Computing and Informatics, Vol. 22, 2003, 473–510

CONTINUOUS ACCESS OF BROADCAST DATA USING ARTIFICIAL POINTERS IN WIRELESS MOBILE COMPUTING

Prem Chandra Saxena

School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi – 110067, India
e-mail: `prem_saxena@hotmail.com`

Inder Jeet Arora

Reader, Deptt. of Statistics
PGDAV College
University of Delhi
New Delhi – 110065, India
e-mail: `inder-arora@lycos.com`

Communicated by Jiří Šafařík

Abstract. In a ubiquitous information environment, massive number of users carrying their portable computers can retrieve information anywhere and anytime using wireless mobile computing technologies. Wireless data broadcasting as a way of disseminating information to the large number of clients, has an inherent advantage by providing all types of users global access to information. An adaptive access method, which tolerates the access failure, has been proposed in an error-prone mobile environment. However the influence of version bits to deal with the updates of the broadcast data has not been exploited for the broadcast with modified but the same size and structure update. The basic idea is to distinguish the type of update that does not influence the change in the size and structure of the broadcast has been introduced. To deal with the types of updates, we classified the users in mobile computing environment into the users in system and the new users. In the proposed continuous algorithms, the user in systems record the previous result and use it efficiently to access the desired records with less number of probes in the broadcast, which is updated by a stream of same size and structure bits. In the performance analysis, the experimental results show that the proposed modified progression method has the best performance, as it requires the minimum cost to access the broadcast data.

Keywords: Data broadcasting, energy conservation, fault tolerance, mobile computing, version number

1 INTRODUCTION

Mobile Computing and wireless networks are the potential technologies, which are making an environment conducive for ubiquitous computing. In this environment, mobile users equipped with compact battery powered palmtops or laptops need to access the large volume of data stored in fixed network through Mobile Support Station [17, 18] by the wireless communication. Mobile Support Station is augmented with a wireless interface to communicate with the mobile units, which are located in a geographical area within the reach of radio coverage. This area is known as a cell. In this environment, user no longer requires maintaining a fixed and known position in the network and enables unobstructed mobility from one cell to another cell while accessing data. A cell could be a wireless local area network, which operates within the area of a building or a small complex. In the general architecture [21] that supports mobile wireless computing, fixed hosts communicate over the fixed network, while the mobile units communicate with other hosts (whether mobile or fixed) via a wireless channel. During this, a new aspect arises when the mobile user has to move during the execution of a transaction and continue its execution in a new cell. This *mobility* is one of the characteristics of mobile computing. There are a number of other special characteristic features in mobile computing environment that make the system unique in the sense of application development. These characteristics are as follows:

Communications Asymmetry: In a wireless communication environment, the bandwidth from server to clients on the downstream channel is significantly higher than the bandwidth available in the reverse direction on the upstream channel. Asymmetry can also arise due to service load asymmetry in which few servers support much larger client population. The third way that asymmetry arises is from the significant difference in the volume of data that is transmitted in each direction.

Frequent Disconnections: Users disconnect from the network by switching off their units as a power saving measure. They switch their terminals on and off regularly as they have the ability of selective tuning. Wireless connection also suffers with frequent interruptions either due to the exhaustion of battery power or to external electromagnetic disturbances such as lightning, spikes of short duration (this kind of noise is noncontinuous consisting of irregular impulses).

Limitation of Resources: The portable units have limited computing power. Compared to fixed wired network, the transmission bandwidth in wireless mobile communications is relatively small. The units have limited battery power and small storage capacity (compressed data use less memory). However, compression of data cannot be encouraged in mobile computing environment, as the battery power consumption is twice as expensive to decompress than to compress. Limitation also exists due to small screen size of the Personal Digital Assistants.

