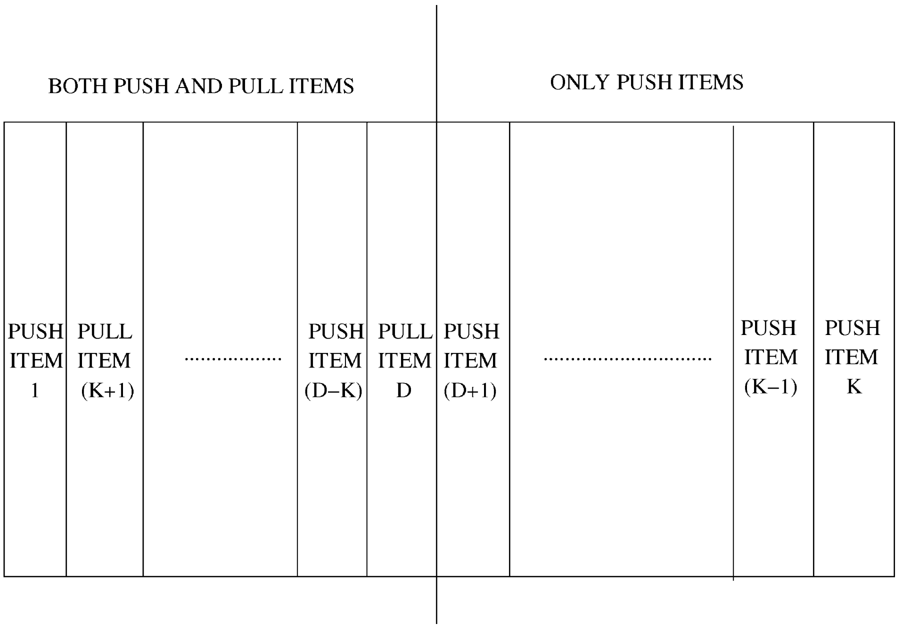

Fig. 2. Frame structure of a broadcast cycle (all pull set items requested).

Hence, $T_{\exp-\text{hyb}} = \sum_{i=1}^{K} S_i P_i + \sum_{i=K+1}^{D} P_i \times (D-K)$, where $S_i = \sum_{j=1}^{K} \sqrt{\widehat{P}_j}/\sqrt{\widehat{P}_i}$. S_i is defined to be the optimal instance space and $\widehat{P}_i$ is defined to be the normalized probability where $\widehat{P}_i = P_i/\sum_{j=1}^{K} P_j$.

In the hybrid algorithm, we try setting K to all values from 1 to D. When $T_{\exp-\text{hyb}}(K) \geq T_{\exp-\text{hyb}}(K+1)$ we stop and we have obtained the K value which will minimize the access/request time for the hybrid algorithm. The $T_{\exp-\text{hyb}}$ values are initialized to D for $K = 0$ and 1.

The pull scheduling is done in such a way that on average we get no more than one request for an item numbered from $(K+1)$ to D during an unit time interval. If we do have more than one item in the pull queue, then we choose the item which has been requested the most. The pull queue is implemented by a max heap (Fig. 2).

Given below is the general outline of the pseudo-code of the hybrid push–pull algorithm.

```
**Integer Function Cut-Off Point** ($D$, $P_1$, $P_2$, ..., $P_D$):$K$
/* The algorithm to determine the cut-off point K between the push and pull items, */
/* is given below and taken directly from [15]. */
$D$ = Number of Database Items
$K$ = Optimal Cut-Off Point
$K := 1$; $T_{\exp-\text{hyb}}(0) := T_{\exp-\text{hyb}}(1) := D$;
while $K \leq D$ and $T_{\exp-\text{hyb}}(K-1) \geq T_{\exp-\text{hyb}}(K)$ do
  begin
    Set $S_i = \frac{\sum_{j=1}^{K}\sqrt{\widehat{P}_j}}{\sqrt{\widehat{P}_i}}$, where $\widehat{P}_i = \frac{P_i}{\sum_{j=1}^{K} P_j}$
```

```
        $T_{\exp-\mathrm{hyb}}(K) = \sum_{i=1}^{K} S_i P_i + \sum_{i=K+1}^{D} P_i * (D - K)$; $K := K + 1$;
    end
return $(K - 1)$
```

Procedure Hybrid Scheduling

```
/* The algorithm at the server which gives us the hybrid scheduling is given below. */
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
```

Procedure Client Request (*i*):

```
/* Finally the algorithm which runs at the client's side is shown below. */
    begin
        send sever request for item $i$
        wait until listen for $i$ on the channel
    end
```

4. Performance analysis of hybrid push–pull algorithm

Earlier performance analysis of the push–pull scheduling algorithm [15] has only examined the expected overall response time. Though, the study did not investigate the expected response times for individual data items in the pull set. This analysis could be useful in systems where meeting deadlines is crucial and/or for better establishing QoS criteria. The expected response time for an individual data item is the time we expect to wait on average for that item to be disseminated. In this section, we will derive a theorem which gives us the expected response time for item *b* to be disseminated. Before we proceed further, let us introduce the following two lemmas.

Lemma 1. *The probability of i requests for data item s during an unit time interval, $Q_{i,s}$, where c is the number of clients in our system, and P_s is the probability of requesting item s during an unit time interval is computed as follows*: $Q_{i,s} = \binom{c}{i}(P_s)^i(1 - P_s)^{c-i}$.

