Frequent-Pattern-Based Broadcast Scheduling for Conflict Avoidance in Multichannel Data Dissemination Systems

Chuan-Chi Lai, Yu-De Lin, and Chuan-Ming Liu

1Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan
2Wistron (Taiwan), Taipei 11469, Taiwan
3Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei 10618, Taiwan

Correspondence should be addressed to Chuan-Ming Liu; cmliu@ntut.edu.tw

1. Introduction

With advances in wireless communications technologies, mobile devices deeply affect our daily lives, such as notebooks, smart phones, and tablets. Users can easily access various information services, such as online news, traffic information, and stock prices. Recently, wireless data dissemination becomes a popular topic [1–3], which can transmit information to a number of users simultaneously. In comparison with the conventional end-to-end transmission (or client-server) model, wireless data dissemination can make use of wireless network channels to reduce the delivery time for obtaining information. Wireless data broadcasting is well-suited to the location-based services (LBS) in an asymmetric communication environment, where a large number of users are interested in popular information such as news [4], traffic reports [5], and multimedia streams [6, 7].

In general, wireless data dissemination can be classified into two modes: push-based and pull-based (on-demand). In push-based wireless data dissemination environments [8–10], data items are disseminated cyclically according to a predefined schedule. In fact, the access pattern of data items may change dynamically, and the broadcast frequency of popular data items may be lower than the broadcast frequency of unpopular data items. Such a case will result in a poor average access latency. In view of this, pull-based wireless data dissemination [11–13] that disseminates data items timely according to the received requests was proposed to overcome the aforementioned drawback. In the pull-based mode, the users first upload their demand information to the server through the uplink channel, and then, the relevant information will be immediately arranged into the broadcasting channels for disseminating data to users. In wireless data dissemination environments, a way of judging the quality of a scheduling approach is to measure the access time of the generated schedule. The access time is a measured time period from starting tuning the channels to obtaining all the requested information. Thus, it is
important to have a better broadcasting schedule for shorter access time.

1.1. Motivation. In early literature, some conventional works [14–16] focus on how to maximize the bandwidth throughout or minimized the access time in single channel environments. Recently, with the advance on antenna techniques, most of works [17–19] has shifted their focus on the similar issues in multiple channel environments. In general, a multichannel wireless data dissemination system can provide a more network bandwidth and a shorter access time for data dissemination than a single-channel wireless data dissemination system can.

However, one new issue, data conflict [20–22], emerges while each client retrieves data items on multiple channels with channel switching in push-based broadcasting environments. Two types of conflicts may occur in multichannel dissemination systems. The first type of conflict is that two required data items are allocated on the same time slot of different channels, so the client cannot download the required data items simultaneously. The second type of conflict occurs if two required data items are allocated on the \( t \) and \( (t+1) \) time slots of different channels, respectively. In such a scenario, the client cannot download both required data items during the time period \([t, t+1]\). The 1st conflict type is obvious. The reason of the 2nd conflict type is that switching from any channel to a different channel takes time. A client cannot download data at time slot \( t+1 \) from one channel if it was downloading data item from another channel at time slot \( t \), because a time slot is already the smallest unit for data retrieving. Note that a client is allowed to access one channel at one time.

Such a data conflict issue makes a client miss its needed data items during the time period for channel switching, thereby leading to a worse access time. On the one hand, some works [20–22] provide some solutions from the client’s point of view. These solutions can make each client schedule itself for retrieving the data items on channels efficiently. On the other hand, only one work [13] provides a server-side scheduling algorithm with consideration of the data conflict issue in on-demand multi-channel environments. The provided algorithm considers the associations between data items and requests while allocating data items on multiple channels and this provides a conflict-free schedule.

Most broadcast scheduling techniques in on-demand multichannel data dissemination environments do not consider the time requirement for channel switching, thereby leading to data conflicts or long access time. This phenomenon motivates us to propose a more efficient server-side scheduling method with conflict avoidance using frequent pattern mining technique, thereby shortening the average access time.

1.2. Contribution. In this study, we discuss how to shorten the average access time on a multichannel wireless data dissemination environment under the data conflict conditions. The contributions of this work are listed as follows:

1. Identify the data broadcast with conflict avoidance (DBCA) problem in on-demand multichannel wireless data dissemination environments and prove the considered DBCA problem is \( \mathcal{NP} \)-complete

2. We propose a heuristic approach, frequent-pattern-based broadcast scheduling (FPBS), for providing an approximate schedule in polynomial time. Inspired by frequent-pattern tree (FP-tree), we suggest a new tree, FP*-tree, for FPBS to schedule the requested data items with the consideration of channel switching

3. We analyze the time complexity and average access time of FPBS in both average case and worst case

4. We verify the performance of FPBS which achieves a shorter average access time in comparison with the existing method, UPF [13]

The rest of this paper is organized as follows. Section 2 gives the background and reviews related research in the literature. Section 3 defines the DBCA problem and proves that the DBCA problem is \( \mathcal{NP} \)-complete. Section 4 explains the proposed approach with examples and algorithms in detail. In Section 5, we discuss the time complexity and access time of the proposed approach in worst case. Section 6 presents the experimental simulation results and validates the correctness and effectiveness of the proposed methods in various situations. Finally, we conclude this work in Section 7.

2. Related Work

In the multichannel dissemination environments, many related research works focused on data scheduling to improve the access time performance [17, 18] from the perspective of spectrum utilization. Yee et al. [17] proposed a greedy algorithm to find the best way to distribute data items into the channels, allowing users to access requested data in a limited time. Zheng et al. [18] considered the data access frequency, data length, and channel bandwidth into a model and proposed a two-level optimization scheduling algorithm (TOSA) to find an appropriate schedule. They also showed that the schedule of TOSA is approximate to the best average time. Yi et al. [19] proposed a method to allow replicating multiple copies of a data item in a broadcasting channel. If there are multiple copies of a popular data item in the channel, the average access time can be effectively reduced.

In addition to the above methods, some works considered the priority of incoming queries and found ways to reduce the access time [12, 14, 15, 23]. Lu et al. [14] proposed some algorithms to schedule data for maximum throughput request selection (MTRS) and minimum latency request ordering (MLRO) problems in a single-channel environment and proved that both problems are \( \mathcal{NP} \)-hard. Xu et al. [15] proposed a \( \alpha \)-algorithm with a set of priority decisions based on the ratio of the length of the expiration time over the amount of information. Lv et al.
The authors provided a profit-based heuristic scheduling algorithm to minimize the request miss rate (or delivery miss rate) considering the access frequency of data. Liu and Su [12] focused on reducing the demand for the loss rate and shortening the access time. Two kinds of algorithms, most popular first heuristic (MPFH) and most popular last heuristic (MPLH), were proposed to solve the problems and they also analyzed differences between the online version (the user demands continuously come in the system, so the scheduling task needs to wait until it starts receiving information of the demands) and offline version (the system already has all the information of demands).

Some works had found that the dependency between requested data items may greatly influence the performance of multichannel data broadcasting. Lin and Liu [24] considered the dependencies among data items as a directed acyclic graph (DAG). They proved that finding the best schedule preserving dependencies between each data item is an \( \mathcal{NP} \)-hard problem and proposed some heuristics for the problem. Qiu et al. [25] proposed a three-layer on-demand data broadcasting (ODDB) system for enhancing the uplink access capacity by introducing a virtual node layer. Each virtual node can merge duplicated requests and help the server reduce huge computational load, there by improve the broadcasting efficiency.