Each of these features has its own impact and poses new challenges in the efficient management of data in a system. These physical restrictions call for power and bandwidth efficient solutions both at hardware and software levels [8]. An inherent asymmetry in the communications (where downstream bandwidth is much greater than the upstream one) and the restriction in power that the mobile units have, make the model of broadcasting data to the larger set of clients a remarkable proposition. This is known as *data dissemination* [12, 15, 30]. In each wireless cell, there are two basic forms of data dissemination: *Broadcasting mode* and *On-demand/Interactive mode*. In broadcasting mode data are periodically broadcast on the downlink channel[1] and the users retrieve their data of interest by just listening to a certain channel. Broadcasting over a wireless medium is also a power saving technique from the end-user's point of view, since they do not have to resort to the power consuming uplink transmission but only listen to the data broadcast over the downlink channel. In an on-demand mode (a traditional client-server approach) a client requests for the data on the uplink channel[2] and the server responds by sending the data of interest on the downlink channel. In this paper, we consider the wireless data broadcasting as a way of disseminating the information to numerous users. Other recent application of data dissemination include Advanced Traveler Information Systems [28], dissemination using satellite networks [9, 16] including two stage data delivery [11], PointCast's webcasting [26], Marimba's Castanet [25] and Teleglobe and AT & T Systems' Internet Delivery Service [29].

In a mobile environment, broadcasting is a powerful medium where information is broadcast to a potentially large set of users. Moreover, the cost of broadcasting over the wireless does not depend upon the number of users who are listening. Therefore this wireless medium is almost an ideal choice for any public information service. Broadcasting also cuts down the number of separate but identical responses to requests, as there are several examples of queries, which are asked frequently and repetitively by a large number of users. Stock market data, local traffic information, weather information, events and news are all examples of information that would rather be broadcasted than to be provided on-demand basis. The periodicity file is very large and if the data record of interest constitutes only a very small portion

[1] Downlink channel refers to the bandwidth reserved for the information flow from server to the clients.

[2] Uplink channel refers to the bandwidth reserved for the information flow from clients to the server.

of it, the user will have to wait (in the worst case) for a period of time equal to the broadcasting time of the entire file. Therefore the index information should be provided along with the data such that the timing of the relevant data record being made available can be predicted. Hashing can also be used to predict the availability of the relevant data record. To conserve the battery power of a mobile unit so that it can run for a long time, a concept of ability of *selective tuning* was introduced [20]. This means the mobile unit comes into *active mode*[3] or tunes into a channel only when the data of relevance is expected to arrive and slips into doze mode for most of the time.

1.1 Related Work

A repetitive broadcast medium for database storage and query processing was investigated in a Datacycle project [15] at Bellcore. Datacycle was intended to exploit high bandwidth, optical communication technology, in which the user queries the data by filtering relevant information using a parallel transceiver capable of filtering data. A system called Boston Community Information System (BCIS) is described in [14] where BCIS broadcasts newspapers and information over an FM channel to the users with personal computers (connected to continuous power supply) equipped with the radio receivers. Both Datacycle and BCIS systems are based upon flat disk approach in which the expected waiting time for an item on the broadcast is the same for all items (half of the broadcast period).

The Broadcast Disk structure [1, 2, 3, 4, 5, 6, 13, 31] incorporates two components: a) a multi-disk structuring mechanism that allows non-uniform broadcast where bandwidth can be allocated to data items, and b) several data caching and prefetching policies, and update mechanism which supports the multi-disk broadcast. In [19] non-uniform broadcast strategies for data items with different access frequencies were provided. To schedule the content of the broadcast data, a demand based random function was used for the various records.

Mobile Computing Group at Rutgers [18, 21] focus on the data broadcasting in mobile environment in which they discussed the different index allocation techniques in order to reduce the battery power consumption at mobile clients. In this a broadcast is constituted when the version of the file containing the data records is interleaved with the index information. In a network transmission, the basic unit of message transfer is a packet whereas fixed number of packets constitutes the smallest logical unit of a broadcast. The smallest logical unit of a broadcast is known as a *bucket* and each bucket is a unit of information, which is sent on the broadcast channel. There are two types of buckets in a broadcast, *index buckets* and *data buckets*. Index information is contained in index buckets whereas data should be

