Efficient Parallel Data Retrieval Protocols with MIMO Antennae for Data Broadcast in 4G Wireless Communications*

Yan Shi[1], Xiaofeng Gao[2], Jiaofei Zhong[1], and Weili Wu[1]

[1] Department of Computer Science, The University of Texas at Dallas
{yanshi,fayzhong,weiliwu}@utdallas.edu
[2] School of Science and Technology, Georgia Gwinnett College
xgao@ggc.edu

Abstract. *Wireless Data Broadcast* is an efficient data dissemination method for public information to a large number of mobile/wireless clients. With the advance of the fourth-generation wireless communication system (4G), mobile devices may embed *multiple-input multiple-output* (MIMO) antennae to setup multi-connections to a base station. In this paper, we deal with data retrieval problem for mobile clients with MIMO antennae to retrieve a set of indexed data from parallel communication channels. Our purpose is to construct fast and energy efficient data retrieval protocols to reduce the response time and energy consumption. We name this problem as *parallel data retrieval scheduling with MIMO Antennae* (PADRS-MIMO), and propose two greedy heuristics named *Least Switch Data Retrieval Protocol* (*Least-Switch*) and *Best First Data Retrieval Protocol* (*Best-First*). We are the first work to deal with data retrieval with MIMO antennae for wireless data broadcast. We analyze the performances of *Least-Switch* and *Best-First* both theoretically and practically. Simulation results prove the efficiency of the two protocols.

1 Introduction

With the explosive increase of wireless/mobile clients and development of wireless network technologies, *wireless data broadcast* has become an important data dissemination method for public information, such as stock activities, traffic conditions, weather reports, and flight schedules. In a typical wireless data broadcast system, a set of data items are broadcasted over several channels at a base station repeatedly. Clients can access onto channels, search for their requested data (usually through index), and then download the corresponding data.

The most important issues in data broadcast are the energy efficiency and query response time for mobile clients, since the majority of mobile devices have limited battery power and constraint lifetime. As a result, *tuning time* and *access latency* are two significant criteria to evaluate the performance of a data broadcast system. According to the architectural enhancements, each

* This work is supported by NSF grant CCF-0829993 and CCF-0514796.

P. García Bringas et al. (Eds.): DEXA 2010, Part II, LNCS 6262, pp. 80–95, 2010.

mobile device has two mode: active mode and doze mode. They can only process operations in active mode, while "sleep" in doze mode to save energy. Consider a process from the time a client requires some data, to the time when this client finishes downloading, *tuning time* denotes the total time a client keeps active, while *access latency* denotes the time interval for the whole process. Intuitively, a critical problem to improve the performance of a wireless data broadcast system is to reduce the access latency and tuning time for clients.

Index technology is an efficient technique to reduce tuning time (e.g., B^+-Tree [6], Huffman Tree [4], Hash Table [14], Exponential Index [13], Signature Index [17], etc). With the help of index technique, clients can first get the estimated waiting time offset and channel information of their requested data, sleep during this offset, wake up and tune in corresponding channel right before the target data appear. Formally, in an indexed parallel wireless data broadcast system, it takes three steps to retrieve a data item:

1. *Initial Probing*: the client tunes in some broadcast channel and decide when the next index is arriving;
2. *Searching*: the client searches through the indices and locate the requested data item on the broadcast channels;
3. *Retrieving*: the client tunes in the channel where the requested data item is at and download the data item when it arrives.

Based on different index techniques, the searching process may vary in different broadcast system. However, no matter what index technique is used, by the end of the searching process, the client should have the knowledge of the time offset and resided channel of the requested data. Since in this paper, we focus on the data retrieving process, we omit the discussion of searching process and index constructions and assume we know the locations of requested data.

If a client requires more than one data items, it needs to get the location of every data item, order them as a permutation, and download them one by one sequentially. If the order of data items to retrieve is not appropriate, the client may spend unnecessarily extra time for downloading. Thus, a time efficient schedule for retrieving data from parallel channels which can reduce access latency is very important for the performance of the whole wireless data broadcast system. Moreover, during the data retrieving process, the tuning time is always the time needed for downloading the requested data. However, a client may switch among broadcasting channels several times (say, disconnect from one communication channel, and then construct connection to another channel). Switching among channels costs additional energy consumption [5], and thus the number of switchings during the data retrieving process also has notable impact on the energy efficiency of a data retrieving protocol. Therefore, a good data retrieving protocol should be able to reduce both access latency and number of switchings among channels during the retrieving process.

Some literatures discussed the data retrieval protocols to download a set of data from multiple broadcasting channels [3][5][9]. Their common assumption is that each client only has one antenna (or one retrieving process). Each time

the antenna can access onto one communication channel to set up one connection. However, only one connection results in narrow bandwidth and small throughput, which is a crucial physical constraint for wireless networks to satisfy increasing requirements with Quality of Service (QoS) guarantees.

To solve this problem, the Fourth-Generation Wireless Communication System (4G) applies *multiple-input multiple-output* (MIMO) technology, which allows different data streams to be transmitted simultaneously from different transmitter antennae. 4G is a complete evolution which will become a total replacement of current Third-Generation Network System (3G) in the next few years. It is predicted that at that time, the majority of mobile clients will be equipped with MIMO antenna for high-speed communications. They can access onto multiple channels in parallel and shorten the query processing time significantly.

