Wireless Networks 9, 239–247, 2003

A Cost-Efficient Scheduling Algorithm of On-Demand Broadcasts

WEIWEI SUN *, WEIBIN SHI and BOLE SHI
Department of Computing and Information Technology, Fudan University, PR China

YIJUN YU
Department of Electrical Engineering, Gent University, Belgium

Abstract. In mobile wireless systems data on air can be accessed by a large number of mobile users. Many of these applications including wireless internets and traffic information systems are pull-based, that is, they respond to on-demand user requests. In this paper, we study the scheduling problems of on-demand broadcast environments. Traditionally, the response time of the requests has been used as a performance measure. In this paper we consider the performance as the average cost of request composed of three kinds of costs – access time cost, tuning time cost, and cost of handling failure request. Our main contribution is a self-adaptive scheduling algorithm named LDFC, which computes the delay cost of data item as the priority of broadcast. It costs less compared with some previous algorithms in this context, and shows good adaptability as well even in pure push-based broadcasts.

Keywords: scheduling algorithm, cost modeling, data broadcast, wireless network, mobile computing

1. Introduction

In a client/server architecture with fixed networks, clients would send requests when they want to retrieve data from the server. Then the server will respond to the requests and send data to clients. Compared with fixed networks, wireless networks have low bandwidth and low communication quality [7,10]. To support numerous mobile users to access data in server concurrently, a new method of data-transmission is put forward, that is, the server broadcasts data on air and clients acquire data that way, which is called *data broadcasting*.

Data broadcast technology has many applications in the field of public information dissemination, such as stock market quotation or traffic and landmark information. One important issue in broadcast technology is to determine an optimal broadcast sequence according to the access probability distribution of mobile users, i.e. the data broadcast scheduling. To evaluate the effectiveness of one broadcast scheduling strategy, we need to consider two basic aspects:

(1) Access Time (shortened as *AT*). It indicates the time elapsed between the query submission and receipt of the response. *AT* determines the response time of query made by mobile users. We need to concentrate on the arrangement of frequency and location of data items in one broadcast period, so as to make the average *AT* smaller, according to various access probabilities of data items. The study on this issue includes [1–3,5,9,14,17], etc.

(2) Tuning Time (shortened as *TT*). It indicates the total time that mobile users spend actively listening to the channel in a complete access period. *TT* determines the power consumption of mobile users because they could slip into doze (stand by) mode when they are not actively listening on the channel. As most of mobile users depend on limited battery supply, the reduction of *TT* would also be an important issue in data broadcast technology. A widespread method is to insert index segments into broadcast period in order to reduce *TT*. The study on this issue includes [11,13,16], etc.

In on-demand broadcasts, we cannot obtain the access profiles of mobile users, that is to say, their access pattern would have some unpredictable changes. Thus, we need a kind of new scheduling algorithm, to determine the contents and organization of data broadcast on the basis of circumstances of recent access and scheduling.

The study of on-demand broadcast scheduling problem includes [4,8], etc. In this broadcast environment, mobile users communicate with the server via wireless channels. These channels include an uplink channel and a downlink channel. Mobile users use this uplink channel to send data access request, and the contents of broadcast will reach the mobile users through downlink channel. First, mobile users make the access request; second, the server considers all pending requests to decide the contents of next broadcast. One core issue is to determine the priorities of data items to be broadcasted, that is, which data items should be broadcasted in the next period. A FCFS (First-Come-First-Served) scheduling algorithm is put forward in [5], which sequences data items according to their requested time. Because of its time sequencing principle, any access request would get responded after waiting for a finite period. There does not exist any case of endless waiting. But it has the deficiency of low average performance, because it considers only the requested time, and does not take into account the difference of access frequency of various data items. MRF (Most-Request-First) scheduling algorithm will broadcast those data items with the

* Corresponding author.
E-mail: wwsun@fudan.edu.cn

largest number of request. As there are most-frequent data items in every broadcast, every broadcast will have the highest response ratio (*number of requests responded/number of total requests*), and we could get much lower *AT*. But it has its own shortcoming: if some data items have few requests, they will always line up behind several most-frequently-requested items, so that the request on these data items could always be unsatisfied and end in endless waiting. In [6], LWF (Long-Wait-First) algorithm is suggested, which chooses the data item that has the largest waiting time (the sum of the total time that all pending requests for that item have been waiting for) to broadcast. It considers both the number of requests and the waiting time, so as to reduce the occurrence of endless waiting. In [4], LTSF (Longest-Total-Stretch-First) algorithm is put forward, which considers the factor of variable-size data items. In [8], a set of self-adaptive broadcast protocols – CBS/VBS protocols (including server broadcast protocol and client receipt protocol) is proposed, and the idea is raised of dynamic adjusting in priority computing formula.

But all these papers mentioned above do not take into account handling of requests waiting for quite a long time. They only consider some measures to reduce the probability of waiting for quite a long time. Being unable to deal with those requests that do not get responded for quite a long time, i.e., the permission of endless waiting, will lead to serious problems. For example, the server would not receive the access requests because of transmission errors, in this case mobile users (the request senders) will wait for impossible responses; responding to an access request sent a long time ago would also lead to ineffectiveness of the response, because the mobile user who sent this request could probably have left the area covered by broadcast, or it would not listen to the channel for the reason of saving power.