[3] The time taken by CPU for shifting from doze mode to the active mode or vice-versa is known as set up time. It also refers to the time needed to switch the receiver on. Generally the set up time is negligible compared to the time it takes to broadcast a single bucket.

stored in data buckets. Thus, each broadcast consists of number of index and data buckets, and *broadcast cycle* is constituted with each version of data file interleaved with the index information. In the broadcast considered by the Computing Group at Rutgers, every data record appears only once. This broadcast is known as uniform broadcast. Power consumption to retrieve the required data is the estimated value of *tuning time* (the amount of time the user has to spend listening to the channel) and the response time is measured by another parameter, *access time* (the amount of time elapsed from the time when the user makes a request and the time when the required record is downloaded in a mobile unit). These parameters (tuning time and access time) are addressed in [20, 22]. In [20], an index allocation method called *distributed indexing* was exploited by efficient multiplexing of a data file with its clustering index. Distributed indexing scheme achieves almost the optimum (minimum) access time. To minimize the battery power, non-clustered indexing scheme for allocating static files and multiple indexes were proposed in [21]. In [23], signature technique to filter information of interest was proposed. Chen et al. in [7], and Saxena and Arora in [27] exploited the use of imbalanced tree based on variant index fanout for the skewed data to reduce the average cost of index probes. In their approach, where technique of Huffman tree was applied, most frequently accessed (hot) data require fewer probes. Mobile environment suffers from frequent occurrence of access failures due to communication noises, disconnections, etc. To address this problem, recently Lo in [24] proposed an adaptive access method, which tolerates the access failures.

The main contributions of this paper are a) the identification of the types of updates in a broadcast data and b) based upon the type of updates various continuous processes to deal with the updates for a set of one group of users are proposed. This can be achieved by caching at the client in the form of an artificial pointer. These algorithms are aimed at reduction of the tuning time. Our approach also works well in an error-prone mobile environment as it has a characteristic to tolerate the access failures. The organization of our work in this paper is as follows: in Section 2, we discuss the basics of distributed indexing scheme and an overview of an adaptive access method. In Section 3, the problems with the data file version number which depends upon two types of updates is identified, and on the basis of types of update of the broadcast data, users can be classified into two groups. Various algorithms for continuous processes for one set of users are discussed in Section 4. Section 5 presents the access methods for handling the updates of broadcast data. In Section 6, performance comparison of the access methods is examined by performing a series of evaluations.

2 ADAPTIVE ACCESS METHOD: AN OVERVIEW

In this section, we first briefly discuss distributed indexing scheme used by Lo and Chen [24] in a method known as adaptive access method. An overview of the progression method is also given in this section. In this method distinct buckets are used and identified by their addresses using the sequence numbers. Therefore, the significance of sequence numbers is also discussed here.

2.1 Sequence Numbers

Each bucket of the current broadcast has an identification number called the address of the bucket – the sequence number of this bucket within the current bucket. These sequence numbers are unsigned integer numbers from 0 (first bucket of the broadcast) to *max_seq_no*, i.e., the last bucket in the current broadcast. The sequence number of a bucket can be interpreted as a relative distance in buckets of this current bucket from the first bucket of the broadcast. This header information is contained in each bucket. The offset of a bucket C from the current bucket is computed as the difference between the sequence number of C and the sequence number of the current bucket. We can also confirm whether the current access in an access sequence is out of sequence by recording the sequence number of the next bucket to be retrieved. Hence it can be used in detecting an interrupt affected by various noises.

Let us now present an overview of the distributed indexing an EPR scheme underlying the importance of sequence number while identifying the buckets.