As a result, the focus of our research is to discuss how to schedule the retrieving process of a set of requested data, given their time offset and resided channels, using a client with multiple antennae. Our target is to minimize the access latency and number of channels switchings for the client. In other words, by the employing protocols proposed in this paper, a client should be able to download a set of requested data using multiple retrieving processes in parallel, with short response time and minimum energy consumption. We name this problem as *Parallel Data Retrieval Scheduling with MIMO Antennae* (PADRS-MIMO). In this paper, we present the communication model, formally define the PADRS-MIMO problem, and construct two greedy heuristics named *Least Switch Data Retrieval Protocol* (*Least-Switch*) and *Best First Data Retrieval Protocol* (*Best-First*). We are the first work to discuss the data retrieval with MIMO antennae for wireless data broadcast problem and propose practical solutions. We analyze the performance of *Least-Switch* and *Best-First* both theoretically and practically, and prove their effectiveness and efficiency by simulation results.

The rest of our paper is organized as follows: in Sec. 2 we list the related works to PADRS-MIMO; In Sec. 3 we propose the communication model and formally define PADRS-MIMO. Section 4 analyzes the nature of PADRS-MIMO, constructs two data retrieval protocols for PADRS-MIMO which are illustrated with detailed examples and theoretical analysis. Section 5 evaluates the two algorithms' performances from several aspects by simulation results. Section 6 concludes the whole work and provides future directions for this topic.

2 Related Works

Multi-channel data broadcast has been a trend in the wireless data broadcast research area since it can significantly reduce the access latency by partitioning data onto multiple channels. Research works on multi-channel wireless data broadcast mainly focus on two aspects from the server side: how to schedule data on multiple channels to reduce access latency and how to design efficient indexing schemes to reduce tuning time.

Several literatures discussed the data scheduling problem on multiple channels in the wireless data broadcast environment. In [1], Ardizzoni et al. developed several algorithms based on dynamic programming techniques to optimally

schedule skewed data on multiple channels while preserving the flat broadcast scheme of each channel. Prabhakara et al. [7] provided a wide range of design considerations for the server which broadcasts over the multi-level multi-channel air cache to improve server's performance. Saxena et al. [8] presented a balanced on-line broadcast scheduling scheme which adopts a hybrid push-pull broadcast schedule per channel. In [15], how to minimize the average access latency by optimally partitioning data among multiple channels was discussed by Yee et al., and an approximation algorithm that is less complex than optimal solution yet with near-optimal performance was developed.

Indexing techniques of multi-channel wireless data broadcast not only discuss how to index data, but also concern about how to allocate indices, given multiple channels. One popular way to allocate index is to assign certain channels as designated index channels and others as data channels. Jung et al. [4] presented a tree-structured index allocation method to allocate index on separate channels from data, which minimized average access latency by broadcasting hot data and their indices more frequently than less hot data and their indices. Waluyo et al. [11] presented a global index schemes where each index channel preserves a part of the index tree with replication among each other. Wang and Chen [12] adopted the distributed indexing technique proposed in [6] to multiple channel environment, by creating an virtual index tree for each data channel and multiplexing them onto one physical index channel.

Despite of various literatures on scheduling and indexing problems on server's side, there is very little research about how to schedule the data retrieval process more efficiently. In [9], Sun et al. presented two algorithms to retrieve a set of requested data from parallel broadcast channels. Hurson et al. [5] gave other two heuristic algorithms to schedule data retrieving from multiple broadcast channels to reduce the access latency and number of switchings among channels. However, both work assume there can be only one process to retrieve data, and data are evenly allocated on multiple channels.

3 Problem Formulation

In this section, we will discuss the communication model of MIMO wireless data broadcast system and formulate the PADRS-MIMO problem.

3.1 Communication Model

A *program* is a complete broadcast cycle which contains a set of data and possibly some index information. Suppose a set of data $D = \{d_1, \cdots, d_{|D|}\}$ are broadcasted in a program. Without the loss of generality, assume the keys of data items $d_i.key$ in D are monotonically increasing. The popularity of data items in D are represented by their access probability. Let $P = \{p_1, \cdots, p_{|D|}\}$ denote the access probability set of D, where each p_i is the access probability of d_i and we have $\sum_{i=1}^{|D|} p_i = 1$. Assume all data items in D have the same size, which fits in one *data bucket* on the broadcast channel.

D is allocated on N channels according to some data allocation method with respect to their access probability. Let $bcast_i$ represent one round of broadcasting all data on the i^{th} channel, and $blength_i = |bcast_i|$. The $blength_i$ of channel i is decided by the data allocation method used. Different from [5] and [9], we assume that $blength_i$ is not necessarily of the same length, which is a common feature of most data allocation methods. Let *bcycle* represent the total length of a program. Note that if any update is needed, it will take place between two consecutive programs. Therefore, within one program, the length of each broadcast channel needs to be the same. This can be achieved by making *bcycle* equal to the least common multiple of $blength_i, i = 1, \cdots, N$. We define the time needed to broadcast one data bucket as one unit time. In a broadcast program, sequence numbers are assigned to each data bucket to represent its time offset from the beginning of the program, denoted as t, $1 \leq t \leq bcycle$.

