Wireless Networks 10, 103–120, 2004

Adaptive Realtime Bandwidth Allocation for Wireless Data Delivery

CHI-WAI LIN, HAIBO HU and DIK-LUN LEE *
Computer Science Department, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

Abstract. The combination of broadcast and on-demand data delivery services is an economic way to build a highly scalable wireless information system with limited bandwidth. The use of data broadcasting should be adaptive so that the system response time can always be minimized. A traditional approach requires the development of a system response time equation in order to find the optimal solution. However, obtaining such an equation is not always possible. We observe that by maintaining a certain level of on-demand request arrival rate, a close approximation to the optimal solution can be obtained. Using this approach, a real-time adaptive data delivery algorithm is developed. Our algorithm does not require the access information of the data items to be known exactly, which is needed normally for this kind of optimization problems. A simple and low overhead bit vector mechanism is able to capture the relative popularities of the data items. With this information, our algorithm can give a performance comparable to the ideal case in which the access information for each data item is known exactly.

Keywords: on-demand broadcasting, bandwidth allocation, mobile computing

1. Introduction

In a mobile computing environment, scarce wireless bandwidth, asymmetric communication, limited battery power and mobility of the mobile clients are the major challenges to the design of a mobile system [12]. The main focus of this paper is on the effective use of limited bandwidth in a wireless communication environment. One of the major research problems in this area is to design a wireless information system to provide mobile clients with wireless access to a central database. A similar access pattern is assumed to be shared among clients. With limited wireless bandwith, the system should be able to provide wireless data access to as many clients as possible. At the same time, the system response time should always be minimized under different system workloads.

A promising approach to building such a system is to use both on-demand and broadcast data delivery services that complement each other [3,6,10,11,13,16,20]. On-demand data delivery is a traditional client-server data delivery model [2,13]. Whenever a data item is requested by a client, the client issues a data request to the server and the server replies the client with the requested data item. On the other hand, data broadcasting is a listen only data delivery service [2,13]. Unlike on-demand data delivery, clients are not required to compete for resources to send data requests to the server. Instead, they just keep listening to the broadcast channel and wait for the data items they required. Since simultaneous data access is possible for an arbitrary number of clients, data broadcasting can scale up with unlimited number of clients. Therefore, it is a bandwidth efficient data delivery techniques for a large client population with common access to data. With a proper balance between the use of these two types of data delivery services, a highly scalable and efficient wireless information system can be built.

To minimize the response time of the system, usage between on-demand and broadcast data delivery services must be balanced. This can be done by providing a certain number of frequently accessed data items on the broadcast channel according to the workload, while the rest are serviced by the on-demand channels. Traditionally, this requires the development of an explicit system response time equation [13,16,20]. Based on this, together with the exact [13,16] or estimated [20] request arrival rates of the data items, the optimal number of broadcast data items can be approximated [13,16,20]. However, developing such an equation is not always possible. Besides, only the best achieved system response time under different system workloads has been previously considered [13,16,20]. None of them have considered the real-time adaptiveness of the system in reaction to changes in system workload.

In this paper, we consider the situation when the development of an explicit system response time equation is impossible, and, the system workload is assumed to be changing dynamically in a real-time fashion. Without knowing the exact request arrival rates but only the relative popularities of the data items, we want to approximate the optimal system response time under different system workloads with a real-time adaptive data delivery algorithm.

The rest of this paper is organized as follows. Section 2 gives an overview of the related works in the literature. Section 3 shows our wireless data delivery model. Section 4 gives a detailed description of our adaptive data delivery algorithm. Section 5 describes our simulation model and discusses the experimental results. Finally, section 6 concludes the paper and suggests possible future research directions.

* Corresponding author.
E-mail: dlee@cs.ust.hk

2. Related work

A number of data delivery models making use of both broadcast and on-demand data delivery services have been proposed in the literature [3,6,10,11,13,16,20]. The basic idea behind these models is to use the broadcast data delivery service to alleviate the use of on-demand data delivery service. However, none of them have considered the real-time adaptiveness of the system in reaction to the changes in system workload or client access pattern. Some of them even do not make use of the broadcast service adaptively.

In [3], a single *shared* uplink channel and a broadcast channel are used. The whole database is broadcast on the broadcast channel. The broadcast data are organized as broadcast disks [1] according to a static client access pattern. They are interleaved together with the on-demand request responses on the broadcast channel. There are no updates for the data items. Caching is used on the client side. For each data request, if the desired item cannot be found in the caches, the client monitors the broadcast channel for a fixed *Threshold* number of frames. If the desired data item cannot be found, an on-demand request will be sent to the server through the shared uplink channel. After that, the client keeps monitoring the broadcast channel until the desired data item arrives. This model is not adaptive to the system workload or the client access pattern.

A single broadcast channel and a shared uplink channel architecture is adopted in [10]. Broadcast disks are used but only a fraction of the database items are used to construct them. There are no updates for the data items. Caching is used on the client side and the *Threshold* data access mechanism is used by the clients. A major feature of this model is that a mechanism to collect client access information called MFA[1] vector mechanism is proposed, such that the content of the broadcast disks are adjusted according to the client access pattern. However, this model is not adaptive to the system workload.

A set of point-to-point on-demand channels is used instead of a single shared uplink channel for the pull-base data requests in [11]. With a single broadcast channel, broadcast disks are used. Only a fraction of the database items are used to construct the broadcast disks. There are no updates for the data items. Caching is used on the client side. MFA vector mechanism is used, so that the broadcast content is adaptive to the client access pattern. In addition, the use of different indexing strategies to improve the *Threshold* data access mechanism are evaluated. Again, this model is not adaptive to the system workload.

In [6], a broadcast channel and a shared uplink channel architecture is used. The broadcast content are constructed based on the received uplink requests with a flat[2] broadcast structure. $(1, m)$ indexing is used to facilitate data access, which will broadcast a complete index m times during a broadcast cycle. A major concern for this model is to minimize the power consumption on data access. Thus a number of power efficient data access protocols are proposed. This model is adaptive to the client access pattern but it is not adaptive to the system workload.

In [13], the architecture consists of a broadcast channel, an on-demand downlink channel and a shared on-demand uplink channel. A flat broadcast is used in this model. If a data item cannot be found on the broadcast channel within a specific period of time, an uplink request is then sent to the server. After that, the client keeps listening to the on-demand downlink channel until the requested data item appears. An algorithm based on the use of a system response time equation is developed to ensure the mean data access time of the system does not exceed a predefined value. Different ways to collect the client access information are suggested, but there are no experimental results presented. The model is adaptive to the system workload, but it assumes that the request arrival rates of the data items are known completely by the server.

A cellular model is assumed in [16]. Each cell has a fixed number of channels. These channels can be used to provide broadcast or pull-based data delivery services. A flat broadcast is used in this model. Depending on the system workload, the number of broadcast data items and the bandwidth allocated for broadcast and pull-based data delivery services are adjusted dynamically to minimize data access time. Normally, for each data request, the client has to send the request to the server. The server then replies either with the requested data item or the broadcast channel access information for the client to retrieve the data item on a specific broadcast channel. System response time equations are developed to find the optimal solution. A static client access pattern is assumed and the request arrival rates of the data items are assumed to be known completely by the server.

The model in [20] is adaptive to both the system workload and client access pattern. It consists of a broadcast channel, an on-demand downlink channel and a shared on-demand uplink channel. Flat broadcast is used. The number of data items on the broadcast channel is dynamically adjusted according to the system workload. The algorithm is based on a system response time equation. A temperature value (request rate) is associate with each data item. When the temperature of a data item gets hot enough, it will be promoted to the broadcast channel. A cool down mechanism is used to control the leaving of the broadcast data from the broadcast channel. Therefore, explicit broadcast data access information is not required. Indexing is used to improve the data access mechanism. To facilitate the development of the system response time equation, the authors assumed that a copy of the index is broadcast with every data item. However, this increases the the overhead of using indexing significantly, so the system performance worsens.

[1] MFA stands for "Most Frequently Accessed".

[2] "Flat" means all objects are of equal importance and disseminated once in a broadcast cycle.

3. Wireless data delivery model

3.1. System architecture

The architecture of the system adopted in our model is the same as the one in [11]. A single broadcast channel and a set of on-demand point-to-point channels are used in a single cell environment. A server (Mobile Support Station) provides mobile clients with wireless access to a central database through the broadcast and on-demand channels. All data items in the database are of the same size. One data item can be stored in a single frame on the broadcast channel. The data items are read-only and will not be updated by the clients or server. Unlike the model in [11], there is no queueing buffer for the on-demand channels.

A flat broadcast is used for the broadcast data [1]. The use of a broadcast channel is to avoid overloading the on-demand channels. As the workload of the on-demand channels changes, the number of data items on the broadcast channel k will be increased or decreased. The value k is embedded in every frame sent on the broadcast channel. It is used by the clients to determine how many frames on the broadcast channel they should listen to before issuing an on-demand request to the server.

3.2. Data access mechanism

It is assumed that each client can have only one single request at a time, so a client will not issue a new request unless the previous request is completed. Besides, each request addresses a single data item only. For each data request, the client first reads a frame from the broadcast channel. If it is not the desired data item, then the client continues to examine up to $k - 1$ succeeding frames on the broadcast channel. If the desired data item is not encountered, the client turns to the on-demand channel and issues an on-demand request to the server. Since there is no queueing buffer for the on-demand channels, once all the on-demand channels are occupied, the client will need to wait for a short random period of time and check the availabilities of the on-demand channels again. This is repeated until an on-demand channel is available and the request is sent to the server. Once the request is received by the server, the requested data item will be sent to the client through the same on-demand channel. Obviously, when too many clients are competing for the on-demand channels, response time of the system will be unbounded.

3.3. Collecting data access information

To make the necessary adaptive adjustments of the broadcast data items, data access information has to be collected continuously. Access information of the data requests made through the on-demand channels can be obtained by the server without extra cost. However, the server does not know which data items of the broadcast channel have been accessed by the clients. In view of this, the MFA vector mechanism proposed in [10] is adopted.

This mechanism requires each client to maintain a MFA vector and a broadcast version number. A MFA vector is basically a bit vector in which each bit represents a data item on the broadcast channel. Whenever a request is answered by a broadcast data item, the bit corresponding to the accessed data item is set. The broadcast version number is used by the server to ensure the validity of the relationship between the bit positions and the data items. It is incremented whenever the set of broadcast data items is changed.

Whenever an on-demand request is sent to the server, the MFA vector and the broadcast version number will also be piggybacked to the server at the same time. The server keeps a request received record for each data item. Whenever an explicit on-demand request for a data item is received or a corresponding bit is set for a data item in a valid MFA vector[3] received, the corresponding request received record of that particular data item is incremented by one.

