Wireless Netw (2010) 16:1817–1836
DOI 10.1007/s11276-009-0230-x

Exploiting replication on dependent data allocation for ordered queries over multiple broadcast channels

Jiun-Long Huang · Jen-He Huang

Published online: 19 December 2009
© Springer Science+Business Media, LLC 2009

Abstract Data broadcasting has been recognized as an important means for information dissemination in mobile computing environments. In some mobile applications, the data items broadcast are dependent upon one another. However, most prior studies on broadcasting dependent data do not employ replication in broadcast program generation. In view of this, we explore in this paper the problem of broadcasting dependent data in multiple broadcast channels, and explicitly investigate the effect of data replication. After analyzing the model of dependent data broadcasting, we derive several theoretical properties to formulate the average access time of broadcast programs. In light of the theoretical results, we develop an algorithm to exploit replication on broadcast program generation. Our experimental results show that the proposed algorithm is able to generate broadcast programs of very high quality. In addition, the results also show that broadcast programs with replication is more robust than those without replication in error-prone environments.

Keywords Data broadcast · Dependent data · Mobile computing · Ordered queries

1 Introduction

To provide power conservation, high scalability and high network bandwidth utilization services in mobile environments, a data delivery architecture in which a server continuously and repeatedly broadcasts data to a client community through a *single* broadcast channel was proposed in [1, 28]. Related research issues about broadcast-based information systems have attracted a considerable amount of studies, including (1) broadcast program generation [1, 48], (2) on-demand broadcast [2, 3, 4, 16, 21], (3) data indexing [13, 28, 50] and (4) location-dependent data broadcasting [32, 35, 52, 55, 56, 57]. As pointed out in [43], one cannot simply merge multiple low-bandwidth physical channels into one high-bandwidth physical channel. Consequently, a significant amount of research effort has been elaborated on the development of index mechanisms [6, 29, 39, 46], data allocation schemes [7, 8, 42, 43, 54] and dynamic data and channel allocation [25, 34] in *multiple* broadcast channels. In addition, the bandwidth allocation for multi-cell environments with frequency reuse and interference considered was studied in [51].

However, most works mentioned above were under the premise that each user requests only one data item at a time and the requests for all data items are independent of one another. That is, the access probability for a user to request a data item in the i-th request is *predetermined* and is *independent* of what have been requested in his/her first, second,…, $(i-1)$-th requests. However, in many real applications [47], some data items are *semantically related* and there exists dependency among the requests of these data items. Broadcast program generation algorithms assuming independent requests might not be able to effectively optimize the performance of such broadcast programs. A query corresponds to a set of semantically related data items which are likely to be requested successively by a user, and we use a set of queries (named a query profile) to model the dependency of all data items. Hence, if several data items are *dependent* upon one

J.-L. Huang (✉) · J.-H. Huang
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC
e-mail: jlhuang@cs.nctu.edu.tw

J.-H. Huang
e-mail: jenho@cs.nctu.edu.tw

another (i.e., within the same query), they are likely to be successively requested by users. According to the constraint of the retrieval sequence of the dependent data items within the same query, queries of dependent data can be categorized into the following two types:

Ordered queries: In an ordered query, the required data items should be retrieved in a predetermined order. Consider a Web page with some images as an example. Once the user requests this Web page by a browser, the browser will retrieve these images automatically in a predetermined order after receiving this Web page [36].
Unordered queries: Similar to an ordered query, an unordered query could be one issued by a mobile user for requesting multiple data items simultaneously. However, unlike in an ordered query, these requested data items may be retrieved in any order. Consider a broadcast system which disseminates the stock information. A mobile user may submit a query like "Show me the stock information of all the LCD companies." As a result, data items in these LCD companies are queried together and displayed without being confined to a specific order [31].

The system architecture of the data broadcast system considered in this paper is shown in Fig. 1 [23, 26]. The system periodically broadcasts data items according to a predetermined broadcast program. In the beginning, without knowing the broadcast program of the system, the mobile device will listen on a broadcast channel to wait for the appearance of the broadcast program. The broadcast program contains some auxiliary information such as data identifiers. The broadcast program is then kept in the mobile device. When a user submits a query to his or her mobile device, the mobile device will retrieve the first required data item from one broadcast channel by referring to the information in the broadcast program. The received data item may possess dependency with other data items,

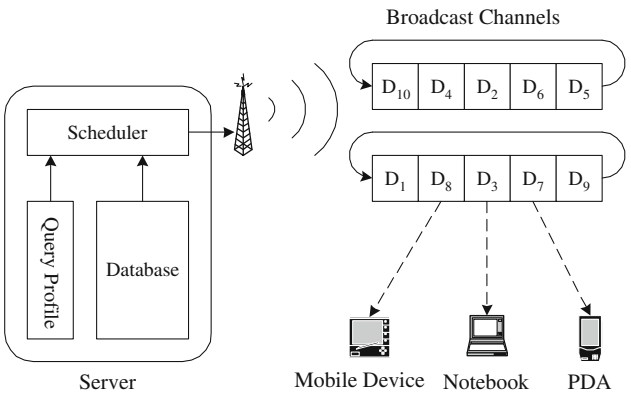

Fig. 1 The architecture of a data broadcast system

(a)

D_1	D_4	D_2
D_6	D_3	D_5

(b)

$D_1(1)$	$D_2(1)$	$D_1(2)$	$D_3(2)$	$D_5(1)$
$D_6(1)$	$D_3(1)$	$D_4(1)$	$D_6(2)$	$D_1(3)$

Fig. 2 Example broadcast programs. **a** Without replication. **b** With replication

and the mobile device will retrieve those dependent data items automatically.

According to the broadcast frequencies of all data items in one broadcast cycle, broadcast programs can also be divided into two categories: without replication and with replication. A broadcast program is said to be *without replication* if all data items are broadcast with equal frequencies. Figure 2(a) shows an example broadcast program without replication. A broadcast program without replication is also called a *flat* broadcast program[1]. On the other hand, as shown in Fig. 2(b), a *broadcast program with replication*[2] corresponds to the case that the data items appear with different frequencies. In most research studies on independent data broadcasting [42, 43, 54], it has been shown that broadcasting data items with higher access frequencies more frequently can achieve the shorter average access time.

As a consequence, prior research studies on broadcast program generation for dependent data can be categorized by the following three properties: (1) the number of broadcast channels (single [11, 47] or multiple channels [23]), (2) the constraint of the sequence of the data items (ordered [11, 23] or unordered [15] queries) and (3) the employment of data replication (without replication or with replication). A brief review and comparison of the prior works on dependent data broadcasting are given in Sect. 2.1. Most prior studies on broadcast program generation for dependent data employ flat broadcast programs (i.e., without replication). We argue that replication should be used in broadcast program generation for dependent data due to the following two advantages:

1. *High performance:*
   It has been shown that, when the data items are independent, broadcast programs with replication usually outperform those without replication. However, prior studies on dependent data broadcasting do not employing replication in broadcast programs. It motivates us to exploit replication on broadcast program generation in order to achieve better performance.
2. *Error resilience:*
   In broadcast programs without replication, each data item appears exactly once in each broadcast cycle.

[1] D_i indicates the i-th data item.

[2] $D_i(j)$ indicates the j-th replica of the i-th data item.

Thus, broadcast programs without replication do not perform well in error-prone environments. Consider the broadcast program shown in Fig. 2(b). Suppose that a network error occurs when a user is retrieving data item D_1. Since D_1 appears only once in each broadcast cycle, the user has to wait almost one full broadcast cycle (two data items in this example) for retrieving D_1.

Consider the broadcast program shown in Fig. 2(b). Suppose that a network error occurs when the user is retrieving the first replica of D_1 (i.e., $D_1(1)$). Since there are many replicas of D_1 in the broadcast program, the user only needs to wait one data item for retrieving the second replica of D_1 (i.e., $D_1(2)$). Based on the above observation, we argue that broadcast programs with replication is more error-resilient than those without replication. Thus, we employ replication in broadcast program generation to ease performance degradation caused by network errors.

In view of this, we explore in this paper the problem of the broadcast program generation for ordered queries in multiple broadcast channels. Note that the problem of broadcasting dependent data in a multiple channel environment is intrinsically difficult in that the factor of data dependency and the efficient use of multiple channels, though being dealt with separately before, are in fact entangled, thus making it more complicated to provide an effective solution to this problem. Note that several special cases of the problem of broadcasting dependent data are shown to be NP-hard [15, 36, 39]. To the best of our knowledge, no prior work on dependent data broadcasting over multi-channel environments employs replication in broadcast programs. This characteristic distinguishes our work from others.

In this paper, we first model the problem of broadcast program generation for ordered queries in multiple channels with replication, and then formulate the average access time of broadcast programs accordingly. Based on these theoretical results, we devise algorithm BPGR (standing for Broadcast Program Generation with Replication) to employ data replication in broadcast program generation. To measure the performance of algorithm BPGR, several experiments are conducted. Our experimental results show that algorithm BPGR is able to obtain broadcast programs of higher quality than those algorithms without employing data replication, showing that employing replication can lead to more efficient use of network bandwidth. In addition, experimental results also show that in error-prone environments, the performance degradation of the programs generated by algorithm BPGR is more slighter than that of the programs generated by algorithms without employing replication.

The rest of this paper is organized as follows. Section 2 presents the preliminaries of this study. Average access time of broadcasting dependent data with ordered queries is formulated in Sect. 3. We then develop in Sect. 4 algorithm BPGR to exploit replication in broadcast program generation for the environments with multiple broadcast channels. Performance evaluation on various parameters is conducted in Sect. 5. Finally, Sect. 6 concludes this paper.

2 Preliminaries

2.1 Related work

2.1.1 On-demand data broadcasting

To give readers more background knowledge about data broadcast, in addition to push-based broadcast architecture which is employed in this paper, pull-based broadcast architecture, also known as on-demand broadcast architecture, is described in this subsection. Figure 3 shows an example on-demand broadcasting system. In an on-demand data broadcasting system [2, 4, 5], a server maintains a data request queue and serves these requests according to the employed scheduling algorithm. When requiring one data item, a mobile client sends a data request to the server. After receiving a data request, the server first checks whether there exists another data request in the data request queue with the same required data object. If yes, the new-coming data request is *merged* into that data request. This phenomenon is called *request merge*. Data requests with the same requested data object can be safely merged since one transmission of the data object in a broadcast channel is able to serve all merged data requests. Therefore, the higher the occurrence probability of request merge is, the more efficient the system is. Otherwise, the new-coming data request is *inserted* into the data request queue.

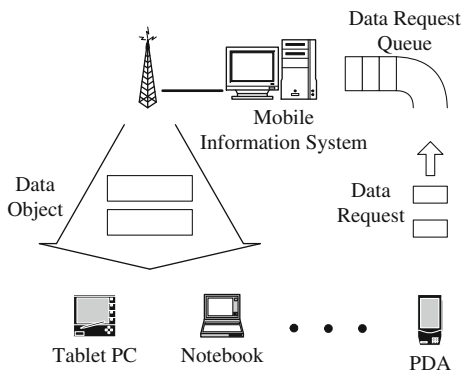

Fig. 3 An example on-demand broadcasting system

A scheduling algorithm is used to prioritize all data requests in the data request queue, and the server will serve these data requests according to their priorities. To serve a data request, the system retrieves the required data object from the corresponding data server, and then broadcasts this object to all its clients via a dedicated and shared broadcast channel. As a result, the on-demand broadcast system is more scalable and can obtain higher network utilization than traditional client-server architecture.