Proof. The above is simply an application of Bernoulli trials where P_s is the probability that a client will make a request for data item *s* during an unit time interval and $(1 - P_s)$ is the probability that the client will not make a request for data item *s* during that time interval. □

Lemma 2. *The probability of pull item b being the ath item to be flushed out of the pull queue during a broadcast cycle, $F_{a,b}$, is determined as follows, where c is the number of clients in our system, is calculated as given below*:

$$F_{a,b} = \sum_{m=1}^{(c-a+1)} Q_{m,b} \sum_{n \in T_{i,a,b,m}} R(n, b)$$

where $Q_{m,b}$ is the probability of m requests for data item b and can be calculated as shown in Lemma 1, P_b is the probability of pull item b being requested, $R(i,b) = Q_{i_1,K+1}Q_{i_2,K+2}\cdots Qi_{D-K,D}/Q_{i_{b-K,b}}$, $i = (i_1, i_2, \ldots, i_{D-K})$ $T_{i,a,b,m} = \{(i_1, i_2, \ldots, i_{D-K}) - (i_{b-K}) |\ \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}] < m$, $\mathrm{Index}(\max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}]) \in Y_{a,b,m} \cup Z_{a,b,m}$, $\max^v[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}] > m$, $1 \le v \le (a-1)$, $0 \le i_j \le c$, $1 \le j \le (D-K)\}$, *where* $Y_{a,b,m} = \{u | i_u = \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}], u < (b-K), i_u < m\}$, $Z_{a,b,m} = \{u | i_u = \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}], u > (b-K), i_u \le m\}$, $\mathrm{Index}(i_n) = n$ *and where our "–" operator for tuples removes one component from our tuple,* $\max^a I$ *is the ath largest element in I, D is the number of items in our database server, and K is the number of items in the push set.*

Proof. For those items in the pull set, we can determine probability distributions for which order the items are released in. We will determine a general formula for $F_{a,b}$. Let us develop our analysis by first looking at $F_{1,k+1}$. Before we proceed further, let us define some notations which will make the equations simpler to write.

Let $I = \{(i_1, i_2, \ldots, i_{D-K-1}) | 0 \le i_j \le c, 1 \le j \le (D-K-1)\}$. The set I represents all the possible combinations of the number of requests that can be made for each item in the pull set. If $i \in I$, we define $\max^a i$ to be the ath largest component of i. Do note that the way we refer to the ath largest value maybe slightly different than the way some people use the term. For example, if we have a tuple (3, 3, 2), the first largest value will be 3 and the second largest value will also be 3 (instead of 2).

So we can say that the probability of item $(K+1)$ being the first one to be flushed out of the pull queue is as follows:

$$\begin{aligned} F_{1,K+1} = & \left(Q_{1,K+1} \sum_{i \in I, (\max^1 i) \le 1} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D} \right) \\ & + \left(Q_{2,K+1} \sum_{i \in I, (\max^1 i) \le 2} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D} \right) \\ & + \left(Q_{3,K+1} \sum_{i \in I, (\max^1 i) \le 3} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D} \right) \\ & + \cdots + \left(Q_{c,K+1} \sum_{i \in I, (\max^1 i) \le c} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D} \right) \end{aligned}$$

The term on the left of the summation symbols represents the probability of having x requests for item $(K+1)$, where x varies from 1 to c, the number of clients. The term on the right of the summation symbols represents the product of the probabilities of having the requests for the other items to be less than or equal to x; this is necessary in order for us to have item $(K+1)$ be the first one to be flushed out.

The following series of computations can determine the probability for item $(K+1)$ to be the second item to be flushed out of the pull queue.

$$\begin{aligned}F_{2,K+1} = {} & \left(Q_{1,K+1} \sum_{i\in I,(\max^2 i)\le 1,(\max^1 i)>1} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D}\right)\\ & + \left(Q_{2,K+1} \sum_{i\in I,(\max^2 i)\le 2,(\max^1 i)>2} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D}\right)\\ & + \left(Q_{3,K+1} \sum_{i\in I,(\max^2 i)\le 3,(\max^1 i)>3} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D}\right)\\ & + \cdots + \left(Q_{c-1,K+1} \sum_{i\in I,(\max^2 i)\le (c-1),(\max^1 i)>(c-1)} Q_{i_1,K+2} Q_{i_2,K+3} \cdots Q_{i_{D-K-1},D}\right)\end{aligned}$$

The reasoning involved in determining the above equation is the same as before. In addition to having the same conditions as before we also make sure that there has to be one and only one item for which the number of requests must exceed x. Another difference is that x varies from 1 to $(c-1)$, where again, c is the number of clients in our system. x cannot be equal to c because that would mean the item referred to by the term which is left of the summation symbol would be flushed out first.

If we introduce a few more terms as given below

$$R(i,b) = \frac{Q_{i_1,K+1} Q_{i_2,K+2} \cdots Q i_{D-K,D}}{Q_{i_{b-K},b}}$$

$$\begin{aligned}T_{i,a,b,m} = {} & \{(i_1, i_2, \ldots, i_{D-K}) - (i_{b-K}) | \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}] < m,\\ & \mathrm{Index}(\max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}]) \in Y_{a,b,m} \cup Z_{a,b,m},\\ & \max^v[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}] > m, 1 \le v \le (a-1), 0 \le i_j \le c, 1 \le j \le (D-K)\}\end{aligned}$$

where $Y_{a,b,m}$, $Z_{a,b,m}$, and extrmIndex are defined as shown below

$$Y_{a,b,m} = \{u | i_u = \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}], u < (b-K), i_u < m\}$$

$$Z_{a,b,m} = \{u | i_u = \max^a[(i_1, i_2, \ldots, i_{D-K}) - i_{b-K}], u > (b-K), i_u \le m\}$$

$$\mathrm{Index}(i_n) = n$$

and where our "−" operator for tuples removes one component from our tuple and also $i = (i_1, i_2, \ldots, i_{D-K})$, we can then using similar reasoning as above, define the probability of item b being the ath one to be flushed out as follows:

$$F_{a,b} = \sum_{m=1}^{(c-a+1)} Q_{m,b} \sum_{n\in T_{i,a,b,m}} R(n,b)$$

We will now determine a formula for the expected response time for item *b* to be disseminated. □

Theorem 1. *The expected response time for item b to be disseminated where $F_{a,b}$ is the probability of pull item b being the ath item to be flushed out of the pull queue, P_b is the probability of item b being requested during an unit time interval, D is the number of data items in our system, and K is the number of items in the push set, is given as follows*:

$$\sum_{g=0}^{D-K}\left[(1-P_b)^g P_b\left(\left(2D-K-g+\frac{1}{2}\right)+\sum_{a=0}^{D-K} aF_{a,b}\right)\right] + \sum_{h=D-K+1}^{D}\left[(1-P_b)^{D-K+h+1} P_b\left((K-h)+\sum_{a=0}^{D-K} aF_{a,b}\right)\right]$$

Proof. The length of the broadcast cycle is at most $[2(D-K)+K]$, where K is the number of items in the push set and D is the total number of data items. The first term is the number of time units taken for $(D-K)$ items to be broadcasted and $(D-K)$ items to be pulled. The second term is the number of time units it will take for us to broadcast the remaining K items. So the length of the broadcast cycle is at most $(2D-K)$.