Therefore, we should set up a Response Time Limit (*RTL*) for every access request. The mobile user sends one request and starts to listen to the contents of broadcast. If it does not get responded within the *RTL*, this request would be identified as a failure and the mobile user would not continue to listen to. Similarly, after the broadcast server received the request sent by the mobile user, if it could not add corresponding data item to broadcast contents within the *RTL*, it would delete the request from the request sequence.

Besides, in the determination of which item should be added to broadcast, the priority computing formula seems unable to explain the exact reason why those data items with low priority should be delayed. And the significance of those cost computing models is vague.

In this paper, we put forward a self-adaptive scheduling algorithm of on-demand broadcast – LDCF (Largest-Delay-Cost-First). It computes the delay cost for every data item and uses it as the priority to schedule the data items, taking into account three kinds of costs – *AT*, *TT* and request failure. The parameters of delay cost computing formula will be adjusted automatically according to recent scheduling circumstances.

The rest of the paper is organized as follows. Section 2 shows the on-demand broadcast model and defines the problem of broadcast scheduling. We also make some basic assumptions here. Section 3 shows the delay cost computing formula of data items, which indicates the increased cost if every data item would not be broadcasted in the following period, including access time cost, tuning time cost and request failure cost. On the basis of this formula, we describe LDCF scheduling algorithm. We describe the simulation experiments and discuss their results in section 4. We make some conclusions in section 5.

2. Problem definition and preliminaries

A typical on-demand broadcast system could be shown as figure 1 [4].

The relationship between radio transmitter (base station) and mobile users could be viewed as server and clients. Mobile users are clients, and radio transmitter is the server. To the convenience of our study, we make some restrictions on the broadcast environment. Our basic assumptions are as follows.

Mobile users communicate with the server via wireless information channels. These channels include an uplink channel and a downlink channel. Mobile users use this uplink channel to send data access request, and the contents of broadcast will arrive at mobile users through downlink channel. After the broadcast server receives an access request, it will respond to this request within a predetermined response time limit, and add the requested data item in broadcasting contents; otherwise this request would be regarded as a failure. (The disposal of a failed request could be in two ways: either the server would do nothing, waiting the mobile user to send request again if the mobile user still want to access the data item; or it could create a separate wireless link to send data item to the mobile user.)

Broadcast server does not know the probability distribution of the access of various data items by mobile users. Therefore, the server could determine suitable broadcast scheduling only after it received those requests. (In that case if we say the server does not know the access pattern of mobile users, we do not mean that the access by mobile users has not any regular patterns; actually, the access by various mobile users do have some patterns.)

The minimal unit of broadcast is data item, and all data items are of identical size.

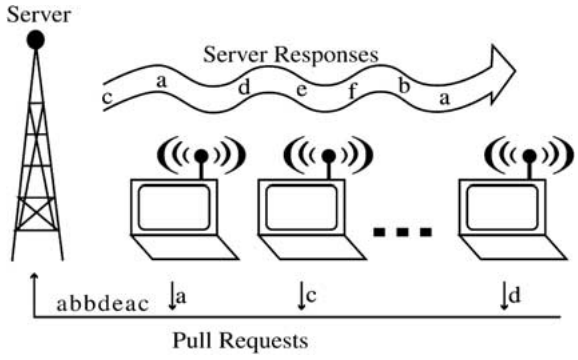


Figure 1. A typical on-demand broadcast system.

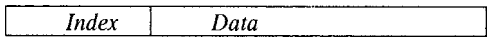


Figure 2. The structure of data broadcast.

Data broadcast uses a kind of constant-period method, that is to say, no matter what changes have taken place in the content of broadcast, the size of every broadcast is fixed.

Mobile users access one data item in each request, and any two accesses are independent.

The following are some definitions and notions that we may use in our further discussion.

- *Unit Time*: suppose the broadcast time of one data item is 1.
- *Data*: the number of data items in one broadcast period.
- *Index*: the index length in one broadcast period.
- *BP*: Broadcast Period. $BP = Data + Index$. The structure of data broadcast: see figure 2. The former is index segment *Index*, and the latter is data segment *Data.* The size of *Index* and *Data* is fixed.
- D_i: data item, $i = 1, \ldots, M$. M indicates the total number of all data items.
- $Q\langle D, T_{\text{req}}\rangle$: indicates one access request. D is the data item that Q requests to be broadcasted; T_{req} is the time when that request is sent.
- *RTL*: Response Time Limit. It indicates the longest time elapsed between the time when the mobile user sends an access request of data item and the time when the server responds to that request. If the server could not add the requested data item to broadcast contents within this time limit, we should say that this request failed. The server could create a separate wireless link and send data item to mobile user. If one mobile user sends a request at T_0, and at T_1 ($T_1 \leqslant T_0 + RTL$) it finds in broadcast index that the requested data item would appear at T_2 ($T_2 > T_0 + RTL$). In this case, we consider that the request gets valid response.
- C_{AT}: access time cost for mobile users to obtain data item (on the basis of unit time). If one mobile user waits for 100 unit time to obtain his requested data item, then total access time cost would be 100 C_{AT}.
- C_{TT}: tuning time cost for mobile users to search for the location of one data item in the index segment. If one mobile user waits for 10 broadcast periods to obtain one data item, and he searches the index for 10 times, then total tuning time cost would be 10 C_{TT}.