Lu et al. [20–22] firstly defined two types of well-known data conflicts in multichannel broadcast applications. They proved the client-side retrieval scheduling problem is \( \mathcal{NP} \)-hard and provided some client-side data retrieval algorithms for helping clients to retrieve data within multiple channels efficiently. Liu et al. [26] firstly proposed a server-side heuristic data scheduling algorithm, dynamic urgency and productivity (DUP), for on-demand multichannel systems with consideration of the request conflict (or request overlapping) issue and the dependency between requests for scheduling at the request level and giving higher priorities to the requests which are close to their deadlines. Such an approach provided a counteracting effect to the request starvation problem and improved the utilization of broadcasting bandwidth. However, they did not consider two types of data conflicts. He et al. [13] proposed a server-side heuristic scheduling approach, most urgent and popular request First (UPF), with the consideration of two types of data conflicts in on-demand systems. Except for UPF method, the hardness of data scheduling problem considering two types of data conflicts from the server perspective is seldom discussed.

The comparisons of the existing works and this paper are summarized in Table 1. In this work, we propose a new server-side heuristic scheduling approach for providing a conflict-free multichannel data broadcast service with a better performance on the average access time.

### 3. Problem Description

The length of a broadcasting cycle is an important factor which is normally predefined in the wireless data dissemination applications. Most of existing data scheduling strategies focus on investigating how to efficiently schedule data items in each broadcasting cycle. To validate the performance of a scheduling strategy, average access time (or average latency), is the commonly and widely used metric. If the average access time is shorter, users generally can obtain all the requested data in a shorter time, meaning that the used scheduling strategy is more efficient. In the following subsections, we will describe the considered system model, define the considered scheduling problem, and then prove the hardness of this problem.

#### 3.1. System Model

In this work, the considered on-demand multichannel data dissemination system is shown in Figure 1 and we only consider the one-hop broadcasting scenario. The considered data dissemination system uses \( |C| + 2 \) antennas with orthogonal frequency division multiplexing (OFDM) technique [27] to provide \( |C| \) downlink broadcast channels, 1 downlink index channel, and 1 uplink request channel, where \( C = \{ c_1, c_2, \cdots, c_{|C|} \} \) and \( |C| > 1 \). The downlink index channel and request uplink channel are denoted as \( c_{\text{index}} \) and \( c_{\text{uplink}} \) respectively. Each user device has two antennas with one for receiving data over the downlink broadcast channels and one for transferring requests via the uplink request channel. We assume that each user device can only access one channel at one time. We assume that all the channels are nonoverlapping, synchronous and discretized into fixed-duration slots. The broadcasting server puts the requests coming from the uplink channel into a buffer with first-come-first-serve (FCFS) strategy and handles all the received requests in a batch manner. In this work, we only focus on the efficiency of (application-layer) data/packet scheduling for users to retrieve the requested data items by accessing the downlink channels.

We assume that all the requested data items are in a dataset \( D = \{ d_1, d_2, \cdots, d_{|D|} \} \), where \( |D| \) is the size of \( D \), and the length of a broadcasting cycle is \( L = |D| \) in default. Suppose that there are \( n \) queries, \( Q = \{ q_1, q_2, \cdots, q_n \} \), and each query \( q_i \) requests \( k \) data items from the dataset \( D \), where \( i = 1, 2, \cdots, n \) and \( k = 1, 2, \cdots, |D| \). We let \( q_i = \{ d_{i1}, d_{i2}, \cdots, d_{ik} \} \) and all the data items have the same data size, where \( d_{ij} \in D \), \( j = 1, 2, \cdots, k \), and \( \bigcup_{i=1}^{n} q_i \subseteq D \). Thus, the system has to arrange the requested data items into \( |C| \) broadcasting channels. Note that each time slot on a broadcasting channel can contain at most one data item and data replication is only allowed on different channels. That is, multiple copies of one data item may be placed within a broadcasting cycle. Suppose \( L \) is the cycle length, each index \( I \), at time slot \( t \) records the informations about all the data items in time slot \( t' \) and the corresponding requests of these data items, where \( t' \) is obtained by

\[
    t' = \begin{cases} 
    (t + 2) \mod L, & \text{if } t + 2 > L, \\
    t + 2, & \text{otherwise}.
    \end{cases}
\]

When a client tunes in the channel, it will access the
data dissemination systems. For example, if one of requested data items for request \( q_i \) is placed at the previous, the same, or the later location of a scheduled data item which is also associated with \( q_i \) on different channels, a data conflict occurs. An example of data conflicts is presented in Figure 3. The data conflict may result in a longer access time and can be defined as Definition 1.

**Definition 1** (data conflict). For a query \( q_i \), two requested data items \( d_{j}^{i} \) and \( d_{j'}^{i} \), \( 1 \leq j \neq j' \leq k \), if \( c_{j}^{i} \neq c_{j'}^{i} \), the conflict occurs when \( p_{j}^{i} = p_{j'}^{i} \) or \( |p_{j}^{i} - p_{j'}^{i}| = 1 \).

Let \( \text{loc}_{\text{min}}^{i} \) denote the minimum value of all the locations of the data items associated with \( q_i \) and \( \text{loc}_{\text{Max}}^{i} \) is the maximum value of all the positions of the data items associated with \( q_i \). In other words, \( \text{loc}_{\text{min}}^{i} = \min_{1 \leq j \leq k} p_{j}^{i} \) and \( \text{loc}_{\text{Max}}^{i} = \max_{1 \leq j \leq k} p_{j}^{i} \). The access time of query \( q_i \), \( \text{acc}(q_i) \), can be defined as \( \text{loc}_{\text{Max}}^{i} - \text{loc}_{\text{min}}^{i} \), while the search starts from the beginning of the broadcasting cycle. The average access time for a mapping \( \mathcal{M} \) is thus \( \text{acc}_{\mathcal{M}} = \frac{\sum_{i=1}^{n} \text{acc}(q_i)}{n} \).

In summary, the problem we want to solve in this work is data broadcast with conflict avoidance (DBCA) problem which can be defined as follows.

### Table 1: Comparisons of related works and the proposed method.

<table>
<thead>
<tr>
<th>Related works</th>
<th>System model</th>
<th>Criterion</th>
<th>Method</th>
<th>Data popularity</th>
<th>Request dependency</th>
<th>Channel switching</th>
<th>Data conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>[17–19]</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[12, 14, 15]</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[21]</td>
<td>Push-based</td>
<td>Latency</td>
<td>Client-side</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[22]</td>
<td>Push-based</td>
<td>Throughput</td>
<td>Client-side</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[23]</td>
<td>On-demand</td>
<td>Request miss rate</td>
<td>Server-side</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[24, 25]</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[26]</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[27]</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Our work</td>
<td>On-demand</td>
<td>Latency</td>
<td>Server-side</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

![Index channel in advance until obtaining information about the first required data item.](Image)

**Figure 1**: The considered on-demand multichannel wireless data dissemination environment.

3.2. Problem Formulation. The considered scheduling problem can be treated as a mapping \( \mathcal{M} \) that data items associated with all the queries to \( |C| \) broadcasting channels. For each data item \( d_{j}^{i} \in D \) associated with a query \( q_{i} \in Q \), let \( \text{pos}(q_{i}, d_{j}^{i}) = (c_{j}^{i}, p_{j}^{i}) \) be the position of data item \( d_{j}^{i} \) in the broadcast, where \( c_{j}^{i} \) is the channel number, \( 1 \leq c_{j}^{i} \leq |C| \), and \( p_{j}^{i} \) is the location of \( d_{j}^{i} \) on that channel, \( 1 \leq p_{j}^{i} \leq |D| \). Such a mapping \( \mathcal{M} : Q \times D \rightarrow \mathbb{N} \times \mathbb{N} \) is a 1-to-1 mapping.