If a request for item *b* to be disseminated comes just prior to the start of a new broadcast cycle, then the expected time for the item to be disseminated is given below: $\sum_{a=0}^{D-K} aF_{a,b}$.

If the broadcast cycle has just commenced and P_b is the probability of item *b* being requested by the client, then we can say that the expected response time for item *b* to be disseminated is as follows:

$$P_b\left(\left(2D-K-\frac{1}{2}\right)+\sum_{a=0}^{D-K} aF_{a,b}\right)$$

If the request for item *b* comes in while the first item is being pulled, then the expected response time for item *b* to be disseminated is as follows:

$$(1-P_b)P_b\left(\left(2D-K-\frac{3}{2}\right)+\sum_{a=0}^{D-K} aF_{a,b}\right)$$

If the request for item *b* comes in while the *g*th item is being pulled then the expected response time for item *b* to be disseminated is as follows:

$$(1-P_b)^{2g+1}P_b\left(\left(2D-K-\frac{(2g+1)}{2}\right)+\sum_{a=0}^{D-K} aF_{a,b}\right)$$

If the request for item *b* comes in when item *h* is broadcasted, where $h > D-K$, the expected response time for item *b* to be disseminated is as follows:

$$(1-P_b)^{2(D-K)+1}(1-P_b)^{h-D+K}P_b\left((2D-K)-(2(D-K)+h)+\sum_{a=0}^{D-K}aF_{a,b}\right)$$

We can simplify this to: $(1-P_b)^{D-K+h+1}P_b((K-h)+\sum_{a=0}^{D-K}aF_{a,b})$.

Therefore, using Lemma 2, the expected response time for item b to be disseminated is given as follows,

$$F_{a,b}=\sum_{g=0}^{D-K}\left[(1-P_b)^g P_b\left(\left(2D-K-g+\frac{1}{2}\right)+\sum_{a=0}^{D-K}aF_{a,b}\right)\right]$$
$$+\sum_{h=D-K+1}^{D}\left[(1-P_b)^{D-K+h+1}P_b\left((K-h)+\sum_{a=0}^{D-K}aF_{a,b}\right)\right]\quad\Box$$

In Fig. 3, we see the response time for a pull item b as a function of the number of items in the push set (K) and the probability of an item b being requested (P_b). As the probability of a pull item being requested increases, the response time for the item increases slightly. This is because the more probable a request for a data item is, the more likely it will be requested towards the beginning of a broadcast cycle, thereby increasing the wait for the next broadcast cycle to start. The start of the next broadcast cycle is when requests made in the previous broadcast cycle are honored. The response time for an item b decreases as we increase the value of the cut-off point K because there are less items in the pull set queue.

We can also study the expected overall time for the pull queue to be flushed out. The analysis done previously [15] gives us only an upper bound. We can refine it further. Computing more accurately the expected overall time for the pull queue to be flushed out can help us to better how to meet QoS requirements.

Theorem 2. *The expected overall time for the pull queue to be flushed out at the start of the broadcast cycle is computed as follows:*

$$\sum_{i=1}^{D-K}2i\sum_{a_1\in H,a_2\in H-a_1,\ldots,a_i\in H-a_1-\cdots-a_{i-1}}P_{a_1}P_{a_2}\cdots P_{a_i}\prod_{m\in H-a_1-\cdots-a_i}(1-P_m)$$

where $H=\{K+1,K+2,\ldots,D\}$, and P_b is the probability of pull item b being requested during an unit time interval.

Proof. To compute the expected overall time for the pull queue more accurately, we need to be able to compute the probability of having g items in the pull queue.

The probability of having 0 items in the queue can be computed as follows: $(1-P_{K+1})(1-P_{K+2})\cdots(1-P_D)$.

Thus, the probability of having only one item in the queue can be computed as follows: $\sum_{h\in H}P_h\prod_{m\in H-h}(1-P_m)$.

The probability of two items in the queue can then be computed as $\sum_{h\in H,i\in H-h}P_hP_i\prod_{m\in H-h-i}(1-P_m)$.