In a pure push-based data dissemination scheduling, we have two main performance metrics: access time and tuning time. In on-demand broadcast scheduling, we should consider not only AT and TT, but the cost of handling failed requests as well, because we have introduced the notion of request failure.

- C_{F}: Cost of handling a failed request. If the server could not respond to the access request within a determined response time *RTL*, we should use C_{F} as the cost of creating a separate wireless link between server and mobile user to obtain data item.

3. LDCF self-adaptive broadcast scheduling algorithm

3.1. Delay cost computing model

The key of LDCF (Largest-Delay-Cost-First) scheduling algorithm is its Delay-Cost computing model. We can compute the cost of every request delayed one broadcast period, according to such parameters as the length of broadcast period, tuning time cost for mobile users to search for needed data item in the index section of broadcast, failure probability of access request, and the cost of handling failed request, etc. This cost is composed of three aspects: access time, tuning time, and request failure.

Some description of several notions is given below. Then we could illustrate the formulas of Delay Cost in our discussion.

- PF_D^T: the popularity factor of data item D at time T, which indicates there are PF_D^T number of mobile users requesting to access data item D. The initial value of PF_D^T is zero; every time when a new request for data item D arrives, PF_D^T will increase by 1; when one request is not satisfied within one *RTL*, PF_D^T will decrease by 1; if the data item D appears in the broadcast line, PF_D^T will be set as zero again.
- SF_Q^T: the safety factor, expressed by remaining broadcast periods, which indicates there are SF_Q^T number of opportunities (excluding the next one) to satisfy request Q by broadcast at time T. The formula of SF_Q^T is

$$SF_Q^T = \left\lfloor \frac{T_{\text{req}} + RTL - T}{BP} \right\rfloor.$$

- T_{req} stands for the sending time of request Q. One thing that we need to mention is that T stands for the time next broadcast begins. If the safety factor is zero, it means that if server does not broadcast the data item Q needs in the next period, then request Q fails. We name it as safety factor, because we want to use it to express the "distance" of request Q to the failure. Obviously, in order to minimize failed requests, the server should respond to those requests with small SF_Q^T .
- $ReqNum(SF)$. It indicates the total number of pending requests whose safety factor equals to SF.
- $RemReqNum(SF)$. It indicates the total number of remained requests that could not be satisfied in the next broadcast period, whose safety factor equals to SF.
- $BroReqNum(SF)$. It indicates the total number of requests that could be satisfied in the next broadcast period, whose safety factor equals to SF. Obviously,

$$ReqNum(SF) = RemReqNum(SF) + BroReqNum(SF).$$

- $R_{\mathrm{R}}(SF)$. It indicates the ratio of requests that could not be satisfied in the next broadcast period, whose safety factor equals to SF:

$$R_{\mathrm{R}}(SF) = \frac{RemReqNum(SF)}{ReqNum(SF)}.$$

- $R_{\mathrm{F}}(SF)$. It indicates the failure probability of the requests whose safety factor equals to SF, if they could not be satisfied in the next broadcast period.
  Apparently, if one request $Q\langle T, D\rangle$ with $SF = 0$ could not be satisfied in the next period, then Q will fail, i.e., $R_{\mathrm{F}}(0) = 1$.
  When $SF > 0$, for request $Q\langle T, D\rangle$, if D could not be satisfied in the next broadcast period, the SF of Q will decrease by 1; the probability of those requests (safety factor $= SF - 1$) that could not be satisfied immediately is $R_{\mathrm{F}}(SF)$, the probability of those requests that could not be satisfied immediately and finally got failed is $R_{\mathrm{F}}(SF-1)$, thus,

$$R_{\mathrm{F}}(SF) = R_{\mathrm{R}}(SF-1) \cdot R_{\mathrm{F}}(SF-1).$$

- $C_D(Q)$: the increased cost if request Q is delayed and could not be satisfied in the next broadcast period.
- P_D: the increased cost if data item D is delayed and does not appear in the next broadcast period, i.e., the priority of data item D. *Data* numbers of data items with highest priority would be broadcasted in the next period.

Lemma 1. The cost of request Q if it would be delayed

$$C_D(Q) = BP \cdot C_{AT} + C_{TT} + R_{\mathrm{F}}\big(SF_Q^T\big) \cdot C_{\mathrm{F}}.$$

$C_D(Q)$ is composed of three parts: the first part indicates the access time cost increased because of delay, the second part indicates the tuning time cost increased because of delay, and the third part indicates the estimated cost of request failure because of delay.