Since there are multiple channels and each user can only tunes into one broadcasting channel at one time instance, each user may switch channels many times for retrieving all the requested data items on different channels. In general, channel switching is a relatively fast operation (in the micro-second range) [28, 29]. For simplicity, we follow the similar assumptions about channel switching in [22], and each channel switching takes one time slot in the considered data dissemination environment. Figure 2 shows an example of the channel switching. However, channel switching may cause a new problem, data conflict, in multichannel wireless data dissemination systems. For example, if one of requested data items for request \( q_i \) is placed at the previous, the same, or the later location of a scheduled data item which is also associated with \( q_i \) on different channels, a data conflict occurs. An example of data conflicts is presented in Figure 3. The data conflict may result in a longer access time and can be defined as Definition 1.

**Definition 1** (data conflict). For a query \( q_i \), two requested data items \( d_{j}^{i} \) and \( d_{j'}^{i} \), \( 1 \leq j \neq j' \leq k \), if \( c_{j}^{i} \neq c_{j'}^{i} \), the conflict occurs when \( p_{j}^{i} = p_{j'}^{i} \) or \( |p_{j}^{i} - p_{j'}^{i}| = 1 \).

Let \( \text{loc}_{\text{min}}^{i} \) denote the minimum value of all the locations of the data items associated with \( q_i \) and \( \text{loc}_{\text{Max}}^{i} \) is the maximum value of all the positions of the data items associated with \( q_i \). In other words, \( \text{loc}_{\text{min}}^{i} = \min_{1 \leq j \leq k} p_{j}^{i} \) and \( \text{loc}_{\text{Max}}^{i} = \max_{1 \leq j \leq k} p_{j}^{i} \). The access time of query \( q_i \), \( \text{acc}(q_i) \), can be defined as \( \text{loc}_{\text{Max}}^{i} - \text{loc}_{\text{min}}^{i} \), while the search starts from the beginning of the broadcasting cycle. The average access time for a mapping \( \mathcal{M} \) is thus \( \text{acc}_{\mathcal{M}} = \frac{\sum_{i=1}^{n} \text{acc}(q_i)}{n} \).

In summary, the problem we want to solve in this work is data broadcast with conflict avoidance (DBCA) problem which can be defined as follows.
De 2: An example of channel switching, where the data items and analyze the schedules with con the existing works only considered the schedules without To the best of our knowledge, most of replicating data items on di multichannel dissemination environment which allows Conversely, our proposed approach, FPBS, considers a is problem. In this subsection, we will prove DBCA problem

\[ D = \{d_1, \ldots, d_3, d_4\} \]

Each request \( q_i \), \( 1 \leq i \leq n \), is associated with \( k \) data items, \( d_{i1}, d_{i2}, \ldots, d_{ik} \), where \( d_{ij} \in D, 1 \leq j \leq k \leq |D| \). Any two data items associated with two different requests are different, and every data item needs an unit time \( u_i \) to be broadcast. Let \( \text{loc}_{\min}^i \) and \( \text{loc}_{\Max}^i \) be the start time and finish time of \( q_i \), respectively.

Question: Does there exist a mapping \( \mathcal{M} : Q \times D \rightarrow \{1, \ldots, |C|\} \times \{1, \ldots, L\} \) such that

\[ \begin{align*}
& (1) \text{ For two data items } d_j^i \text{ and } d_j'^i \text{ associated with } q_i, |p_j^i - p_j'^i| > 1 \text{ and}
& (2) \text{ the average access time, } \sum_{i=1}^{n} |\text{loc}_{\Max}^i - \text{loc}_{\min}^i/n|, \text{ is minimized}
\end{align*} \]

In the definition of CDBML problem, the first objective indicates that the broadcasting schedule avoids the data conflict problem. The second objective is to reduce the average access time and all of the data items associated some request \( q_i \) should be broadcasting before the end of the broadcasting cycle. \( W_i \) is an indication function used to present if a request is served or not. To show further that the CDBML problem is \( \mathcal{N} \mathcal{P} \)-complete, we consider a special case of it, where the number of data items associated with each request is the same and equal to the number of channels. That is, we consider the case \( k = |C| \). The data items associated with different requests are all different. The following gives the definition of the decision problem for the above special case.

**Definition 4 (CDBMLp problem).** Instance: There are \( |C| \) data broadcasting channels with cycle length \( L \), a set of \( |D| \) data items \( D = \{d_1, \ldots, d_{|D|}\} \), a set of \( n \) requests \( Q = \{q_1, \ldots, q_n\} \), and an integer \( h \). Each request \( q_i, 1 \leq i \leq n \), is associated with \( |C| \) data items, \( d_{i1}, d_{i2}, \ldots, d_{i|C|} \), where \( d_{ij} \in D, 1 \leq j \leq |C| \leq |D| \). Any two data items associated with two different requests are different, and every data item needs an unit time \( u_i \) to be broadcast. Let \( \text{loc}_{\min}^i \) and \( \text{loc}_{\Max}^i \) be the start time and finish time of \( q_i \), respectively.

Question: Does there exist a mapping \( \mathcal{M} : Q \times D \rightarrow \{1, \ldots, |C|\} \times \{1, \ldots, L\} \) such that

\[ \begin{align*}
& (1) \text{ For two data items } d_j^i \text{ and } d_j'^i \text{ associated with } q_i, |p_j^i - p_j'^i| > 1 \text{ and}
& (2) \sum_{i=1}^{n} \text{acc}(q_i)/n \leq h, \text{ where } \text{acc}(q_i) = |\text{loc}_{\Max}^i - \text{loc}_{\min}^i|
\end{align*} \]

To show that the CDBMLp problem is \( \mathcal{N} \mathcal{P} \)-complete, we reduce the minimizing mean flow time in unit time open
shop (MMUOS) scheduling problem with preemption \((O \mid P_j \in \{0, 1\}; \text{ptmn} \mid \Sigma C_t)\) to the CDBML \(\rho\) problem. \cite{30} has proved such a problem \((O \mid P_j \in \{0, 1\}; \text{ptmn} \mid \Sigma C_t)\) is \(\mathcal{NP}\)-hard by the reduction from the graph coloring problem, and thus, the CDBML problem is \(\mathcal{NP}\)-hard. The MMUOS problem is defined as follows.

**Definition 5 (MMUOS problem).** Instance: Given \(m\) machines, a set of \(n\) jobs \(J = \{ J_1, J_2, \ldots, J_m \}\), a set of \(O\) unit operations \(O = \{ o_1, o_2, \ldots, o_O \}\), and an integer \(T\). Each job \(J_i\), \(1 \leq i \leq n\), consists of \(m\) unit operations \(o_{ij}^1, o_{ij}^2, \ldots, o_{ij}^m\). The \(j_{th}\) operation, \(1 \leq j \leq m\), has to be processed on the \(j_{th}\) machine. Job \(J_i\) will be processed in a window defined by a release time \(r_i\) and a finish time \(c_i\).