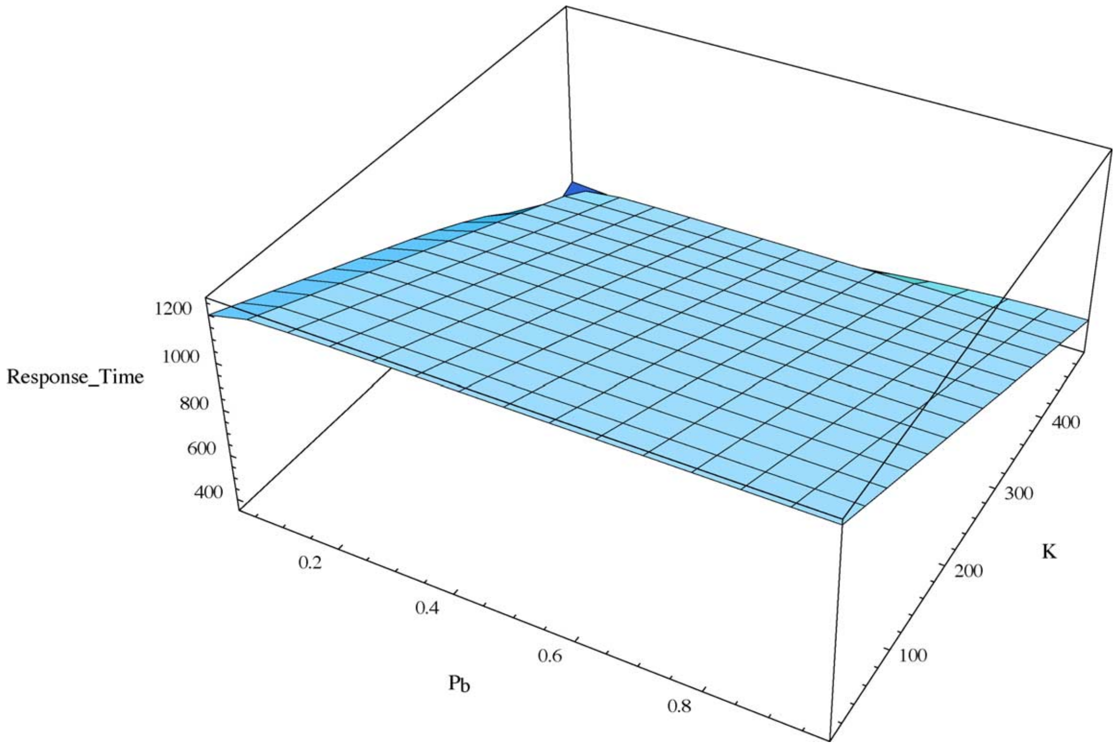

Fig. 3. Response time for pull item b as a function of the number of items in the push set (K) and the probability of item b being requested during an unit time interval (P_b).

So the probability of having g items in the queue is determined as follows

$$\sum_{a_1\in H, a_2\in H-a_1, a_g\in H-a_1-\cdots-a_{g-1}} P_{a_1}P_{a_2}\cdots P_{a_g} \prod_{m\in H-a_1-\cdots-a_g}(1-P_m)$$

Thus, the expected overall time for the pull queue to be flushed out at the start of the broadcast cycle is computed as follows:

$$\sum_{i=1}^{D-K} 2i \sum_{a_1\in H, a_2\in H-a_1,\ldots,a_i\in H-a_1-\cdots-a_{i-1}} P_{a_1}P_{a_2}\cdots P_{a_i} \prod_{m\in H-a_1-\cdots-a_i}(1-P_m) \quad \square$$

5. Enhanced hybrid push–pull algorithm

In this section, we propose an enhanced hybrid push–pull algorithm which can be used in systems with non-uniform data sizes and in which clients can have multiple priority levels. Furthermore, the enhanced hybrid push–pull algorithm does not require any a priori knowledge of data access probabilities. The enhanced hybrid push–pull algorithm is also able to handle spurious cases as well. For example, if only one normal or low priority client always requests a particular data item, then that item will be placed in the pull set instead of the push set as would likely be done instead with other algorithms. The explanation of the algorithm's pseudo-code (given in Section 5) is explained below. The reader will find it helpful to investigate while reading the description of the algorithm below.

In procedure client request, whenever a client wants an item, it sends a message to the server identifying itself and which item number it wants. The server identifies whether the item requested is part of the broadcast cycle or something which needs to be pulled. This is determined by looking at the item number assigned to the item. If we have D items in our database server, they are numbered from 1 to D. We assign the variable K to be the number given to the highest numbered item which is broadcasted. In other words items 1 to K are broadcasted and items $(K + 1)$ to D are pulled. The K value is recomputed at the end of each broadcast cycle. Clients are also numbered from 1 to c, where c is the number of clients in our system. The numbering of clients is purely arbitrary.

If client j requests an item numbered greater than K, then the request made by client j is added to the pull request queue. We keep a running total of the number of requests made for an item by a client. We also keep track of when the earliest request for an item has been made during a broadcast cycle. In our system, requests made during a broadcast cycle which has already commenced, do not get honored until the next one starts. We alternate between broadcasting an item and pulling an item if there is something that is waiting to be pulled. Another thing that we keep track of is which is the most important or highest priority client that has made a request for item i during a broadcast cycle.

When the system has reached steady state, we check to see whether the total number of requests for item i by client j is an outlier or too much greater than the average number of requests made by all the clients for that item i. If the number of requests made for item i by client j is too much greater than above the average we disregard it in our calculations for the effective number of requests (shown in the enhanced push–pull algorithm with the subscript "eff"). If things do change later on, and it turns out that what was considered to be too much greater than the average is no longer so, then we include it in our calculations. The $B_{i,j}$ boolean values indicate to us whether or not to include it in our calculations.

What constitutes as being "too much greater than the average" depends on the priority value assigned to the client. The higher the priority a client is, the more the number of requests it can make for a particular item before it is considered too far above the average. We define a strictly monotonically increasing function, α, which is a function of the client's priority value. This α value tells us how many multiples of the standard deviation for the average number of requests made for item i we can stray before being considered as going too far above the average. The rationale for this is that for example, if a data item is requested 300 times by a certain client and the other clients seldom request that same item and also if that certain client is important enough, we will include the 300 requests as part of the probability calculations and possibly include the corresponding item in the push set as well. However, if it turns out that certain client is not important enough, then we will not consider the 300 requests made as part of our probability calculations and very likely instead include that item in the pull set.

At the end of every broadcast cycle, the number of requests made for an item by a client is reinitialized to its current value minus the value it had T time units ago. The pre-defined time interval, T, that we choose should not be too small nor too large. If it is too large, then we are not able to properly capture the current state of what is going on. On the other hand, if T is too small, then we will not be able to weed out spurious cases properly.

The scheduler broadcasts each item in descending order of effective access probabilities. We say effective access probabilities instead of just access probabilities because a request made by a client is not always counted as one request. It could be counted as two or one-half requests for example, depending upon if the client is a higher than normal priority client or if it is a lower than normal priority client. The client priorities assigned, w, are positive values. A normal priority client is assigned a w value of 1. Important or high priority clients are assigned w values greater than 1. Low priority clients are assigned values greater than 0 but less than 1. In Fig. 4, we see that six clients with different priority levels, indicated by their w values, requesting either data item 1, 2, or 3. From the figure, we can calculate $R_{\mathrm{eff}_1} = 0.5 + 1 + 1 = 2.5$, $R_{\mathrm{eff}_2} = 1 + 2 = 3$, and $R_{\mathrm{eff}_3} = 0.5$.

After we broadcast an item, we check to see if there are any items in the pull queue. We pull the item which has the largest modified optimal stretch value. The optimal stretch value is the number of effective requests made for an item divided by the square of its length. The modified optimal stretch value is the product of the optimal stretch value and the product of the time that the earliest request was made for that item. In the past, the stretch optimal algorithm has been applied only to the push set, but now we are applying it to the pull set and we are also looking at the earliest request made for that item (not just the number of effective requests which have been made). If there is a tie, then we propose to break the tie by pulling the item requested by the highest priority client. If there is still a tie, then we break the tie by pulling the item which was requested the earliest. If again we still have a tie, then we pull the item which has been numbered with the smaller value. There is no possibility of further ties as each item is distinctly numbered.