Example 1. A data set D with 10 items are to be broadcasted. Their keys are $1, \cdots, 10$ respectively. The access probability is $P = \{0.1138, 0.0488, 0.0427, 0.3414, 0.0854, 0.0682, 0.0379, 0.0569, 0.0341, 0.1707\}$, which follows *zipf* distribution, a typical distribution used to model non-uniform access patterns of web data [4]. D is allocated on $N = 4$ broadcast channels using the *Dynamic Weight-Schedule* allocation method described in [3]. The data allocation result is illustrated in Fig. 1. Each data item is represented by its key value.

t:	1	2	3	4	5	6	7	8	9	10	11	12	
C_1	4	4	4	4	4	4	4	4	4	4	4	4	$bcast_1 = \{4\}$; $blength_1 = 1$
C_2	10	1	10	1	10	1	10	1	10	1	10	1	$bcast_2 = \{10,1\}$; $blength_2 = 2$
C_3	5	6	8	5	6	8	5	6	8	5	6	8	$bcast_3 = \{5,6,8\}$; $blength_3 = 3$
C_4	2	3	7	9	2	3	7	9	2	3	7	9	$bcast_4 = \{2,3,7,9\}$; $blength_4 = 4$
	←				*bcycle* = 12							→	

Fig. 1. Communication Model Example: Data Allocation of D

A mobile client has M antennae, which enables it to retrieve data using at most M processes in parallel. Due to technical constraints, the number of antennae is usually limited because of the size of mobile devices. The common number of antennae for mobile handsets is 2 or 3 [10]. On the other hand, the number of channels for a base station to broadcast data is relatively large. Therefore, we assume $M < N$.

As mentioned in Sec. 1, there are three steps to retrieve requested data from broadcasting channels. After *Initial Probing* and *Searching*, the client should know the location of each requested datum, which includes: key, resided channel, sequence number within the channel, and the channel blength, represented by a four-tuple $< key, ch, sq, blength >$ respectively. Based on the data allocation method, a data item d_i may be broadcasted multiple times in a program. The sq included in the tuple is defined as the sequence number of the first appearance of d_i in the broadcast program. For instance, if d_i is broadcasted on channel j,

its sq should be $1 \leq sq \leq blength_j$. The sequence numbers of all data buckets broadcasting d_i in a program can be easily computed given its sq and $blength$.

3.2 Parallel Data Retrieval Scheduling Problem

The parallel data retrieval scheduling problem in a MIMO wireless data broadcast system can be addressed as follows. Given a base station broadcasting a set of data on multiple channels, a client with multiple antennae has a request of a subset of the data broadcasted. The client would like to start the data retrieving from a certain starting time. The problem is: 1) how to assign antennae to retrieve different data items in the request; and 2) how to order the retrieval of data items for each antenna, so that we can reduce

- the access latency of the data retrieval,
- number of switchings among broadcast channels.

Obviously, to reduce access latency is to reduce the time needed to retrieve a request. The reason why we want to reduce the number of switchings among channels is for the sake of energy efficiency. As discussed in [5], switching among channels also consumes energy. In general, one switching takes 10% of the active mode power consumption. However, the objectives of reducing access latency and reducing number of switchings can be contradictive to each other.

Example 2. In the broadcast program shown in Fig. 2, suppose the grey circled data items {1,2,3,4} are of request. The client has only one antenna and the starting point of retrieving process is at $t = 1$. If we want to minimize the access latency, the request should be retrieved in the order of "$3 \rightarrow 1 \rightarrow 4 \rightarrow 2$" which takes only 7 time units but needs 3 switchings. However, if we want to minimize the switchings, the best retrieving order should be "$3 \rightarrow 4 \rightarrow 2 \rightarrow 1$" which needs only 1 switching but takes 12 time units. This example shows that access latency and number of switchings can not be minimized at the same time.

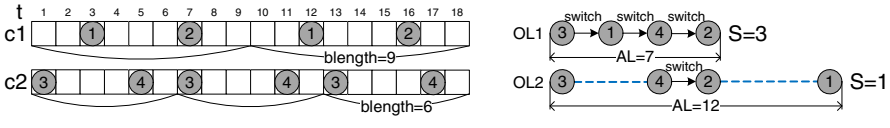

Fig. 2. Example of Possible Objective Contradiction

To balance the two objectives mentioned above, we introduce a cost function that takes both of them into consideration:

Definition 1. *The cost of a parallel data retrieving schedule is evaluated as* $c = \alpha \cdot AT + \beta \cdot S$, *where* AT *is the access latency of the data retrieving process from the starting time (initial probing and searching processes not included);* S *is the number of switches among channels during the retrieving process.*

α and β are adjustable parameters which can reflect the user's preference. For example, if the user's first priority is to get the requested data faster, α should be increased; if energy is of more concern, β should be increased.

Definition 2. *A request* $\mathbf{R} = \{R_1, \cdots, R_w\}$ *is a client request to retrieve* w *data items in a broadcast program. Each* $R_i \in \mathbf{R}$ *is a four-tuple:* $< key, ch, sq, blength >$. $\{R_i.key | R_i \in \mathbf{R}\} \subseteq \{d_j.key | d_j \in D\}$.

Given a request $\mathbf{R}$, there may be some channels that do not contain any data items in $\mathbf{R}$. During data retrieving, these channels will not affect the retrieving process at all, and thus can be ignored. In other words, when retrieving a request, we are only interested in broadcast channels which has requested data on them. Therefore, we have the following definition:

Definition 3. *Given* $\mathbf{R}$*, let* $CH = \{ch_1, \cdots, ch_K\}$, $K = |\bigcup_{R_i \in \mathbf{R}} \{R_i.ch\}|$. CH *is a set of requested channels where requested data reside.* $\forall ch_i \in CH$*, define:*

- $ch_i.begin = \min_{R_j.ch = ch_i} \{R_j.sq\}$*: starting sequence number of required data on* ch_i*;*
- $ch_i.end = \max_{R_j.ch = ch_i} \{R_j.sq\}$*: ending sequence number of required data on* ch_i*;*
- $ch_i.blength$*: blength of* ch_i*.*

Definition 4. *Given* $\mathbf{R}$*, a data retrieval schedule is a set of ordered lists of data items (represented by their key values)* $\mathbf{OL} = \{OL_1, \cdots, OL_M\}$ *for* M *processes, where each* OL_i *provides a schedule for one process. There should be* $\bigcup_{OL_j \in \mathbf{OL}} \{d_i.key | d_i \in OL_j\} = \bigcup_{R_j \in \mathbf{R}} \{R_j.key\}$. $\forall OL_i \in \mathbf{OL}$*, define:*

- $OL_i.t$ *as the first time slot after retrieving the last data item in* OL_i*;*
- $OL_i.ch$ *as the channel of the last data item in* OL_i*;*
- $OL_i.c$ *as the total downloading cost for* OL_i*, where* $c = \alpha AT + \beta S$.