Theorem 1. The cost of data item D if it would be delayed

$$\begin{aligned} P_D &= \sum_{Q\langle D, T_{\mathrm{req}}\rangle} C_D(Q) \\ &= PF_D^T \cdot (BP \cdot C_{AT}) + PF_D^T \cdot C_{TT} \\ &\quad + \sum_{Q\langle D, T_{\mathrm{req}}\rangle} R_{\mathrm{F}}\big(SF^T_{Q\langle D, T_{\mathrm{req}}\rangle}\big) \cdot C_{\mathrm{F}}. \end{aligned}$$

Both P_D and $C_D(Q)$ include three parts: Access Time Cost, Tuning Time Cost, and Request Failure Cost.

BP, $C_{AT}, C_{TT}, C_{\mathrm{F}}$ are predetermined constants, while PF_D^T, $R_{\mathrm{F}}(SF^T_{Q\langle D, T_{\mathrm{req}}\rangle})$ will change along with recent circumstances of access and broadcast. When the failure rate increases, those requests with low SF will be satisfied first; when the failure rate decreases, those data items with high PF will be broadcasted priorly.

If $C_{\mathrm{F}} = 0$, the priority of data item D, P_D, is in direct proportion to the number of pending requests for data item D, PF_D, and thus, LDCF degenerates to MRF.

3.2. LDCF scheduling algorithm

We describe LDCF scheduling algorithm as follows:

Algorithm 1 (LDCF).

Input: request sequence;
Output: a broadcast scheduling;
Proceeding:

```
main( )
{
    fail_rate[ ]:=[1,0,0,...,0]
    time = 0;
    while true do
    {
        for i := 1 to BP
        {time = time + 1;
            receive the new requests {req⟨Di, time⟩},
                and add them to RequestSequence;
            }
            LDCF(time);
        }
    }
    procedure LDCF(time)
    {
        for each data item Di
            DataItem[Di].priority := 0;
        for each pending request req⟨Di, req_time⟩
            in RequestSequence
        {
            SF := ⌊(req_time + RTL − time) / BP⌋;
            DataItem[Di].priority :=
                DataItem[Di].priority + BP * C_AT
                + C_TT + fail_rate[SF] * C_F;
        }
        select Data number of data items with largest
            priority from DataItem[ ];
        add these data items into broadcast period sorted
            by the value of PF (in descending order), and
            make the index;
        compute fail_rate[ ] once again;
        delete those requests that have been responded or
            failed in RequestSequence;
    }
```

In the above description, we mainly focus on the illustration of LDCF algorithm, therefore, some implementation details have been omitted. For example, when one request $req\langle Di, req\_time\rangle$ would not get responded and failed because of time out, we did not make a concrete analysis on creating direct wireless link between server and mobile user to send requested data items. Also, we will not fully explain how to select data items with largest priority, how to add broadcast contents and make index, etc.

4. Experiments and comparisons

We intend to use simulation method to compare LDCF scheduling algorithm with MRF, FCFS and LWF algorithms, so as to evaluate the performance of LDCF scheduling algorithm. And in experiments 6 and 7, we also try to use LDCF in pure push-based broadcasts and compare LDCF to other push-based scheduling algorithms. By studying its performance, we may analyze the adaptability of LDCF.

At each time, the server will receive access requests from mobile users, compute the priority of every data item on the basis of all pending requests, then select *Data* number of data items with largest priority and add them to broadcast contents.

4.1. Experimental data

4.1.1. The numerical distribution of newly-arriving requests during one time period

Suppose the probability that mobile user will send request during one time period is p $(0 \leqslant p \leqslant 1)$, the number of mobile users is m, then the probability that number of new requests equals to r is:

$$p^r \cdot C_m^r \cdot (1-p)^{m-r} \cdot C_m^{m-r}.$$

4.1.2. The numerical distribution of data items required by new requests during one time period

We use function *Zipf*(k) to describe the skewed distribution of data access. In generating the distribution of data access with *Zipf*(k), we suppose the skewness k at any time could change randomly in one interval. Besides, we would randomly select 10% data items, multiply their distribution results by a random number between 0 to 10.

4.1.3. Benchmark random numbers

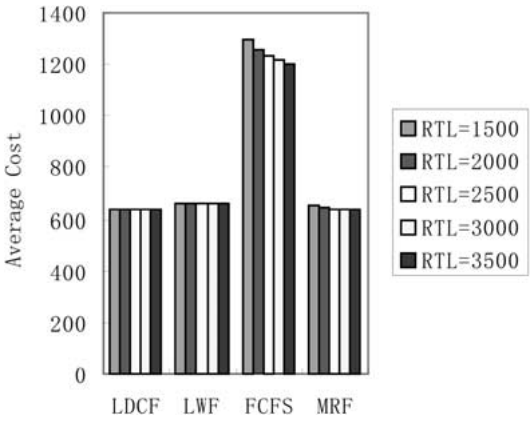
We use the randomizer provided by `http://www.randomizer.org` to generate benchmark random numbers for our experiments.