After a broadcast cycle has completed, we recalculate the item access probabilities, the normalized item access probabilities, and also the optimal instance space between the broadcasted data items. We then determine the cut-off point between the items to be broadcasted and the items which have to be pulled. The constraint imposed is that we have to make sure that the expected aggregate length of all the pull items which need to be disseminated during our broadcast cycle is less than or equal to the average length of a broadcasted item.

In Table 1, we show the notation used throughout the enhanced hybrid push–pull algorithm. Given below is the general outline of the pseudo-code of the enhanced hybrid push–pull algorithm.

```
**Procedure Initialize-Values** [
  $K = \frac{D}{2}$
   $R_{eff_{TOTAL}} = 0$
   for $i = 1$ to $D$ [
     $Time[i]$ = "Negative-Infinity"
     $HighestPriority[i]$ = "Negative-Infinity"
     $P_i = \frac{1}{D}$
     $R_{eff_i} = 0$
     for $j = 1$ to $c$
       $R_{i,j} = 0$
  ]
]

**Procedure Client-Request** [
  Request $(i, j)$;
]
```

```
Procedure Request (i, j) [
    if ($i > K$)
        then add-pull-queue (j)
    $R_{i,j} = R_{i,j} + 1$
    if ($Time[i]$ == “Negative Infinity”)
        then $Time[i] = CurrentTime$
    if ($w[j] > HighestPriority[i]$)
        then $HighestPriority[i] = j$
    if (not (steady-state))
        then $B_{i,j} = 1$
        else if (Out-of-Range ($i, j, R_{i,j}$))
                then $B_{i,j} = 0$
                else [$B_{i,j} = 1$
                    $R_{eff_i} = \sum_{j=1}^{c} R_{i,j} w_j B_{i,j}$
                    $R_{eff_{total}} = \sum_{i=1}^{D} R_{eff_i}$
                    ]
]
Procedure Scheduler [
    for $item = 1$ to $num.items.in.broadcast.cycle$
        Broadcast (next broadcast item)
        if (not (empty(pull-queue)))
            then [
                pull (item with largest $\frac{R_{eff_i} Time[i]}{L_i{}^2}$)
                if TIE
                    then [
                        pull (item with largest $HighestPriority[i]$ value)
                        if TIE
                            then [
                                pull (item with smallest $Time[i]$)
                                if TIE
                                    then pull (item with smallest index)
                            ]
                    ]
        ]
    for $i = 1$ to $D$ [
        $P_i = \frac{R_{eff_i}}{R_{eff_{total}}}$
        $\widehat{P}_i = \frac{L_i P_i}{\sum_{j=1}^{K} L_j P_j}$
        $S_i = \frac{\sum_{j=1}^{K} \sqrt{\widehat{P}_j}}{\sqrt{\widehat{P}_i}}$
    ]
    find $K$ which minimizes $\sum_{i=1}^{K} S_i L_i P_i + \sum_{i=K+1}^{D} L_i P_i$ subject to $\sum_{i=K+1}^{D} L_i P_i \leq \frac{1}{K} \sum_{i=1}^{K} L_i$
    Renumber $i$ values in descending order of $P_i$
```

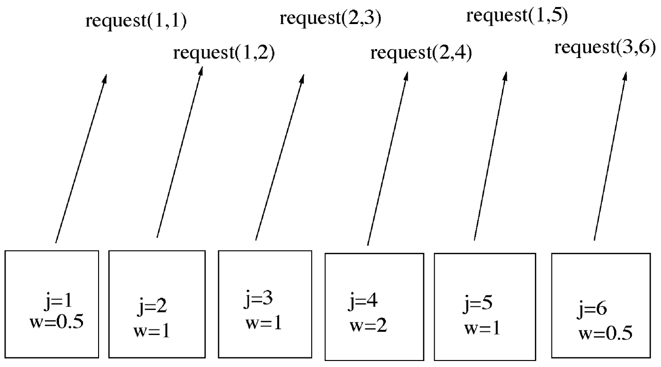

Fig. 4. Multiple priority level clients requesting data items.

```
      if TIE
         then [
            assign ($i$ values in descending order of $HighestPriority$)
            if TIE
               then [
               assign ($i$ values in ascending order of $Time$)
               if TIE
                  then [ randomly decide
               ]
         ]
      ]
   /* Reset at end of every broadcast cycle */
   for $i = 1$ to $D$ [
      $R_{i,j} = R_{i,j}(CurrentTime) - R_{i,j}(CurrentTime - T)$
      $Time[i] = $ "$Negative\ Infinity$"
      $HighestPriority[i] = $ "$Negative\ Infinity$"
   ]
]
```

```
Procedure Out-of-Range ($number, i, j$) [
   if $(number + w_j) \geq (\overline{R}_i + \alpha(w_j)\sigma_{R_i})$
      then return TRUE
      else return FALSE
]
```

6. Performance analysis of the enhanced hybrid push–pull algorithm

Let us now analyze the performance of the enhanced hybrid push–pull algorithm. Recall that the enhanced algorithm we also consider differing priority values for clients and non-uniform data sizes.

Table 1
Nomenclature

Symbol	Meaning
N	Number of clients in system
D	Number of data items in server database
K	Cut-off point between push and pull data
T	A pre-defined time interval or quantum we select
P_i	Probability of client requesting item *i* in one time unit
$\underline{R}_{i,j}$	Number of requests for item *i* by client *j*
$\bar{R}_i$	The average number of requests made for item *i* by a client in time period (*CurrentTime*, *T*)
R_{eff_i}	The effective number of requests for data item *i*
$R_{eff_{total}}$	The effective number of total requests for the data items
W_j	Priority or weight given to client *j*
L_i	The length of data item *i*
i	Refers to data item
j	Refers to client number
c	Refers to total number of clients in the system
α	A function of client priority
$B_{i,j}$	A boolean variable which tells us whether or not to include $R_{i,j}$ in the calculations for R_{eff_i}
$Time[i]$	Contains the time that earliest request was made for item *i*
$HighestPriority[i]$	Contains the number of the highest priority client which requested item *i*

This certainly makes the mathematical analysis far more complex. However, we will make things much simpler by using the mathematical analysis done in Section 4 as a foundation from which to build upon. Finding out the response times for individual data items may help us how to meet deadlines and also achieve QoS requirements. Once again, the expected response time for an individual data item is the time we expect to wait on average for that item to be disseminated.