Even with these information, the exact request arrival rate for each broadcast data item still cannot be known. There are two reasons. First of all, not all the MFA vectors are sent to the server. For instance, if the requests of a client can all be answered by the broadcast data, the MFA vector will not be sent to the server. Second, even a MFA vector is received by the server, if it is not of the same broadcast version currently used by the server, it will be discarded. Therefore, the MFA vector mechanism is unable to collect the exact access frequencies of the broadcast data items, so that the exact request arrival rates for the broadcast data items cannot be known exactly. Nevertheless, the relative popularities of data items can still be estimated with this mechanism.

4. Real-time adaptive data delivery

4.1. Optimal system response time

The use of data broadcasting should be adaptive such that system response time can always be minimised under different system workloads. A traditional approach would require the development of a system response time equation, based on which, the optimal solution for assigning data to the broadcast channel that lead to an optimal system response time can be found. However, obtaining such an equation is not always possible.

For instance, we assumed that there are no queueing buffers for the on-demand channels, so an $M/M/c$ queueing model [15] cannot be used. Also, we assumed that requests are not lost even when all the on-demand channels are occupied, so an $M/M/c/c$ queueing model [15] cannot be used either. Therefore, it is impossible to obtain a system response time equation for our system without making some simplifying assumptions.

To approximate the optimal system response time, we have made the following observations. Since on-demand channels can provide instant access to data items, they should be fully

[3] A valid MFA vector is defined as a MFA vector with the same broadcast version number as the current broadcast version number used by the server.

utilized, but must not be overloaded. Otherwise, the system response time will be unbounded. Thus, the idea of utilizing the on-demand channels as much as possible while not overloading them requires a certain level of on-demand request arrival rate to be maintained.

4.2. Difficulty in maintaining on-demand request arrival rate

Assuming that there are n data items in the server database and λ_d is the set containing the request arrival rate for each of the data items such that λ_{d_i} is the request arrival rate for data item i. The workload of the system is characterized by the aggregate request arrival rate of the data items in the server database. It is given in the following equation [16]:

$$\lambda = \sum_{i=1}^{n} \lambda_{d_i}. \tag{1}$$

Suppose, an on-demand request arrival rate λ'_o is to be maintained. When $\lambda > \lambda'_o$, the k most popular data items have to be identified and put onto the broadcast channel, while the rest of the data items are serviced by the on-demand channels so that:

$$k = \text{Min}\left\{ j \,\Big|\, \sum_{i=1}^{j} \gamma_i \geqslant \lambda - \lambda'_o \right\} \tag{2}$$

where $\gamma_i \in \lambda_d$ and $\gamma_1 \geqslant \gamma_2 \geqslant \cdots \geqslant \gamma_n$. The objective is to control the amount of data on the broadcast channel k so that the on-demand request arrival rate is less than or equal to λ'_o.

When $\lambda \leqslant \lambda'_o$, the system workload is so low that all requests can be served by the on-demand channels. In this case, a frame containing one of the most frequently accessed data items will still be broadcast. It is because according to the data access mechanism, each client is required to read a frame from the broadcast channel first. Broadcasting the most frequently accessed data item is better than broadcasting a dummy frame, even though the performance gain is not significant. The algorithm for the selection of broadcast data is shown in algorithm 1.

Algorithm 1. Algorithm for broadcast data selection
Select-Broadcast-Data(λ_d, λ'_o)
$\lambda \leftarrow j \leftarrow k \leftarrow 0$
List-Clear(List-Of-Broadcast-Data)
for $i \leftarrow 1$ **to** n **do**
    $\lambda \leftarrow \lambda + \lambda_{d_i}$
$\gamma[] \leftarrow$ Sort-In-Descending-Order(λ_d)
if $\lambda \leqslant \lambda'_o$ **then**
    List-Insert(List-Of-Broadcast-Data, $\gamma[1]$.data-id)
else
    while $j < \lambda - \lambda'_o$ **do**
      $k \leftarrow k + 1$
      $j \leftarrow j + \gamma[k]$.value
      List-Insert(List-Of-Broadcast-Data, $\gamma[k]$.data-id)
return List-Of-Broadcast-Data

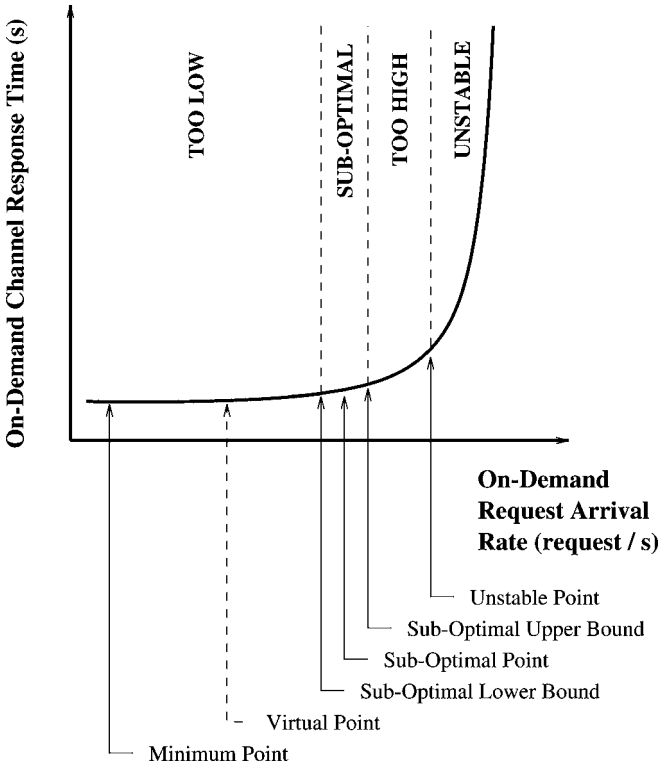


Figure 1. On-demand channel response time vs. on-demand request arrival rate.

This technique requires the request arrival rates, and hence, access frequencies of data items, to be known exactly by the server. However, the exact access information cannot be obtained by using the MFA vector mechanism. Therefore, the algorithm cannot be used directly to maintain the on-demand request arrival rate in our model.

4.3. Techniques to maintain on-demand request arrival rate

4.3.1. Basic idea

Figure 1 shows the relationship between the on-demand channel response time and the on-demand request arrival rate. According to the on-demand request arrival rate, four on-demand channel states are defined: *TOO LOW*, *SUB-OPTIMAL*, *TOO HIGH* and *UNSTABLE*. The *Sub-Optimal Point* is the on-demand request arrival rate that we want to maintain. It is assumed that when either one of the *previous state* or *current state* of the on-demand channels or both are within the range between the *Sub-Optimal Upper Bound* and *Sub-Optimal Lower Bound*, the on-demand request arrival rate at the *Sub-Optimal Point* is regarded as successfully maintained.

The basic idea of our real-time adaptive data delivery algorithm is that, whenever the on-demand request arrival rate cannot be maintained at the *Sub-Optimal Point*, the amount of data on the broadcast channel will be increased or decreased according to the on-demand channel states. To decide which data item is to be put on or removed from the broadcast channel next, relative popularities of data items are used.

If real-time system performance is taken into consideration, increasing or decreasing the number of data items on the

broadcast channel one at a time until the on-demand channels reach the *SUB-OPTIMAL* state will be unacceptable. In fact, whenever the on-demand request arrival rate cannot be maintained at the desired level, the adaptive algorithm should be able to bring the on-demand channels back to the *SUB-OPTIMAL* state as soon as possible. Therefore, the number of broadcast data items to increase or decrease should be determined by the algorithm effectively and efficiently such that a stable system performance can always be maintained while the system can still be highly reactive to the changes in system workload or client access pattern.

4.3.2. Controlling the number of broadcast data items

To control the increase or decrease of the number of broadcast data items, the broadcast data selection algorithm shown in algorithm 1 is used indirectly by maintaining a virtual request arrival rate called *Virtual Point*. The name virtual is used because the *Virtual Point* is used to assist the system to maintain the on-demand request arrival rate at the *Sub-Optimal Point*, rather than at the *Virtual Point* itself.

Request arrival rates of the data items are needed by the broadcast data selection algorithm. They are computed based on the incomplete access information collected by the MFA vector mechanism. Although the computed request arrival rates may not represent the actual values, they can still preserve the information about the relative popularities of the data items with certain accuracy. Since the computed request arrival rates for the broadcast data items are normally underestimated, if on-demand request arrival rate is maintained at the *Sub-Optimal Point* explicitly using the broadcast data selection algorithm, the resulted on-demand request arrival rate would very likely exceed the *Sub-Optimal Point*.

Suppose the current on-demand request arrival rate is too high, then the *Virtual Point* will be decreased until the on-demand request arrival rate can be maintained at the *Sub-Optimal Point*. In fact, decreasing the *Virtual Point* has the effect of increasing the number of broadcast data items. Once enough data items are broadcast, the desired on-demand request arrival rate can be maintained. Similarly, when the current on-demand request arrival rate is too low, then the *Virtual Point* will be increased until the on-demand request arrival rate can be maintained at the *Sub-Optimal Point*.

Obviously, even when the exact access frequencies of the broadcast data items cannot be obtained, if the accuracy of the captured popularities of the data items is high, a highly comparable performance to the ideal case in which the access information is known exactly can still be achieved. The remaining problem is to design an algorithm to adjust the *Vitual Point* so that it can give a stable and steady real-time performance in a highly dynamic environment.

4.3.3. Adjusting the virtual point to control the number of broadcast data items

Before describing the algorithm for the adjustment of the *Virtual Point*, the control parameters and on-demand channel states shown in figure 1 are formally defined. The following parameters are used to control the adjustment of the *Virtual Point*:

- *Sub-Optimal Point*: The request arrival rate level to be maintained. It is also the maximum value for the *Virtual Point*.
- *Sub-Optimal Upper and Lower Bounds*: Within this range, the *Sub-Optimal Point* is regarded as successfully maintained.
- *Unstable Point*: The request arrival rate level at which the on-demand channels start getting overloaded.
- *Minimum Point*: The minimum value for the *Virtual Point*.

Corresponding to these control points, four on-demand channel states are defined:

- *SUB-OPTIMAL*: When the on-demand request arrival rate is within the range between the *Sub-Optimal Upper and Lower Bounds*.
- *TOO HIGH*: When the on-demand request arrival rate is larger than the *Sub-Optimal Upper Bound* and less than the *Unstable Point*.
- *TOO LOW*: When the on-demand request arrival rate is less than the *Sub-Optimal Lower Bound*.
- *UNSTABLE*: When the on-demand request arrival rate is larger than or equal to the *Unstable Point*.

Two additional control parameters which are not shown in figure 1 are used to control the adjustment speed of the *Virtual Point*:

- *Normal Step*: A request arrival rate stepping used for the adjustment of the *Virtual Point* when the on-demand channels are not overloaded.
- *Large Step*: A larger request arrival rate stepping used for the adjustment of the *Virtual Point* when the on-demand channels are overloaded.