4.2. Experiment results and analysis

4.2.1. Parameter settings

The following are some common parameters:

- M: the number of data items that the server could use to broadcast. Suppose it is 1000 in the following experiments.
- *Data*: the number of data items in one broadcast period.
- *Index*: the length of index section in one broadcast period. Suppose it is 6 in the following experiments.
- *RN*: the average number of requests that the server would receive at each time.
- k: parameter of function *Zipf*, indicating the skewness of data access distribution.
- *RTL*: response time limit.
- C_{AT}: cost of *AT* per unit time. Suppose it is 1 in the following experiments.



Parameter	Value
Data	100
RN	10.10
k	0
RTL	1500–3500

Figure 3. Performance for various algorithms when fail rate of request is low.

- C_{TT}: cost of *TT* per seeking index. Suppose it is 20 in the following experiments.
- C_{F}: cost of handling a failed request. Suppose it is 2000 in the following experiments.

4.2.2. Experiment 1: Performance when fail rate of request is low

First, we discuss the performance comparison of LDCF algorithm with the other three algorithms in the situation of low workloads. We will consider the effect of various *RTL* on Average Cost of request. The setting of parameters and the results are shown in figure 3.

As *RTL* increases, Average Cost of request will decrease. When $RTL \geqslant 2500$, there are not any failed requests and all requests are satisfied in MRF scheduling, that is, all data items whose $SF = 0$ belongs to those data items with largest PF. In LDCF scheduling, all data items with $SF = 0$ belong to those data items with largest P_D, too. Therefore, at this time LDCF and MRF are identical, of which both are optimal scheduling algorithm having the least average *AT* and *TT*. The performance of LWF is a little worse than LDCF and MRF, while FCFS is the worst.

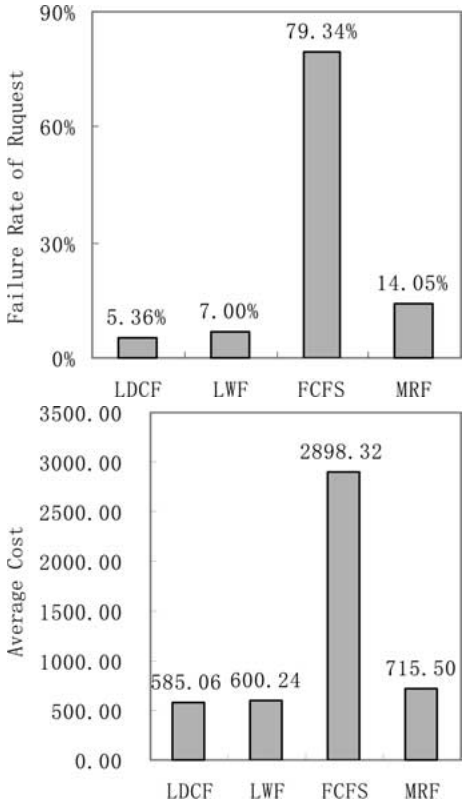
In short, LDCF and MRF occupy the first place, LWF comes the second, and FCFS is the worst.

4.2.3. Experiment 2: Performance when fail rate of request is high

In this experiment, we discuss the performance comparison between LDCF and the other three algorithms when fail rate of request is high. The setting of parameters and the results are shown in figure 4.

If the skewness k of data access randomly changes between $[-1.5, 1.5]$, lots of requests will get failed when there are many requests at each unit time.

The fail rate of LDCF scheduling is the lowest, and its average cost is the lowest, too. The performance of LWF scheduling is a little worse than that of LDCF.



Parameter	Value
Data	120
RN	247.15
k	−1.5–1.5
RTL	1500

Figure 4. Performance for various algorithms when fail rate of request is high.

FCFS scheduling has much higher fail rate and larger average cost. It shows that the average performance of FCFS scheduling is unsatisfactory, because it considers only the time factor, not the number of requests.

The performance of MRF scheduling still lags behind LDCF and LWF. It also shows that it is insufficient to consider only the number of requests, not the time factor.

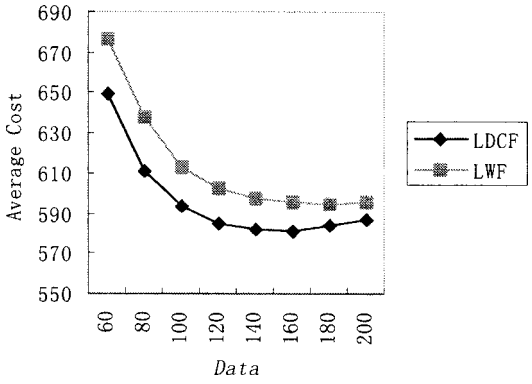
We will not compare FCFS and MRF with our algorithm in further discussion.

In short, LDCF could efficiently reduce the number of failed requests, and it has the least average cost. Consideration of only one factor (request number or time, as in the case of MRF and FCFS) will lead to lots of request failures.