- Question: Does there exist a mapping \(M : Q \times D \longrightarrow \{1, \ldots, |C| \} \times \{1, \ldots, L\}\) such that
  1. For operations \(o_{ij}^1\) and \(o_{ij}^2\) in job \(J_i\), \(M(o_{ij}^1) \neq M(o_{ij}^2)\) and
  2. \(\sum_{i=1}^{n} C_i/n \leq T\), where \(C_i = |c_i - r_i|\)

**Theorem 6.** The CDBML \(\rho\) problem is \(\mathcal{NP}\)-complete.

**Proof.** It is easy to see that the CDBML \(\rho\) problem is in \(\mathcal{NP}\), since validating the existence of an given conflict-free schedule simply needs polynomial time. In order to prove the CDBML \(\rho\) problem is \(\mathcal{NP}\)-hard, a reduction from the MMUOS problem can be made. Suppose that \(I'\) is an instance of the MNUOS problem. A corresponding instance \(I\) of the CDBML \(\rho\) problem can be constructed from \(I'\) as follows:

1. An unit operation time is equal to the unit time slot to broadcasting a data item
2. Let a job \(J_i\) correspond to a request \(q_i\), \(1 \leq i \leq m\) and operations \(o_{ij}^1\) in \(J_i\) be the data item \(d_{ij}\) associated with \(q_i\)
3. Let \(m\) machines be the \(|C|\) data broadcasting channels (i.e., \(m = |C|\))
4. Let \(J_i's\) release time \(r_i\) and \(q_i's\) start time \(lo_c_{min}\) in the schedule
5. Let \(J_i's\) finish time \(c_i\) be \(q_i's\) finish time \(lo_c_{max}\) in the schedule
6. Let integer \(T\) be the integer \(h\) in CDBML \(\rho\) problem
7. Let the unit time \(u_i\) in MMUOS problem be three times of \(u_i\) in CDBML \(\rho\) problem \((u_i' = 3 \times u_i)\)

According to the last step of the construction, the first objective of MMUOS problem can be equivalent to the first objective of CDBML \(\rho\) problem and the above construction can be done in polynomial time. It is straightforward to show that there is a solution for an instance \(I'\) of the MMUOS problem if and only if there is a solution for the instance \(I\) of the CDBML \(\rho\) problem since the reduction is a one-to-one mapping for the variables from the MMUOS problem to the CDBML \(\rho\) problem. Hence, the CDBML \(\rho\) problem is \(\mathcal{NP}\)-complete.

Thus, we can conclude the following theorem.

**Theorem 7.** The CDBML problem is \(\mathcal{NP}\)-complete.

### 4. Frequent-Pattern-Based Broadcast Scheduling

In this section, we propose an approach, the frequent-pattern-based broadcast scheduling (FPBS), to shorten the average access time per user for the DBCA problem. In FPBS, we construct a new tree with the frequent patterns of queries. This tree is named as FP*-tree. FPBS includes four stages: (1) sorting requested data items, (2) constructing the FP*-tree’s backbone, (3) constructing the FP*-tree's accelerating branches, and (4) schedule mapping. In the following, the proposed method will be introduced with a running example in detail.

#### 4.1. Stage 1: Sorting Requested Data Items

We consider a running example which uses two data broadcasting channels \(c_1\), \(c_2\) and an additional index channel \(c_{index}\). The data dissemination server receives five queries \(q_1 = \{d_1, d_2, d_3, d_4\}\), \(q_2 = \{d_4, d_3, d_1\}\), \(q_3 = \{d_3, d_1, d_2\}\), \(q_4 = \{d_2, d_3, d_4\}\), and \(q_5 = \{d_2, d_5, d_5\}\) and then derives the access frequency \(f_{d_i}\) of each data item \(d_i\) in these queries. After that, the server sorts all the data items in each query according to the descending order of their access frequencies and also derives the statistical average access frequency \(f_q\) for each query \(q_i\).

For example, \(f_{q_1} = (f_{d_1} + f_{d_2} + f_{d_3})/|q_1| = (3 + 3 + 1)/3 = 2.33\). Hence, the final result is presented in Table 2.

The detailed process, FPBS.StatisticAndSort \((Q)\), for the first stage is presented in Algorithm 1. Line 2 and Line 3 analyze the received query set \(Q\), derive the statistical information, and save it as a temporary set \(S\). The operations from Line 4 to Line 6 sort every requested data item of each query according to the access frequency of the data item. As the example shown in Table 2, the orders of requested data items in queries \(q_4\) and \(q_5\) change after the sorting. Line 7 and Line 8, respectively, store the results in two lists, \(l_{sortedWithSize}\) and \(l_{sortedWithFreq}\), in different orders. Finally, the process returns these two lists at Line 9 for the use in following stages.

#### 4.2. Stage 2: Constructing the FP*-Tree’s Backbone

After deriving some statistical information and the sorting result in Table 2, the system starts to create the backbone of a FP*-tree. In this stage, the system will always select the query which requests the most number of data items to be inserted into the FP*-tree in advance. If there are multiple queries which request the same number of data items, the system will select the one which has the maximum average access frequency \(f_q\). Thus, the system select \(q_i\) as the first query to construct the backbone of a FP*-tree and the result...
is shown in Figure 4(a). After adding $q_4$ to the FP$^*$-tree, the system will update the statistical information of unhandled queries, as shown in Table 3.

After updating the statistical information, the system will select the next query to handle in the same way. In the previously mentioned, both $q_1$ and $q_3$ request 2 data items so the system will compare the remaining average access frequencies of $q_1$ and $q_3$ ($f_{q_1} = 2$, $f_{q_3} = 2$) and both values are the same. Then, the query which comes into the system first will be selected, so $q_1$ becomes the next one in this step. Note that the numbering of $q_1$ is smaller than $q_3$’s and it means that $q_1$ comes into the system earlier. Thus, the handling priority of the remaining queries is $q_1 \rightarrow q_3 \rightarrow q_5$.

While adding data item $d_2$ into the FP$^*$-tree, the system needs to consider the relations between $d_2$ and the other queries. In this case, $q_1$ and $q_3$ also request the data item $d_2$. The system then checks the other data items which are in the request list of both queries and have been added into the FP$^*$-tree. Since the level of $d_2$ is larger than $d_4$’s level, the system will insert $d_2$ as $d_4$’s child. Such a way can avoid increasing the access time of $q_4$ which has been handled. After handling $d_2$, the system handles $d_3$ in the same way and the result of FP$^*$-tree is shown in Figure 4(b). The system then updates the statistical information which is presented in Table 4.

The next query which will be handled is $q_5$. Since there are no other queries relating to the requested data item $d_6$, the system needs to add $d_8$ after $d_2$ according to the order of $q_5$’s requested list. However, $d_6$ is also requested by $q_1$, and thus, $d_6$ already has one branch and the position is occupied by $d_5$. Therefore, $d_6$ can only be scheduled in the level (time slot) after $d_2$ and $d_5$. In this case, the system creates a new branch of $d_2$ and inserts an empty node between $d_2$ and $d_5$. Note that an empty node is a node without saving any data item. After handling $q_5$, the results are shown in Figure 4(c). The last query is $q_3$, and there are no other queries relating to $d_4$. Hence, the system has to add $d_4$ after $d_1$ according to the order of $q_3$’s requested list. However, $d_4$ is also requested by $q_1$, and thus, $d_4$ needs to be scheduled after $d_4$. In this case, the system creates a new branch of $d_4$ and inserts an empty node between $d_1$ and $d_4$. Finally, the construction of FP$^*$-tree’s backbone is finished and the result is shown in Figure 4(d).

Algorithm 2 presents two functions for the backbone construction. FPBS_CreateBackbone($S$) describes the main process of an FP$^*$-tree’s backbone construction and FPBS_AddNodeForBackbone ($\mathcal{F}, N_p, d$) is the function of adding a node during the backbone construction. From Line 3 to Line 5, the operations initialize an empty FP$^*$-tree $\mathcal{F}$ and create a sorted query table $Q_{table}$ with the derived sorted result in the stage 1. The operations from Line 6 to Line 8 handle each requested data item of the first query in the sorted query set. The first query is the most important and has maximum number of requested data items. As shown as the above example in Figure 4(a), the query $q_1$ is the first to be handled. At Line 9, the remaining information of unhandled queries and data items in the query table $Q_{table}$ will be updated. From Line 10 to Line 17, the operations continuous inserting the unhandled data items of $Q_{table}$ into the backbone of $\mathcal{F}$. At Line 13, the operation finds the right position of $\mathcal{F}$’s backbone to insert the unhandled data item with the consideration of query dependency and the order of data items. The operations from Line 21 to Line 35 presents the detailed process of adding a data node to the backbone of $\mathcal{F}$. Note that the operation, $\mathcal{F}.isOverload()$, at Line 26 is used to avoid scheduling data items out of $|C|$ data broadcasting channels. Figures 4(c) and 4(d) are the running examples for such operations.