Virtual Point can be adjusted between the *Sub-Optimal Point* and the *Minimum Point*. The decision for the adjustment of the *Virtual Point* is made according to the *previous state* and *current state* of the on-demand channels:

- *Case I*: If both the *previous state* and *current state* of the on-demand channels are UNSTABLE, then the *Virtual Point* would be decreased with a *Large Step*. It is because when the on-demand channels are overloaded, system response time can increase exponentially. Therefore, the decrease of the Virtual Point must be able to stabilize the system as soon as possible.
- *Case II*: If only the *current state* of the on-demand channels is *UNSTABLE*, or both the *previous state* and *current states* are *TOO HIGH*, then the *Virtual Point* would be decreased by one or more *Normal Steps*. The amount of decrease is accumulative[4].

[4] When the same condition is sustained consecutively, more and more *Normal Steps* would be used. For instance, when the condition holds for the first time, one *Normal Step* would be used. When the condition holds con-

- *Case III*: If both the *previous state* and *current state* of the on-demand channels are *TOO LOW*, then the *Virtual Point* would be increased by one or more *Normal Step*. The amount of increase is also accumulative.
- *Case IV*: If either the *previous state* is *TOO LOW* and the *current state* is *TOO HIGH*, or the *previous state* is *TOO HIGH* and the *current state* is *TOO LOW*, then it means that the previous increase or decrease made to the *Virtual Point* has been excessive. Instead of decreasing or increasing the *Virtual Point* incrementally from the very beginning, a mid-point would be taken from the previous and current *Virtual Points*. This approach helps to speed up the time required for the on-demand channels to reach the *SUB-OPTIMAL* state.

Algorithm 2 shows the pseudo-code for the adjustment of the *Virtual Point*.

Algorithm 2. Algorithm for the virtual point adjustment

```
Adjust-Virtual-Point(Previous-State, Current-State)
temp = Virtual-Point
if Previous-State = UNSTABLE and
Current-State = UNSTABLE then
    Increase ← Decrease ← 0
    Virtual-Point ← Virtual-Point − Large-Step
else if ((Previous-State = UNSTABLE or
Previous-State = TOO-HIGH) and Current-State =
TOO-HIGH) or Current-State = UNSTABLE then
    Increase ← 0
    Decrease ← Decrease + Normal-Step
    Virtual-Point ← Virtual-Point − Decrease
else if Previous-State = TOO-LOW and
Current-State = TOO-LOW then
    Decrease ← 0
    Increase ← Increase + Normal-Step
    Virtual-Point ← Virtual-Point + Increase
else
    Increase ← Decrease ← 0
    if Previous-State = TOO-HIGH and Current-State
    = TOO-LOW or (Previous-State = TOO-LOW
    and Current-State = TOO-HIGH) then
      Virtual-Point ← (Previous-Virtual-Point
                      + Virtual-Point) / 2
Previous-Virtual-Point ← temp
if Virtual-Point > Sub-Optimal-Point then
    Virtual-Point ← Sub-Optimal-Point
if Virtual-Point < Minimum-Point then
    Virtual-Point ← Minimum-Point
return Virtual-Point
```

secutively for the second time, two *Normal Steps* would be used and so on.

4.4. Computation of request arrival rates

4.4.1. Request arrival rates for the data items

Request arrival rate λ_{d_i} for data item i is computed with an exponentially weighted moving average technique [20]:

$$\lambda_{d_i,j} = (1-\alpha)\lambda_{d_i,j-1} + \alpha\frac{r_{i,j}}{t_j} \tag{3}$$

where $\lambda_{d_i,j}$ is the request arrival rate of data item i for the jth evaluation period, $0 < \alpha \leqslant 1$, $r_{i,j}$ is the number of requests received corresponding to data item i during the jth evaluation period and t_j is the length of the jth evaluation period.

4.4.2. On-demand request arrival rate

On-demand request arrival rate is computed with the following equation:

$$\lambda_{o,j} = \frac{o_j}{t_j} \tag{4}$$

where $\lambda_{o,j}$ is the on-demand request arrival rate for the jth evaluation period, o_j is the total requests received through the on-demand channels over the jth evaluation period and t_j is the length of the jth evaluation period.

4.5. Evaluation time for the system

To determine the evaluation period, two parameters are required:

- *Maximum Evaluation Cycle*: the maximum number of broadcast cycles allowed for an evaluation period.
- *Maximum Evaluation Time*: the maximum time duration allowed for an evaluation period.

Evaluation period e_{cycle} in terms of number of broadcast cycles is obtained with the following equation:

$$e_{\text{cycle}} = \text{Min}\left(c_{\max}, \left\lfloor \frac{t_{\max}}{t_{\text{cycle}}} \right\rfloor\right) \tag{5}$$

where $c_{\max}$ is the *Maximum Evaluation Cycle*, $t_{\max}$ is the *Maximum Evaluation Time*, t_{cycle} is the time required for a complete broadcast of the current set of broadcast data items and we have assumed $t_{\max} \geqslant t_{\text{cycle}}$. Basically, this equation states that evaluation period equals to an integer multiple of broadcast cycles not more than the *Maximum Evaluation Cycle* and the total length in time does not exceed the *Maximum Evaluation Time*.

4.6. Real-time adaptive data delivery algorithm

With all the techniques described above, a real-time adaptive data delivery algorithm is developed. Once the system starts, a dummy data item is first broadcast for a short period of time, so that the system can compute the request arrival rates based on the number of requests received through the on-demand channels. As long as the system is up, the following steps are repeated. The request arrival rate for each data item and the on-demand request arrival rate are computed with equations (3) and (4), respectively. Then the *current state* of

the on-demand channels is determined according to the on-demand request arrival rate. After that, the *Virtual Point* is adjusted according to the *previous state* and *current state* of the on-demand channels using the *Virtual Point* adjustment algorithm shown in algorithm 2. Next, the set of broadcast data items is determined with the broadcast data selection algorithm shown in algorithm 1, according to the *Virtual Point* and the request arrival rates of the data items. The selected set of broadcast data items will then be broadcast for a specific number of cycles. The number of cycles is determined by equation (5).

The running time of our algorithm is $O(n \log n)$. It is attributed to the need for sorting the request arrival rates of the data items in the broadcast data selection algorithm shown in algorithm 1. The pseudo-code for our real-time adaptive data delivery algorithm is shown in algorithm 3.

Algorithm 3. Algorithm for adaptive broadcast

Adaptive-Broadcast()
Evaluation-Start-Time ← Clock
Broadcast(Dummy-Items)
while SYSTEM-UP **do**
  Evaluation-Time ← Clock − Evaluation-Start-Time
  for $i \leftarrow 1$ **to** n **do**
    $\lambda_{d_i} \leftarrow (1-\alpha)\lambda_{d_i} + \alpha \dfrac{\text{Request-Received-For-Data-Item}_i}{\text{Evaluation-Time}}$
  $\lambda_o \leftarrow \dfrac{\text{On-Demand-Requests-Received}}{\text{Evaluation-Time}}$
  Previous-State ← Current-State
  if $\lambda_o \geqslant$ Unstable-Point **then**
    Current-State ← UNSTABLE
  else if $\lambda_o >$ Sub-Optimal-Upper-Bound **then**
    Current-State ← TOO-HIGH
  else if $\lambda_o <$ Sub-Optimal-Lower-Bound **then**
    Current-State ← TOO-LOW
  else
    Current-State ← SUB-OPTIMAL
  Virtual-Point ← Adjust-Virtual-Point(Previous-State, Current-State)
  Set-Of-Broadcast-Data ← Select-Broadcast-Data(λ_d, Virtual-Point)
  cycle ← Min(Maximum-Evaluation-Cycle, $\left\lfloor \dfrac{\text{Maximum-Evaluation-Period}}{\text{Time-For-A-Broadcast-Cycle}} \right\rfloor$)
  Evaluation-Start-Time ← Clock
  for $i \leftarrow 1$ **to** cycle **do**
    Broadcast(Set-Of-Broadcast-Data)

5. Experiments and results

5.1. Simulation model

Our simulation program is implemented with C and Csim [18]. The simulation model used in the experiments is shown in figure 2.

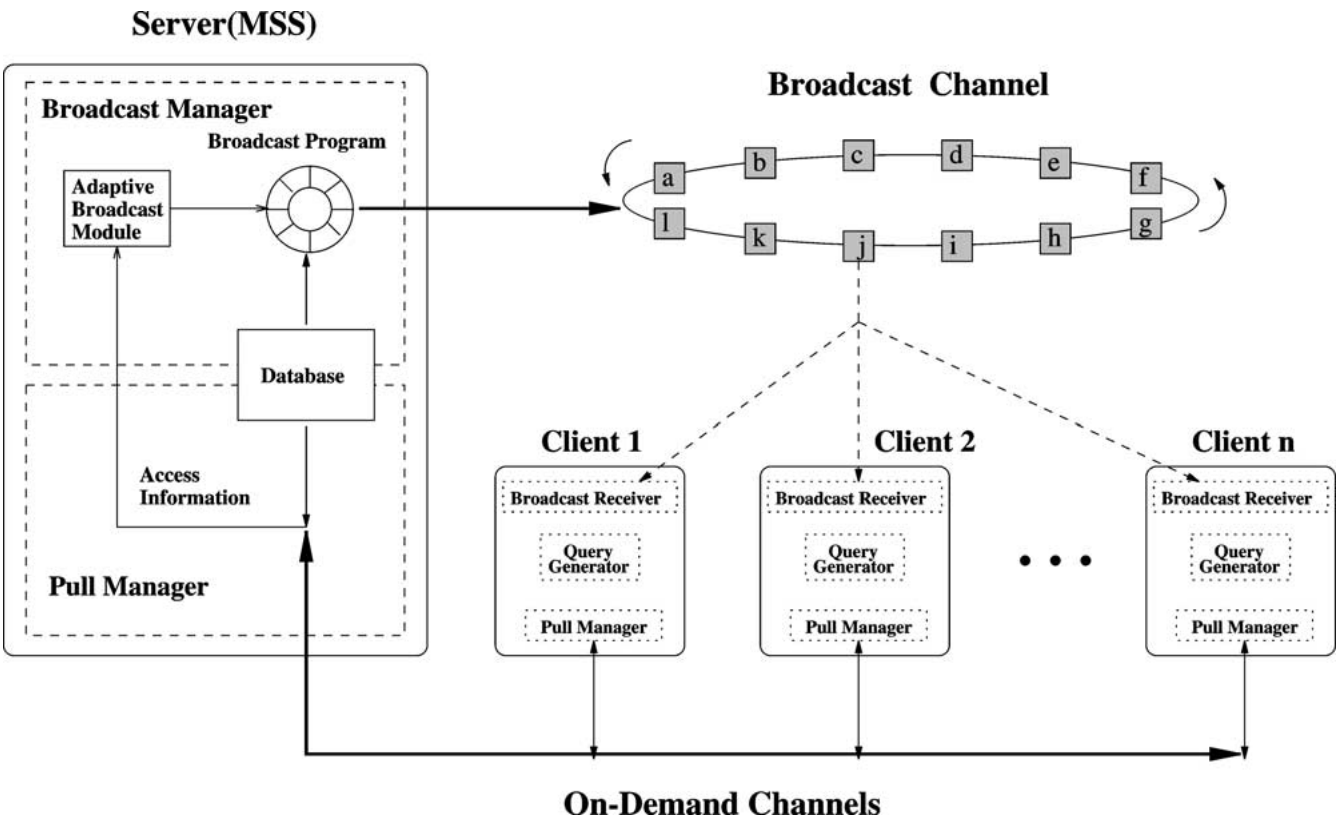


Figure 2. Simulation model.