Dykeman et al. pointed out in [19] that traditional FCFS scheduling would produce long average access time for an on-demand broadcast system when the access frequencies of all data items were not uniformly distributed. They proposed several scheduling algorithms and concluded that LWF could provide the best performance among the proposed algorithms. Aksoy et al. pointed out in [4] that although being able to produce the shortest average access time, LWF is not efficient when the number of data requests is large. To address this problem, they proposed algorithm $R \times W$ which is able to schedule the received data requests efficiently by employing a pruning technique. In [2], Acharya et al. addressed the broadcast scheduling problem in the environments with variable-size data items. They defined a new metric, stretch, as the ratio of the response time of a request to its service time. Based on stretch, they proposed a scheduling algorithm, called LTSF, to minimize the stretch. Wu et al. [49] argued that algorithm LTSF is not optimal in terms of overall stretch. In addition, algorithm LTSF is not scalable in a large-scale environment. Therefore, they proposed a scheduling algorithm to optimize the system performance in terms of stretch. Moreover, the proposed scheduling algorithm is more scalable than LTSF, and hence, is suitable for practical use. In recent years, several researches are focused on scheduling for dependent data in on-demand broadcast environments [12, 14, 17, 44]. In addition to devising scheduling algorithms, several studies consider the employment of data indexing in on-demand data broadcasting environments [21, 24, 33]. Since the sizes of index items are much smaller than those of data items, employing data indexing is able to greatly reduce the average tuning time at the cost of a slight increase in the average access time.

2.1.2 *Broadcast program generation for unordered queries*

For unordered queries, the authors in [15] modelled the dependency among data items as a set of ⟨query, access probability⟩ pairs, and proposed a greedy algorithm called QEM to generate broadcast programs for unordered queries in single channel environments. The authors in [30] extended algorithm QEM by proposing several heuristics. In addition, a data oriented broadcast program generation algorithm is also proposed. The authors in [53] modelled the data dependency as a dependency graph and transformed this problem into a travelling salesman problem. The authors in [18] presented a theoretical analysis of data dependencies and proposed a polynomial algorithm to generate broadcast programs in a single channel environment. The authors in [22] formulated the average access time of broadcast programs on multiple channel environments, and then, proposed a genetic-based algorithm to generate broadcast programs. Based on the theoretical results derived in [22], the authors in [27] proposed a heuristic algorithm to generate broadcast programs on multiple channel environments. It is worth mentioning that the prior studies on dependent data broadcasting for unordered queries did not consider data replication.

2.1.3 *Broadcast program generation for ordered queries*

In [11], the problem of dependent data broadcasting for ordered queries is transformed into an optimal linear ordering problem and two heuristic algorithms are then proposed to generate broadcast programs in a single-channel environment. However, the queries are assumed to be *acyclic* (i.e., the dependency graph must be a dag), and such an assumption limits the applicability of the proposed algorithms. The author in [36] first modelled data dependencies as a dag and then modelled the problem of dependent data broadcasting for ordered queries in a single broadcast channel as a minimum circular arrangement problem. A heuristic was proposed to generate broadcast programs. The authors in [47] proposed a branch-and-bound searching algorithm to search a broadcast program in one broadcast channel. A randomized algorithm was proposed in [9] to consider the dependency of *two* data items in a single broadcast channel. The authors in [31] first transformed the problem of dependent data broadcasting for ordered queries into an optimal cycle ordering problem, and then proposed a broadcast program generation algorithm in a single-channel environment. In [38], the dependencies among data items are modelled as a dag and broadcast program generation in a single broadcast channel is modelled as a topological ordering problem. Two strategies, called level-oriented strategy and greedy-oriented strategy, are then proposed to generate broadcast programs. With similar formulations as [38], the authors in [37] proposed a level-based scheduling algorithm and several heuristics to generation broadcast programs on multi-channels. In [45], Shih et al. extended the problem modeled in [38] to further consider the problem of fairness and proposed two algorithms to solve such problem. In [23], Huang et. al. derived several theoretical results of generating broadcast programs for ordered queries, and proposed a genetic algorithm to generate broadcast programs

Table 1 Comparison among our study and related studies

Study	No. of channels	Replication
Chehadel et al. [11]	Single	No
Si et al. [47]	Single	No
Noy et al. [9]	Single	No
Lee et al. [31]	Single	No
Liberatore [36]	Single	Yes
Liu et al. [38]	Single	No
Huang et al. [23]	Multiple	No
Lin et al. [37]	Multiple	No
Hung et al. [26]	Multiple	No
Our study	Multiple	Yes

accordingly. In [26], Hung et. al. proposed a two-phase framework called MULS for dependent data broadcasting for ordered queries. Note that all prior studies for multiple broadcast channels did not employ data replication. A comparison of our study and these related studies is given in Table 1.

2.2 Problem description and formulation

We in this subsection formulate the problem of broadcasting dependent data in multiple broadcast channels. For better readability, a list of symbols used is shown in Table 2. Same as in [25], it is assumed that the database D contains $|D|$ data items, D_1, D_2,…, $D_{|D|}$ and each data item is read-only. An ordered query is defined as follows.

Definition 1 An *ordered* query $Q_i = \{D_{q^i(1)}, D_{q^i(2)}, \cdots, D_{q^i(|Q_i|)}\}$ is an *ordered*, *non-empty subset* of all data items where $|Q_i|$ represents the number of required data items in Q_i. Note that $1 \le q^i(j) \le |D|$ for all $1 \le j \le |Q_i|$, and

Table 2 Description of symbols

Symbol	Description
$\|D\|$	Size of the original database
$\|D^*\|$	Size of the revised database
L^*	Length of the broadcast program w. replication
n	Number of broadcast channels
D_j	The j-th data item
Q_i	The i-th query
$Pr(Q_i)$	Access probability of the i-th query
$D_{q^i(j)}$	The j-th required data item in Q_i
$D_i(j)$	The j-th copy of data item D_i
$n(D_i)$	Number of replicas of data item D_i
s	Size of each data item
B	Channel bandwidth

Table 3 An example query profile

Query(Q_i)	$Pr(Q_i)$ (%)
$Q_1 = \{D_1, D_2, D_3\}$	30
$Q_2 = \{D_1, D_3, D_4\}$	20
$Q_3 = \{D_4, D_6, D_1\}$	20
$Q_4 = \{D_1, D_2, D_5\}$	10
$Q_5 = \{D_5, D_3, D_4, D_6\}$	10
$Q_6 = \{D_6, D_5, D_2\}$	10

$q^i(j) = k$ represents that the j-th required data item in Q_i is D_k.

A query profile, which is an aggregation of the access behavior of all users, is defined as below.

Definition 2 A query profile Q consists of a set of $\langle Q_i, Pr(Q_i)\rangle$ pairs where $|Q|$ represents the number of queries in Q and Q_i is the i-th query in Q. In addition, $Pr(Q_i)$ is defined as

$$Pr(Q_i) = \frac{\text{No. of } Q_i \text{ issued by users}}{\text{No. of queries issued by users}},$$

$$\text{where } \sum_{i=1}^{|Q|} Pr(Q_i) = 1.$$

For the sake of simplicity, we assume that the queries is sorted by their access probabilities in descending order. That is, $Pr(Q_i) \ge Pr(Q_j)$ if $i \le j$.

Example 1 Consider a database D containing six data items, D_1, D_2,…, D_6. Table 3 shows an example query profile containing six queries on the database D. The query $Q_3 = \{D_4, D_6, D_1\}$ indicates that the mobile device will retrieve D_4, D_6 and D_1 sequentially while the user submits Q_3. $Pr(Q_3) = 20\%$ shows that 20% of the queries submitted by users are Q_3*s*.

Note that the capture of the query profile is a challenging problem since the data items are dependent upon one another. Similar to [20], when an uplink channel is provided, the mobile device can store hot queries of its owner and send its query statistics to the server. When the mobile device is about to connect to the server (e.g., to register some mobile services), these statistics can be transferred by piggybacks. Clearly, dependent on applications, different methods to capture the query profile of a broadcast-based information system are considerable.

The bandwidth of each channel is divided into slots of equal size s. A data item D_i will occupy $\left\lceil \frac{|D_i|}{s} \right\rceil$ slots where $|D_i|$ is the size of D_i. Let $r = \left\lceil \frac{|D_i|}{s} \right\rceil$ and D_i occupy slots D_i^1, D_i^2,…, D_i^r. If a mobile user requests D_i, the system will retrieve $D_i^1 \rightarrow D_i^2 \rightarrow \cdots \rightarrow D_i^r$, sequentially. Therefore, a

query containing a multi-slot data item D_i is expanded from $\{\ldots, D_i, \ldots\}$ into $\{\ldots, D^1{}_i, D^2{}_i, \ldots, D^r{}_i, \ldots\}$. For ease of the presentation, each data item is assumed to be of equal size s (i.e., $|D_i| = s$ for all i).

The problem of broadcast program generation with replication can be divided into two subproblems: (1) determining the number of replicas needed for each data item and (2) determining the placement of these replicas into the broadcast program. After determining the number of replicas of each data item D_i, the original database D is revised according to the number of replicas of each data item. The revised database, denoted as D^*, can be defined as

$$D^* = \bigcup_{i=1}^{|D|}\left(\bigcup_{j=1}^{n(D_i)}\{D_i(j)\}\right),$$

where $n(D_i)$ is the number of replicas of D_i and $D_i(j)$ indicates the j-th copy of data item D_i. Accordingly, the size of the revised database D^* (denoted as $|D^*|$) is as follows

$$|D^*| = \sum_{i=1}^{|D|} n(D_i) = n \times L^*,$$

where L^* is the length of the broadcast program. Finally, the broadcast program with replication can be stated as follows.

Definition 3 A broadcast program P with replication is the placement of all data items in D^* into an n by L^* array. L^* is specified by the system designers and should be larger than or equal to $\left\lceil \frac{|D|}{n} \right\rceil$. In addition, each data item D_i will appear $n(D_i)$ times in the broadcast program P.

To facilitate the following discussion, we have the following definition.

Definition 4 Let position $(D_i(j))$ be the position of $D_i(j)$. The *distance* from the j_1-th replica of D_{i_1} (denoted as $D_{i\_1}(j_1)$) to the j_2-th replica of D_{i_2} (denoted as $D_{i\_2}(j_2)$) is defined as

$$\begin{aligned} dist(D_2(1), D_6(2)) &= \text{position}(D_6(2)) - \text{position}(D_2(1)) \\ &\quad - 1 \\ &= 4 - 2 - 1 = 1, \end{aligned}$$

while the distance from $D_6(2)$ to $D_2(1)$ is

$$\begin{aligned} dist(D_6(2), D_2(1)) &= L^* - \text{position}(D_6(2)) \\ &\quad + \text{position}(D_2(1)) - 1 \\ &= 5 - 4 + 2 - 1 = 2. \end{aligned}$$

□

Two metrics, *access time* and *tuning time* are introduced in [28] to evaluate the performance of broadcast programs. The access time is the time elapsed from the moment a client issues a query to the point that all the relevant data are read. The tuning time is the amount of time spent by the client listening on the broadcast channels, which is a measurement of power consumption. Here we take the access time as the measurement for the quality of broadcast programs. Denote the average access time of a query Q_i as $T_{\text{Access}}(Q_i)$ and the average access time of a query profile Q as $T_{\text{Access}}(Q)$. Then we have the following equation,

$$T_{\text{Access}}(Q) = \sum_{i=1}^{|Q|}[T_{\text{Access}}(Q_i) \times Pr(Q_i)]. \quad (1)$$

With the above definitions, the problem of broadcast program generation with replication is formulated as follows.

Definition 5 Given the number of broadcast channels, a database D and a query profile Q, the problem of broadcast program generation with replication is to determine a broadcast program P with replication which minimizes the average access time of the query profile Q.