### 4.3. Stage 3: Constructing the FP$^*$-Tree’s Accelerating Branches.

After the construction of FP$^*$-tree’s backbone, the system starts to create the accelerating branches to optimize the schedule. The purpose of constructing the accelerating branch is to increase the chance of each user getting the requested data item earlier after switching channels.

In this stage, we propose two different ordering rules, request-number-first and frequency-first, to insert data items in the FP$^*$-tree’s accelerating branches. The priority of a query for the insertion of FP$^*$-tree is decided by following values: number of requested data items, average access frequency, and arrival time. With request-number-first rule, the system will select the query which requests the maximum number of data items to handle first. If multiple queries request the maximum number of data items, the system will select the one of them that has the maximum average access frequency. If multiple queries has the maximum average access frequency unfortunately, the system will select the query according to its arrival order. Conversely, with frequency-first rule, the system will first select the query which has the maximum average access frequency. If multiple queries has the maximum average access frequency, the system will select the one of them that requests the maximum number of data items. If multiple queries requests the maximum number of data items unfortunately, the system will select the query according to its arrival order. Note that the construction of the FP$^*$-tree’s backbone always follows the request-number-first rule in our design. The system can use different rules only when constructing accelerated branches of the FP$^*$-tree.

Since different orders of handling queries and data items make the process constructs different accelerating branches of FP$^*$-trees, we will compare the performance results of different schedules generated by using different rules. By default, the system uses frequency-first rule to select the query for constructing the FP$^*$-tree’s accelerating branches. Due to limitations on space and the similar process, we only
introduce the proposed approach with frequency-first in detail. In this example, the system follows the frequency-first rule and gets the following handling sequence: \( q_2 \rightarrow q_4 \rightarrow q_1 \rightarrow q_3 \rightarrow q_5 \). Note that the value of \( f_{q_i} \) is shown in Table 2.

The system first handles query \( q_2 \) and \( q_2 \)'s sorted requested data items are \( d_2, d_3, \) and \( d_4 \). Hence, the system sequentially schedules \( d_3 \), \( d_4 \), and \( d_2 \). When scheduling \( d_4 \), the system temporarily inserts \( d_4 \) into level (or slot) 1 and the position is a right child of the root. Then, the system searches \( d_2 \) in the backbone and check whether \( p_2 > p_2' \) and \( p_2 - p_2' > 1 \) or not. In this case, \( p_2 > p_2' \) and \( p_2 - p_2' = 4 > 1 \) is hold, so \( d_2 \) can be inserted into the position of \( d_2' \).

For the next requested data item \( d_3 \), the system inserts \( d_3 \) after \( d_2 \) in the accelerating branch and then checks whether the position is legal or not in the same way. In this case, \( d_3 \) can be inserted into the position of \( d_3' \). For the last requested data item \( d_4 \) by query \( q_4 \), the system tries to temporarily insert \( d_4 \) after \( d_3 \) in the accelerating branch. However, the system can find \( d_4 \) in the backbone that \( p_4 - p_4' \leq 1 \). Thus, \( d_4 \) can not be inserted into the accelerating branch. After handling \( q_2 \), the result of FP* -tree is shown in Figure 5(a).

For the next query \( q_4 \), the system will do nothing in the accelerating branch. The reason is that \( q_4 \) is the first query handled in the backbone and the schedule, \( d_3 \rightarrow d_2 \rightarrow d_4 \rightarrow \ldots \).
1: Function FPBS_CreateBackbone (S)
   Input: a sorted set of queries (clients) S;
   Output: a basic FP*-tree $T$;
2: create a empty FP*-tree $T$ and the root $R$ of $T$;
3: set $S$ into a query table $Q_{table}$;
4: let $q$ ← 5.first();
5: let a temporary pointer $N_{curr}$ ← $R$;
6: for each requested data $d$ in $q$ do;
7:   $N_{curr}$ ← FPBS_AddNodeForBackbone ($T$, $N_{curr}$, $d$);
8: end;
9: update $Q_{table}$;
10: while $Q_{table}$ contains any unhandled required data do
11:   $q_{un}$ ← the query with the maximum number of unhandled data items in $Q_{table}$;
12:   for each unhandled requested data $d'$ in $q_{un}$ do;
13:     $N_{d',d}^*$ ← find the other queries which also needs data $d'$ and then choose one of the handled data nodes whose slot is maximum in $T$
14:     FPBS_AddNodeForBackbone ($T$, $N_{d',d}^*$, $d'$);
15: end;
16: update $Q_{table}$;
17: end;
18: return $T$;
19: end;
20: Function FPBS_AddNodeForBackbone ($T$, $N_{p}$, $d$)
   Input: an FP*-tree $T$, the parent node $N_{p}$, and a new data item $d$;
   Output: an added node $N_{d}$;
21: create a new node $N_{q}$ with data item $d$
22: if $N_{p}$ has children then
23:   create an empty node $N_{i}$;
24:   $N_{p}.addChild(N_{i})$;
25:   let a temporary pointer $N_{temp}$ ← $N_{i}$;
26:   while $T$.isOverload($N_{temp}.slot+1$) do
27:     create an empty node $N_{i}$;
28:     $N_{p}.addChild(N_{i})$;
29:     $N_{temp}$ ← $N_{i}$;
30: end;
31: $N_{temp}.addChild(N_{d})$;
32: else
33:   $N_{p}.addChild(N_{d})$;
34: end;
35: return $N_{d}$;
36: end;

Algorithm 2: Functions used for the FP*-tree’s backbone construction.

$d_1 \rightarrow d_4$, has been optimized. Go on the next step, $q_4$ is going to be handled and $q_4$’s requested data items are $d_2$, $d_3$, and $d_7$. Since $d_2$ has been inserted into the accelerating branch, the system skips $d_1$ and tries to insert $d_2$ in this step. According to the order of $q_4$’s requested list, $d_3$ needs to be inserted after $d_2$. In the accelerating branch, node $d_2$ already has a child, so the system creates a new branch of $d_2$, inserts an empty node as $d_2$’s right child, and then add temporary $d_3$ after the empty node. Since there is no $d_5$ whose $p_5 > p_9'$ in the backbone, it is legal to insert $d_5$ at the position of $d_5'$. For the last requested data item $d_7$ in $q_4$, $d_7$ is inserted in the same way. The system inserts $d_7'$ after $d_5$ in advance and check whether the backbone contains $d_7$ or not. Since $p_7 > p_7'$ and $p_7 - p_7' = 2 > 1$, it is legal to insert $d_7$ at the position of $d_7'$. After handling all the requested data items in $q_4$, the result of FP*-tree is shown in Figure 5(b)

After handling $q_4$, the system will start to handle $q_3$. The sorted requested data items are $d_3$, $d_5$, and $d_8$. Since $d_2$ has been scheduled at the first slot (level) in the accelerating branch, the system skips $d_2$ in this step. The next data item $d_3$ also has been scheduled in the accelerating branch while handing the previous query $q_3$. Hence, the system only needs to handle $d_3$ for $q_3$. According to the requested list of $q_3$, $d_8$ needs to be inserted at a position that is after $d_2$ and $d_3$. In the accelerating branch, $p_3 > p_2$ so that $d_8$ will be inserted under the $d_3$. Since $d_3$ already has a branch, the system creates a new branch of $d_3$, inserts an empty node after $d_3$, and tries to inserts a temporary $d_8'$ after the empty node.
the number of channels, inauguring to the indexing rule de
definition index channel and the result is shown in Figure 6. Accord-
From Lines 25 to 29, the process schedules the index items
strategy. The
system will
insert an empty node again and try to add a temporary
d nodes after
d in the accelerating branch. Hence, the final
FP-tree is shown in Figure 5(c).