4.2.4. Experiment 3: Effect of Data

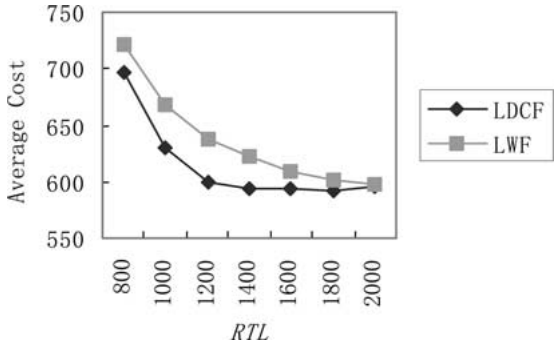
In this experiment, we discuss the effect of length of data segment in one broadcast period. The setting of parameters and the results are shown in figure 5.

In this experiment, our conclusion is: the number of data items contained in one broadcast period should be moderate. Too small a value will drastically increase the average cost, but once its value increases above one certain point, average cost will rise instead. Still, the performance of LDCF is better than that of LWF.



Parameter	Value
Data	60–200
RN	98.86
k	0
RTL	1500

Figure 5. Effect of *Data*.



Parameter	Value
Data	100
RN	98.86
k	−1.5–1.5
RTL	800–2000

Figure 6. Effect of *RTL*.

4.2.5. Experiment 4: Effect of RTL

In this experiment, we discuss the effect of Response Time Limit. The setting of parameters and the result are shown in figure 6.

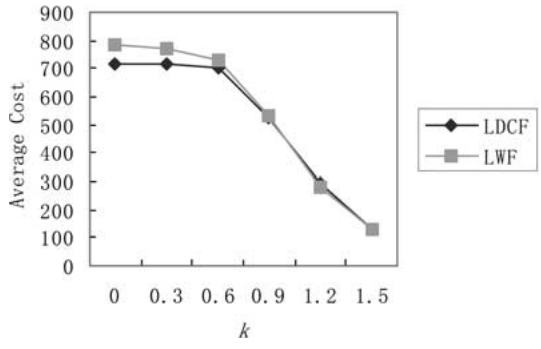
In this experiment, we conclude that the larger *RTL* is, the lower the average cost is. When $RTL > 1200$, the performance gap between two algorithms is very small. Again, LDCF shows some advantages over LWF.

4.2.6. Experiment 5: Effect of skewness of data access distribution

In this experiment, we discuss the effect of skewness k of data access distribution.

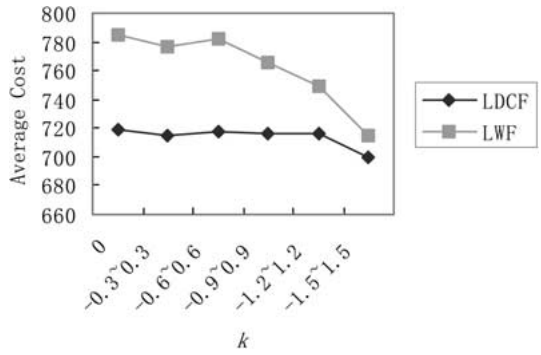
First, we will consider the cases when the value of skewness k is certain. The setting of parameters is shown as follows. The result is shown in figure 7.

Second, we consider the cases when skewness k is a random number in a certain interval centered on zero. The skewness k might be different at any time.



Parameter	Value
Data	100
RN	98.86
k	0–1.5
RTL	1000

Figure 7. Effect of skewness k of a certain value.



Parameter	Value
Data	100
RN	98.86
k	[0, 0], . . . , [−1.5, 1.5]
RTL	1000

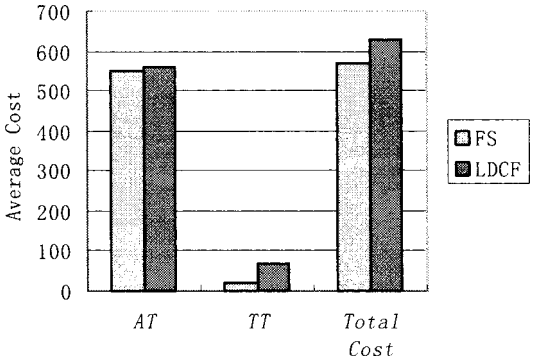
Figure 8. Effect of skewness k in a certain interval.

The setting of parameters and the results are shown in figure 8.

In this experiment, our conclusion is: when skewness k holds one certain value, the average cost of LDCF will decrease as k increases. Its overall performance is superior to that of LWF.

When skewness k is a random number in a certain interval centered on zero, the average cost of LDCF shows little influence of interval size.

The LDCF algorithm is designed for on-demand broadcasts where the server does not have the access profiles of mobile users. The experiments and analyses above show that LDCF has good performance. In a pure push-based data dissemination scheduling, we often assume that the access pattern of mobile users is certain and predictable. We try to apply LDCF to pure push-based broadcasts where the access pattern of mobile users is certain and compare LDCF with the flat schedule (FS) and the skewed schedules (such as MDS – Multi-disk schedule) [1]. Different from the previous experiments, the requests used in the next two experiments are generated according to the access pattern without any disturbance.