Having a system where clients have different priority levels instead of having a system where each client is of the same importance will make achieving quality of service requirements far easier. Clients who want higher priority levels can subscribe to premium services. Those clients who need to meet some deadlines can be assigned higher priority levels, so that each request for a pull item is counted as more than just one request. By the same token, for those clients where time is not so crucial, can be assigned lower priority levels. In most practical systems, the sizes of data items will vary considerably. Our mathematical analysis takes this into consideration.

Let us define the effective number of clients, as $c_{\text{eff}} = \sum_{i=1}^{c} w_i$. For example, if there are three clients, with priorities 1 (normal), 2 (high), and 0.5 (low), then the effective number of clients is 3.5. In what follow, we will introduce a theorem which allows us to compute the expected response time for dissemination of item *b*. Before we do that, we will introduce two lemmas.

Lemma 3. *The probability of i effective requests for item j, $U_{i,j}$, where item j belongs to the pull set and c is the number of clients in our system is calculated as such*: $U_{a,b} = \sum_{i=1}^{c} W_{i,a} Q_{i,b}$, *where $Q_{a,b}$ is the probability of i requests for item b and is calculated as shown in Lemma 1, $W_{i,a}$ is the probability that the sum of i distinct elements taken from W is equal to a, where* $W = \{w_j | w_j \geq 0, 1 \leq j \leq c\}$.

Proof. We can start by saying that the probability of receiving one effective request for item number $(K+1)$, where c is the total number of clients in the system can be determined as follows:

$$U_{1,K+1} = W_{1,1}Q_{1,K+1} + W_{2,1}Q_{2,K+1} + \cdots + W_{c,1}Q_{c,K+1}$$

where $Q_{a,b}$ is calculated as shown in Lemma 1 and is the probability of a number of requests made for item b, $W = \{w_j | w_j \geq 0, 1 \leq j \leq c\}$, $W_{i,a}$ is the probability that the sum of i distinct elements taken from W is equal to a and w_j is the priority level of client j.

We can, therefore, easily generalize the probability of a effective requests from item b as follows: $U_{a,b} = \sum_{i=1}^{c} W_{i,a}Q_{i,b}$. □

Lemma 4. *The probability that item b will be the ath one to be flushed out, $G_{a,b}$, in a system with different priority levels for clients is computed as follows:*

$$G_{a,b} = \sum_{m=\max^{c} A, m\in A}^{c_{eff}} U_{m,b} \sum_{n\in X_{i,a,b,m}} Y(n,b)$$

where $U_{m,b}$ is the probability of m effective requests for item b, and is calculated as shown in Lemma 3, $Y(i,b) = U_{i_1,K+1}U_{i_2,K+2}\cdots Ui_{D-K,D}/U_{i_{b-K},b}$, $X_{i,a,b,m} = \{(i_1, i_2, \ldots, i_{D-K}) - (i_{b-K}) | \max^{a}[(m_{K+1}, m_{K+2}, \ldots, m_D) - m_b] < mTime[b]/{L_b}^2, \max^{v}[(m_{K+1}, m_{K+2}, \ldots, m_D) - m_b] > mTime[b]/{L_b}^2, 1 \leq v \leq (a-1), 0 \leq i_j \leq c_{eff}, 1 \leq j \leq (D-K)\}$, $\max^{v} M$ is the vth largest element in the set M, c is the number of clients, $c_{eff} = \sum_{i=1}^{c} w_i$, $Time[b]$ is defined to be the earliest recorded time for which item b was requested, $i = (i_1, i_2, \ldots, i_{D-K})$, $m = (m_{K+1}, m_{K+2}, \ldots, m_{K+D})$, $m_n = i_{n+k}Time[n]/{L_{n+k}}^2$, $A = \{\sum_{j=1}^{c} a_j w_j | 1 \leq j \leq c, a_j \in Z\}$, Z is the set of non-negative integers, L_b is the length of item b, D is the number of items in the database server, and K is the number of items in the push set.

Proof. The reasoning is similar to the reasoning used to derive the proof for Lemma 2. □

Now that we have presented general formula for $G_{a,b}$, let us use it to develop the expected response time for item b, where item b is an item in the pull set. The analysis is more complex than what we had for Theorem 1, but follows a similar train of thought.

Theorem 3. *The expected response time for item b to be disseminated is given as follows:*

$$\sum_{g=1}^{D-K}\left(P_b(1-P_b)^{L_{\mathrm{p}}g}\left(2DL_{\mathrm{p}} - \sum_{i=1}^{g} L_{\mathrm{p}} - \frac{gL_a}{2} + \sum_{i=0}^{D-K}(i(L_a+L_b)G_{i,b})\right)\right)$$
$$+ \sum_{h=D-K+1}^{D}\left(P_b(1-P_b)^{L_{\mathrm{p}}(h-D+K)}\left(DL_{\mathrm{p}} - \sum_{i=1}^{h-1} L_{\mathrm{p}} + \sum_{i=0}^{D-K}(i(L_a+L_b)G_{i,b})\right)\right)$$

where D is the number of items in the database server, K the number of items in the push set, P_b the probability of pull item b being requested by a client during an unit time interval, L_{p} the average length of an item in the pull set, L_a the average length of all the items, L_b the length of pull item b, and $G_{a,b}$ is the probability that pull item b will be the ath one to be flushed out in a system with non-uniform sized data items.

Proof. Suppose, a request for pull item b is given just after the start of a new broadcast cycle. We can then say that the expected response time for item b can be calculated as such

$$P_b\left(2DL_\mathrm{p} + \sum_{i=0}^{D-K} iG_{i,b}(L_b + L_a)\right)$$

where L_a is the average length of all the items in the database server, L_p the average length of an item in the pull set, D the number of items in the database server, K the number of items in the push set, L_b the length of pull item b, and $G_{a,b}$ is the probability that item b will be the ath one to be flushed out of the pull set.