5.1.1. Server model

The server has a database with size, *Database Size*. Each data item has a size of *Data Item Size*. The server is modeled by two sub-processes, namely *broadcast manager* and *pull manager*.

There is a single broadcast channel with bandwidth, *Broadcast Bandwidth*. The *broadcast manager* is responsible for broadcasting data adaptively on the broadcast channel according to the system workload. The adaptive broadcast algorithm shown in algorithm 3 is used.

There are *On-Demand Channel* number of on-demand channels, each having a bandwidth of *On-Demand Bandwidth*. The *pull manager* is responsible for granting free on-demand channels to clients.

5.1.2. Client model

To simulate the contention of the on-demand channels among clients, each client is modeled by an independent process. It is assumed that each client can have a single request at a time, so a client will not issue a new request unless the previous request is completed. Each request addresses a single data item only.

For each request, a client waits for a random amount of time drawn from a negative exponential distribution with the mean of *Think Time* before generating another data request with the *query generator*. Since Gaussian distribution [7] is commonly used to model the client access pattern [11,20], we make the *query generator* randomly selects a data item from a Gaussian distribution, such that 68% of the generated requests fall within a particular set of data items with size, *Hot Spot Size*. After that, the request is directed to the *broadcast receiver*.

The *broadcast receiver* first reads a frame from the broadcast channel. The frame contains (1) a data item, (2) a number k, which specifies the number of data on the broadcast channel, (3) the bit position in the MFA vector corresponding to this particular data item, and (4) the broadcast version number. Whenever a new broadcast version number is detected, the MFA vector will be reset. If the data item received is not the desired one, the *broadcast receiver* continues to examine up to $k - 1$ succeeding frames on the broadcast channel. If the desired data item is encountered, the corresponding bit in the MFA vector is set. Otherwise, the request is directed to the *pull manager*.

The *pull manager* first checks if an on-demand channel is available. If not, it waits for a random period of time which is drawn from a uniform [0, *Wait Time*) distribution. This is repeated until a point-to-point connection is set up between the client and the server. After the connection is established, the *pull manager* sends a pull request with size *Pull Request Size* to the server. At the same time, the broadcast version number with size, *Version Size* and the MFA vector are also sent to the server. The server then replies with the requested data item to the *pull manager* through the same channel.

5.1.3. Modeling the client access pattern

As described in the client model, client access pattern is modeled by the Gaussian distribution [7]. In this section, we will define the hot spot of the data items using the Gaussian distribution.

As shown in figure 3, the shaded region is the hot spot of the data items. It is defined as follows. The mean of the Gaussian distribution equals to the hot spot center and the standard deviation σ is defined as

$$\sigma = \frac{\text{Hot Spot Size}}{2}. \tag{6}$$

Therefore, 68% of the generated requests fall within the hot spot of the data items [7].

To model the dynamic changes of the client access pattern, we have assumed that the hot spot center is shifted by one data item every s seconds. Rather than changing the center of the distribution explicitly, a shifting offset is used. This offset is increased by one every s seconds and it is modular by the total number of data items. Then the offset is added to each of the generated requested data item id to simulate the effect of shifting the hot spot center.

5.2. Parameter settings

The system parameters and control parameters used in the simulations are shown in tables 1 and 2, respectively. The control parameters marked by asterisks in table 2 are derived analytically from the system parameters listed in table 1. In the appendix, we give the derivation for four of them, namely, Sub-Optimal Point, Unstable Point, Maximum

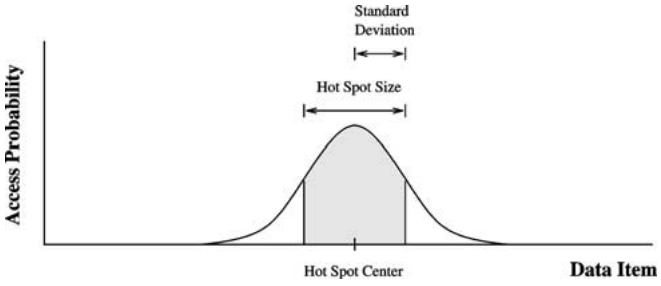


Figure 3. Client access pattern.

Table 1
System parameter settings.

Parameter	Value
Broadcast Bandwidth	80000 bps
On-Demand Bandwidth	8000 bps
Broadcast Channel	1
On-Demand Channel	10
Database Size	1000 data items
Hot Spot Size	50 data items
Data Item Size	8000 bits
Pull Request Size	128 bits
Version Size	16 bits
Think Time	10 s
Wait Time	2 s
α	0.5

Table 2
Control parameter settings.

Parameter	Value
Sub-Optimal Point*	8 requests/s
Sub-Optimal Upper Bound*	8.3 requests/s
Sub-Optimal Lower Bound*	7.7 requests/s
Unstable Point*	9.4 requests/s
Minimum Point*	0.5 request/s
Maximum Evaluation Cycle*	20 cycles
Maximum Evaluation Time*	120 s
Normal Step	0.2 request/s
Large Step	2.5 requests/s

Evaluation Cycle and Maximum Evaluation Time. Since the other three parameters, namely, Sub-Optimal Upper Bound, Sub-Optimal Lower Bound, and Minimum Point, can be derived in a similar manner, their derivations are not given.

The remaining two parameters, Normal Step and Large Step, are determined based on the following rationale. Since step movements are accumulative (see footnote 4), Normal Step should be small enough so that the Virtual Point (and hence the system) would not jolt around the optimal value. Therefore, we set it to 0.2 requests/s. The Large Step, on the contrary, should be large enough so that quick adaption can be achieved. Since Virtual Point ranges between the Sub-Optimal Point and the Minimum Point, which are 8 and 0.5 request/s, respectively, we use 2.5 requests/s as the Large Step value so that even in the extremely overloaded case, 3 steps are enough for the system to recover from being overloaded.

5.3. Notations used in the experiments

The following notations are used in the experiments:

- *IDEAL*: It represents the ideal case in which the access information of the data items are known completely.
- *MFA*: It represents the case when the MFA vector mechanism is used to collect the data access information.
- *Evaluation($c_{\max}$, $t_{\max}$)*: It shows the values of the *Maximum Evaluation Cycle* $c_{\max}$ and the *Maximum Evaluation Time* $t_{\max}$ used to determine the evaluation period in equation (5).

5.4. Experiment 1: Relationship between optimal system response time and on-demand request arrival rate

This experiment shows the relationship between the optimal system response time and on-demand request arrival rate. Since an explicit system response time equation cannot be obtained for our model, the optimal system response time is found by simulation only. Client access pattern is assumed to be unchanged. To find the optimal system response time, a fixed number of most frequently accessed data items are broadcast under a specific client population. The number of data items is tried exhaustively from 1 to n which is the number of data items in the database. The one which gives the minimum system response time is regarded as the optimal solution[5].

As shown in figure 4(a), for the IDEAL case, maintaining the on-demand request arrival rate at 8 requests/s can generate a nearly optimal system response time under different client populations. Even for the MFA case, as shown in figure 4(b), maintaining the on-demand request arrival rate at 8 requests/s can also give a performance close to the optimal solution.

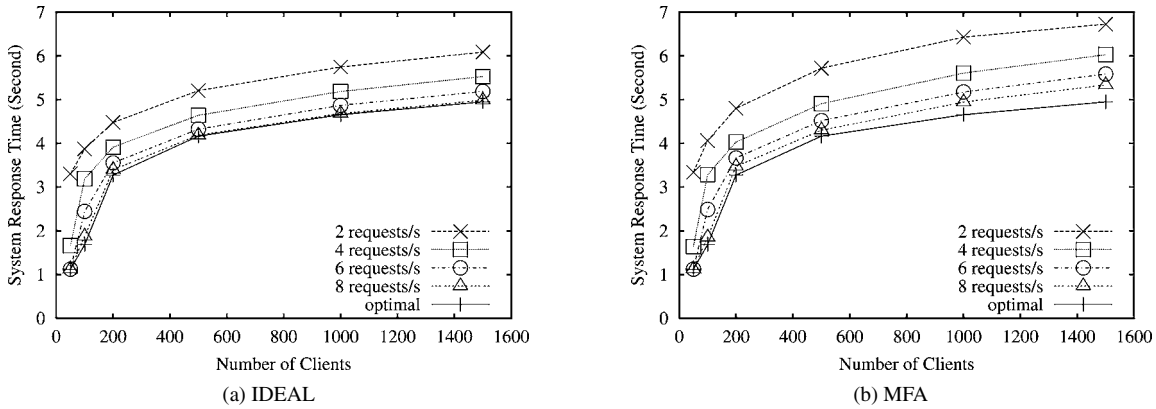
It should be noted that when the on-demand request arrival rate is maintained at 6 requests/s, the performance loss is not too large. It means that, the choice of the *Sub-Optimal Point* is not too critical to system performance. As long as the setting of the *Sub-Optimal Point* is far enough away from the *Unstable Point*, a reasonably good system performance can be achieved.

From figure 4(b), it is found that the gap between the 8 requests/s line and the optimal line becomes larger when the system workload is higher. It is because when the system workload is higher, more data items will be broadcast, so the fraction of MFA vectors received by the server is smaller[6]. Hence, the accuracy of the captured relative popularities of the data items is lower, so the system performance worsens.

Since the IDEAL case can give a nearly identical performance to the optimal solution, hereafter, the IDEAL case will be used as a comparison baseline for the MFA case.

[5] The optimal lines shown in both figures 4(a) and 4(b) are identical.

[6] If the requests of a client can all be answered by the broadcast data, the MFA vector would not be sent to the server.



(a) IDEAL (b) MFA

Figure 4. Different on-demand request arrival rates.

5.5. Experiment 2: Real-time adaptive performance of the system

To see the real-time adaptive performance of the system, we would like to know how it reacts to the changes in client population and client access pattern in a real-time environment. To achieve this, two dynamic environments are used to simulate these two changes separately.