In data retrieving, we need to pay attention to possible conflicts of retrieving data buckets on parallel channels. If a retrieving process is on ch_i at time t, there will be a conflict if it tries to download a datum on ch_j $(j \neq i)$ which also appears at t, because the time needed to switch between channels is not neglectable, which is almost equivalent to broadcasting one data bucket [5]. More formally:

Definition 5. *For a retrieving process* OL_i*, a conflict will occur if it wants to retrieve* R_j *at* $OL_i.t$*, where* $R_j.sq = OL_i.t \% R_j.blength$*, and* $R_j.ch \neq OL_i.ch$.

Based on the above definitions, PADRS-MIMO can be more formally defined:

Definition 6. *Given* D *broadcasted on* N *channels, a client with* M *antennae has a request* $\mathbf{R}$*.* t_0 *is a predefined starting point in a program for the client to start retrieving* $\mathbf{R}$*, which is not necessarily the beginning of a program. The* parallel data retrieval scheduling with MIMO Antennae *problem (PADRS-MIMO) is to develop a function* $f : \{\mathbf{R}, t_0\} \rightarrow \mathbf{OL}$ *to produce a schedule* $\mathbf{OL}$ *without any conflicts, so that the cost function* c *defined in Def. 1 is as small as possible.*

Let us use an example to illustrate the above definitions.

Example 3. In the broadcast program in Example 1, assume a client with $M = 2$ antennae has a request $R = \{1, 2, 6, 8, 9\}$. The predefined starting time is $t_0 = 3$. The cost function parameters are $\alpha = 1$ and $\beta = 2$. The request channel set CH and a schedule $\mathbf{OL} = \{OL_1, OL_2\}$ are shown in Fig. 3. If OL_1 wants to retrieve d_4 at time 9, there will be a conflict because $OL_1.t = 9$ and $OL_1.ch = 2 \neq 1$.

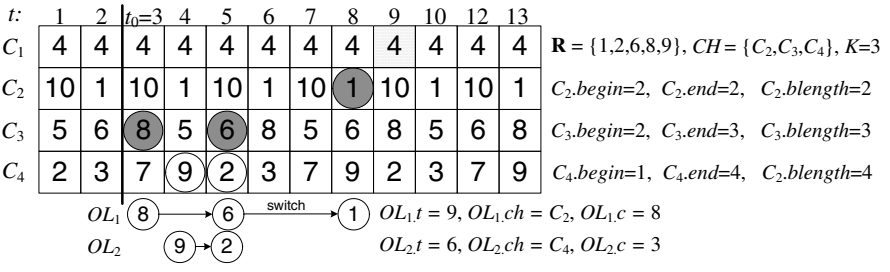

Fig. 3. Example of R, CH, **OL** and conflict

4 Efficient Parallel Data Retrieval Scheduling

As discussed in Sec. 3, how to schedule the request to multiple retrieving processes can significantly influence the access latency and energy consumption for clients. In this section, we introduce several algorithms to solve PARDRS-MIMO.

We first consider the relation between K: number of channels where requested data **R** locates; and M: number of available processes a client can use to retrieve data in parallel. When $K \leq M$, it is obvious that the optimal schedule is to assign a different process to retrieve requested data on each channel. This schedule requires only K processes. In the following discussion for algorithm construction, we assume that $K > M$. We will present two scheduling algorithms of parallel data retrieval for clients equipped with MIMO antennae. The first algorithm is *Least Switch Data Retrieval Scheduling* (*Least-Switch*), which guarantees the least switching number S, and tries to minimize AL, while the second algorithm is *Best First Data Retrieval Scheduling* (*Best-First*), which minimizes global cost function c.

4.1 Least Switch Data Retrieval Scheduling (*Least-Switch*)

For one retrieving process, switching among channels not only will consume energy, but also may introduce unnecessary conflicts. An intuitive thought is to minimize the total number of switchings among channels during the complete data retrieval. Therefore, we develop an algorithm which guarantees the least number of switchings among channels. Alg. 1 presents the *Least-Switch* protocol and Alg. 2 computes the cost function used in Alg. 1.

Least-Switch (Alg. 1) can be decomposed into two steps. The first step (Line 2 to 4) is to find the M channels to read. The criteria is the time needed for each channel to retrieve all requested data on the channel starting from t_0,

which is evaluated by cost function c. M channels with the longest time will be assigned to the M processes. The second step (Line 5 to 8) is to read the rest $K - M$ channels. Choose the process with least cost, append to it the channel which requires the longest time to retrieve all requested data on it. If several processes have the same cost, randomly choose one to proceed. The M processes keep reading the remaining channels until all K channels are read and R is completely retrieved. Note that in Algorithm 1, once we append a channel ch'_j to an OL_i, it means appending every data item on this channel to OL_i with the order calculated from $OL_i.t$ and $ch'_j.blength$. $OL_i.t$, $OL_i.c$ and $OL_i.ch$ should also be changed correspondingly afterwards.