Continuing down this road of analysis, we can say that the expected time taken for a request for item b to be satisfied soon after the first item is pulled to be as follows:

$$P_b(1 - P_b)\left(2DL_\mathrm{p} - L_1 - \frac{L_a}{2} + \sum_{i=0}^{D-K} iG_{i,b}(L_b + L_a)\right)$$

Therefore, the expected time for a request for item b to be fulfilled when the request is made after the final possible item is pulled is given below as

$$\sum_{g=1}^{D-K}\left(P_b(1 - P_b)^{L_\mathrm{p}g}\left(2DL_\mathrm{p} - \sum_{i=1}^{g} L_\mathrm{p} - \frac{gL_a}{2} + \sum_{i=0}^{D-K}(i(L_a + L_b)G_{i,b})\right)\right)$$

So the expected response time for item b to be disseminated after the wth item in the push set has been broadcasted, where $w > K$, is given below as

$$\sum_{g=1}^{D-K}\left(P_b(1 - P_b)^{L_\mathrm{p}g}\left(2DL_\mathrm{p} - \sum_{i=1}^{g} L_\mathrm{p} - \frac{gL_a}{2} + \sum_{i=0}^{D-K}(i(L_a + L_b)G_{i,b})\right)\right)$$
$$+ \sum_{h=D-K+1}^{w}\left(P_b(1 - P_b)^{L_\mathrm{p}(h-D+K)}\left(DL_\mathrm{p} - \sum_{i=1}^{h-1} L_\mathrm{p} + \sum_{i=0}^{D-K}(i(L_a + L_b)G_{i,b})\right)\right)$$

Therefore, we can conclude that the expected response time for item b to be disseminated is as follows:

$$\sum_{g=1}^{D-K}\left(P_b(1 - P_b)^{L_\mathrm{p}g}\left(2DL_\mathrm{p} - \sum_{i=1}^{g} L_\mathrm{p} - \frac{gL_a}{2} + \sum_{i=0}^{D-K}(i(L_a + L_b)G_{i,b})\right)\right)$$
$$+ \sum_{h=D-K+1}^{D}\left(P_b(1 - P_b)^{L_\mathrm{p}(h-D+K)}\left(DL_\mathrm{p} - \sum_{i=1}^{h-1} L_\mathrm{p} + \sum_{i=0}^{D-K}(i(L_a + L_b)G_{i,b})\right)\right) \quad \Box$$

where $G_{i,b}$ is computed using Lemma 3.

Fig. 5 shows the response time for item 5 after the previous broadcast cycle has ended as a function of the average data item size, L_a, and the number of items in the push set, K, when the probability of item

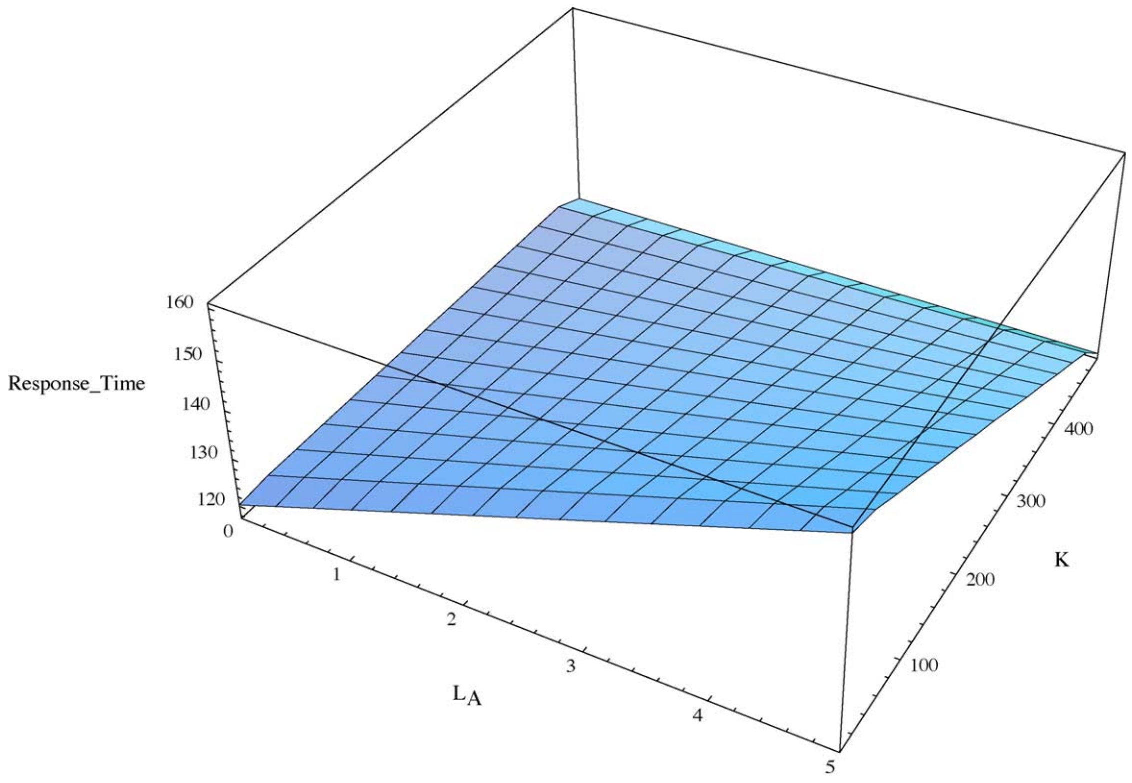

Fig. 5. Response time for item 5 after previous broadcast cycle ends as a function of average data item size (L_a) and number of items in push set (K) where $P_b = 0.5$ (probability of item 5 being requested during an unit time interval), $L_\mathrm{p} = 5$ (the average length of an item in the pull set), and $D = 500$ (number of items in database server).

5 being requested during an unit time interval, P_b is 0.5, the total number of items in the database server both push and pull, D, is 500, and the average length of a data item in the pull set is L_p. We can see that as the average data item size increases the response time also increases. This is because it generally takes longer for the previous items to be sent out if the average data size of an item is larger. Similarly, we can see that, as the number of items in the push set, K, increases, there are less items in the pull set to be flushed out, so the response time will decrease.