Algorithm 3 presents the pseudocodes for the functions
of accelerating branch construction. FPBS. CreateAcceleratingBranch (Ț, S) is the main function
for constructing accelerating branch. The process calls the
subfunction FPBS_AddNodeForAcceleratingBranch (Ț, Ncurr, d) to insert a data item into the accelerating branch
of Ț at Line 6. Such a process is similar to the function
FPBS_AddNodeForBackbone (Ț, Np, d) in the backbone
construction. The operation at Line 7 calls another
subfunction FPBS_RangeSearch (Ț, Np) to check whether
the inserted data item is in the search range (or levels)
or not. The insertion will be illegal if the same data item
in the backbone of Ț locates at one of search levels. If
the insertion is illegal, the inserted nodes (including the
data item and empty node(s)) will be deleted at Line 47.

4.4. Stage 4: Schedule Mapping. After finishing stage III, the
system will map every slot (or level) of FP-tree into the
broadcasting channels using the breadth-first-search (BFS)
strategy. The final results are shown in Figure 6. Note that
the maximum number of data items in each slot (level) is
the number of channels, |C|. The mapping process is
described as the operations before Line 24 in Algorithm 4.
From Lines 25 to 29, the process schedules the index items
in index channel and the result is shown in Figure 6. Accord-
ing to the indexing rule defined in (1), the index I1 records
the information about who requests the data items in slot
3 and the index I6 records the similar information corre-
sponding to the data items in slot 1.

Consider the example of Table 1, for the request
q2 = \{d2, d3, d4\}, the final schedule in Figure 6 generated by the
proposed FPBS shows that the user can retrieve all the
requested data items d2, d3 (on c2), and d4 (on c1) within 4
time slots including a channel switching. If there is no accel-
inating branch, the user needs 5 time slots to retrieve data
items d2, d3, and d4 on c1. This result shows that the
proposed FP-tree can indeed reduce the access time.

5. Analysis and Discussion

In this section, we analyze the performance of FPBS in terms
of time complexity, space complexity, and access time.

5.1. Time Complexity. Suppose that the notations are defined
as above and the FP-tree is denoted as Ț, then, the time
complexity of the Ț’s construction will be \(\Theta(nk)\). The idea
of FP-tree design comes up from the FP-tree and only
one difference between them is that FP-tree needs to
insert an empty node when creating a new branch except
for the root node. In the last stage of the proposed method,
schedule mapping needs to maps all the data nodes of Ț to
the broadcasting channels and |Ț| \(\leq nk\), so the time com-
plexity of schedule mapping is also \(\Theta(nk)\). Due to the
nature of the FP-tree which is evolved from FP-tree, FPBS
costs \(\Theta(nk)\) in both average case and worst case. In sum-
mary, FPBS provides a polynomial algorithm for solving
the DBCA problem.

5.2. Space Complexity. After discussing the time complexity
of FPBS, we start to analyze the space complexity of FPBS.
In this part, we only consider the temporary space for FPBS
process. In the stage 1 of FPBS process, the system uses a
\(\Theta(nk)\) size table to store the sorted requests and the statisti-
cal information. In the stage 2, the system uses the obtained
sorted table to construct the backbone of an FP-tree and it
also costs \(\Theta(nk)\) space. In the stage 3, the system constructs
accelerating branches of the FP-tree and it costs \(\Theta(nk')\)
space, where 1 \(\leq k' \leq k\). In the last stage, the system just
maps the FP-tree to the channels and only costs \(\Theta(1)\) addi-
tional temporary space for traversing the FP-tree. That is,
the temporary space complexity during the scheduling
process is \(\Theta(nk)\).
1: **Function** FPBS_CreateAcceleratingBranch \((\mathcal{F}, S)\)
   **Input:** an FP*-tree \(\mathcal{F}\) and a sorted set of queries (clients) \(S\)
   **Output:** a final FP*-tree \(\mathcal{F}\)
2: let a temporary pointer \(N_{\text{curr}} \leftarrow R\);
3: create a temporary list \(q\) and a temporary node \(N_{\text{temp}}\);
4: for each query \(q\) in \(S\) do
5:   for each requested data \(d\) in \(q\) do
6:     \(N_{\text{temp}} \leftarrow \text{FPBS>AddNodeForAcceleratingBranch}(\mathcal{F}, N_{\text{curr}}, d)\);
7:     \(N_{\text{curr}} \leftarrow \text{RangeSearch}(\mathcal{F}, N_{\text{temp}})\);
8:     \(\text{list}_q\).add\((N_{\text{curr}})\);
9: if \(N_{\text{curr}}.\text{slot} > \mathcal{T}.\text{slot}\) then
10:   delete the path of \(\text{list}_q\) in \(\mathcal{T}\);
11: break;
12: end
13: end
14: \(\text{list}_q\).clear();
15: end
16: return \(\mathcal{T}\);
17: end
18: **Function** FPBS_AddNodeForAcceleratingBranch \((\mathcal{F}, N_p, d)\)
   **Input:** an FP*-tree \(\mathcal{F}\), the parent node \(N_p\), and a new data item \(d\)
   **Output:** an added node \(N_d\)
19: create a new node \(N_d\) with data item \(d\);
20: if \(N_p\) has children then
21:   if \(N_p\) has a child \(N_d'\) with \(d\) then
22:     return \(N_d'\);
23: else
24:   create an empty node \(N_e\);
25:   \(N_p\).addChild\((N_e)\);
26:   let a temporary pointer \(N_{\text{temp}} \leftarrow N_e\);
27:   while \(\mathcal{T}.\text{isOverload}(N_{\text{temp}}.\text{slot} + 1)\) do
28:     create an empty node \(N_e\);
29:     \(N_p\).addChild\((N_e)\);
30:     \(N_{\text{temp}} \leftarrow N_e\);
31: end
32: create a new node \(N_d\) with data item \(d\);
33: \(N_{\text{temp}}\).addChild\((N_d)\);
34: end
35: else
36: create a new node \(N_d\) with data item \(d\);
37: \(N_p\).addChild\((N_d)\);
38: end
39: return \(N_d\);
40: end
41: **Function** FPBS_RangeSearch \((\mathcal{F}, N_d)\)
   **Input:** an FP*-tree \(\mathcal{F}\) and a search node \(N_d\)
   **Output:** a result node \(N_d\) within the search range
42: int Num\(_e\) \leftarrow the number of \(N_d\)'s ancestors which are empty;
43: int startSlot \leftarrow \(N_{\text{temp}}.\text{slot} - \text{Num}_e + 1\);
44: int endSlot \leftarrow \(N_{\text{temp}}.\text{slot} + 1\);
45: for \(i \leftarrow \text{startSlot} \text{ to } \text{endSlot} \) do
46:   if find a node \(N_{\text{temp}}\) that has the same data as \(N_d\) does at level \(i\) of \(\mathcal{F}\) then
47:     delete the path that contains \(N_d\) and all the empty connected ancestors of \(N_d\);
48:     return \(N_{\text{temp}}\);
49: end
50: end

Algorithm 3: Continued.
5.3. Access Time. In wireless data dissemination environments, access time (or latency) is an important metric for validating the efficiency of scheduling. In FPBS, the system always first selects the request, whose size and average access frequency are maximum, and then schedules it in the backbone of FP*-tree. We then treat it as the base of schedule. That is, the access time for a request $q_i$ can be formulated as Theorem 8.