Algorithm 1. Least Switch Data Retrieval Scheduling (*Least-Switch*)

Input: R; CH; t_0.
Output: OL.
1: $\forall 1 \leq i \leq M$, $OL_i.t = t_0$, $OL_i.c = 0$, $OL_i.ch = \emptyset$.
2: Sort CH in descending order by $c(OL_1, ch_i, 1, 0)$ as CH'.
3: Append $ch'_1, \cdots, ch'_M$ to $OL_1, \cdots, OL_M$ correspondingly.
4: $CH' = CH' \setminus \{ch'_1, \cdots, ch'_M\}$.
5: **while** $CH' \neq \emptyset$ **do**
6: $OL_i^* = \arg\min_{OL_i \in \mathbf{OL}}\{OL_i.c\}$; append $ch_j'^* = \arg\max_{ch'_j \in CH'}\{c(OL_i^*, ch'_j, 1, 0)\}$ to OL_i^*.
7: $CH' = CH' \backslash \{ch_j'^*\}$.
8: **end while**

The reason why in *Least-Switch* we every time append the longest channel to the fastest process is that access latency of the complete data retrieval is determined by the process which takes the longest time. Therefore, we would like to balance the time needed by each process to avoid delay caused by some process which is much more slower than others. This can be achieved by keeping appending the "longest" channel remained to the fastest process.

Algorithm 2, the computation of the cost function plays an importance role in the *Least-Switch*. The input of the cost function is candidate process OL_i, channel ch_j, and parameters α and β. It will return the cost of downloading all requested data on ch_j starting from $OL_i.t$. Since *Least-Switch* always guarantees the minimum number of switchings, it is not necessary to consider switching during evaluating the cost function. Therefore, we set $\alpha = 1$ and $\beta = 0$.

For initial channel assignment to M processes, no switching is needed to tune in the target channel and thus no conflicts will occur. There are three possible cases (Line 3 to 7):

Case 1: If the starting point $local = OL_i.t\%ch_j.blength$ is before or at $ch_j.begin$ in a broadcast round on ch_j, the access latency is simply the distance between $ch_j.end$ and the starting point (Line 10).

Case 2: If the starting point is between $ch_j.begin$ and $ch_j.end$ or at $ch_j.end$, the access latency will be $ch_j.blength - local + g_2 + 1$, where g_2 is the requested data appeared right before or at $local$.

Case 3: If *local* is after $ch_j.end$, the process needs to wait till the next $ch_j.end$ to finish downloading all requested data on ch_j, that is, $ch_j.blength - local + ch_j.end + 1$ time slots.

When $OL_i.ch \neq ch_j$ (Line 8 to 13), possible conflicts should be considered. The computation of cost is similar except for two differences: 1) If *local* lies exactly at $ch_j.begin$, the access latency should be computed as Case 2. This is because the data item at $ch_j.begin$ will be the last requested data on ch_j available for OL_i due to the conflict; 2) Similarly, if *local* lies at $ch_j.end$, the access latency should be computed as Case 3.

Algorithm 2. Cost Function $c(OL_i, ch_j, \alpha, \beta)$

Input: OL_i, ch_j, α, β.
Output: Cost for OL_i to download data on channel ch_j.
1: $local = OL_i.t\%ch_j.blength$ ▷ if $local = 0$, let $local = blength$
2: $g_1 = \max_{R_k.ch=ch_j}\{R_k.sq | R_k.sq < local\}$; $g_2 = \max_{R_k.ch=ch_j}\{R_k.sq | R_k.sq \leq local\}$.
3: **if** $OL_i.ch = \emptyset$ **then** ▷ calculate cost at initial stage
4: $S = 0$;
5: **if** $local \leq ch_j.begin$ **then** $AL = ch_j.end - local + 1$;
6: **else if** $ch_j.begin < local \leq ch_j.end$ **then** $AL = ch_j.blength - local + g_1 + 1$;
7: **else** $AL = ch_j.blength - local + ch_j.end + 1$; **end if**
8: **else** ▷ calculate cost with switch
9: $S = 1$;
10: **if** $local < ch_j.begin$ **then** $AL = ch_j.end - local + 1$;
11: **else if** $ch_j.begin \leq local < ch_j.end$ **then** $AL = ch_j.blength - local + g_2 + 1$;
12: **else** $AL = ch_j.blength - local + ch_j.end + 1$; **end if**
13: **end if**
14: Return $c = \alpha \times AL + \beta \times S$

Lemma 1. *The minimum number of switchings for a client with M processes to download requested data allocated on K channels is $K - M$ (when $K > M$).*

Proof. Consider one process first. If data are located on K channels, the process has to visit each channel at least once. Suppose it first accesses ch_i, it has to switch to the rest $K-1$ channels. Hence, $\min S = K-1$. If we have M processes, only the first M channels accessed do not need switchings, so $\min S = K-M$. □

Theorem 1. Least-Switch *guarantees minimum switchings to download* **R**.

Proof. From Line 3 in Alg. 1, M processes will access onto M channels without switching. According to the procedure between Line 5 and 8, and the fact that each data will only be downloaded once, each candidate channel $ch_i \in CH'$ will only be visited once. Therefore, in total *Least-Switch* has $S = K - M$. By Lemma 1, this algorithm guarantees the minimum number of switchings. □

Example 4. Fig. 4 is the result of *Least-Switch* with the setting in Example 3. The eclipses are the decision procedure and rectangles are the resulting schedules.