3 Formulation of average access time

We extend our formulation approach in [22] to formulate the average access time of a query profile on a broadcast program with replication in this section to facilitate the design of the proposed broadcast program generation algorithm for dependent data. We first decompose the access time in Sect. 3.1, and then derive the average access time of broadcast programs with replication in Sect. 3.2.

$$\text{dist}(D_{i_1}(j_1), D_{i_2}(j_2)) = \begin{cases} \text{position}(D_{i_2}(j_2)) - \text{position}(D_{i_1}(j_1)) - 1, & \text{if position}(D_{i_1}(j_1)) < \text{position}(D_{i_2}(j_2)); \\ L^* - \text{position}(D_{i_1}(j_1)) + \text{position}(D_{i_2}(j_2)) - 1, & \text{if position}(D_{i_1}(j_1)) \geq \text{position}(D_{i_2}(j_2)). \end{cases}$$

Example 2 In the broadcast program shown in Fig. 2(b), the positions of $D_2(1)$ and $D_6(2)$ are 2 and 4, respectively. That is, $\text{position}(D_2(1)) = 2$ and $\text{position}(D_6(2)) = 4$. Then, the distance from $D_2(1)$ to $D_6(2)$ is

3.1 Decomposition of access time

To facilitate the derivation of the average access time of an arbitrary query Q_i, we decompose the access time of Q_i into three parts: *startup time, waiting time* and *retrieval time*. When a mobile user submits a query Q_i, the mobile device should wait until the system starts to broadcast the first required data item of Q_i (i.e., $D_{q^i(1)}$). This time interval is called the startup time. The waiting time is defined as the summation of the time intervals between the moment that the mobile device completes the retrieval of the data item $D_{q^i(j)}$ and the moment that mobile device starts to retrieve the next data item $D_{q^i(j+1)}$. The retrieval time is the aggregated time while the mobile device indeed reads data items from broadcast channels. It is noted that the retrieval time of a query is proportional to the number of data items required by the query.

Example 3 Consider the scenario shown in Fig. 4. The corresponding broadcast program is shown in Fig. 2(b) and a mobile user issues the query $\{D_1, D_2, D_6\}$ shown in Table 3. In this example, the startup time, the waiting time and the retrieval time are $s_1 = 1 \times \frac{s}{B}, s_3 + s_5 = 2 \times \frac{s}{B}$ and $s_2 + s_4 + s_6 = 3 \times \frac{s}{B}$, respectively. As a result, the access time (i.e., the summation of startup time, waiting time and retrieval time) of the query in this scenario is equal to $(1 + 2 + 3) \times \frac{s}{B} = 6 \times \frac{s}{B}$. □

3.2 Broadcast program with replication

According to the decomposition of access time mentioned above, the average access time of the query Q_i (denoted as $T_{\text{Access}}(Q_i)$) can be formulated as

$$T_{\text{Access}}(Q_i) = T_{\text{Startup}}(Q_i) + T_{\text{Wait}}(Q_i) + T_{\text{Retr.}}(Q_i),$$

where $T_{\text{Startup}}(Q_i)$, $T_{\text{Wait}}(Q_i)$ and $T_{\text{Retr.}}(Q_i)$ represent the average startup, waiting and retrieval time of Q_i, respectively. Thus, Eq. (1) is rewritten as

$$T_{\text{Access}}(Q) = T_{\text{Startup}}(Q) + T_{\text{Wait}}(Q) + T_{\text{Retr.}}(Q), \quad (2)$$

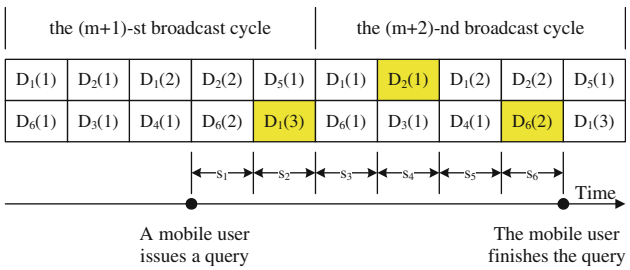

Fig. 4 An example scenario of a query

where

$$T_{\text{Startup}}(Q) = \sum_{i=1}^{|Q|} T_{\text{Startup}}(Q_i) \times Pr(Q_i), \quad (3)$$

$$T_{\text{Wait}}(Q) = \sum_{i=1}^{|Q|} T_{\text{Wait}}(Q_i) \times Pr(Q_i), \quad (4)$$

$$T_{\text{Retr.}}(Q) = \sum_{i=1}^{|Q|} T_{\text{Retr.}}(Q_i) \times Pr(Q_i). \quad (5)$$

Then, we will formulate the average startup, waiting and retrieval time in the following subsections.

3.2.1 Derivation of average startup time

Suppose that the user submits Q_i in the m-th broadcast cycle and the time interval between the start time of the m-broadcast cycle and the time that the user submits Q_i is a uniform distribution over $(0, L^*)$. Let b be the number of replicas of the first requested data item of Q_1. That is, $b = n(D_{q^i(1)})$. To simplify the further derivation, we define a series $a(j), j = 1, 2, \ldots, n(D_{q^i(1)})$, to represent the *sorted* positions of all copies of $D_{q^i(1)}$. Therefore, $T_{\text{Startup}}(Q_i)$ can be formulated as follows.

Lemma 1 *In a broadcast program with replication, $T_{\text{Startup}}(Q_i)$ can be formulated as*

$$T_{\text{Startup}}(Q_i) = \frac{s}{L^* \times B} \times \left\{ \sum_{i=1}^{b-1} \frac{[a(i+1) - a(i)]^2}{2} + \frac{[L^* - a(b) + a(1)]^2}{2} \right\}.$$

With the result of Lemma 1, $T_{\text{Startup}}(Q)$ can be obtained by Eq. (3).

Example 4 Consider the broadcast program shown in Fig. 2(b) and a query $Q_1 = \{D_1, D_2, D_3\}$. As shown in Table 1, the length of the broadcast program is five. Moreover, we have $b = 3$ since there are three copies of the first required data item in Q_1 (i.e., $D_1(1)$, $D_1(2)$, and $D_1(3)$). In addition, since $a(1)$ is the first broadcast data item among all copies of D_1, we have $a(1) = 1$. Similarly, we also have $a(2) = 3$ and $a(3) = 5$. Then, $T_{\text{Startup}}(Q_1)$ can be determined as

$$T_{\text{Startup}}(Q_1) = \frac{s}{5 \times B} \left\{ \frac{[3-1]^2}{2} + \frac{[5-3]^2}{2} + \frac{[5-5+1]^2}{2} \right\} = 0.9 \times \frac{s}{B}.$$

□

To facilitate the following derivation, $p_i(j)$ is defined as the probability that the user retrieves the j-th

copy of $D_{q^i(1)}$ as the first required data item of Q_i, and $T_{\text{Startup}}(Q_i(j))$ is defined as the average startup time as the user retrieves the j-th copy of $D_{q^i(1)}$ as the first required data item of Q_i. Then, we have the following lemma.

Lemma 2 *In a broadcast program with replication, $p_i(j)$ and $T_{\text{Startup}}(Q_i(j))$ can be formulated as follows:*

$$p_i(j) = \begin{cases} \frac{L^* - a(b) + a(1)}{L^*} : & \text{if } j = 1, \\ \frac{a(j) - a(j-1)}{L^*} : & \text{otherwise.} \end{cases}$$

$$T_{\text{Startup}}(Q^i(j)) = \begin{cases} \frac{L^* - a(b) + a(1)}{2} \times \frac{s}{B} : & \text{if } j = 1, \\ \frac{a(j) - a(j-1)}{2} \times \frac{s}{B} : & \text{otherwise.} \end{cases}$$

With Lemma 2, $T_{\text{Startup}}(Q_i)$ can be formulated as

$$T_{\text{Startup}}(Q_i) = \sum_{j=1}^{n(D_{q^i}(1))} p_i(j) \times T_{\text{Startup}}(Q_i(j)).$$

Finally, the following lemma can be obtained with the above lemmas.

Lemma 3 *The lower bound of $T_{\text{Startup}}(Q_i)$ is $\frac{L^*}{2b} \times \frac{s}{B}$. In addition, the lower bound occurs in the situation that all copies of $D_{q^i(1)}$ are placed in the broadcast program with equal space $\frac{L^*}{b}$. That is, $a(j+1) - a(j) = \frac{L^*}{b}$ for* j = 1, 2, …, $b - 1$ *and $L^* - a(b) + a(1) = \frac{L^*}{b}$.*

3.2.2 Formulation of average waiting time

Let $T_{\text{Wait}}(Q_i, j)$ be the waiting time of Q_i under the condition that the user retrieves the j-th copy of $D_{q^i(1)}$ as the first required data item of Q_i. Thus, the average waiting time of Q_i can be formulated as below.

$$T_{\text{Wait}}(Q_i) = \sum_{j=1}^{n(D_{q^i}(1))} p_i(j) \times T_{\text{Wait}}(Q_i, j)$$

In the above equation, $T_{\text{Wait}}(Q_i, j)$ is too difficult to be obtained by simple mathematical equations. Thus, we devise function CAL$T_{\text{Wait}}(i, j)$ to calculate $T_{\text{Wait}}(Q_i, j)$.

The behavior of function CAL$T_{\text{Wait}}(i, j)$ is as follows. We first set *pos*1 to the position of the j-th replica of the first required data item of Q_i (i.e., $D_{q^i(1)}$) and set *waiting* to 0. We then start from the position *pos*1 to find the nearest (with the shortest distance from position *pos*1) replica of $D_{q^i(2)}$. Suppose that the position of the nearest $D_{q^i(2)}$ replica is *pos*2. We add the distance from *pos*1 to *pos*2 into *waiting*. Then, we set the value of *pos*1 to *pos*2, and start from the position *pos*1 to find the nearest (with the shortest distance from position *pos*1) replica of $D_{q^i(3)}$. The above procedure is repeated until all data items in Q_i have been processed. Finally, $T_{\text{Wait}}(Q_i, j)$ is equal to waiting $\times \frac{s}{B}$. The algorithmic form of function CAL$T_{\text{Wait}}(i, j)$ is as below.

Function CAL$T_{\text{Wait}}(i, j)$

```
1:  waiting←0
2:  Let pos1 be the position of the j-th replica of D_{q^i(1)}
3:  for k=1 to |Q_i| − 1 do
4:    Starting from the position pos1, find the nearest (with the
      shortest distance from position pos1) replica of
      D_{q^i(k+1)}
5:    Let pos2 be the position of the found replica
6:    if pos2 > pos1 then
7:      waiting←waiting + (pos2 − pos1 − 1)
8:    else  /*pos2 ≤ pos1*/
9:      waiting←waiting + (L* − pos1 + pos2 − 1)
10:   pos1←pos2
11:   return waiting × s/B
```

Finally, we use the following calculation to illustrate the behavior of function CAL$T_{\text{Wait}}(i, j)$.

Example 5 Consider the broadcast program shown in Fig. 2(b) and a query $Q_1 = \{D_1, D_2, D_3\}$. The steps to use function CAL$T_{\text{Wait}}(i, j)$ to calculate $T_{\text{Wait}}(Q_1, 1)$ are as follows. The position of the first replica of D_1 is 1. From position 1, we try to find the nearest replica of D_2. As shown in Table 1, the first replica of D_2 is of the shortest distance. The distance from $D_1(1)$ to $D_2(1)$ is position($D_2(1)$) − position($D_1(1)$) − 1 = 2 − 1 − 1 = 0. Then, from position($D_2(1)$), we try to find the nearest replica of D_3 and the first replica of D_3 is of the shortest distance. The distance from $D_2(1)$ to $D_3(1)$ is L^* − position($D_2(1)$) + position($D_3(1)$) − 1 = 5 − 2 + 2 − 1 = 4. Finally, $T_{\text{Wait}}(Q_1, 1)$ is determined as $(0 + 4) \times \frac{s}{B} = 4 \times \frac{s}{B}$. □

3.2.3 Formulation of average retrieval time

Since retrieval time is the time that the mobile device indeed retrieves data from broadcast channels, $T_{\text{Retr.}}(Q_i)$ is independent of the broadcast programs and can be determined as follows.

$$T_{\text{Retr.}}(Q_i) = |Q_i| \times \frac{s}{B}$$

According to the above formulations, we have the following guidelines for designing broadcast program generation algorithms.

1. To reduce average startup time, for each query Q_i, the replicas of $D_{q^i}(1)$ should be placed with equal space.

2. To reduce average waiting time, for each query Q_i, each replica of $D_{q^i(j+1)}$, $j > 1$ should be placed in back of one replica of $D_{q^i(j)}$.
3. Average retrieval time is determined by the query profile and cannot be reduced by broadcast program generation algorithms.
4. According to Eqs. (3, 4) and (5), when two queries suggest conflict replica replacements, the placement suggested by the query with higher access probability is of higher priority than that suggested by the other query.