2.2 An Introduction of Distributed Indexing and EPR Scheme

The distributed indexing [20] and [21] is a novel way of organizing a file and its index on a broadcast channel. In this algorithm, an index tree with M as its arity (Figure 1) is multiplexed with the data by subdividing it into two parts – a) the replicated index and b) the non-replicated index. The replicated index constitutes the top r levels of the index tree, while the non-replicated index contains the rest of the $(L-r)$ levels. The index buckets at the $(r+1)^{\text{th}}$ level are called non-replicated roots and are collectively denoted by NRR. In each broadcast every index subtree rooted in NRR appears once in the whole broadcast. In an Entire Path Replication (EPR) scheme let us define

Rep(a_{ri}): replicated path from root of the index tree to the non-replicated bucket a_{ri} (excluding a bucket a_{ri}), i.e.,

$$Rep(a_{ri}) = (R, a_{1,l^{(1)}}, a_{2,l^{(2)}}, \dots, a_{r-1,l^{(r-1)}}) \text{ where } (j-1) < i \le jM; \quad j = 1, 2, \dots, M^{r-1}$$

and

$$\ell^{(s)} = \left\lceil \frac{i}{M^{r-s}} \right\rceil, s = 1, 2, \dots, r-1.$$

Ind(a_{ri}): buckets of the subtree rooted at a_{ri} by the level order traversal (including a_{ri}), i.e., for $i = 1, 2, \dots, M^r$,

$$Ind(a_{ri}) = (a_{ri}, (a_{r+1,M\cdot(i-1)+1}, \dots, a_{r+1,i\cdot M}), (a_{r+2,M^2\cdot(i-1)+1}, \dots,$$

$$a_{r+2,i\cdot M^2}), \ldots, (a_{L-1,M^{L-r}\cdot(i-1)+1}, \ldots, a_{L-1,i\cdot M^{L-r}}))$$

$Data(a_{ri})$: Data buckets indexed by a_{ri} (from left to right), and
$Segment(i)$: i^{th} broadcast segment that is constituted for iteration of a loop.

Broadly, the following algorithm gives the organization of the data file and index information into a broadcast format under the distributed indexing scheme.