Let us now determine the expected overall time for the pull queue to be flushed out for the general case. The analysis is quite similar to what we did in Theorem 2.

Theorem 4. *The expected overall time for the pull queue to be flushed when we have non-uniform data sizes, where L_x is the length of item x and P_x is the probability of item x being requested, and $H = \{K+1, \dots, D\}$ is computed as follows*:

$$\sum_{i=1}^{D-K} i \sum_{a_1 \in H, a_2 \in H - a_1, \dots a_i \in H - a_1 - \cdots - a_{i-1}} (L_{a_1} + \cdots + L_{a_i}) P_{a_1} P_{a_2} \cdots P_{a_i} \prod_{m \in H - a_1 - \cdots - a_i} (1 - P_m)$$

Proof. From Theorem 2 we know that the expected overall time for the pull queue with all items of uniform size, namely unity, to be flushed out is:

$$\sum_{i=1}^{D-K} i \sum_{a_1 \in H, a_2 \in H-a_1, \ldots, a_K \in H-a_1-\cdots-a_{K-1}} P_{a_1} P_{a_2} \cdots P_{a_K} \prod_{m \in H-a_1-\cdots-a_K} (1-P_m)$$

Based on the above results, we can determine that the expected overall time for the pull queue to be flushed is given below when we have non-uniform data sizes. The L_x values after the second summation symbol below, refer to the different possible aggregate lengths of the items requested to be disseminated.

$$\sum_{i=1}^{D-K} i \sum_{a_1 \in H, a_2 \in H-a_1, \ldots, a_i \in H-a_1-\cdots-a_{i-1}} (L_{a_1} + \cdots + L_{a_i}) P_{a_1} P_{a_2} \cdots P_{a_i} \prod_{m \in H-a_1-\cdots-a_i} (1-P_m) \quad \square$$

7. Conclusions

A generalized and novel hybrid scheduling algorithm has been developed for an asymmetric communication environment which consists of one server and multiple clients. The enhanced hybrid push–pull algorithm has taken into consideration multiple data sizes, QoS criteria such as priority levels, computing data access probabilities on the fly, and handling aberrant or stray situations which can lead to distorted probability calculations. Our scheme makes sure that these stray situations will be taken care of by the pull set instead of the push set. To the best of our knowledge this is the first hybrid scheduling algorithm to effectively handle all these these factors simultaneously. We then extended the performance analysis on our algorithm by looking at the response time for individual data items in the pull set, as well as fine tuning the analysis for the expected overall time for the pull queue to be flushed out. We have also presented a performance evaluation of the enhanced hybrid push–pull algorithm. The performance analysis will help us to better understand how to meet QoS requirements and also be of use in systems with deadlines.

Acknowledgements

This work was partially supported by NSERC and Canada Research Chair (CRC) Program, Canadian Foundation for Innovation, and OIT/Distinguished Researcher Award. Trivikram Dash work was part of his thesis under the supervision of Dr. A. Boukerche.

References

[1] S. Acharya, M. Franklin, S. Zdonik, Dissemination-based data delivery using broadcast disks, IEEE Personal Communications, December 1995, pp. 50–60.
[2] S. Acharya, M. Franklin, S. Zdonik, Prefetching from a broadcast disk, in: Proceedings of the 12th International Conference on Data Engineering, Feburary 1996, pp. 276–285.

[3] S. Acharya, S. Muthukrishnan, Scheduling on-demand broadcasts: new metrics and algorithms, in: Proceedings of the Fourth Annual ACM/IEEE Mobicom, 1998, pp. 43–54.
[4] D. Aksoy, M. Franklin, R×W: a scheduling approach for large-scale on-demand data broadcast, IEEE/ACM Trans. Netw. 7 (6) (1999) 846–860.
[5] A. Bar-Noy, B. Patt-Shamir, I. Ziper, Broadcast disks with polynomial cost functions, in: IEEE INFOCOM'2000, 2000, pp. 575–584.
[6] S. Baruah, A. Bestavros, Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems, in: Proceedings of the IEEE International Conference on Data Engineering, April 1997, pp. 543–551.
[7] S. Baruah, A. Bestavros, Real-time mutable broadcast disks, in: Proceedings of the Second International Workshop on Real-Time Databases, 1997.
[8] J.C.R. Bennett, H. Zhang, Hierarchical packet fair queueing algorithms, in: Proceedings of the ACM SIGMOD International Conference, 1994, pp. 1–12.
[9] A. Bestavros, Aida-based real-time fault-tolerant broadcast disks, in: Proceedings of the Second IEEE Real-Time Technology and Applications Symposium, 1996.
[10] Y. Guo, M.C. Pinotti, S.K. Das, A new hybrid broadcast scheduling algorithm for asymmetric communication systems, ACM SIGMOBILE Mobile Comput. Commun. Rev. 5 (3) (2001) 39–44.
[11] S. Hameed, N.H. Vaidya, Efficient algorithms for scheduling data broadcast, Wireless Netw. 5 (3) (1999) 183–193.
[12] S. Hameed, N.H. Vaidya, Scheduling data broadcast in asymmetric communication environments, Wireless Netw. 5 (3) (1999) 171–182.
[13] J. Hu, K.L. Yeung, G. Feng, K.F. Leung, A novel push-and-pull hybrid data broadcast scheme for wireless information networks, in: IEEE International Conference on Communications, June 2000, pp. 1778–1782.
[14] R. Jain, J. Werth, Airdisks and airraid: modeling and scheduling periodic wireless data broadcast (extended abstract), DIMACS Technical Report 95–11, Rutgers University, DIMACS, 1995.
[15] M.C. Pinotti, N. Saxena, Push less and pull the current highest demanded data item to decrease the waiting time in asymmetric communication environments, 4th International Workshop on Distributed Computing, Special Day on Wireless Networks, in LNCS 2571, December 28–31, 2002, Calcutta, India.
[16] Y. Wu, G. Cao, Stretch-optimal scheduling for on-demand data broadcasts, in: IEEE International Conference on Computer Communications and Networks, October 2001, pp. 500–504.