Parameter	Value
Data	160
k	0
RTL	2000

Figure 9. Comparing LDCF with FS in pure push-based broadcasts.

4.2.7. Experiment 6: Comparing with FS in pure push-based broadcasts with uniform access

In a pure push-based information system, if each data item has the same access probability, FS is used to disseminate the information; if the access pattern of mobile users is not available, usually we assume that each data item has the same access probability, and FS is also used.

In this experiment, we make the condition that each data item has same access probability, adopt $(1, m)$ index [11] to insert index segments into broadcasts and set a sufficient *RTL* so that in FS no access failure will occur. Apparently, in this case, FS is the best schedule. The setting of parameters of LDCF is shown in figure 9 and the unlisted parameters are set as in the previous experiments. The result is shown in figure 9.

In this experiment, when $m = \sqrt{M/Index} = 10$, average cost of FS is minimal [11], and equals to

$$\frac{M + m \cdot Index}{2} + C_{TT} = 550 + 20 = 570.$$

Average cost of LDCF is

$$C_{AT} + C_{TT} + C_{\mathrm{F}} = 557 + 68 + 0 = 625.$$

The C_{AT} of LCDF is very close to the C_{AT} of FS, and the C_{TT} of LCDF is several times larger than the C_{AT} of FS, just because each index segment in LDCF includes the addresses of the data items only in the subsequent data segment. In order to find a data item the MU often needs to read several index segments, while in the $(1, m)$ index each index segment includes the addresses of all data items, so in FS we can obtain the address of the data item by reading only one index segment.

LDCF's performance is related to *Data*'s value. To the proper *Data,* its performance is close to the theoretically best schedule – FS, in this case.

4.2.8. Experiment 7: Comparing with skewed schedules in pure push-based broadcasts with skewed access

When the access probability of data item is different from each other, we should use other more efficient schedules to replace FS. One of the most famous approaches is MDS proposed by Acharya et al. in [1]. However, MDS counts little on index and optimizing, Li improves MDS and proposes HMDS in [12] which has greatly optimized its *AT*, Sun et al. propose a new skewed schedule named NAS in [15], whose average *AT* is close to the theoretical minimum.

Different from FS, the broadcast period of MDS and HMDS can not be pre-set, various access pattern and parameter values will cause different broadcast periods, so LDCF's *RTL* and MDS and HMDS's broadcast period cannot be simply set equally.

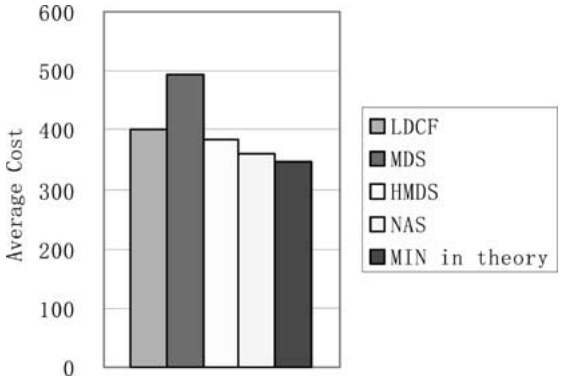
The 80–20 pattern is a most typical access pattern in database system, i.e., the 80% accesses concentrate on the 20% data items, nearly close to the distribution of *Zipf*(0.95), so we make contrast between LDCF and MDS, HMDS, NAS with this access pattern.

There's little discussion on index in Acharya's paper, and we imitate $(1, m)$ index to insert index segments. With skewed access, we use *Data* to indicate the broadcast period without index, and make the indexless average *AT* to be $\alpha \cdot Data$, so if $m = \sqrt{Data/(2\alpha \cdot Index)}$ and *AT* is minimal, we can see that this formula is still correct in FS with uniform access.

The setting of parameters of LDCF is shown in figure 10 and the unlisted parameters are set as in the previous experiments. The result is shown in figure 10.

In Figure 10, we select the best results of MDS and HMDS with the varying *num_disks*. Set a sufficient *RTL* so that in these skewed schedules no access failure will occur. The deduction of theoretical optimizing is mentioned in [12]. Average cost of LDCF is counted according to various *RN* from 51.82 to 1340.65.

In the two above experiments, we conclude that LDCF performs well even in pure push-based broadcasts. Its average cost is close to the schedules with access pattern preset, which proves its fine adaptability.



Parameter	Value
Data	160
RN	51.82–1340.65
k	0.95
RTL	2000

Figure 10. Comparing LDCF with other skewed schedules in pure push-based broadcasts.

5. Conclusions

In this paper, we have studied the problem of scheduling on-demand broadcasts. Compared with pure push-based data scheduling, we need uplink channel to send data access request in an on-demand broadcast-based environment. The server would not know the access profiles of mobile users, and it should take into account the situation when request fails because of time out.