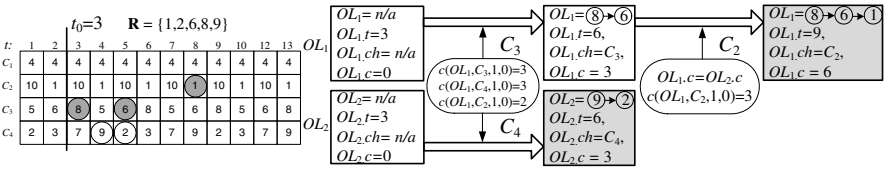

Fig. 4. Example of *Least-Switch* scheduling scheme

4.2 Best First Data Retrieval Scheduling (*Best-First*)

Least-Switch discussed in Section 4.1 can minimize the switching among channels during the entire data retrieval. However, if two requested data items on one channel are far from each other, the retrieving process reading that channel may miss the chance to retrieve available data items on other channels during waiting and thus increase the access latency. In order to take account of both the access latency and number of switching, we introduce the second algorithm which adopts the idea of best-first search, as illustrated in Alg. 3 and Alg. 4.

Algorithm 3. Best First Data Retrieval Scheduling(*Best-First*)

Input: $\mathbf{R}$; CH; t_0.
Output: $\mathbf{OL}$.
1: $\forall 1 \leq i \leq M$, $OL_i.t = t_0$, $OL_i.c = 0$, $OL_i.ch = \emptyset$.
2: $\forall 1 \leq i \leq K$, let $ch_i.R^*$ be the R_j that appears first on ch_i after t_0.
3: Sort CH ascendingly by $c(OL_1, ch_i.R^*, \alpha, \beta)$ as CH'. To break tie, select the one with longer $ch_i.blength$ if two channels have the same cost.
4: Append $ch'_1.R^*, \cdots, ch'_M.R^*$ to $OL_1, \cdots, OL_M$ correspondingly.
5: $\mathbf{R} = \mathbf{R} \setminus \{ch'_1.R^*, \cdots, ch'_M.R^*\}$; Update CH.
6: **while** $|CH| > M$ **do**
7: $OL_i^* = \arg\min_{OL_i \in \mathbf{OL}}\{OL_i.c\}$; append $R_j^* = \arg\min_{R_j \in \mathbf{R}}\{c(OL_i^*, R_j, \alpha, \beta)\}$ to OL_i^*.
8: $\mathbf{R} = \mathbf{R} \backslash \{R_j^*\}$; Update CH.
9: **end while**
10: Sort $\mathbf{OL}$ descendingly by $OL_i.c$.
11: **for** $i = 1$ to M **do**
12: Append $ch_j'^* = \arg\min_{ch'_j \in CH'}\{c(OL_i, ch'_j, \alpha, \beta)\}$ to OL_i; $CH = CH \setminus \{ch_j'^*\}$.
13: **end for**

Best-First (Alg. 3) can be interpreted as three phases:

Phase 1: *Initial Assignment* (Line 1 to 5). Starting from t_0, for each channel $ch_i \in CH$, $ch_i.R^*$ is its first appeared requested data item. All K channels will be sorted in ascending order by their $ch_i.R^*$ into CH'. $ch_i.R^*$ of the first M ch'_i will be assigned to M processes respectively and removed from request list R. $OL_i.t$, $OL_i.c$ and $OL_i.ch$ should also be updated correspondingly. If there is any tie occurred, the requested data whose channel has longer blength will be selected. This is out of the consideration that requested data on channels with

longer blength appear less frequently in the broadcast program, and thus the client will need longer time to wait for it.

Phase 2: *Best First Assignment* (Line 6 to 9). The idea of best first search is adopted in this phase to choose the next data item to retrieve. We first choose the process OL_i^* with least cost, and append R_j^* which needs the least cost for OL_i^* to retrieve. Note that for OL_i^*, there might be multiple R_j^* with the same anticipated cost to retrieve. To break the tie, R_j^* with longer blength will be chosen. $OL_i.t$, $OL_i.c$ and $OL_i.ch$ should be updated after the assignment. If ch_i does not contain requested data any more, it should be removed from CH. This procedure will continue until there are only M channels with requested data left.

Phase 3: *Final Assignment* (Line 10 to 13). When there are only M channels with requested data on them, they will be assigned to the M processes with the same reason as if $K \leq M$. M processes will first be sorted by their total cost so far. Starting from the process with the most cost, it will append the channel that costs least for it to finish retrieving. Once again, the reason of doing this is to balance the cost of each process in order to avoid unnecessary delay caused by possible "super-costly" process.

Algorithm 4. Cost Function $c(OL_i, R_j, \alpha, \beta)$

Input: OL_i, R_j, α, β.
Output: Cost for OL_i to download data R_j.

1: $local = OL_i.t\%R_j.blength$ ▷ if $local = 0$, let $local = blength$
2: **if** $OL_i.ch = R_j.ch \vee OL_i.ch = \emptyset$ **then** ▷ initial stage or no switch
3: $S = 0$;
4: **if** $local \leq R_j.sq$ **then** $AL = R_j.sq - local + 1$;
5: **else** $AL = R_j.blength - local + R_j.sq + 1$; **end if**
6: **else** ▷ switch from one channel to another
7: $S = 1$;
8: **if** $local \leq R_j.sq - 1$ **then** $AL = R_j.sq - local + 1$;
9: **else** $AL = R_j.blength - local + R_j.sq + 1$; **end if**
10: **end if**
11: Return $c = \alpha \times AL + \beta \times S$

Different from Alg. 2, the cost computed in *Best-First* is related to single requested data item, instead of a channel. As described in Al. 4, For each OL_i and R_j pair, there are two possible cases: 1) OL_i is on the same channel as R_j or OL_i is still empty (Line 2 to 5) and 2) they are on different channels (Line 6 to 10). In the first case, no switching between channels is needed, and access latency can be easily computed. In the second case, one switching is required, and the possible conflict should be considered.