4 Algorithm BPGR: broadcast program generation with replication

In this section, we propose algorithm BPGR (standing for Broadcast Program Generation with Replication) to employ replication to solve the problem of dependent data allocation for ordered queries based on the design guidelines mentioned in Sect. 3. Algorithm BPGR consists of the following two phases: *replica number determination phase* and *data item allocation phase*. Algorithm BPGR determines the replica number of each data item according to the query profile in replica number determination phase, and allocates the data items into the broadcast program in data item allocation phase. The details of replica number determination phase and data item allocation phase are described in Sects. 4.1 and 4.2, respectively.

4.1 Replica number determination phase

According to the property shown in [48], the average access time for data items of equal size will be minimized if the copies of each data item are equally spaced and for any two data items D_i and D_j,

$$\frac{\text{Freq}}{(D_i)}\text{Freq}(D_j) = \frac{\sqrt{\frac{Pr(D_i)}{\text{Size(D}_\text{i})}}}{\sqrt{\frac{Pr(D_j)}{\text{Size(D}_\text{j})}}}, \tag{6}$$

where $Freq(D_i)$ is the broadcast frequency of D_i and $Pr(D_i)$ is the access frequency of D_i. This property is called square-root property in [48].

Since each data item is of equal size, each query Q_i can be viewed as a data item with size $|Q_i|$. Thus, the ratio of broadcast frequencies of any two queries, Q_i and Q_j is

$$\frac{\text{Freq}}{(Q_i)}\text{Freq}(Q_j) = \frac{\sqrt{\frac{Pr(Q_i)}{|Q_i|}}}{\sqrt{\frac{Pr(Q_j)}{|Q_j|}}}, \text{ where } \sum_{j=1}^{|Q|}\text{Freq}(Q_j) = 1. \tag{7}$$

To facilitate the following discussion, we have the following definition.

Definition 6 A data item D_i is said to be *in* a query Q_j if there exists an integer k so that the k-th broadcast data item of Q_j is D_i (i..e, $D_{q^j(k)} = D_i$). Thus, the *broadcast weight* of data item Weight(D_i) is defined as

$$\text{Weight}(D_i) = \sum_{\forall_{Q_j \text{ where } D_i \text{ is in } Q_j}} Freq(Q_j).$$

Let L^* be the expected length of the broadcast program with replication specified by the system designers. Since all data items should be broadcast at least once, each data item is first assigned one slot and there will be $n \times L^* - |D|$ slots left. Then, the remaining slots are assigned to all data items according to broadcast weights. Therefore, the number of slots assigned to D_i (denoted as $n(D_i)$) can be calculated by the following equation:

$$n(D_i) = \left\lfloor \frac{\text{Weight}}{(D_i)} \sum_{j=1}^{|D|} \text{Weight}(D_j) \times (n \times L^* - |D|) \right\rfloor + 1. \tag{8}$$

Since the *floor* function is used, some slots may be unassigned. Suppose that there are k unassigned slots where $k = n \times L^* - \sum_{i=1}^{|D|} n(D_i)$. These unassigned slots are assigned to the top k weightiest data items.

The desired length of the resultant broadcast program (i.e., L^*) is an important user-specified parameter in algorithm BPGR. However, it is a little bit difficult to specify a proper value of L^*. Thus, *replication factor* (denoted as *rf*), which indicates the degree of replication, is used to aid users to specify a proper value of L^*. Replication factor is defined as the ratio of the number of slots in the broadcast program to the number of data items. Since each data item should appear at least once in the broadcast program, *rf* should be larger than or equal to one. It is clear that when *rf* becomes larger than one, algorithm BPGR has more space to perform replication. The resultant broadcast program is flat when *rf* is set to one. After users specify the value of *rf*, the value of L^* can be determined by the following equation.

$$L^* = \left\lceil \frac{\max\{rf, 1\} \times |D|}{n} \right\rceil \tag{9}$$

Finally, the algorithmic form the proposed replica number determination algorithm is as below.

Procedure REPLICA-NUMBER-DETERMINATION

1: Calculate L^* according to Eq. (9)
2: **for** each query Q_j **do**
3: Calculate Freq(Q_j)
4: **for** each data item D_i **do**
5: Calculate Weight(D_i)

Table b continued

Procedure REPLICA-NUMBER-DETERMINATION

6: **for** each data item D_i **do**
7: Calculate $n(D_i)$ according to Eq. (8)
8: $k \leftarrow n \times L^* - \sum_{i=1}^{|D|} n(D_i)$
9: **for** each data item, say D_i, in the top k weightiest data items **do**
10: $n(D_i) \leftarrow n(D_i) + 1$

4.2 Data item allocation phase

A replica is said to be *allocated* if its position in the broadcast program has been determined. Let $Alloc(D_i)$ be the number of allocated replicas of D_i. For each data item D_i, all its replicas are initially unallocated. That is, $Alloc(D_i) = 0$ for $1 \le i \le |D|$. According to the design guidelines in Sect. 3, queries with higher access probabilities should be of higher priority in data item allocation. Thus, in data item allocation phase, the queries are considered sequentially according to their access probabilities in descending order, and only one query is considered in each iteration.

When Q_j is considered, the data items in Q_j are also considered sequentially. That is, the consideration order of the data items of Q_j is $D_{q^j(1)} \rightarrow D_{q^j(2)} \rightarrow \cdots \rightarrow D_{q^j(|Q_j|)}$. When $D_{q^j(1)}$ is considered, procedure LEADING-ITEM-ALLOCATION is invoked to allocate the replicas of $D_{q^j(1)}$ if there are some unallocated replicas of $D_{q^j(1)}$ (i.e., $Alloc(D_{q^j(1)}) < n(D_{q^j(1)})$). On the other hand, for each $1 < i \le |Q_j|$, procedure SUCCESSIVE-ITEM-ALLOCATION is invoked to allocate the replicas of $D_{q^j(i)}$ if $Alloc(D_{q^j(i)}) < n(D_{q^j(i)})$. Procedure LEADING-ITEM-ALLOCATION is designed to reduce average startup time of a query while procedure SUCCESSIVE-ITEM-ALLOCATION is designed to reduce average waiting time of a query. Finally, for each position p, for $j = 1, 2, \ldots n$, the j-th replica assigned to position p is placed to the p-th slot of the j-th broadcast channel. The details of procedure LEADING-ITEM-ALLOCATION and procedure SUCCESSIVE-ITEM-ALLOCATION are described in the following subsections, while the algorithmic form of data item allocation phase is as below.

Procedure DATA-ITEM-ALLOCATION

1: **for** each data item D_i **do**
2: $Alloc(D_i) \backslash, \leftarrow \backslash, 0$
3: **for** each query Q_j **do**
4: **if** $Alloc(D_{q^j(1)}) < n(D_{q^j(1)})$ **then**
5: Invoke Procedure LEADING-ITEM-ALLOCATION to allocate $D_{q^j(1)}$

Table c continued

Procedure DATA-ITEM-ALLOCATION

6: **for** $i = 2$ to $|Q_j|$ **do**
7: **if** $Alloc(D_{q^j(i)}) < n(D_{q^j(i)})$
8: Invoke Procedure SUCCESSIVE-ITEM-ALLOCATION to allocate $D_{q^j(i)}$
9: **for** $p = 1$ to $|L^*|$ **do**
10: **for** $j = 1$ to n **do**
11: Place the j-th replica assigned to position p into the p-th slot of the j-th broadcast channel

4.2.1 Leading item allocation

Although there are $n(D_{q^j(1)})$ replicas of $D_{q^j(1)}$, only a portion of replicas will be assigned to Q_j. According to Eq. (3), the portion is determined as ratio of the contribution of Q_j on the weight of $D_{q^j(1)}$ (i.e., $\frac{\text{Freq}}{(Q_j)}$ $\text{Weight}(D_{q^j(1)})$). Let *AssignNo* be the number of replicas of $D_{q^j(1)}$ assigned to Q_j. That is, $AssignNo = \frac{\text{Freq}}{(Q_j)} \text{Weight}(D_{q^j(1)}) \times n(D_{q^j(1)})$. In procedure LEADING-ITEM-ALLOCATION, we have to determine the positions of the *AssignNo* replicas of $n(D_{q^j(1)})$. A position p is said *legal* if the number of replicas allocated in position p is smaller than the number of broadcast channels (i.e., n). It means that if position p is not legal, we cannot allocate any data item into position p.

According to whether some replicas of $D_{q^j(1)}$ have been allocated, the behavior of procedure LEADING-ITEM-ALLOCATION can be divided into the following two cases.

Case 1: No replica of $D_{q^j(1)}$ has been allocated

The positions of the replicas of $D_{q^j(1)}$ only affect average startup time of Q_j. According to Lemma 3, the replicas of $D_{q^j(1)}$ should be placed with equal space in order to minimize average startup time. Thus, the best interval between every two consecutive replicas of $D_{q^j(1)}$ should be $\frac{L^*}{AssignNo}$.

The proposed allocation method to allocate *AssignNo* replicas of $D_{q^j(i)}$ is as below. We first find the smallest legal position *pos* and allocate the first replica of $D_{q^j(i)}$ in position *pos*. Then, we will try to allocate the next replica in position $pos + \frac{L^*}{AssignNo}$. Since position $pos + \frac{L^*}{AssignNo}$ may be illegal, we will find a legal position around $pos + \frac{L^*}{AssignNo}$ in a zig-zag manner. That is, the order of searching a legal position is $pos + \frac{L^*}{AssignNo} \rightarrow pos + \frac{L^*}{AssignNo} + 1 \rightarrow pos + \frac{L^*}{AssignNo} - 1 \rightarrow pos + \frac{L^*}{AssignNo} + 2 \rightarrow pos + \frac{L^*}{AssignNo} - 2 \cdots$. Then, the replica is placed in the first legal position we meet, and the value of *pos* is updated to the legal position found. The above method is applied repeatedly until all assigned replicas are allocated.

Case 2: Some replicas of $\mathbf{D_{q^j(1)}}$ have been allocated
To minimize $T_{\text{Startup}}(Q_j)$, the replicas of $D_{q^j(1)}$ should be placed with equal space. Since $Alloc(D_{q^j(1)})$ replicas have been allocated, what we should do is inserting *AssignNo* replicas into these allocated replicas so that the spaces of the allocated replicas of $D_{q^j(1)}$ are as equal as possible.
The proposed allocation method is as follows. Only one replica is allocated in each iteration. In each iteration, we first calculate the lengths of the intervals of every two consecutive allocated replicas of $D_{q^j(1)}$. Let p be the middle point of the longest interval. We then find a legal position *pos* around p in a zig-zag manner, and place the replica in position *pos*. The above method is applied repeatedly until *AssignNo* replicas of $D_{q^j(1)}$ have been allocated. The algorithmic form of procedure LEADING-ITEM-ALLOCATION is as below.

Procedure LEADING-ITEM-ALLOCATION $(D_{q^j(1)})$

1: $AssignNo \leftarrow \frac{\text{Freq}(Q_j)}{\text{Weight}(D_{q^j(1)})} \times n(D_{q^j(1)})$
2: **if** $Alloc(D_{q^j(1)}) = 0$ **then** /*Case 1*/
3: $pos \leftarrow$ the smallest legal position
4: Place a replica of $D_{q^j(1)}$ in position *pos*
5: **for** k = 2 to *AssignNo* **do**
6: Find legal position p around $pos + \frac{L^*}{AssignNo}$ in a zig-zag manner
7: $pos \leftarrow p$
8: Place a replica of $D_{q^j(1)}$ in position *pos*
9: **else**/* Case 2*/
10: **for** k = 1 to *AssignNo*
11: Calculate the lengths of every two consecutive replicas of $D_{q^j(1)}$
12: Let p be the middle point of the longest interval
13: Find a legal position *pos* around p in a zig-zag manner
14: Place a replica of $D_{q^j(1)}$ in position *pos*
15: $Alloc(D_{q^j(1)}) \leftarrow Alloc(D_{q^j(1)}) + AssignNo$

4.2.2 Successive item allocation

We now consider the allocations of the replicas of $D_{q^j(i)}$ for $i > 1$. Let *AssignNo* be the number of replicas of $D_{q^j(i)}$ assigned to Q_j and $AssignNo_{Prev}$ be the number of replicas of $D_{q^j(i-1)}$ assigned to Q_j. According to the relationship of *AssignNo* and $AssignNo_{Prev}$, the behavior of procedure SUCCESSIVE-ITEM-ALLOCATION can be divided into the following two cases.