5.5.1. Real-time change of client population

In this experiment, system performance under real-time change of client population is studied. The client access pattern is assumed to be unchanged and the system response time is sampled every 60 seconds in the experiment.

Figure 5 shows the real-time system performance under the change of client populations. Each line in the sub-figure shown in figure 5 represents a different number of clients in the server during the intermediate period of the simulation time.

From 0 to 1000 seconds, there are only 50 clients in the server. Then the number of clients increases linearly during the period 1000 to 2000 seconds from 50 clients to 100, 200, 500, 1000 and 2000 clients respectively for each line shown in each sub-figure in figure 5. After that, the number of clients remains unchanged until 5000 seconds. Finally, the number of clients decreases linearly back to 50 clients during the period 5000 to 6000 seconds. In fact, each line models a different rate change of client population[7].

It is found that the length of evaluation period, and hence, the choice of the *Maximum Evaluation Cycle* and the *Maximum Evaluation Time*, is very critical to the real-time system performance. As shown in figure 5(a), when an overall short evaluation period[8]is used in the IDEAL case, the transition of the system to the change of client population is the smoothest. It is because the system is evaluated most frequently, so appropriate adjustment can always be made before the on-demand channels approach the overloaded state. However, if a short evaluation period is used in the MFA case, as shown in figure 5(b), the system response time is significantly longer and more fluctuating than the IDEAL case. Since the evaluation period is too short, the relative popularities of the data items captured are not accurate enough. Therefore, some less popular data items are broadcast while some more popular data items are not broadcast, so the system performance is very poor.

On the other hand, if the *Maximum Evaluation Cycle* chosen is too large, and hence, the overall evaluation period is too long, as shown in figures 5(c) and (d), the evaluation period used during the system transition from 50 to 1000 clients and 50 to 2000 clients becomes too long. Therefore, the on-demand channels can get into the overloaded state easily before any adjustment can be made by the system. Nevertheless, once the overloading of the on-demand channels is detected, our adaptive data delivery algorithm can take the on-demand channels back to stable quickly.

20 cycles seem to be an appropriate choice for the *Maximum Evaluation Cycle* because both the stability and reactiveness of the system can be maintained. As shown in figures 5(e) and (f), with a proper setting of the evaluation period such that the overall evaluation period is just long enough for a fairly accurate capturing of the relative popularities of the data items, the performance of the MFA case can be very close to the IDEAL case. Since the relative popularities captured are not 100% accurate, so the performance of the MFA case is still slightly worse than the IDEAL case.

As shown in figures 5(g) and (h), when the *Maximum Evaluation Time*, and hence, the upper bound of the evaluation period, is increased, the system response times for the MFA case get even closer to the IDEAL case comparing to figures 5(e) and (f). However, the improvement is not very significant. It means that once the accuracy of the captured popularities of the data items are high already, further increasing the length of the evaluation period does not help to further improve the accuracy much.

5.5.2. Real-time change of client access pattern

In this experiment, system performance under real-time change of client access pattern is studied. The client population is fixed at 200 clients and the system response time[9] is sampled every 60 seconds in the experiment.

Figure 6 shows the real-time system performance under a dynamic changing client access pattern environment. Each line in the sub-figure shown in figure 6 represents a different rate change of client access pattern during the intermediate period of the simulation time.

Initially, the client access pattern changes every 60 seconds. From 1000 to 4000 seconds, the client access pattern changes every 1, 2, 5, 10 and 20 seconds, respectively, for each line shown in each sub-figure in figure 6. Hereafter, 1 s shift, 2 s shift and so on are used to represent these lines.

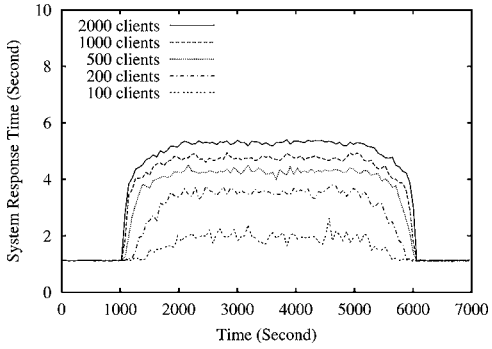
In an environment where the client access pattern is changing dynamically, the more frequent the evaluation of the system, the better the system performance. It is because the server can update the broadcast content more frequently to meet the needs of the clients. Therefore, using an overall short evaluation period, as shown in figures 6(a) and (b), would give the best system performance among the others.

It is interesting to note that when the broadcast content is updated frequently enough to meet the needs of the client, the performance of the MFA case, as shown in figure 6(b), is even better than the performance of the IDEAL case, as shown in figure 6(a). The reason is due to the use of an exponentially weighted computation in equation (3) for the data request arrival rates. When an exponentially weighted computation is

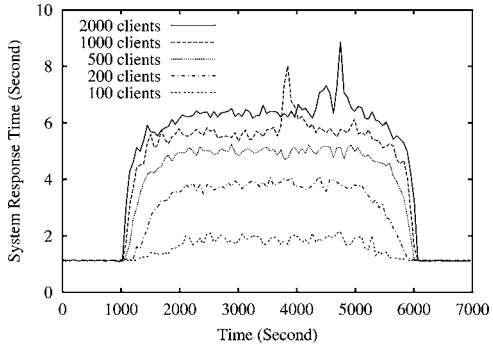
[7] The modeling for these changes is done by associating each client with a *Wake Up Time* and *Leave Time* so that each client can enter and leave the server accordingly.

[8] Since evaluation period is determined according to the length of broadcast cycle, so it can be long or short. Overall short evaluation period means the evaluation period used for different lengths of broadcast cycle is relatively short.

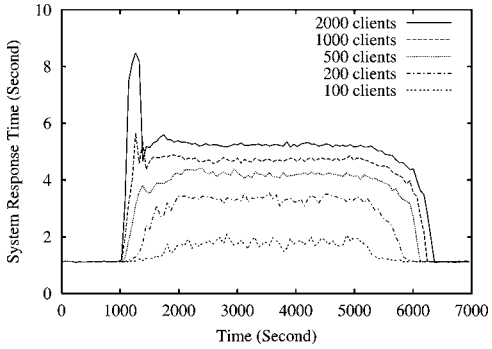
[9] Actually all clients record server's response time for each request issued, and the system response time is simply the averaged response time among all clients.



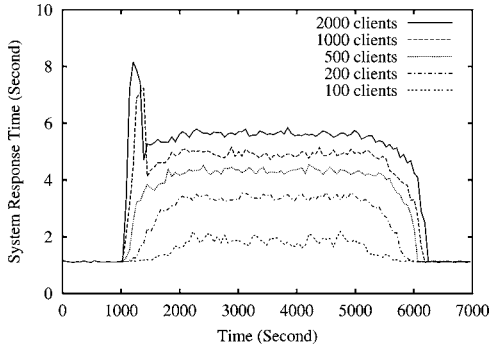
(a) IDEAl, Evaluation (5 cycles, 120 s)



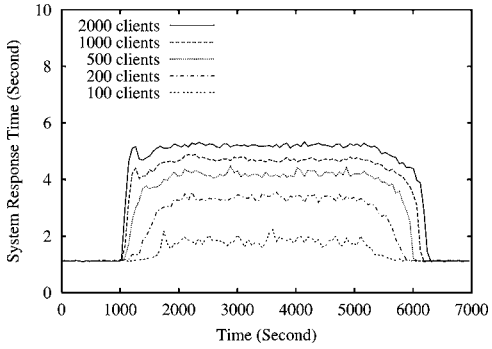
(b) MFA, Evaluation (5 cycles, 120 s)



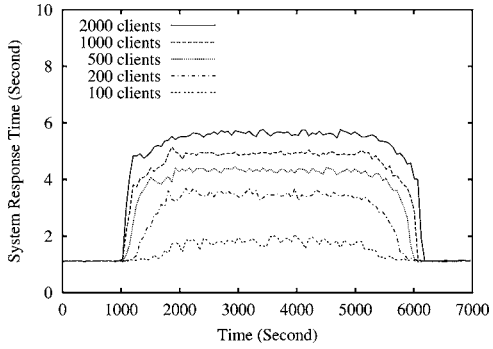
(c) IDEAl, Evaluation (40 cycles, 120 s)



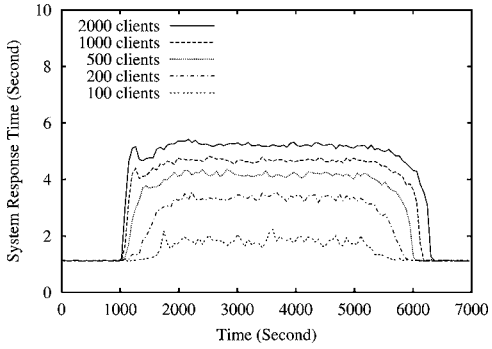
(d) MFA, Evaluation (40 cycles, 120 s)



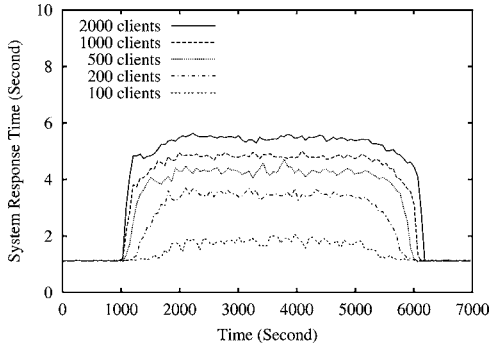
(e) IDEAl, Evaluation (20 cycles, 120 s)



(f) MFA, Evaluation (20 cycles, 120 s)

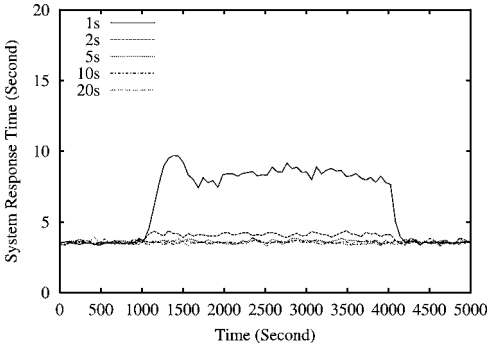


(g) IDEAl, Evaluation (20 cycles, 180 s)

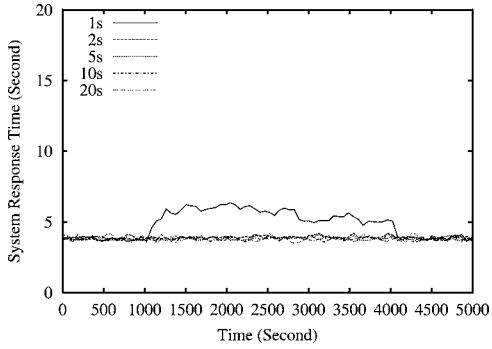


(h) MFA, Evaluation (20 cycles, 180 s)

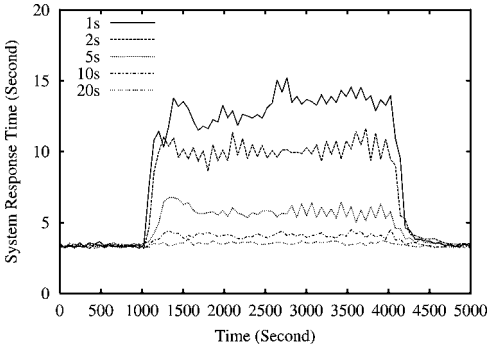
Figure 5. Real-time change of client population.