Example 5. With the setting in Example 3, *Best-First* is performed as shown in Fig. 5, with $\alpha = 1$ and $\beta = 2$. The eclipses are the decision procedure and rectangles are the resulting schedule.

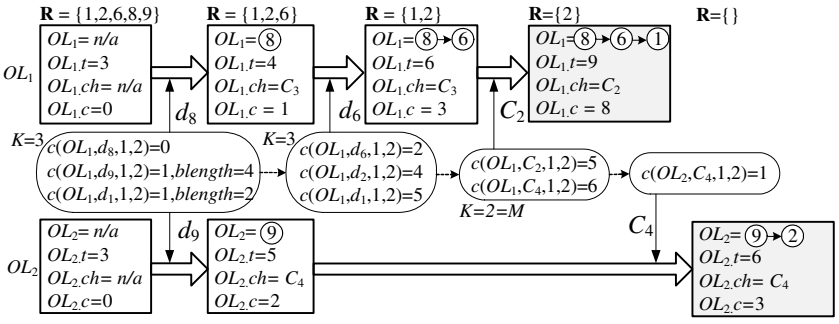

Fig. 5. Example of *Best-First* scheduling scheme

5 Performance Analysis

In this section, we will use simulation result to discuss the characteristics of PADRS-MIMO and evaluate the performances of *Least-Switch* and *Best-First* data retrieval scheduling schemes.

5.1 Simulation Setup

Simulation is implemented in Java 1.6.0 16 on an Intel(R) Xeon(R) E5520 computer with 6.00GB memory, with Windows 7 version 6.1 operating system. We simulate a base station with N broadcast channels, and multiple clients with various requests of data. The database to be broadcast has 10000 items [16], each of size 1KB. The access probability of the database follows zipf distribution [2], which is a typical model for non-uniform access patterns [4,12]. We adopt *Dynamic Weight-Schedule* [3] for data allocation. N varies from 5 to 30. (α, β) are set to $(1, 0)$ and $(1, 2)$ for *Least-Switch* and *Best-First*.

Clients have multiple antennae to retrieve the data from broadcast channels. The number of antennae varies from 1 to 10. The size of a request varies from 10 to 1000. For each experiment, we generate 100 requests to get their average access latency and number of switchings during data retrieval. Requests are generated according to data's access probability.

5.2 Simulation Results

When $N = 20$, $|\mathbf{R}| = 100$, we vary the number of antennae M.Access latency is measured in unit time (the time needed to broadcast one data bucket). Fig. 6 shows that when M is small (M =1, 2 or 3), *Best-First* takes much shorter access latency than *Least-Switch*, while when M is relatively large ($M \geq 4$), both protocol takes similar access latency. Due to current technique constraints of the number of antennae in mobile devices (usually 2 or 3 antennae [10]), we can claim that *Best-First* protocol has advantage in reducing response time of request retrieving. However, from Fig. 9, it is clear that *Best-First* needs much more

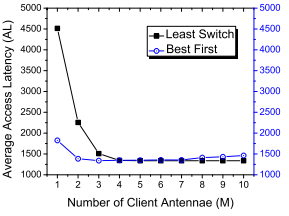

Fig. 6. Impact of M: Average AL ($N = 20$, $|\mathbf{R}| = 100$)

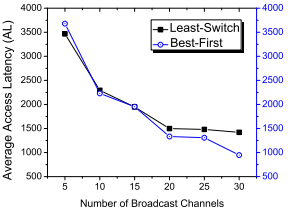

Fig. 7. Impact of N: Average AL ($M = 3$, $|\mathbf{R}| = 100$)

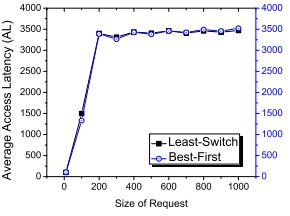

Fig. 8. Impact of $|\mathbf{R}|$: Average AL ($M = 3$, $N = 20$)

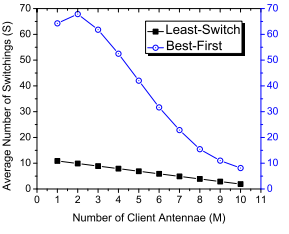

Fig. 9. Impact of M: Average S ($N = 20$, $|\mathbf{R}| = 100$)

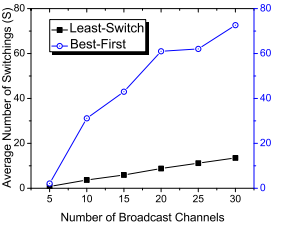

Fig. 10. Impact of N: Average S ($M = 3$, $|\mathbf{R}| = 100$)

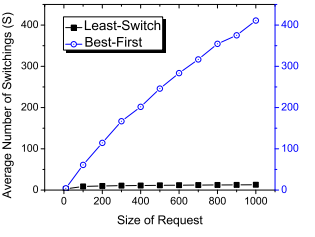

Fig. 11. Impact of $|\mathbf{R}|$: Average S ($M = 3$, $N = 20$)

switchings during retrieval than *Least-Switch*, which guarantees the minimum number of switchings. When M decreases, the number of switchings needed by *Best-First* is dropping dramatically, because more data will be retrieved during its third phase and thus it needs not switch among channels any more.