Case 1: The number of assigned $\mathbf{D_{q^j(i)}}$ replicas is larger than or equal to that of assigned $\mathbf{D_{q^j(i-1)}}$ replicas (i.e., $AssignNo \geq AssignNo_{Prev}$)

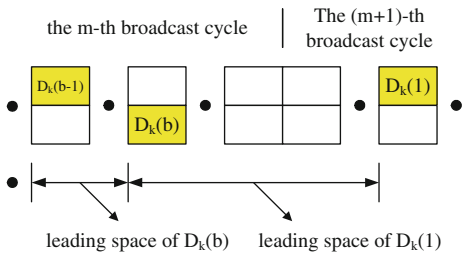

Fig. 5 An illustration of leading space

According to the guidelines in Sect. 3, since the devices will retrieve $D_{q^j(i-1)} \rightarrow D_{q^j(i)}$ sequentially when the user submits Q_j, one replica of $D_{q^j(i)}$ should be placed in back of a replica of $D_{q^j(i-1)}$ in order to reduce average waiting time of Q_j. Since the number of assigned $D_{q^j(i)}$ replicas is larger than or equal to the number of assigned $D_{q^j(i-1)}$ replicas (i.e., $AssignNo \geq AssignNo_{Prev}$), we first place one replica of $D_{q^j(i)}$ in back of each allocated replica of $D_{q^j(i-1)}$ and several replicas of $D_{q^j(i)}$ may remain unallocated. We then use the method in Case 2 of procedure LEADING-ITEM-ALLOCATION to place the remaining $D_{q^j(i)}$ replicas to further reduce the average startup time of $D_{q^j(i)}$.
Case 2: The number of assigned $\mathbf{D_{q^j(i)}}$ replicas is smaller than that of assigned $\mathbf{D_{q^j(i-1)}}$ replicas (i.e., $AssignNo < AssignNo_{Prev}$)
Since the number of assigned $D_{q^j(i)}$ replicas is smaller than the number of assigned $D_{q^j(i-1)}$ replicas, we can only place the assigned $D_{q^j(i)}$ replicas in back of a portion of assigned $D_{q^j(i-1)}$ replicas. We use the *leading space* of each $D_{q^j(i-1)}$ replica to determine the importance of the assigned $D_{q^j(i)}$ replicas. As shown in Fig. 5, the leading space of a $D_{q^j(i-1)}$ replica, say p, is defined as the distance between p and the $D_{q^j(i-1)}$ replica in front of p. A $D_{q^j(i-1)}$ replica with longer leading space indicates that the $D_{q^j(i-1)}$ replica is of higher likelihood to be retrieved when a mobile device is reading all data items of Q_j. Thus, when placing the assigned $D_{q^j(i)}$ replicas, we first calculate the leading space of the assigned $D_{q^j(i-1)}$ replicas, pick the $AssignNoD_{q^j(i-1)}$ replicas with longest leading space, and place one $D_{q^j(i)}$ replica in back of each picked $D_{q^j(i-1)}$ replica. The algorithmic form of procedure SUCCESSIVE-ITEM-ALLOCATION is as below.

Procedure SUCCESSIVE-ITEM-ALLOCATION $(D_{q^j(i)})$

1: $AssignNo \leftarrow \frac{\text{Freq}(Q_j)}{\text{Weight}(D_{q^j(i)})} \times n(D_{q^j(i)})$
2: $AssignNo_{Prev} \leftarrow \frac{\text{Freq}(Q_j)}{\text{Weight}(D_{q^j(i-1)})} \times n(D_{q^j(i-1)})$
3: **if** $AssignNo \geq AssignNo_{Prev}$ **then** /*Case 1*/
4: **for** each replica of $D_{q^j(i-1)}$, say r, which is assigned to Q_j
5: Let p be the position of r

Table e continued

Procedure SUCCESSIVE-ITEM-ALLOCATION ($D_{q^j(i)}$)

6: Find the a legal position *pos* in the order $p+1 \backslash, \rightarrow \backslash, p+2 \backslash, \rightarrow \backslash, p+3 \ldots$
7: Place a replica of $D_{q^j(i)}$ in the first legal position we meet
8: **for each** unplaced assigned replica of $D_{q^j(i)}$ **do**
9: Calculate the lengths of every two consecutive replicas of $D_{q^j(1)}$
10: Let p be the middle point of the longest interval
11: Find a legal position *pos* around p in a zig-zag manner
12: Place a replica of $D_{q^j(1)}$ in position *pos*
13: **else** /*Case 2 */
14: Calculate the leading space of the replicas of $D_{q^j(i-1)}$ which are assigned to Q_j
15: **for** $k = 1$ to *AssignNo* **do**
16: Let r the replica of $D_{q^j(i-1)}$ with the largest leading space.
17: Let p be the position of r
18: Find the a legal position *pos* in the order $p+1 \rightarrow p+2 \rightarrow p+3 \ldots$
19: Place a replica of $D_{q^j(i)}$ in the first legal position we meet
20: Recalculate the leading space of all $D_{q^j(i-1)}$ replicas
21: $Alloc(D_{q^j(i)}) \leftarrow Alloc(D_{q^j(i)}) + AssignNo$

4.3 Time complexity

We analyze the worst case time complexity of algorithm BPGR in this subsection. There are four loops in procedure REPLICA-NUMBER-DETERMINATION. The time complexity of the first loop is $O(|Q|)$, while the time complexities of the other three loops are $O(|D|)$. Therefore, the time complexity of the procedure REPLICA-NUMBER-DETERMINATION is $O(|Q| + 3 \times |D|) = O(|Q| + |D|)$. Let the average value of *AssignNo* be *AvgAssignNo*. On average, the loops in procedure LEADING-ITEM-ALLOCATION will iterate *AvgAssignNo* runs. In each iteration, procedure LEADING-ITEM-ALLOCATION will find a legal position within an interval with average length $\frac{L^*}{AvgAssignNo}$ in line 6. Thus, the time complexity of procedure LEADING-ITEM-ALLOCATION is $O(AvgAssignNo \times \frac{1}{2} \times \frac{L^*}{AvgAssignNo}) = O(L^*)$. We can derive the time complexity of procedure SUCCESSIVE-ITEM-ALLOCATION with similar analysis, and the complexity of procedure SUCCESSIVE-ITEM-ALLOCATION is also $O(L^*)$. Let the average query length be *AvgQLen*. In procedure DATA-ITEM-ALLOCATION, procedure LEADING-ITEM-ALLOCATION will be invoked to allocate the leading item of each query and procedure SUCCESSIVE-ITEM-ALLOCATION will be invoked to allocate the rest required data items of the query. Thus, the time complexity of procedure DATA-ITEM-ALLOCATION is $O(|D| + |Q| \times (O(L^*) + (AvgQLen - 1) \times O(L^*))) = O(|D| + AvgQLen \times |Q| \times L^*)$. Finally, the time complexity of algorithm BPGR is $O(|Q| + |D|) + O(|D| + AvgQLen \times |Q| \times L^*) = O(|D| + AvgQLen \times |Q| \times L^*)$.

In practice, it is unnecessary for algorithm BPGR to consider all queries in the query profile due to the reason that the access frequencies of all queries are usually skewed [10] (i.e., only several queries are of high access probabilities and many queries are of very small access probabilities). According to Eq. (1), optimizing broadcast programs for queries with very small access probabilities does not have significant contribution on minimizing average access time. Thus, for the cases that the number of queries is very large, the execution time of algorithm BPGR is able to be controlled in an acceptable range by ignoring the queries with small access probabilities. For example, the users can ask algorithm BPGR to ignore the queries whose access probabilities are smaller than a threshold (e.g., $\frac{1}{10,000}$). Besides, users can select top-k queries such that the summation of the access probabilities of the k selected queries is larger than a threshold (e.g., 80%), and ask algorithm BPGR to consider the selected queries only.

4.4 Integration with index allocation

To the best of our knowledge, there is no study dealing with data allocation and index allocation together in the literature of *dependent data broadcast*. It is due to the reason that in dependent data broadcast, considering data allocation and index allocation together usually makes the problem too difficult to be dealt with. Therefore, a two-phase approach [6, 29], consisting of data allocation phase and index allocation phase, is usually applied to simplify the process of generating broadcast programs with index. The responsibility of data allocation is to organize data items (e.g., perform replication, place data items/replicas…) into broadcast programs without index to minimize average access time, while the responsibility of index allocation is to generate broadcast programs with index by inserting index items into broadcast programs without index to reduce average tuning time at the cost of increasing average access time.

The effect of index items is not considered in algorithm BPGR and its competitors [23, 26] since all these algorithms focus on data allocation, and the effect of index items will be considered in the algorithm used in index allocation phase. It is true that there is a trade-off between average access time and average tuning time, and it is the index allocation algorithm's responsibility to strike a balance between them. We use $(1, m)$ indexing [28] as an example to demonstrate how to integrate algorithm BPGR with an index allocation algorithm. In $(1, m)$ indexing, all index information is clustered into one index item, say I,

Table 4 System parameters

Parameters	Values
Size of each data item (s)	8 kb
Bandwidth of each channel (B)	80 kb/s.
Number of data items ($\|D\|$)	2000
Number of channels (n)	6
Replication factor (rf)	3
Number of queries ($\|Q\|$)	100
Average query length	10
Zipf distribution (θ)	0.75
Data error rate	0%
Simulation time	12 h

and the size of the index item, denoted as $|I|$, is usually smaller than the size of a data item [6]. A broadcast program without index is first divided into m equal-size partitions and one replica of the index item I is inserted into the front of each partition. Based on [28], the value of m is suggested to be $\sqrt{\frac{L^* \times s}{|I|}}$ to achieve a better balance between average access time and average tuning time. If the sizes of the index item and the data item are close, the index item can be treated as a special data item. Thus, we can extend each query $\{D_{q^i(1)}, D_{q^i(2)}, \ldots, D_{q^i(|Q_i|)}\}$ to $\{I, D_{q^i(1)}, D_{q^i(2)}, \ldots, D_{q^i(|Q_i|)}\}$ and apply algorithm BPGR to generate broadcast programs with index.

5 Performance evaluation

5.1 The simulation model

For performance studies, we implemented algorithm BPGR with JSIM[3], and also a query profile generator based on the approach mentioned in [40]. The probability of the query Q_i issued by users is assumed to be $Pr(Q_i) = \frac{\left(\frac{1}{i}\right)^{\theta}}{\sum_{j=1}^{n}\left(\frac{1}{j}\right)^{\theta}}$ where θ is the parameter of the Zipf distribution. The default value of θ is set to 0.75 according to the analyses of real Web traces [10]. Similar to [43], the number of channels and data items are set to six and 2,000, respectively. The channel bandwidth is set to 80 kb/s and the data size is set to 8 kb [26]. By default, we consider ideal broadcast channels by setting data error rate to zero. In addition, the average query length and query number are set to 10 and 100, respectively [26]. The simulation executes for 12 h. For better readability, Table 4 shows the system parameters in our experiments.

In addition to algorithm BPGR, algorithm MULS is also implemented. As shown in [26], algorithm MULS is of the best performance among all other broadcast program generation algorithms for ordered queries. The experimental results are shown in the following subsections. We use average access time as the performance metric of both algorithms. The performance gain of algorithm BPGR is defined as the reduction rate of algorithm BPGR over algorithm MULS in average access time. Thus, performance gain is formulated as

$$\frac{\text{Avg. access time of algorithm MULS} - \text{Avg. access time of algorithm BPGR}}{\text{Avg. access time of the algorithm MULS}}.$$

Since algorithm MULS contains several parameters to adjust execution time, for comparison purpose, the execution time of algorithm MULS is controlled to be equal to that of algorithm BPGR. Thus, only the execution time of algorithm BPGR is drawn in the following figures.