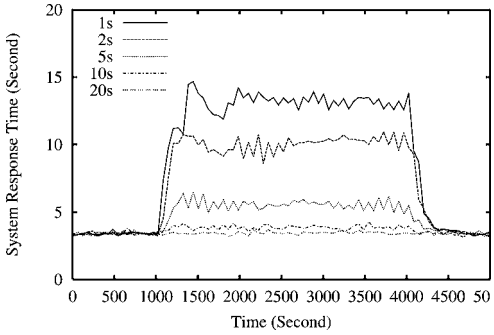
(a) IDEAl, Evaluation (5 cycles, 120 s)



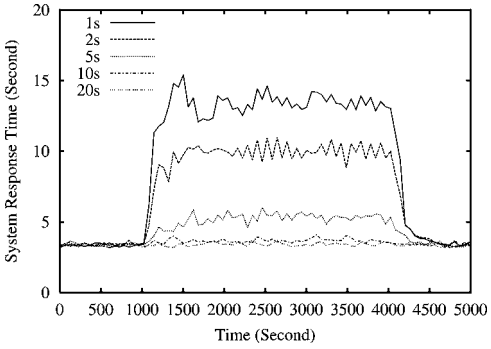
(b) MFA, Evaluation (5 cycles, 120 s)



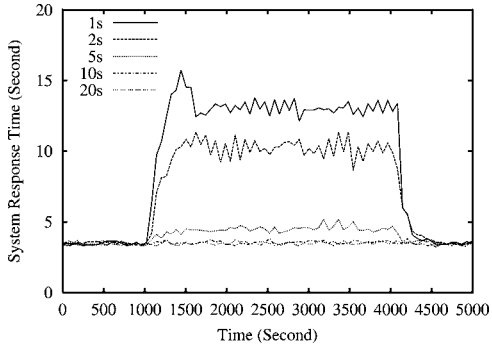
(c) IDEAl, Evaluation (40 cycles, 120 s)



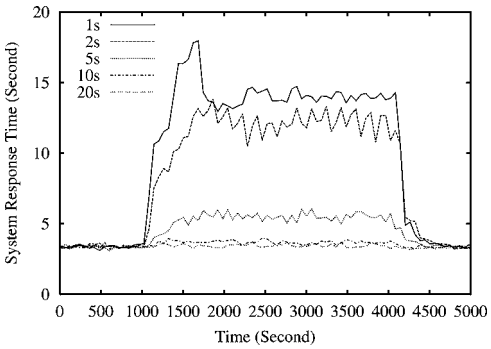
(d) MFA, Evaluation (40 cycles, 120 s)



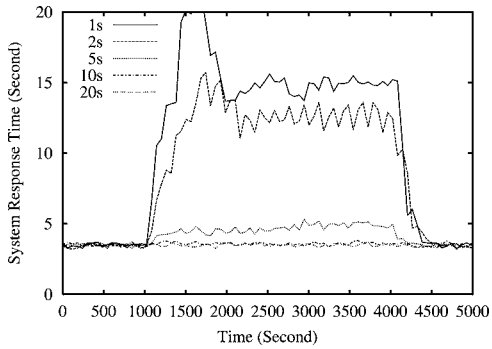
(e) IDEAl, Evaluation (20 cycles, 120 s)



(f) MFA, Evaluation (20 cycles, 120 s)



(g) IDEAl, Evaluation (20 cycles, 180 s)



(h) MFA, Evaluation (20 cycles, 180 s)

Figure 6. Real-time change of client access pattern.

used, even when a broadcast data item is no longer needed by the clients, if it had a large request arrival rate value in the previous system evaluation, it may still be broadcast. It is because the value is so large that it may take a few more system evaluations for this value to decrease to a certain level, such that it will not be selected by the server to broadcast. Since this data item may still be broadcast for some times which becomes a hindrance to the broadcast of some other useful data items, so the system performance will be affected. In fact, the smaller the weighting factor α used in equation (3), the more significant the problem. As a result, for a particular broadcast data item, the request arrival rate computed in the IDEAL case is always larger than the one computed in the MFA case. It is because the exact data access frequencies are used in the IDEAL case while only partial data access frequencies[10] are used in the MFA case to compute the request arrival rates of the data items. Therefore, the problem caused by the used of an exponentially weighted computation to calculate the request arrival rates in the IDEAL case is more serious than in the MFA case. Thus, the performance of the MFA case, as shown in figure 6(b), is much better than that of the IDEAL case, as shown in figure 6(a).

As shown in figures 6(e) and (f), when the *Maximum Evaluation Cycle* increases, and hence, the overall evaluation period increases, the performance of the system worsens significantly for 1 s shift and 2 s shift. Also, the MFA case does not show any advantages over the IDEAL case for these two shift cases. It is because the client access pattern is changing too fast compared to the frequency of system evaluation such that most of the data items on the broadcast channel originally interesting to the client become uninteresting to the clients after a short period of time. Therefore, the advantage of the MFA case over the IDEAL case disappears. Nevertheless, for the 5 s shift, the performance of the MFA case is still better than the IDEAL case. It is because the content of the broadcast channel can be updated frequently enough comparing to the client access pattern changing rate, so the ability of the system to select the most frequently access data item to broadcast becomes critical to system performance.

When the *Maximum Evaluation Cycle* is further increased, and hence, the overall evaluation period increases, as shown in figures 6(c) and (d), the performance for 5 s shift also becomes similar for both the IDEAL and MFA cases. It is because the evaluation period increased for the 5 s shift now becomes too long compared to the rate change of the client access pattern. However, 1 s shift and 2 s shift shown in figures 6(c) and (d) do not have much performance difference compared to figures 6(e) and (f), respectively. It is because the evaluation period for these two shifts are both bounded by the *Maximum Evaluation Time* but not the *Maximum Evaluation Cycle*.

When the *Maximum Evaluation Time* increases, and hence, the upper bound of the evaluation period increases, as shown in figures 6(g) and (h), the performance for 1 s shift and 2 s shift becomes the worst compared to figures 6(e) and (f), respectively, because the evaluation period for these two shifts are now longer. Thus, the frequency of system evaluation is even less, making it more difficult for the broadcast data to catch up with the changing access pattern of the clients. Therefore, the setting of an upper bound Maximum Evaluation Time is very important in a dynamic changing client access pattern environment.

In fact, data broadcasting can also be seen as a kind of caching which is stored in the air. Like any other caching techniques, the more the cached data items are used, the more the performance gain can be achieved by the use of caching. Therefore, if the client access pattern changes too fast, the effectiveness of using data broadcasting is lost. Thus, the use of data broadcasting may not be suitable when there is no common access pattern among clients or the client access pattern changes too fast.

5.6. Experiment 3: Effect on the weighting factor used for the computation of data request arrival rates

In the following experiments, we would like to study the effect of using different values of weighting factor α in equation (3) to compute the data request arrival rates. Specifically, we would like to examine how it affects system performance under different client populations and different rate changes of client access pattern with different evaluation period settings. Only the MFA case is considered. The case when $\alpha = 1$ is used as the baseline for comparisons. It represents the case where the computation of the data request arrival rates is not weighted and based only on the current request arrival rates.

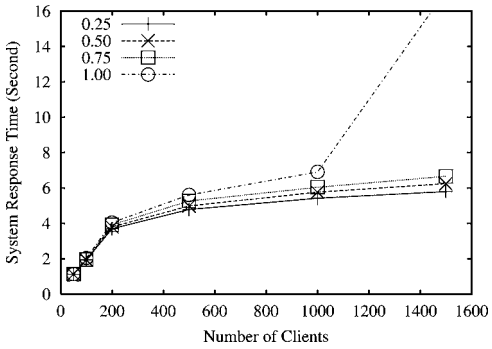
5.6.1. Different client populations

In this experiment, client access pattern is assumed to be unchanged and different values of weighting factor are used. The effect on system performance under different client populations are studied.

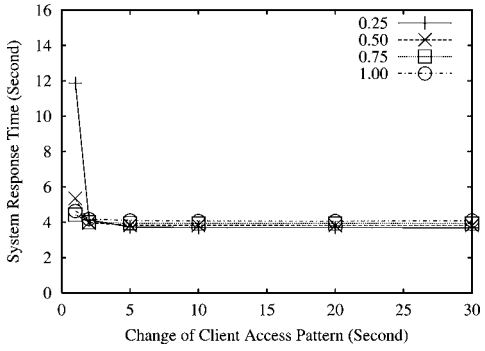
As shown in figure 7(a), when a short evaluation period is used, the use of an exponentially weighted computation for the data request arrival rates can help to improve system performance. Without it, when $\alpha = 1$, the system performance is poor, especially when the client population equals to 1500 clients. The is because the exponentially weighted computation can help to improve the accuracy of the captured relative popularities of data items. Figure 7(a) also shows that the smaller the weighting factor, the better it helps to improve the accuracy of the relative popularities of the data items captured, and thus, the better the system performance.

When the evaluation period is long enough, as shown in figures 7(c), (e) and (g), the ability of the exponentially weighted computation to further improve the accuracy of the captured relative popularities of the data items reduces significantly. This is because the data access information gathered during a long evaluation period can already preserve the relative popularities of the data items with high accuracy. Therefore, the need of using the exponentially weighted computation to improve the accuracy diminishes. Although perfor-

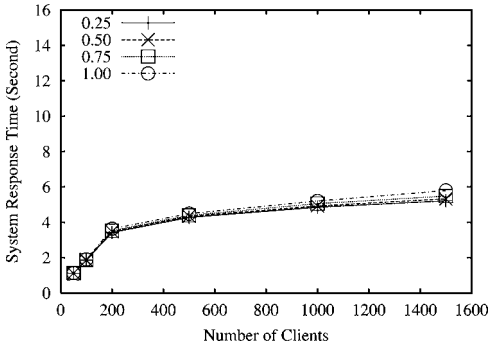
[10] Depending on the fraction of valid MFA vector received by the server, the access frequency collected for a broadcast data item may just represent a certain fraction of the actual value.