Next, we evaluate the impact of the number of broadcast channels on the two protocols' performances. The average access latency and number of switchings needed are shown in Fig. 7 and Fig. 10, given $M = 3$ and $|\mathbf{R}| = 100$. We observe that when N increases, access latency of both protocols decrease similarly. This shows the benefit of using more broadcast channels with respect to reducing the access latency. For *Least-Switch* protocol, the number of switchings increases only a bit when N increases. However, for *Best-First*, number of switchings increases significantly when N increases.

We are also interested in how the size of requests can influence the performances of two protocols. Fig. 8 presents the change of average access latency with the increasing of request size. Both protocols behave similarly. The access latency increases rapidly when request size first increase from 10 to 200. After that, it becomes relatively stable despite of the increasing request size. This is because when the number of requested data increases, the requested data will appear more frequently on broadcast channels and thus retrieval protocols can download them more continuously without having to wait extra time. As regards to number of switchings as shown in Fig. 11, *Best-First* have to switch much

more times when the request size increases, while *Least-Switch* remains similar amount of switchings.

Based on the above observations, we can conclude that the two protocols proposed in this paper have their own advantages. With limited number of antennae in mobile devices, *Best-First* can significantly reduce the response time needed to download client requests, while *Least-Switch* can guarantee minimum energy consumption during data retrieving and also provide similar response time as *Best-First* when there are reasonable amount of antennae available.

6 Conclusions

In this paper, we are the first to propose data retrieval scheduling problem for mobile clients with MIMO antennae (PADRS-MIMO), which is a promising technique within the emerging 4G wireless network. We formally define PADRS-MIMO, analyze its nature, and then design two data retrieval scheduling protocols: *Least-Switch* and *Best-First* to minimize the access latency and energy consumption of clients. We proof that *Least-Switch* guarantees minimum number of switchings during the data retrieval process. The performance of two protocols are evaluated by simulation. Simulation results show the advantages of two protocols: *Least-Switch* is more energy effective while *Best-First* reduces response time significantly when the number of antennae in the mobile devices are limited. Our future work includes developing more advanced data retrieving scheduling scheme and provide more theoretical analysis on PADRS-MIMO.

References

1. Ardizzoni, E., Bertossi, A.A., Ramaprasad, S., Rizzi, R., Shashanka, M.V.S.: Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel. IEEE Transactions on Computers 54(5), 558–572 (2005)
2. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999)
3. Gao, X., Shi, Y., Zhong, J., Zhang, X., Wu, W.: SAMBox: A Smart Asynchronous Multi-Channel Black BOX for B^+-Tree based Data Broadcast System under Wireless Communication Environment. Distributed & Parallel Databases (2009) (in review)
4. Jung, S., Lee, B., Pramanik, S.: A Tree-Structured Index Allocation Method with Replication over Multiple Broadcast Channels in Wireless Environment. IEEE Transaction on Knowledge and Data Engineering 17(3), 311–325 (2005)
5. Hurson, A.R., Muñoz-Avila, A.M., Orchowski, N., Shirazi, B., Jiao, Y.: Power-Aware Data Retrieval Protocols for Indexed Broadcast Parallel Channels. Pervasive and Mobile Computing 2(1), 85–107 (2006)
6. Imielinski, T., Viswanathan, S., Badrinath, B.: Data on Air: Organization and Access. IEEE Transactions on Knowledge and Data Engineering 9, 353–372 (1996)
7. Prabhakara, K., Hua, K.A., Oh, J.: Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment. In: ICDE 2000, pp. 167–176 (2000)
8. Saxena, N., Pinottti, M.C.: On-line Balanced k-Channel Data Allocation with Hybrid Schedule per Channel. In: MDM 2005, New York, NY, USA, pp. 239–246 (2005)

9. Sun, B., Hurson, A.R., Hannan, J.: Energy-Efficient Scheduling Algorithms of Object Retrieval on Indexed Parallel Broadcast Channels. In: ICPP 2004, Montreal, Quebec, Canada, pp. 440–447 (2004)
10. Usman, M., Abd-Alhameed, R.A., Excell, P.S.: Design Considerations of MIMO Antennae for Mobile Phones. PIERS online 4(1), 121–125 (2008)
11. Waluyo, A.B., Srinivasan, B., Taniar, D.: Global Indexing Scheme for Location-Dependent Queries in Multi Channels Mobile Broadcast Environment. In: AINA 2005, pp. 1011–1016 (2005)
12. Wang, S., Chen, H.-L.: Tmbt: An Efficient Index Allocation Method for Multi-Channel Data Broadcast. In: AINAW 2007, vol. 2, pp. 236–242 (2007)
13. Xu, J., Lee, W.-C., Tang, X., Gao, Q., Li, S.: An Error-Resilient and Tunable Distributed Indexing Scheme for Wireless Data Broadcast. IEEE Transactions on Knowledge and Data Engineering 18(3), 392–404 (2006)
14. Yao, Y., Tang, X., Lim, E.-P., Sun, A.: An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast. IEEE Transactions on Knowledge and Data Engineering 18(8), 1111–1124 (2006)
15. Yee, W.G., Navathe, S.B., Omiecinski, E., Jermaine, C.: Efficient Data Allocation over Multiple Channels at Broadcast Servers. IEEE Transactions on Computers 51(10), 1231–1236 (2002)
16. Yee, W.G., Navathe, S.B.: Efficient data access to multi-channel broadcast programs. In: CIKM 2003, New York, NY, USA, pp. 153–160 (2003)
17. Zheng, B., Lee, W.-C., Liu, P., Lee, D.L., Ding, X.: Tuning On-Air Signatures for Balancing Performance and Confidentiality. IEEE Transactions on Knowledge and Data Engineering 21(12), 1783–1797 (2009)