5.2 Effect of channel number

Figure 6 shows the experimental results with the number of broadcast channels varied. We observe from Fig. 6(a) that the average access time of both algorithms decreases as the number of channels increases. This agrees with the intuition that increasing bandwidth will decrease the average access time. However, the improvement in average access time decreases as the number of broadcast channels increases. As a result, the determination of the number of broadcast channels has to strike a compromise between the performance gain and the number of channels used. The number of broadcast channels suggested by this experiment is around four. In addition, algorithm BPGR leads to better broadcast programs than algorithm MULS even when algorithm MULS uses more broadcast channels. In this experiment, the average access time of algorithm BPGR with four broadcast channels is close to that of algorithm MULS with eight channel channels. This result shows that the higher bandwidth utilization can be attained by employing replication.

Figure 6(b) shows that the execution time of algorithm BPGR increases as channel number decreases. It is because

[3] http://www.cs.uga.edu/jam/jsim/.

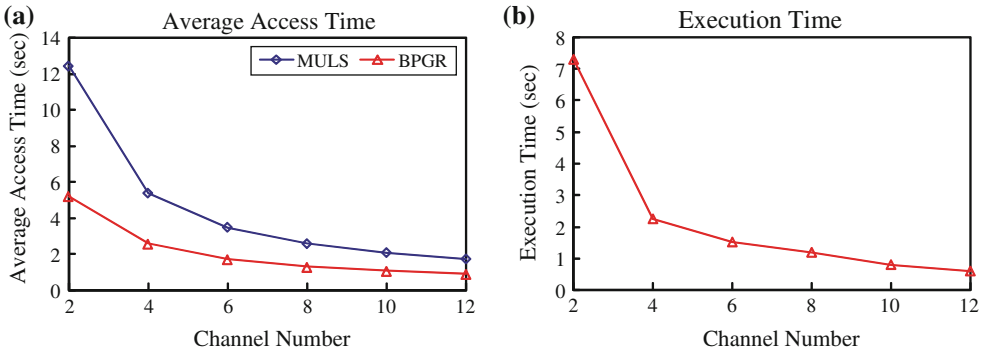

Fig. 6 Effect of channel number

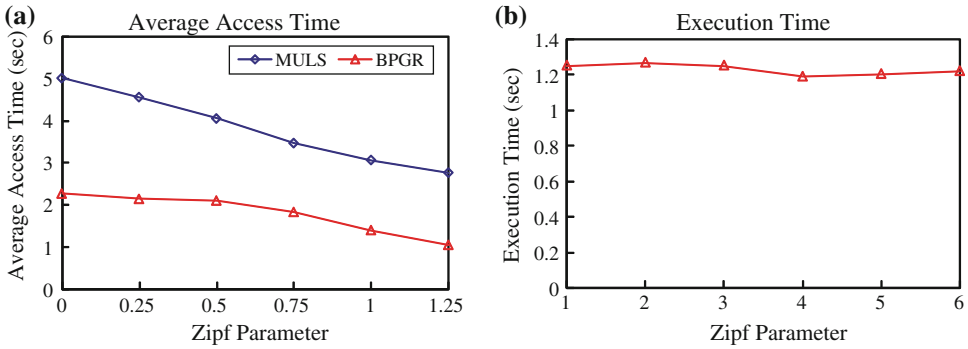

Fig. 7 Effect of query skewness

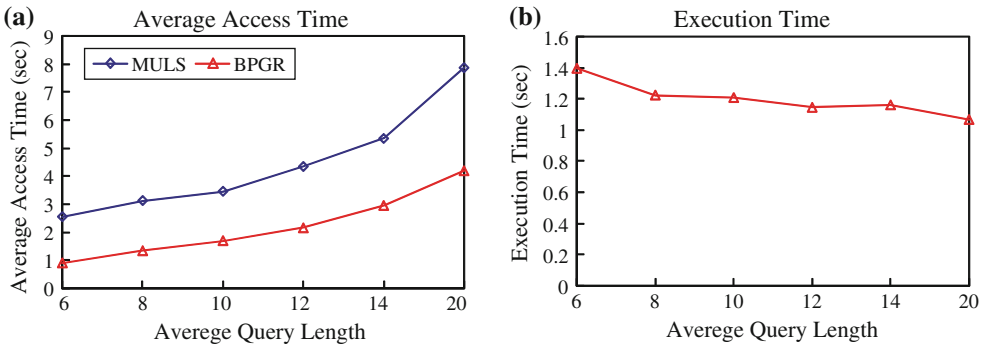

Fig. 8 Effect of average query length

that with the same replication factor, fewer broadcast channels imply longer broadcast programs. Algorithm BPGR has more possibilities to place each replica, thus increasing the execution time. In our experiment, the execution time of algorithm BPGR increases from 0.58 to 7.3 s as channel number decreases from 12 to 2.

5.3 Effect of query skewness

We use Zipf distribution to model skewness of queries. As observed in [41], θ is usually larger than one for busy web sites. Thus, we set θ from 0 to 1.25 to investigate the effect of query skewness to both algorithms. Note that setting θ to 0 indicates that the access frequencies of all queries are set to be equal.

From Fig. 7(a), we observe that the average access time of both algorithms decreases as the query frequencies become more skewed. It is because that when query frequencies become more skewed, optimizing a smaller amount of queries is enough to minimize average access time of all queries. Thus, both algorithms perform well when query skewness is high. Algorithm BPGR outperforms algorithm MULS in all cases, showing the advantage of employing replication in broadcast programs. In addition, the performance gain of algorithm BPGR is between 48.16 and 62.61%. As shown in Fig. 8(b), query skewness does not significantly affect on the execution time of algorithm BPGR. The execution time of algorithm BPGR is ranging from 1.19 to 1.27 s in this experiment.

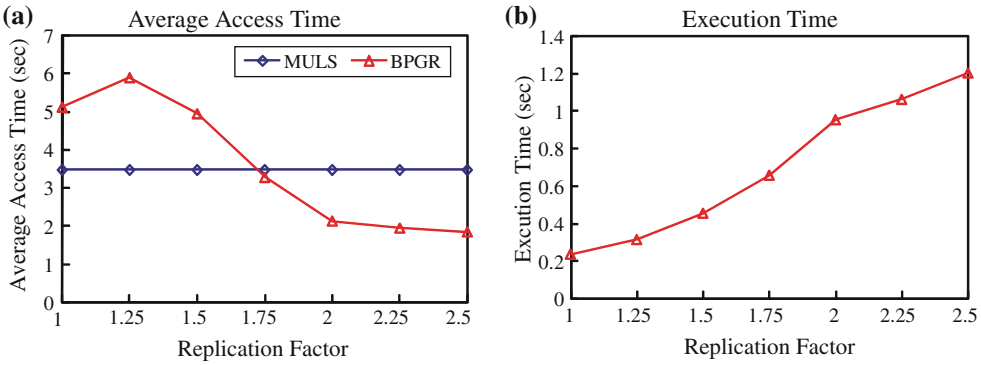

Fig. 9 Effect of replication factor

5.4 Effect of average query length

Given a query profile, average query length is used as an indicator to measure strength of data item dependency. A query profile of high average query length means that the data items are highly dependent upon others. In this experiment, average query length is set from 6 to 20.

Figure 8(a) shows the average access time of both algorithms as average query length varies. We observe that the average access time of both algorithms becomes longer as the data dependency of query profiles becomes stronger. The reason is that in a query profile with higher data dependency, the relationship among data items is more entangled and minimizing average access time becomes more difficult. In our experiment, the average access time of algorithm MULS increases from 2.56 to 7.85 s while the average access time of algorithm BPGR is from 0.89 to 4.17 s. In addition, the performance gain of algorithm BPGR slightly decreases from 64.96 to 46.84%. As shown in Fig. 9(b), average query length only slightly affects the execution time of algorithm BPGR.

5.5 Effect of replication factor

We now consider the effect of the degree of replication on average access time and execution time. In this experiment, replication factor is set from 1 to 2.5. Setting replication factor to 1 indicates that replication is disabled in algorithm BPGR.

Figure 9(a) shows the average access time of both algorithms as the value of replication factor varies. Since not employing data replication, algorithm MULS is not affected by the change of replication factor. We observe that when replication factor is smaller than 1.75, algorithm BPGR performs worse than algorithm MULS because algorithm BPGR has insufficient free slots to allocate replicas. When replication factor is larger than 1.75, algorithm BPGR has enough free slots to allocate the replicas into good positions, thus being able to generate broadcast programs of high quality. In addition, the improvement of performance gain of algorithm BPGR becomes insignificant as replication factor is larger than 2.

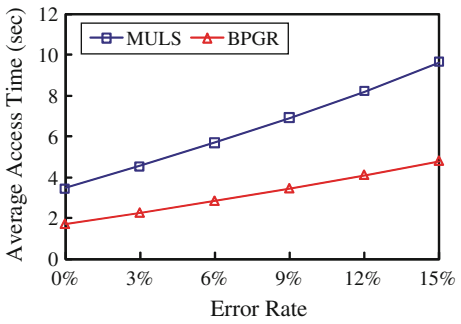

Fig. 10 Effect of data error rate

Figure 9(b) shows that the execution time of algorithm BPGR increases as replication factor is getting larger. It is because that high replication factor indicates that there are many slots to place replicas, thus increasing the execution time of algorithm BPGR. In our experiment, the execution time of algorithm BPGR is still under 2 s even replication factor is set to 2.5. Since replication factor is a user-controllable parameter, the value of replication factor should be set carefully to strike a balance between the quality of broadcast programs and execution time.

5.6 Effect of data error rate

Since wireless networks are usually error-prone, we investigate in this subsection how data error rate affects the performance of algorithm BPGR and algorithm MULS. Since data error rate does not affect the execution time of both algorithms, only average access time is given. We set data error rate from 0 to 15% and the experimental results is shown in Fig. 10. It is obvious that the average access time of both algorithms increase as data error rate increases. Since each data item may have multiple replicas in broadcast programs, the performance degradation of algorithm BPGR caused by data error is slighter than the performance degradation of algorithm MULS. In our

experiment, as data error rate is set from 0 to 15%, the average access time of algorithm BPGR increases from 1.69 to 4.79 s while the average access time of algorithm MULS increases from 3.45 to 9.62 s. In addition, the performance gain of algorithm BPGR over algorithm MULS is around 50%. This result shows that besides reducing average access time, employing replication in broadcast programs also makes broadcast programs more resilient to data error, showing the advantage of algorithm BPGR in error-prone environments.

6 Conclusion

We explored in this paper the problem of broadcasting dependent data in multiple broadcast channels, and explicitly investigated the effect of data replication. By analyzing the model of dependent data broadcasting, we formulated the average access time of a broadcast program in a multiple channel environment. In light of the theoretical results, a two-phase algorithm called BPGR is developed to generate broadcast programs with replicas. Our experimental results showed that, in most cases, algorithm BPGR outperformed algorithm MULS in terms of the average access time of the resultant broadcast programs. In addition, experimental results also showed that in error-prone environments, performance degradation of algorithm BPGR is slighter than that of algorithm MULS, showing the robustness of algorithm BPGR over algorithm MULS and the advantage of replication in broadcast programs.

Appendix: Proof of all Lemmas

Proof of Lemma 1 Consider a broadcast program with replication and a query Q_i. Suppose that the first required data item of Q_i is D_k (i.e., $k = q^i(1)$). Since each data item is of multiple replicas, the mobile device will retrieve the nearest copy of D_k in order to minimize the access time of the Q_i. Without loss of generality, we order each copy of D_k by its position in the broadcast program, and let $a(j)$ be position($D_k(j)$).

Suppose that a mobile user submits Q_i in the m-th broadcast cycle and the time interval between the start time of the m-th broadcast cycle and the moment when the user submits Q_i to be x. As shown in Fig. 11, the broadcast program can be divided into $b + 1$ parts, where b is the number of replicas of D_k, by the start time of each copy of D_k. It is obvious that the mobile device will read $D_k(j)$ when x lies on part j, $1 \le j \le b$. In addition, the mobile device will read $D_k(1)$ in $(m + 1)$-st broadcast cycle when x lies on part $(b + 1)$.