**Theorem 8.** Suppose that $\mathcal{F}$ is the maximal frequent item-set in the first-scheduled request, $\hat{t}$ is the minimum cost for channel switching, and $t_{\text{wait}}$ is the average waiting time from tuning into the channel to receiving the first required data item for a request, the access time for a request $q_i$ can be expressed as

$$
\text{acc}(q_i) = \begin{cases} 
  t_{\text{wait}} + |q_i| + \hat{t} + \sigma_2, & \text{if } (q_i \subseteq \mathcal{F}) \lor (q_i \cap \mathcal{F} = \emptyset), \\
  t_{\text{wait}} + |q_i \cap \mathcal{F}| + |q_i \setminus \mathcal{F}| + \hat{t} + \sigma_2, & \text{otherwise},
\end{cases}
$$

where $\hat{t}$ is the frequency of channel switching and $\sigma_2$ is the frequency of occupied slot (empty node in the FP*-tree) skipping.

**Proof.** With the use of index channel in FPBS, the average waiting time can be reduced efficiently. If $q_i \subseteq \mathcal{F}$, it means that all the required data items for $q_i$ can be obtained before the end of broadcasting all the data items in $\mathcal{F}$. In such a case, the access time for $q_i$ will be $t_{\text{wait}} + |q_i| + \hat{t} + \sigma_2$, where $|q_i| + \hat{t} + \sigma_2 \leq |\mathcal{F}|$. If $q_i \cap \mathcal{F} = \emptyset$ (is equivalent to $|q_i \cap \mathcal{F}| = 0$), it means that $q_i$ and $\mathcal{F}$ are two disjoint sets. In this case, the data items requested by $q_i$ only can be allocated after the first-scheduled maximal frequent item-set, so the access time for $q_i$ will be $t_{\text{wait}} + |\mathcal{F}| + |q_i| + \hat{t} + \sigma_2$.

However, the time $|\mathcal{F}|$ can be merged into the average waiting time $t_{\text{wait}}$ until accessing the first data item requested by $q_i$. Otherwise, for the case of $|q_i \setminus \mathcal{F}| > 0$, $q_i$ and $\mathcal{F}$ are two partially overlapping. It means that some required data items for $q_i$ will be scheduled after $\mathcal{F}$. Hence, the access time for $q_i$ will be $t_{\text{wait}} + |q_i \cap \mathcal{F}| + |q_i \setminus \mathcal{F}| + \hat{t} + \sigma_2$, where $|q_i \cap \mathcal{F}| + |q_i \setminus \mathcal{F}| + \hat{t} + \sigma_2 \geq |\mathcal{F}|$. 

After discussing the general case of access time, we also discuss the worst case in following Theorem 9.

**Theorem 9.** Suppose all the notations are defined as above. The worst case of access time will be

$$
\text{acc}_{\text{worst}} = t_{\text{wait}} + \sum_{j=1}^{n} |q_j|.
$$

**Proof.** In general, the worse case is the scenario that a client access the channels from the first time slot to the last time slot. In other words, the worse access time of FPBS will be the height of the FP*-tree. According to the design of FPBS approach, the accelerating branches of FP*-tree is impossible to be longer than the backbone of FP*-tree. Hence, the height of the FP*-tree $\mathcal{F}$ will be the height of the backbone, $\bigcup_{j=1}^{n} |q_j|$. In practice, each client tunes in channel at random time slot, so the access time in worst case $\text{acc}_{\text{worst}}$ will be $t_{\text{wait}} + \sum_{j=1}^{n} |q_j|$.
In FPBS, each data item is not replicated in the FP*-tree’s backbone and \(|\bigcup_{i=1}^{n} q_i|\). In this work, we focus on minimizing the average access time and the proposed FPBS approach can effectively shorten the access time of each request using the accelerating branches. In (2), the terms \(|q_i \cap F|\) and \(|q_i \setminus F|\) are uncertain since the relation between request \(q_i\) and the maximal frequent item-set \(F\) is unpredictable. Hence, FPBS focus on minimizing the frequencies of channel switching or occupied slot (empty node in the FP*-tree) skipping, such as \(\sigma_1\) and \(\sigma_2\) in (2). This problem is solved by FP*-tree using the accelerating branches in our proposed approach. In other words, FPBS is proposed for effectively make the upper bound of access time be tighter. Thus, the worst case becomes a very rare occurrence.

6. Simulation Results

We validate and discuss the performance of FPBS in terms of average access time by running the experimental simulations in different scenarios. The unit of time is a time slot. All the simulations are written in C++ and executed on a Windows 7 server which is equipped with an Intel (R) Core (TM) i7-3770 CPU @ 3.4 GHZ and 12G RAM. We use Quandl databases [31] to extract the U.S stock prices and then use the obtained stock dataset as the input of our simulation.

We assume that the maximum number of channels is 10 (|C| = 2, 3, …, 10) in the simulation. Therefore, we assume that one of the channels is the uplink for receiving the request and the remaining 10 channels are used as the downlink broadcasting channel. The detailed parameters of our simulations are shown in Table 5.

In the simulations, FPBS is conducted in online and offline modes. In the online mode, the system will use a buffer to keep the information of queries and request data items. When the buffer becomes full, the system will start to schedule data into the broadcasting channels. The scheduled data items will be removed from the buffer and new user demands are continuously coming in the buffer. It means that the FP*-tree and schedule may change during the simulation. Conversely, we assume that the system in the offline mode schedules the data after storing all the requested information in the buffer.

Note that there are two selecting strategies during scheduling process of FPBS, request-number-first and frequency-

---

**Algorithm 4:** The function used for the schedule mapping

```
1: Function FPBS.ScheduleMapping(\(\mathcal{F}, S, |C|\))
2:   Input: an FP*-tree \(\mathcal{F}\), a sorted query set \(S\), and the number of channels \(|C|\)
3:   Output: a scheduled channel set \(S_{channel}\) and a index channel \(I_{channel}\)
4:   let a list \(\text{list}\text{handling}\) ← \(\mathcal{T}.\text{root}.\text{children}\);
5:   let a temporary list \(\text{list}\text{next}\) ← ∅;
6:   create a data channel \(S_{channel}\) with \(|C|\) data broadcasting channels (or rows);
7:   create an index channel \(I_{channel}\) ← ∅;
8:   \(\text{int } i\);
9:   while \(\text{list}\text{handling}\) is not empty do
10:      \(i \leftarrow i + 1;\) /*\(i\) is used as a pointer to the current channel*/
11:      for each node \(N\) in \(\text{list}\text{handling}\) do
12:         if \(N\) is an empty node then
13:            break;
14:         else if \(N\).parent is an empty node then
15:            insert \(N\) into \(S_{channel}\) whose slot \(N\).slot is not occupied;
16:         else
17:            insert \(N\) into the \(i\)th channel;
18:         end
19:      end
20:      \(i \leftarrow i + 1;\)
21:   end
22:   copy every node of \(\text{list}\text{next}\) to \(\text{list}\text{handling}\);
23:   \(\text{list}\text{next}\).clear();
24:   \(\text{end}\)
25:   for \(j \leftarrow 1\) to \(\mathcal{T}.\text{height()}\) do
26:      for \(i \leftarrow 1\) to \(|C|\) do
27:         Use \(S\) to check who requests the data item in the slot determined by (1) and channel \(C_i\) of \(S_{channel}\) and then update this information to \(I_{channel}[j];\)
28:   end
29:   \(\text{end}\)
30:   return \(I_{channel}, S_{channel}\)
31: \(\text{end}\)
```
first. Request-number-first strategy is to select the query according to the length of its requested data items first and then selecting the query according to its average access frequency if multiple queries request same number of data items. Frequency-first strategy is to select the query according to its average access frequency first and then select the query according to the length of its requested data items if multiple queries have the same average access frequency. Hence, we discuss the above two strategies in online and offline modes, respectively.