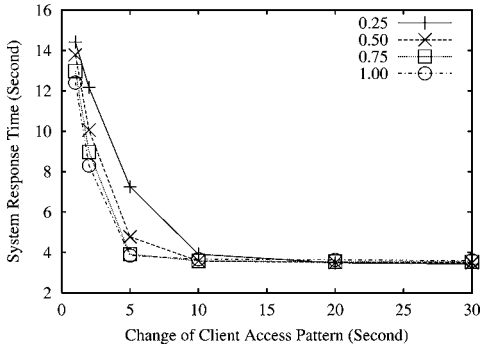
(a) MFA, Evaluation (5 cycles, 120 s)



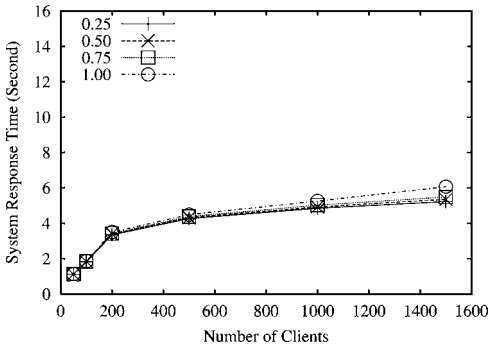
(b) MFA, Evaluation (5 cycles, 120 s)



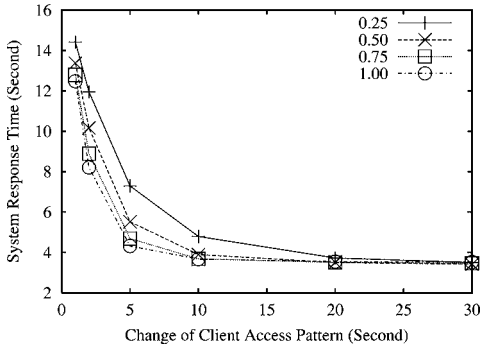
(c) MFA, Evaluation (20 cycles, 120 s)



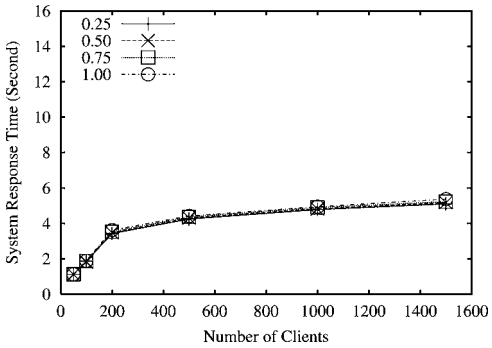
(d) MFA, Evaluation (20 cycles, 120 s)



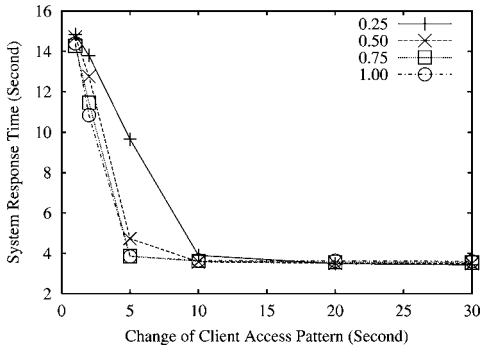
(e) MFA, Evaluation (40 cycles, 120 s)



(f) MFA, Evaluation (40 cycles, 120 s)



(g) MFA, Evaluation (20 cycles, 180 s)



(h) MFA, Evaluation (20 cycles, 180 s)

Figure 7. Using different values of weighting factor.

mance improvement is not significant, the smaller the weighting factor the better the system performance phenomenon still holds.

5.6.2. Different rate changes of client access pattern

In this experiment, client population is fixed at 200 clients and different values of weighting factor are used. The effect on system performance under different rate change of client access pattern is studied.

The results presented in this section helps to strengthen our argument used in section 5.5.2 to claim the disadvantage of using an exponentially weighted computation in a dynamically changing client access pattern environment.

As shown in figures 6(b), (d), (f) and (h), when the client access pattern is changing too fast, i.e. 1 s shift and 2 s shift, the use of an exponentially weighted computation always give poorer result than without using it. In addition, the smaller the weighting factor, the poorer the system performance. When a small evaluation period is used, as shown in figure 6(b), the negative effect of using the exponentially weighted computation disappears, except when a very small weighting factor is used and the client access pattern is changing very fast.

However, when the client access pattern is not changing too fast relative to the evaluation period settings, for example 5 s shift or longer in figure 6(b) and 20 s shift or longer in figure 6(d), using an exponentially weighted computation can give a better system performance than without using it, although the performance improvement is not significant. This suggests that relative popularities of the data items can be better captured by the use of an exponentially weighted computation, provided that, the client access pattern is not changing too fast relative to the evaluation period settings.

6. Conclusion

In this paper, a real-time adaptive data delivery algorithm is presented. By maintaining a certain level of on-demand request arrival rate, our algorithm can approximate the optimal system response time under different system workloads. Most importantly, it does not require the development of a system response time equation nor require to know the exact request arrival rate for each data item. These features are highly advantageous because the development of a system response time equation is not always possible and the requirement for the system to know the exact request arrival rate for each broadcast data item is impractical.

Given the relative popularities of the data items, on-demand channel utilization is maintained by adjusting the number of broadcast data items dynamically according to the changes in system workload and client access pattern. Apart from the algorithm to control the number of broadcast data items, it is found that the length of the evaluation period is also an important factor to the real-time performance of the system. With a proper choice of the control parameters and an appropriate setting of the evaluating period, our real-time adaptive data delivery algorithm can give a performance comparable to the ideal case in which the exact request arrival rates of the data items are known by the server.

Our algorithm can also be applied to other hybrid data delivery models with different system architectures. For instance, models which use a shared uplink channel to receive requests from the clients can maintain the uplink channel utilization with our algorithm to obtain a reasonably close to optimal system response time.

Since our adaptive data delivery algorithm is developed on an ad hoc basis, for future work, we would like to develop an algorithm with the use of control theory. For instance, we would like to see how fuzzy control theory [23] can be applied to maintain the on-demand request arrival rate.

Furthermore, indexing [4,14,17,19] should be incorporated into the data access mechanism in our model. The use of indexing can help to reduce the power consumption of the clients because they do not have to examine every data item on the broadcast channel. Also, clients can turn to the on-demand channels directly once they determine from the index that the requested data item will not appear on the broadcast channel, thus, system performance can also be improved.

Once indexing is used to assist the retrieval of data items on the broadcast channel, broadcast scheduling [8,9,21,22] or broadcast disks [1] can be used to improve the access time of the data on the broadcast channel.

Finally, we would like to study the effect of allowing each client to have more than one data request at a time, and those requests will be buffered in the client's local queue. Once the on-demand point-to-point connection is set up, all the buffered requests will be sent to the server. Depending on the system workload, the server may reply only a few of them. Those unanswered requests will be answered by the broadcast channel or the on-demand channels some time later according to a more dynamic data access mechanism.

Acknowledgements

Supported by Research Grants Council, Hong Kong SAR under grant HKUST6079/01E.

Appendix A. Derivation of sub-optimal upper bound and unstable point

In order to derive the two threshold control parameters in our system, we use the $M/M/m$ model to represent the service model [5]. The client generates requests with a negative exponential distribution. There are $m = 10$ servers (channels) and an infinite number of clients in our system.[11] This model is known as the *Erlang Delay System*. We will focus on the *average waiting time*, W, in terms of ρ. Specifically, we focus on its slope, i.e., W's change rate. We define the Sub-Optimal Upper Bound Point to be a point where $\partial W/\partial\lambda$ equals to a threshold K_0. Similarly, the Unstable Point represents a

[11] In our experiment, the number of clients N is at least 200. Since $N \gg m$, the infinite client source model is adopted.

point where $\partial W/\partial\lambda$ equals to another threshold K_1. The two thresholds are the system's *QoS* parameters which can be decided based on how stable we want the system to be in face of varying workloads. In our experiment, we set K_0 in such a way that when the request rate increases by 2 requests/s the response delay increases by 1 second. We set K_1 to be 10 times larger than K_0, i.e., $K_0 = 0.5$, $K_1 = 5$.

The *average waiting time*, W, is equivalent to the following [5]:

$$W = \frac{p_m}{m\mu(1-\rho)^2}. \tag{A.1}$$

In addition, we can derive the following [5]:

$$p_m = \frac{m^m\rho^m}{m!}p_0, \tag{A.2}$$

$$p_0 = \left[\sum_{n=0}^{m-1}\frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!(1-\rho)}\right]^{-1}. \tag{A.3}$$

We then rewrite equation (A.1) to make ρ the only independent variable so that we can calculate its derivative. However, since equation (A.3) is very complicated, we have to simplify it. Since we know ρ should be close to 1 ($\rho < 1$), the last factor of the summation in equation (A.3) is much larger than the other factors.[12] Thus, we replace the summation with the last factor to simplify equation (A.3):

$$p_0 = \left[\frac{(m\rho)^m}{m!(1-\rho)}\right]^{-1} = \frac{m!(1-\rho)}{m^m\rho^m}. \tag{A.4}$$

Plugging equation (A.4) into equation (A.2), we have:

$$p_m = \frac{m^m\rho^m}{m!} \times \frac{m!(1-\rho)}{m^m\rho^m} = 1-\rho. \tag{A.5}$$

Plugging equation (A.5) into equation (A.1), we have:

$$W = \frac{1-\rho}{m\mu(1-\rho)^2} = \frac{1}{m\mu(1-\rho)}. \tag{A.6}$$

Therefore, we have:

$$\frac{\partial W}{\partial\lambda} = \frac{\partial W}{\partial\rho}\frac{\partial\rho}{\partial\lambda} = \frac{1}{m\mu(1-\rho)^2}\frac{\partial(\lambda/m\mu)}{\partial\lambda} = \frac{1}{(m\mu)^2(1-\rho)^2}. \tag{A.7}$$

We then derive ρ_0 for the Sub-Optimal Upper Bound according to the following equation:

$$\frac{1}{(m\mu)^2(1-\rho)^2} = K. \tag{A.8}$$

Plugging $m = 10$, $\mu = 0.984$,[13] $K_0 = 0.5$ into equation (A.8), we get $\rho_0 = 0.85$. Therefore,

$$\lambda_{\textit{Sub-Optimal Upper Bound}} = \rho_0 m\mu = 8.3 \text{ request/s}.$$

[12] This can be verified to be valid using the ρ values derived later.

[13] $\mu = \frac{8000}{8000+128} = 0.984$ request/s.

Similarly, plugging $m = 10$, $\mu = 0.984$, $K_1 = 5$ into equation (A.8), we get $\rho_1 = 0.955$. Therefore, $\lambda_{\textit{Unstable Point}} = \rho_1 m\mu = 9.4$ request/s.

Appendix B. Maximum evaluation time

As we know, the maximum evaluation time (MET) is an important tuning parameter for the performance of the proposed scheme. In order to adapt to the changing workload, MET should be set to the smallest possible value. However, an MET value that is too small will lead to inadequate statistics for calculating the relative popularity of the data items,[14] as we will see in the experiments. Therefore, keeping MET just long enough to have adequate access statistics for the current broadcast version is our main criteria for obtaining an appropriate value for MET.