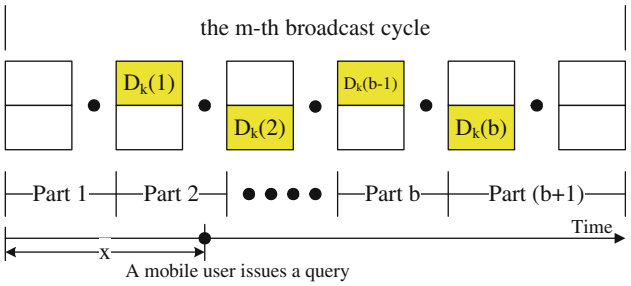

Fig. 11 The illustration for the Proof of Lemma 1 and 2

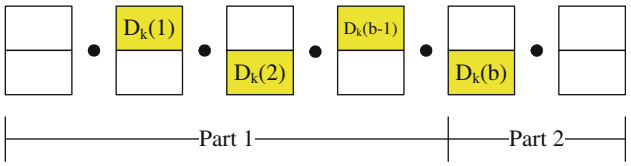

Fig. 12 The illustration for the Proof of Lemma 3

Recall that the time interval between the start time of the m-th broadcast cycle and the moment of the appearance of the j-th copy of D_k is $(\text{position}(\text{D}_\text{k}(\text{j})) - 1) \times \frac{\text{s}}{\text{B}} = (\text{a}(\text{j}) - 1) \times \frac{s}{B}$. Suppose that x is a uniform distribution over $(0, L^*)$. The average startup time of Q_i on the broadcast program with replication can be formulated as below.

$$
\begin{aligned}
&T_{\text{Startup}}(Q_i) \\
&= \frac{1}{L^*} \times \left\{ \int_{x=0}^{a(1)-1} [a(1) - 1 - x]dx \right. \\
&\qquad + \int_{x=a(1)-1}^{a(2)-1} [a(2) - 1 - x]dx \\
&\qquad + \cdots + \int_{x=a(b-1)-1}^{a(b)-1} [a(b) - 1 - x]dx \\
&\qquad \left. + \int_{x=a(b)-1}^{L^*} [L^* - x + a(1) - 1]dx \right\} \times \frac{s}{B} \\
&= \frac{s}{L^* \times B} \times \left\{ \sum_{i=1}^{b-1} \frac{[a(i+1) - a(i)]^2}{2} + \frac{[L^* - a(b) + a(1)]^2}{2} \right\}
\end{aligned}
$$

□

Proof of Lemma 2 Consider the same scenario in the Proof of Lemma 1 and Fig. 11. Suppose that the first required data item of Q_i is D_k (i.e., $k = q^i(1)$) and the mobile device will try its best to minimize the access time. It is obvious that the mobile device will retrieve the j-th copy of D_k, $j = 2, 3, \ldots, b$, as the first required data item when x lies on part j. In addition, the mobile device will retrieve the first copy of D_k when x lies on part 1 and part $b + 1$. Suppose that x is a uniform distribution over $(0, L^*)$.

The probability that x lies on part j is equal to the ratio of the length of part j over the length of the broadcast program. Therefore, we can formulate $p_i(j)$ as follows:

$$p_i(j) = \begin{cases} \frac{L^* - a(b) + a(1)}{L^*} & : \text{ if } j = 1, \\ \frac{a(j) - a(j-1)}{L^*} & : \text{ otherwise.} \end{cases}$$

Since x is a uniform distribution over $(0, L^*)$, for $j = 1$, $T_{\text{Startup}}(Q^i(j))$ can be formulated as
Similarly, for $j = 2, 3, \ldots, b$, $T_{\text{Startup}}(Q^i(j))$ can be formulated as

$$\begin{aligned} &T_{\text{Startup}}(Q^i(j)) \\ &= \frac{1}{L^* - a(b) + a(1)} \\ &\quad \times \left(\int_{x=a(b)}^{L^*} (L^* - x + a(1))dx + \int_{x=0}^{a(1)} (a(1) - x) \right) dx \times \frac{s}{B} \\ &= \frac{1}{L^* - a(b) + a(1)} \\ &\quad \times \left[(L^* - a(1)) \times \left(\frac{1}{2}L^* + a(1) - \frac{1}{2}a(b) \right) + \frac{1}{2}a(1)^2 \right] \times \frac{s}{B} \\ &= \frac{L^* - a(b) + a(1)}{2} \times \frac{s}{B}. \end{aligned}$$

$$\begin{aligned} &T_{\text{Startup}}(Q^i(j)) \\ &= \frac{1}{a(j) - a(j-1)} \times \int_{x=a(j-1)}^{a(j)} (a(j) - x)dx \times \frac{s}{B} \\ &= \frac{1}{a(j) - a(j-1)} \times \left(a(j) \int_{x=a(j-1)}^{a(j)} 1dx - \int_{x=a(j-1)}^{a(j)} xdx \right) \times \frac{s}{B} \\ &= \frac{a(j) - a(j-1)}{2} \times \frac{s}{B}. \end{aligned}$$

To summarize the above results, we have

$$T_{\text{Startup}}(Q^i(j)) = \begin{cases} \frac{L^* - a(b) + a(1)}{2} \times \frac{s}{B} & : \text{ if } j = 1, \\ \frac{a(j) - a(j-1)}{2} \times \frac{s}{B} & : \text{ otherwise.} \end{cases}$$

□

Proof of Lemma 3 We prove this lemma by mathematical induction on the value of b.

Basis

We assume that the first data item of Q_i is D_k (that is, $D_k = D_{q^i(j)}$) and the number of the replicas D_k is b. Consider the case that $b = 1$. In this case, there is the only one copy of D_k in the broadcast program. On average, the user has to wait for a half broadcast cycle to retrieve D_k. Thus, $T_{\text{Startup}}(Q_i) = \frac{L^*}{2} \times \frac{s}{B} = \frac{L^*}{2b} \times \frac{s}{B}$, and Lemma 3 is true when $b = 1$.

Consider the case that $b = 2$. Without loss of generality, we reorder the replicas of D_k based on their positions in ascending order and assume that the first copy of D_k is placed in the first slot of the broadcast program (i.e., $a(1) = 1$). Thus, the $T_{\text{Startup}}(Q_i)$ can be formulated as

$$\begin{aligned} &T_{\text{Startup}}(Q_i) \\ &= \sum_{j=1}^{b} (p_i(j) \times Q^i(j)) \\ &= \left[\frac{a(2) - a(1)}{L^*} \times \frac{a(2) - a(1)}{2} + \frac{L^* - (a(2) - a(1))}{L^*} \right. \\ &\quad \left. \times \frac{L^* - (a(2) - a(1))}{2} \right] \times \frac{s}{B} \\ &= \frac{1}{2 \times L^*} \times (a(2)^2 - 2a(2) + 1) + \frac{1}{2 \times L^*} \\ &\quad \times (L^{*2} - 2 \times L^* \times (a(2) - 1) + (a(2) - 1)^2) \times \frac{s}{B} \\ &= \frac{1}{2 \times L^*} \times \left(2a(2)^2 - (2L^* + 4)a(2) + L^{*2} + 2L^* + 2 \right) \times \frac{s}{B}. \end{aligned} \tag{10}$$

Since only $a(2)$ is unknown in the above equation, we have $T''_{\text{Startup}}(Q_i) = \frac{2}{L^*} \times \frac{s}{B} > 0$. Thus, $T_{\text{Startup}}(Q_i)$ has a minimal value and the minimal value is in the point that $T'_{\text{Startup}}(Q_i) = 0$. By the following derivation, the minimal value of $T_{\text{Startup}}(Q_i)$ is in the point that $a(2) = \frac{L^*}{2} + 1$.

$$\begin{aligned} T'_{\text{Startup}}(Q_i) &= 0 \\ 4a(2) - (2L^* + 4) &= 0 \\ a(2) &= \frac{L^*}{2} + 1 \end{aligned}$$

Then, we have $a(2) - a(1) = \frac{L^*}{2}$ and $L^* - a(b) + a(1) = \frac{L^*}{2}$. In addition, by substituting $a(2) = \frac{L^*}{2} + 1$ into Eq. (10), we obtain that the lower bound of $T_{\text{Startup}}(Q_i)$ is $\frac{L^*}{4} \times \frac{s}{B} = \frac{L^*}{2b} \times \frac{s}{B}$. Thus, Lemma 3 is true when $b = 2$.

Induction step

Assume that Lemma 3 is true for $b = 1, 2, \ldots, k$. We would like to use this hypothesis to prove that Lemma 3 is also true when $b = k + 1$. Without loss of generality, we assume that $a(1) = 1$. To facilitate the following derivation, we reorder the replicas of D_k based on their positions in ascending order. In addition, as shown in Fig. 12, we divide the broadcast program into two parts according to the position of the b-th replica of D_k (i.e., $a(b)$).

Part 1 is equivalent to a broadcast program of length $a(b) - 1$ and with $b - 1$ replicas of D_k. Based on the hypothesis, the lower bound of the average startup time of part 1 is $\frac{a(b)-1}{2(b-1)} = \frac{a(b)-1}{2k}$. In addition, we also have

$$a(j+1)-a(j)=\frac{a(b)-1}{b}=\frac{a(b)-1}{k},\quad \text{for } j = 1,2,\ldots,b-2, \text{and} \tag{11}$$

$$a(b-1)=\frac{a(b)-1}{k}\times(b-2)+1 = \frac{k-1}{k}\times(a(b)-1)+1. \tag{12}$$

Thus, $T_{\text{Startup}}(Q_i)$ can be formulated as

$$\begin{aligned} T_{\text{Startup}}(Q_i) &= \left[\frac{a(b)-1}{L^*}\times\frac{a(b)-1}{2k}+\frac{L^*-a(b)+1}{L^*}\right. \\ &\quad \left.\times\frac{L^*-a(b)+1}{2}\right]\times\frac{s}{B} \\ &= \frac{1}{2\times L^*}\times\left[\left(1+\frac{1}{k}\right)a(b)^2-\left(\frac{2}{k}+2L^*+2\right)a(b)\right. \\ &\quad \left.+\frac{1}{k}+L^{*2}+2L^*+1\right]\times\frac{s}{B} \end{aligned} \tag{13}$$

Similarly, only $a(2)$ is unknown in the above equation. Since $T''_{\text{Startup}}(Q_i)=\frac{1+\frac{1}{k}}{L^*}\times\frac{s}{B}>0$. the minimal value of $T_{\text{Startup}}(Q_i)$ is in the point that $T'_{\text{Startup}}(Q_i)=0$. By the following derivation, the minimal value of $T_{\text{Startup}}(Q_i)$ is in the point that $a(b)=\frac{k}{k+1}L^*+1$.

$$\begin{aligned} &T'_{\text{Startup}}(Q_i)=0 \\ &\frac{1}{2\times L^*}\times\left[\left(2+\frac{2}{k}\right)a(b)-\left(\frac{2}{k}+2L^*+2\right)\right]\times\frac{s}{B}=0 \\ &a(b)=\frac{\frac{2}{k}}{+}2L^*+22+\frac{2}{k}=\frac{k}{k+1}L^*+1 \end{aligned}$$

By substituting the value of $a(b)$ into Eq. (12), we have $a(b-1)=\frac{k-1}{k}\times(a(b)-1)+1=\frac{k-1}{k+1}L^*+1$ and $a(b)-a(b-1)=\frac{1}{k+1}L^*$. The following equations can be obtained by summarizing this result and Eq. (11).

$$\begin{aligned} &a(j+1)-a(j)=\frac{a(b)-1}{k}=\frac{L^*}{k+1}, j=1,2,\ldots b-1 \\ &L^*-a(b)+a(1)=\frac{L^*}{k+1}=\frac{L^*}{b} \end{aligned}$$

By substituting $a(b)=\frac{k}{k+1}L^*+1$ into Eq. (13), we obtain that the lower bound of $T_{\text{Startup}}(Q_i)$ is $\frac{L^*}{2(k+1)}\times\frac{s}{B}=\frac{L^*}{2b}\times\frac{s}{B}$. To summarize the above results, Lemma 3 is true for $b=k+1$ when Lemma 3 is true for $b=1, 2,\ldots, k$.