Previous works in this context mainly discuss how to reduce the average *AT* of mobile users. In practical applications, the handling of a request waiting for quite a long time must be considered and we introduce the notion *request failure*. While discussing the performance of a scheduling algorithm of on-demand broadcasts, we take into account not only *AT*, but also *TT* and request failure. We put forward a self-adaptive scheduling algorithm – LDCF, which computes the delay cost for every data item and uses it as the priority to schedule the data items. The parameters of delay cost computing formula will be adjusted automatically according to recent scheduling circumstances.

Our work raises the open algorithmic problem of determining a schedule that minimizes the average cost of request considering all kinds of cost – *AT*, *TT* and failure. We compare LDCF with LWF, FCFS and MRF through several experiments, which indicate the average cost of LDCF be the least. We also make contrast between LDCF and some skewed schedules in pure push-based broadcasts, and LDCF acts well and shows fine adaptability.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grant No. 69933010.

References

[1] S. Acharya, R. Alonso, M.J. Franklin and S.B. Zdonik, Broadcast Disks: Data management for asymmetric communications environments, in: *SIGMOD Conference* (1995) pp. 199–210.

[2] S. Acharya, M.J. Franklin and S.B. Zdonik, Disseminating updates on Broadcast Disks, in: *VLDB* (1996) pp. 354–365.

[3] S. Acharya, M.J. Franklin and S.B. Zdonik, Dissemination-based data delivery using Broadcast Disks, IEEE Personal Communications 2(6) (1995) 50–60.

[4] S. Acharya and S. Muthukrishnan, Scheduling on-demand broadcasts: New metrics and algorithms, in: *MOBICOM* (1998) pp. 43–54.

[5] D. Aksoy and M.J. Franklin, Scheduling for large-scale on-demand data broadcasting, in: *INFOCOM*, Vol. 2 (1998) pp. 651–659.

[6] M.A. Bender, S. Chakrabarti and S. Muthukrishnan, Flow and stretch metrics for scheduling continuous job streams, SODA (1998) 270–279.

[7] C. Dhawan, *Mobile Computing* (McGraw-Hill, 1997).

[8] A. Datta, D.E. Vandermeer, A. Celik and B.V. Kumar, Broadcast protocols to support efficient retrieval from database by mobile users, ACM Transactions on Distributed Systems 24(1) (1999) 1–79.
[9] V.A. Gondhalekar, Scheduling periodic wireless data broadcast, MS thesis, University of Texas at Austin (1995).
[10] T. Imielinski and B.R. Badrinath, Mobile wireless computing: Challenges in data management, Communications of the ACM 37(10) (1994) 18–28.
[11] T. Imielinski, S. Viswanathan and B.R. Badrinath, Energy efficient indexing on air, in: *SIGMOD Conference* (1994) pp. 25–36.
[12] L. Li, The research on data broadcast and data replication/caching in mobile database systems, Ph.D. thesis, National University of Defense Technology, PR China (1999).
[13] N. Shivakumar and S. Venka, Efficient indexing for broadcast-based wireless systems, Mobile Networks and Applications 1(4) (1996) 433–446.
[14] C. Su and L. Tassiulas, Broadcast scheduling for information distribution, in: *INFOCOM*, Vol. 1 (1997) pp. 109–117.
[15] W. Sun, W. Shi and B. Shi, Optimizing the access time of data broadcast in mobile computing environments, Mini-Micro Systems (to appear).
[16] S. Viswanathan and T. Imielinski, Pyramid broadcasting for video on demand service, Technical report DCS TR-311, Rutgers University (1994).
[17] J.W. Wong, Broadcast delivery, Proceedings of the IEEE 76(12) (1988) 1566–1577.

Weiwei Sun is a Ph.D. candidate in the Department of Computing and Information Technologies, Fudan University, Shanghai, China. He received M.S. degree in computer science from Fudan University in 1999. His current research is broadcast scheduling for wireless mobile computing systems.
E-mail: wwsun@online.sh.cn

Weibin Shi is a Ph.D. candidate in the Department of Computing and Information Technologies, Fudan University, Shanghai, China. He received M.S. degree from Shanghai University for Science and Technology in 1992. His research interests include object-oriented database systems, mobile databases and query evaluation techniques for XML data.
E-mail: ly008136@online.sh.cn

Bole Shi is a Chief Professor of the Department of Computing and Information of Fudan University, Superintendent of Computer Science Research Institute of Fudan University, Director of Shanghai International Research Center, Vice Chairman of Computer Science Education Guide Commission of Education Ministry of China. His research area includes relational databases, knowledge bases, object-oriented databases, and mobile databases.
E-mail: bshi@fudan.edu.cn

Yijun Yu graduated from the Computer Science Department of Fudan University, China (B.S., 1992). After graduation, he participated the Laboratory of Software Engineering in Fudan University and did R&D for the Jadebird CASE tools. After receiving his M.S. degree in 1995, he continued his postgraduate course on computer software in Fudan University and did research work on parallel processing at the Institute of Parallel Processing. In July, 1998, he received his Ph.D. from Fudan University. Since 1999, he joined the ELIS research group in University of Ghent, Belgium. His research interests include compilers, distributed systems, wireless networks, computer graphics and visualization.
E-mail: yijun@linmp1.elis.rug.ac.be