To the best of our knowledge, none of existing works model the optimal performance of the multi-item request scheduling simultaneously considering the channel switching and dependencies between different requests over multi-channel dissemination environments. Only [13] provides a heuristic algorithm, UPF, to discuss the similar problem. This is the reason that we choose UPF as the comparative baseline in the simulations.

6.1. Size of Dataset. In the first simulation, we discuss the performance of FPBS with different sizes of dataset in terms of average access time. Note that the size of dataset indicates the number of different data items stored in the dataset. Figure 7 shows the results in three different cases if the

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Range (type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of dataset,</td>
<td>500</td>
<td>100, 300, 500, 700, 900</td>
</tr>
<tr>
<td>Number of users</td>
<td>5000</td>
<td>—</td>
</tr>
<tr>
<td>Maximum number of requested data items,</td>
<td>10</td>
<td>2, 4, 6, 8, 10</td>
</tr>
<tr>
<td>Number of downlink broadcast channels,</td>
<td>6</td>
<td>2, 3, …, 10</td>
</tr>
<tr>
<td>Size of buffer</td>
<td>3000</td>
<td>500, 1000, …, 4500</td>
</tr>
</tbody>
</table>

Figure 7: Effect of the different sizes of dataset with different number of channels: (a) |C| = 3, (b) |C| = 6, and (c) |C| = 9.
number of channels $|C| = 3$, $|C| = 6$, and $|C| = 9$, respectively. In the $|C| = 3$ channels environment, as shown in Figure 7(a), UPF can outperform the online FPBS approaches, FPBS-Fre-Online and FPBS-Rn-Online, if the size of dataset, $|D|$, is smaller than 800. The offline FPBS, FPBS-Fre and FPBS-Rn, can always have a better performance than UPF does in all different sizes of dataset.

The results depicted from Figures 7(a)–7(c) show that UPF has similar performances in different number of channels environments and the trends of UPF’s average access
time are always linear increasing. According to the results in Figures 7(b) and 7(c), we can know that both of online and offline FPBS approaches can outperform UPF in different sizes of datasets when $|C| \geq 6$. Additionally, the frequency-first strategy, FPBS-Fre, always has the best performance in different scenarios.

6.2. Number of Channels. In this part, we discuss the performance of FPBS in different scenarios that the number of broadcasting channels is set from 2 to 20 and the results are shown in Figure 8. The results indicate the existing method, UPF, is not suitable to multiple channel ($C \geq 4$) broadcasting environments and UPF cannot dynamically
schedule data items with the consideration of each user’s requests. That is to say, in comparison with the proposed approach, UPF can not utilize these channels if \( C \geq 4 \). Figures 8(a) and 8(b) show that UPF has a stable performance in the broadcasting environments with different number of channels when the size of dataset is small \( |D| \leq 300 \). Conversely, the results from Figures 8(c)–8(e) show that the average access time of UPF is unstable and becomes a slightly increasing trend when the size of dataset becomes large \( |D| \geq 500 \). The possible reason for this result is that UPF aims to minimize the request miss rate, not the average access time. There may be a trade-off between minimizing the request miss rate and the average access time.

Figures 8(a) and 8(b) shows the results of each approach in small dataset. FPBS in the offline mode, FPBS-Fre and FPBS-Rn, can have a better performance since the system consider all the requests while constructing the FP∗-tree. According to the results in Figures 8(c)–8(e), the frequency-first strategies, FPBS-Fre and FPBS-Fre-Online, have better performances than the request-number-first strategies, FPBS-Rn and FPBS-Rn-Online, when the size of dataset becomes large \( |D| \geq 500 \).

6.3. Number of Requested Data Items. If the number of requested data items becomes larger, the possibility of data dependency between each query becomes higher. In this subsection, we consider the effect of the different number of requested data items on the average access time. As shown in Figure 9, one can observe that all the FPBS-based approaches can outperform UPF when the maximum number of requested data items \( q_{max} \) is smaller than 11. When \( q_{max} \) is 2, all the FPBS-based approaches have similar performances on the average access time. As the value of \( q_{max} \) increases, the average access time in all the FPBS-based approaches also increases linearly.

According to the result in Figure 9, we can know that the frequency-first strategies are better than the request-number-first strategies since the performances of FPBS-Fre and FPBS-Fre-Online are more smoothly increasing than the performances of FPBS-Rn and FPBS-Rn-Online. In addition, FPBS-Fre can has the best performance and its trend is almost parallel to the trend of UPS’s performance.

6.4. Buffer Size. In the last simulation, we discuss the effect of the different size of buffer on the average access time for comparing two proposed online approaches, FPBS-Rn-Online and FPBS-Fre-Online. We also consider the trend of performance in some scenarios that the number of channel is, respectively, set to 3, 6, and 9.
The result in Figure 10 indicates that both FPBS-Rn-Online and FPBS-Fre-Online can have shorter average access time as the size of buffer increases. In an environment providing small number \( (C = 3) \) of channels, as shown as Figure 10(a), FPBS-Fre-Online can has a slightly better performance than FPBS-Rn-Online does when the buffer can store more than 2500 data items. The results in Figures 10(b) and 10(c) show that FPBS-Fre-Online is much better than FPBS-Rn-Online with different size of buffer when the number of channels increases \( (C \geq 6) \).

### 6.5. Open Issues
In this subsection, we summarize some remaining issues (or potential challenges) in on-demand multi-channel data dissemination systems as follows:

(i) Hardware constraint: although the minimum cost \( \tilde{t} \) for channel switching is normalized as one time slot in FPBS, it is difficult to implement a broadcasting system that meets this condition due to hardware limitations.

(ii) Cross-layer system design: in this paper, we design a server-side data scheduling for serving the multi-item requests. For wireless networks, the time-varying and uncertain nature of wireless channels can be considered in the scheduling. Thus, the server needs a new cross-layer system design to simultaneously access the request information in the application layer and channel information in the physical layer and then schedule data items more efficiently.

### 7. Conclusion
In this paper, we investigate and formulate an emerging problem, DBCA, in multichannel wireless data dissemination environments. We also prove that the DBCA problem is \( \mathcal{NP} \)-complete. Then, we present a heuristic scheduling approach, FPBS, to avoid data conflicts on multiple broadcasting channels. In FPBS, we use frequent patterns of requested data items to build a FP*-tree for extracting the correlation between each received request. Thus, data conflicts can be avoided. During the construction of FP*-tree’s accelerating branch, adding empty nodes at appropriate positions makes the user client have sufficient time to switch the channel for obtaining the required data. We not only analyze that FPBS can be done in polynomial time but also present the upperbound of access time of a request which is related to size of dataset. According to the simulation results, FPBS is much better than the existing work, UPF, in most of cases.

### Data Availability
The stock dataset used to support this study is available online and is cited as a reference [31] in relevant places in the text. The program data used to support the findings of this study are available from the corresponding author upon request.

### Conflicts of Interest
The authors declare that they have no conflicts of interest.

### Acknowledgments
This research was supported by the Ministry of Science and Technology, Taiwan under Grant Nos. MOST 107-2221-E-027-099-MY2, MOST 109-2221-E-027-095-MY3, and MOST 110-2222-E-035-004-MY2.

### References