Finally, Lemma 3 is proven by mathematical induction.

References

1. Acharya, S., Alonso, R., Franklin, M., & Zdonik, S. (1995, March). Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the ACM SIGMOD Conference* (pp. 198–210).
2. Acharya, S., & Muthukrishnan S. (1998, October). Scheduling On-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking* (pp. 43–94).
3. Agrawal, M., Manjhi, A., Bansal, N., & Seshan, S. (2003, March–April). Improving web performance in broadcast-unicast networks. In *Proceedings of the IEEE INFOCOM Conference*.
4. Aksoy, D., & Franklin, M. J. (1998, March). Scheduling for large-scale on-demand data broadcasting. In *Proceedings of IEEE INFOCOM Conference* (pp. 651–659).
5. Aksoy, D., Franklin, M. J., & Zdonik, S. (2001, September). Data staging for on-demand broadcast. In *Proceedings of the 27th International Conference on Very Large Data Bases*, (pp. 571–580).
6. Amarmend, D., Aritsugi, M., & Kanamori, Y. (2006, April). An air index for data access over multiple wireless broadcast channels. In *Proceedings of the 22nd IEEE International Conference on Data Engineering*.
7. Anticaglia, S., Barsi, F., Bertossi, A. A., Iamele, L., & Pinotti, M. C. (2008). Efficient heuristics for data broadcasting on multiple channels. *Wireless Networks, 14*(2), 219–231.
8. Ardizzoni, E., Bertossi, A. A., Pinotti, M. C., Ramaprasad, S., Rizzi, R., & Shashanka, M. V. S. (2005). Optimal skewed data allocation on multiple channels with flat broadcast per channel. *IEEE Transactions on Computers, 54*(5),558–572.
9. Bar-Noy, A., Naor, J., & Schieber, B. (2000, August). Pushing dependent data in clients-providers-servers systems. In *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking* (pp. 222–230).
10. Breslau, L., Cao, P., Phillips, G., & Shenker, S. (1999, March). Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM Conference*.
11. Chehadeh, Y. C., Hurson, A. R., & Kavehrad, M. (1999). Object organization on a single broadcast channel in the mobile computing environment. *Multimedia Tools and Applications, 9*(1), 69–94.
12. Chen, J., Huang, G., & Lee, V. C. S. (2007). Scheduling algorithm for multi-item requests with time constraints in mobile computing environments. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems*.
13. Chen, M. -S., Wu, K. -L., & Yu, P. S. (2003, February). Optimizing index allocation for sequential data broadcasting in wireless mobile computing. *IEEE Transactions on Knowledge and Data Engineering, 15*(1), 161–173.
14. Chu, C. -H., Yang, D. -N., & Chen, M. -S. (2007, November). Using network coding for dependent data broadcasting in a mobile environment. In *Proceedings of IEEE GLOBECOM Conference*.
15. Chung Y. D., & Kim, M. H. (2001). Effective data placement for wireless broadcast. *Distributed and Parallel Databases, 9*(2),133–150.
16. Dewri, R., Ray, I., Ray, I., & Whitley, D. (2008, March). Optimizing on-demand data broadcast scheduling in pervasive environments. In *Proceedings of the 11th International Conference on Extending Database Technology*.
17. Dewri, R., Whitley, D., Ray, I., & Ray, I. (2008, September). Optimizing real-time ordered-data broadcasts in pervasive environments using evolution strategy. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*.
18. Dey, S., & Schabanel, N. (2006, March) Customized newspaper problem: Data broadcast with dependancies. In *Proceedings of the 7th Latin American Theoretical Informatics Conference*.

19. Dykeman, H. D., Ammar, M. H., & Wong, J. W. (1986). Scheduling algorithms for videotex systems under broadcast delivery. In *Proceedings of IEEE ICC Conference*.
20. Hu, Q. L., Lee, D. L., & Lee, W. -C. (1998, November). Dynamic data delivery in wireless communication environments. In *Proceedings of International Workshop on Mobile Data Access* (pp. 218–229).
21. Huang, J. -L. (2008). AIDOA: An adaptive and energy-conserving indexing method for on-demand data broadcasting systems. *IEEE transactions on systems, man, and cybernetics, part A: Systems and humans, 38*(2).
22. Huang, J. -L., & Chen, M. -S. (2004). Dependent data broadcasting for unordered queries in a multiple channel mobile environment. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1143–1156.
23. Huang, J. -L., Chen, M. -S., & Peng, W. -C. (2003, March). Broadcast dependent data for ordered queries without replication in a multi-channel mobile environment. In *Proceedings of the 19th IEEE International Conference on Data Engineering*.
24. Huang, J. -L., & Peng, W. -C. (2005, May). An energy-conserved on-demand data broadcasting system. In *Proceedings of the 6th International Conference on Mobile Data Management*.
25. Huang, J. -L., Peng, W. -C., & Chen, M. -S. (2001, November). Binary interpolation search for solution mapping on broadcast and on-demand channels in a mobile computing environment. In *Proceedings of the 10th ACM International Conference on Information and Knowledge management*.
26. Hung, H.-P., & Chen, M.-S. (2007, October). MULS: A general framework of providing multi-level service quality in sequential data broadcasting. *IEEE Transactions on Knowledge and Data Engineering*.
27. Hung, H. -P., Huang, J. -W., Huang, J. -L., & Chen, M. -S. (2006, April). Scheduling dependent items in data broadcasting environment. In *Proceedings of the 21st ACM Symposium on Applied Computing*, (pp. 23–27).
28. Imielinski, T., Viswanathan, S., & Badrinath, B. R. (1997). Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering, 9*(9), 353–372.
29. Jung, S., Lee, B., Pramanik, S. (2005). A tree-structured index allocation method with replication over multiple broadcast channels in wireless environments. *IEEE Transactions on Knowledge and Data Engineering, 17*(4).
30. Lee, G., & Lo, S. -C. (2003). Broadcast data allocation for efficient access of multiple data items in mobile environments. *Mobile Networks and Applications, 8*(4), 365–375.
31. Lee, G., Lo, S. -C., & Chen, A. L. P. (2002, October). Data allocation on the wireless broadcast channel for efficient query processing. *IEEE Transactions on Computers, 51*(10), 1237–1252.
32. Lee, K. C. K., Schiffman, J., Zheng, B., & Lee, W. -C. (2008, October). Valid scope computation for location-dependent spatial query in mobile broadcast environments. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*.
33. Lee, S., Carney, D. P., & Zdonik S. (2003, March). Index hint for on-demand broadcasting. In *Proceedings of the 19th IEEE International Conference on Data Engineering*.
34. Lee, W. -C., Hu, Q. L., & Lee, D. L. (1999). A study on channel allocation for data dissemination in mobile computing environments. *Mobile Networks and Applications, 4*(5), 117–129.
35. Lee, W. -C., & Zheng, B. (2005, June). DSI: A fully distributed spatial index for location-based wireless broadcast services. In *Proceedings of the 25nd IEEE International Conference on Distributed Computing Systems*.
36. Liberatore, V. (2002, June). Multicast scheduling for list requests. In *Proceedings of IEEE INFOCOM Conference*.
37. Lin, K. -F., & Liu, C. -M. (2006, June). Schedules with minimized access latency for disseminating dependent information on multiple channels. In *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*.
38. Liu, C. -M., & Lin, K. -F. (2007, January). Disseminating dependent data in wireless broadcast environments. *Distributed and Parallel Databases, 22*(1), 1–25.
39. Lo, S. -C., & Chen, A. L. P. (2000, March). Optimal index and data allocation in multiple broadcast channels. In *Proceedings of the 16th International Conference on Data Engineering* (pp. 293–702).
40. Nanopoulos, A., Katsaros, D., & Manolopoulos, Y. (2001, August). Effective prediction of web-user accesses: A data mining approach. In *Proceedings of the WEBKDD Workshop*.
41. Padmanabhan, V., & Qiu, L. (2000, August–September). The content and access dynamics of a busy web site: Findings and implications. In *Proceedings of the IEEE SIGCOMM Conference* (pp. 111–123).
42. Peng, W. -C., & Chen, M. -S. (2003). Efficient channel allocation tree generation for data broadcasting in a mobile computing environment. *Wireless Networks, 9*(2), 117–129.
43. Prabhakara, K., Hua, K. A., & Oh, J. H. (2000, February-March). Multi-level multi-channel air cache designs for broadcasting in a mobile environment. In *Proceedings of the 16th International Conference on Data Engineering* (pp. 167–186).
44. Robert, J., & Schabanel, N. (2007, January). Pull-based data broadcast with dependencies: Be fair to users, not to items. In *Proceedings of the 18th ACM/SIAM Symposium on Discrete Algorithms* (pp. 238–247).
45. Shih, M. -T., & Liu, C. -M. (2008, December). Fair broadcasting schedules on dependent data in wireless environments. In *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*.
46. Shivakumar, N., & Venkatasubramanian, S. (1996). Efficient indexing for broadcast based wireless systems. *Mobile Networks and Applications, 4*(6), 433–446.
47. Si, A., & Leong, H. V. (1999). Query optimization for broadcast database. *Data and Knowledge Engineering, 29*(3), 351–380.
48. Vaidya, N. H., & Hameed, S. (1999). Scheduling data broadcast in asymmetric communication environments. *Wireless Networks, 5*(3), 171–182.
49. Wu, Y., & Cao, G. (2001). Stretch-optimal scheduling for on-demand data broadcasts. In *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*.
50. Xu, J., Lee, W. -C., & Tang, X. (2004, June). Exponential index: A parameterized distributed indexing scheme for data on air. In *Proceedings of the 2nd ACM/USENIX International Conference on Mobile Systems*.
51. Xu, J. L., Lee, D. L., & Li, B. (2003, March). On bandwidth allocation for data dissemination in celluar mobile networks. *Wireless Networks, 9*(2), 103–116.
52. Xu, J. L., Zheng, B., Lee, W.-C., & Lee, D. K. (2003, March). Energy efficient index for querying location-dependent data in mobile broadcast environments. In *Proceedings of the 19th International Conference on Data Engineering*.
53. Yajima, E., Hara, T., Tsukamoto, M., & Nishio, S. (2002, March). Scheduling and caching strategies for broadcasting correlated data. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 504–509).
54. Yee, W. G., Navathe, S. B., Omiecinski, E., & Jermaine, C. (2002, October). Efficient data allocation over multiple channels at broadcast servers. *IEEE Transactions on Computers, 51*(10), 1231–1236.
55. Zhang, X., Lee, W. -C., Mitra, P., & Zheng, B. (2008, March). Processing transitive nearest neighbor queries in multi-channel

access environments. In *Proceedings of the 33rd International Conference on Extending Database Technology*.
56. Zheng, B., Lee, W. -C., Lee, K. C. K., Lee, D. L., & Shao, M. (2009). A distributed spatial index for error-prone wireless data broadcast. *VLDB Journal, 18*(4), 959–986.
57. Zheng, B., Xu, J., Lee, W. -C., & Lee, D. L. (2004, March). Energy-conserving air indexes for nearest neighbor search. In *Proceedings of the 9th International Conference on Extending Database Technology*.

Author Biographies

Jiun-Long Huang Jiun-Long Huang received his B.S. and M.S. degrees in Computer Science and Information Engineering Department in National Chiao Tung University in 1997 and 1999, respectively, and his Ph.D. degree in Electrical Engineering Department in National Taiwan University in 2003. Currently, he is an assistant professor in Computer Science Department in National Chiao Tung University. His research interests include: mobile computing, wireless networks and data mining.

Jen-He Huang Jen-He Huang received the B.S. and M.S. degrees in Department of Computer Science in National Chiao Tung University, Taiwan, in 2006 and 2008, respectively. His research interests include sensor networks and wireless networks.