The basic idea is the following: we should set MET long enough that even for a very unpopular item, at least one record of accessing the item can be sent to the server among all N clients during the MET period. In this way, the relative popularity of almost all items can be recorded. The following equation helps us to derive t_{MET}:

$$\lambda_{d_i} \cdot t_{\text{MET}} = p^* \tag{B.1}$$

where λ_{d_i} is the request arrival rate for a very unpopular item i, p^* is the required access probability so that among N clients, one record of access can likely be obtained and sent to the server. In the following subsections, we will derive p^* and λ_{d_i}, respectively.

B.1. Derivation of p^*

We restate the problem according to the probability theory: N clients are independently performing the same trial (with positive probability p^*), i.e., Bernoulli trials. What's the minimum value of p^* such that with a specified confidence (probability c), positive results will occur at least once during the N trials?

Let f denote the proportion of positive results, i.e., $f = \#\textit{ of positive results}/N$. We would like $f \geqslant 1/N$. It is the same as

$$\frac{p^* - f}{\sqrt{p^*(1-p^*)/N}} \leqslant \frac{p^* - 1/N}{\sqrt{p^*(1-p^*)/N}}.$$

Since we know from probability theory that $(p^* - f)/\sqrt{p^*(1-p^*)/N}$ satisfies Gaussian distribution with $\mu = 0$, $\sigma = 1$. Therefore, in order for this inequation holds with the probability of confidence c (we set $c = 0.95$ in the experiment), the following equation must satisfy:

$$\frac{p^* - 1/N}{\sqrt{p^*(1-p^*)/N}} = -z \tag{B.2}$$

where $-z = -1.65$ for $c = 0.95$.

Having solved this equation, we have

[14] According to the MFA mechanism, a MFA vector will be discarded if it doesn't correspond to the current broadcast version.

$$p^* = \frac{z^2 + 2 \pm \sqrt{z^4 + 4z^2 - 4z^2/N}}{2(N + z^2)}.$$

For $z = 1.65$, $N = 200$,[15] we have $p^* = 1.096 \times 10^{-3}$.

B.2. Derivation of λ_{d_i}

In this subsection, we define the threshold for the request arrival rates of the objects, above which each of the objects is expected to have at least one record of access among all N clients to be sent to the server during the MET period. Since the objects access pattern is also modeled by a Gaussian distribution, we similarly set a confidence c representing the proportion of items that exceeds the threshold. For high accuracy, we set $c = 0.999$ here, resulting in $z \approx 3$. If we put $x = \mu + 3\sigma$, $\sigma = 25$ into the Gaussian probability density function (PDF): $\frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$, we get $p_{d_i} = 1.773 \times 10^{-4}$.

Since $\lambda_{d_i} = p_{d_i} \times \lambda$, we approximate the request fulfilling time with the mean Think Time, we have $\lambda = \frac{1}{10+10} = \frac{1}{20}$. Therefore, $\lambda_{d_i} = 1.773 \times 10^{-4} \times \frac{1}{20} = 8.87 \times 10^{-6}$ requests/sec.

B.3. Derivation of MET

Finally,

$$t_{\mathrm{MET}} = \frac{p^*}{\lambda_{d_i}} = \frac{1.096 \times 10^{-3}}{8.87 \times 10^{-6}} = 123.6 \text{ sec.}$$

Therefore, we set $t_{\mathrm{MET}} = 120$ sec in our experiments.

Appendix C. Maximum evaluation cycle

We derive Maximum Evaluation Cycle's counterpart – Minimum Cycle Length (MCL) first. A Maximum Evaluation Cycle (MEC) will be used instead of MET to bound the evaluation time when the broadcast cycle length is below the lower bound of MCL. This scenario occurs when all the broadcast items (even the coolest one) have such a large access probability p^* that during the MET period, each client has accessed more than once the unpopular items. This scenario should be prevented since it will make the MFA mechanism less effective.[16] This is the intrinsic reason why we should set a Minimum Cycle Length (MCL) in our scheme.

We still use equation (B.1) to derive t_{MCL}. Again, to achieve a high accuracy, we set our confidence to 0.95, i.e., $p^* = 0.05$. Here λ_{d_i} is the request arrival rate for the most unpopular item in the broadcast channel. As we know, during an t_{MCL} period, $10t_{\mathrm{MCL}}$ different items will be broadcast,[17] which are items with x-coordinate in the range of $[\mu - 5t_{\mathrm{MCL}}, \mu + 5t_{\mathrm{MCL}}]$ in figure 3. Meanwhile, the average request fulfilling time for all broadcast items is $t_{\mathrm{MCL}}/2$, resulting in the total request arrival rate, i.e., λ, equal to $\frac{1}{10+t_{\mathrm{MCL}}/2}$.

In a way similar to the derivation of equation (B.1), we derive the following equation:

$$p_{d_i} \times \lambda \times t_{\mathrm{MET}} = p^*. \tag{C.1}$$

We put $x = \mu + 5t_{\mathrm{MCL}}$, $\sigma = 25$ into equation (C.1) and obtain:

$$\frac{1}{25\sqrt{2\pi}} e^{-25t_{\mathrm{MCL}}^2/(2\times 25^2)} \times \frac{1}{10 + t_{\mathrm{MCL}}/2} \times 120 = 0.05. \tag{C.2}$$

After solving this equation with approximation, we obtain $t_{\mathrm{MCL}} \approx 7$. Therefore, $\mathrm{MEC} = \frac{t_{\mathrm{MET}}}{t_{\mathrm{MCL}}} = \frac{120}{7} \approx 20$.

[15] In the experiment, the minimum number of clients is 200. Since MET is fixed for all numbers of clients, we must guarantee that it works for each setting. Since $N = 200$ is the most strict case to satisfy, we set $N = 200$ here.

[16] The MFA mechanism has only 1 bit for each item. It can not tell the difference between accessing an item once or more than once.

[17] According to the bandwidth and object size set in the experiment, *# of objects* $= \frac{80{,}000 * t_{\mathrm{MCL}}}{8000} = 10t_{\mathrm{MCL}}$.

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*, San Jose, CA (May 1995) pp. 199–210.

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

[3] 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*, Tuscon, AZ (May 1997) pp. 183–194.

[4] M.-S. Chen, P.S. Yu and K.-L. Wu, Indexed sequential data broadcasting in wireless mobile computing, in: *The 17th International Conference on Distributed Computing Systems (ICDCS'97)*, Baltimore, MD (27–30 May 1997).

[5] R.B. Cooper, *Introduction to Queueing Theory*, 3rd edn (CEE Press Books, Washington, DC, 1990).

[6] A. Datta, A. Celik, J. Kim, D. VanderMeer and V. Kumar, Adaptive broadcast protocols to support efficient and energy conserving retrieval from databases in mobile computing environments, in: *Proceedings of IEEE International Conference on Data Engineering*, Birmingham, England (April 1997).

[7] S. Goldberg, *Probability: An Introduction* (Dover, New York, 1986).

[8] S. Hameed and N.H. Vaidya, Efficient algorithms for scheduling single and multiple channel data broadcast. Technical Report 97-002, Computer Science, Texas A&M University (February 1997).

[9] S. Hameed and N.H. Vaidya, Log-time algorithms for scheduling single and multiple channel data broadcast, in: *The Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97)* (1997).

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

[11] Q.L. Hu, D.L. Lee and W.-C. Lee, Performance evaluation of a wireless hierarchical data dissemination system, in: *Proceedings of the Fifth Annual International Conference on Mobie Computing and Networking*, Seattle, USA (August 1999).

[12] T. Imielinski and B.R. Badrinath, Mobile wireless computing: Challenges in data management, Communications of ACM 37(10) (1994) 18–28.

[13] T. Imielinski and S. Viswanathan, Adaptive wireless information systems, in: *Proceedings of SIGDBS Conference*, Tokyo, Japan (October 1994).

[14] T. Imielinski, S. Viswanathan and B.R. Badrinath, Data on the air – organization and access, IEEE Transactions on Knowledge and Data Engineering 9(3) (1997) 353–372.

[15] R. Jain, *The Art of Computer Systems Performance Analysis* (Wiley, New York, 1991).
[16] W.-C. Lee, Q.L. Hu and D.L. Lee, A study of channel allocation methods for data dissemination in mobile computing environments, Mobile Networks and Applications (MONET) 4(2) (1999) 117–129.
[17] W.-C. Lee and D.L. Lee, Using signature techniques for information filtering in wireless and mobile environments, Journal on Distributed and Parallel Databases 4(3), Special Issue on Database and Mobile Computing (1996) 205–227.
[18] H. Schwetman, *Csim User's Guide (Version 17)* (MCC Corporation, 1992).
[19] N. Shivakumar and S. Venkatasubramanian, Efficient indexing for broadcast based wireless systems. Mobile Networks and Applications (MONET) 1(4) (1996) 433–446.
[20] K. Stathatos, N. Roussopoulos and J.S. Baras, Adaptive data broadcast in hybrid networks, in: *Proceedings of 23rd VLDB Conference*, Athens, Greece (August 1997) pp. 326–335.
[21] C. Su and L. Tassiulas, Broadcast scheduling for information distribution, in: *Proceedings of IEEE INFOCOM'97*, Kobe, Japan (April 1997) pp. 109–117.
[22] N.H. Vaidya and S. Hameed, Scheduling data broadcast in asymmetric communication environments. Technical Report 96-022, Computer Science, Texas A&M University (November 1996).
[23] L.X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1994).

Chi-Wai Lin received his B.Eng. degree in computer engineering and M.Phil. degree in computer science from the Hong Kong University of Science and Technology. His area of interests include data communication, operating system, and computer and network security. He is a Certified Information Systems Security Professional.
E-mail: lcw@cs.ust.hk

Haibo Hu is now a Ph.D. candidate in the Department of Computer Science, Hong Kong University of Science and Technology. He received his B.Eng. degree in computer science and engineering from Shanghai Jiaotong University, China. His research interests include wireless data dissemination techniques, location-based services, and mobile spatial databases.
E-mail: haibo@cs.ust.hk

Dik Lun Lee received the M.S. and Ph.D. degrees in computer science from the University of Toronto in 1981 and 1985, respectively. He is a Professor in the Department of Computer Science at the Hong Kong University of Science and Technology, and was an Associate Professor in the Department of Computer and Information Science at the Ohio State University, Columbus, OH, USA. He has served as a guest editor for several special issues on database-related topics, and as a program committee member and chair for numerous international conferences. He was the founding conference chair for the International Conference on Mobile Data Management. His research interests include document retrieval and management, discovery, management and integration of information resources on Internet, and mobile and pervasive computing. He was the Chairman of the ACM Hong Kong Chapter.
E-mail: dlee@cs.ust.hk