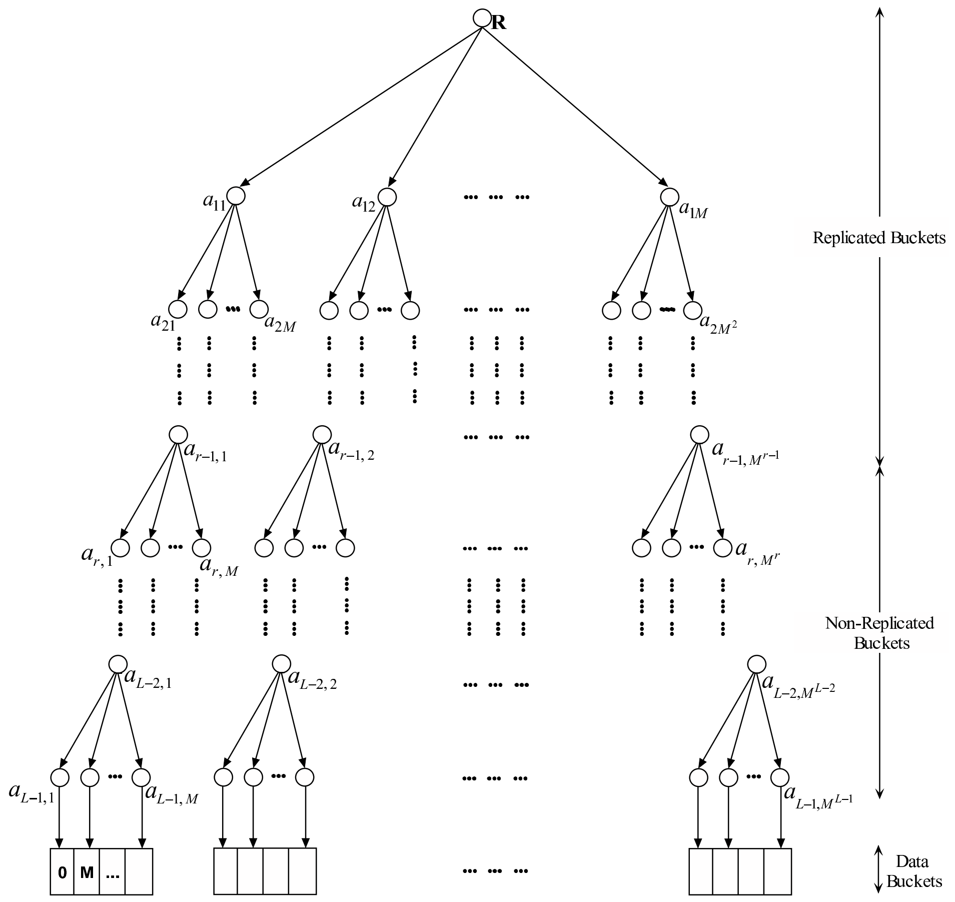


Fig. 1. An index tree

Procedure: Initialization of the broadcast under EPR Scheme.
Begin

```
 1. Initialize $r$.            /* level $r$ is for for the separation of index tree
                               into replicated index and non-replicated index */
```

```
2. Collect Non-replicated Roots $NRR = \{a_{r_1}, a_{r_2}, \ldots, a_{r_n}\}$.
                                        /* where $n = M^r$ */
3. Broadcast = ‘Null’.
4. For $i$=1 to $n$
   begin
      Segment($i$) = Rep($a_{ri}$) || Ind($a_{ri}$) || Data($a_{ri}$).
                                        /* || denotes concatenation */
      Broadcast = Broadcast || Segment($i$).
   end.
End.
```

In the above procedure, each broadcast segment is constructed for iteration of a loop. Every bucket in the broadcast is self-identifying and contains the following information:

- *Bucket_id* (sequence number): The offset of the bucket from the beginning of the broadcast, i.e. the sequence number of the bucket.
- *Broadcast_index*: The offset to the beginning of the next broadcast cycle.
- *Segment_pointer*: The offset to the beginning of the next broadcast segment.
- *Bucket_type*: Type of the bucket (replicated index bucket, non-replicated index bucket, data bucket).
- *version number*: Version bits to deal with updated information of the broadcast cycle.

The index bucket is arranged as a sequence of (*pointer*, *attribute_value*), i.e., (P_i, k_i), where P_i be a pointer to the bucket that contains the record identified by *attribute_value* k_i. Therefore the offset value of a bucket T from a current bucket B is computed as a difference between the *bucket_id* of T and the *bucket_id* of B, i.e.,

$$OFFSET(\text{pointer to } T) = S(T) - S(B).$$

Also, the first bucket in each broadcast segment has an additional tuple. This tuple includes the largest attribute value of the previous segment, which is used to find a data miss (the desired record has already passed and may be available in the next broadcast) and instruct the user to wait for the next broadcast cycle.

In the rest of this paper, for simplification, we consider an index tree (Figure 2a) with top two levels of replicated index buckets in an index tree of four levels, and a broadcast as given in Figure 2b under entire path replication scheme.

Now we present the general access protocol for a record with attribute value k under an entire path replication scheme as follows:

1. Tune to the current bucket of the broadcast. Get the pointer to the next broadcast segment.

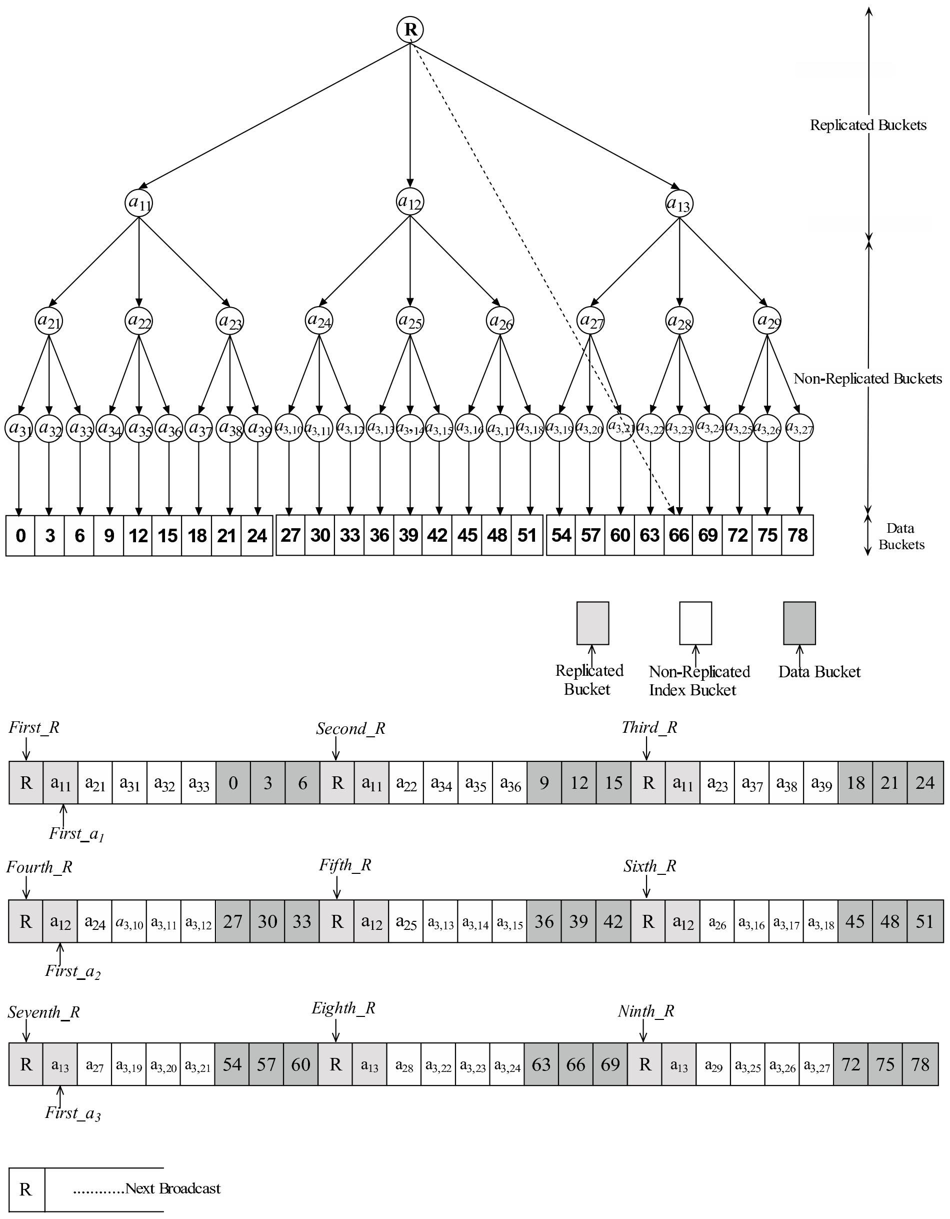


Fig. 2. a) Two replicated level index tree; b) A broadcast cycle under an EPR scheme

2. Tune in again to the beginning of the next broadcast segment either of the same broadcast or of the next broadcast. Determine whether a data miss occurs, on the basis of attribute value k and the largest attribute value of the previous broadcast segment.
   (a) When a data miss occurs, wait until the beginning of the next broadcast. Tune in to the beginning of the next broadcast and proceed as in step 3.
   (b) Tune in again for the appropriate higher-level index broadcast and proceed as in step 3.
3. Probe the designed index bucket and follow a sequence of pointers to find when the data bucket containing the first record with k as the *attribute_value* is to be broacast.
4. Tune in again when the bucket containing the first record with k as the value of the attribute is broadcast and download all the records with k as the value of the attribute.

2.2.1 Example of the Access Sequence

To show how this scheme works, we consider the following two cases:

1. If the user wishes to access the desired record from the data bucket 48 and makes the first probe at data bucket 32, then the access sequence by referencing Figure 2a and Figure 2b and using the access protocol is: 32, *fifth_R*, *second_*a_{12}, a_{26}, $a_{3,17}$, and 48. In this case no data miss happens.
2. Similarly the access sequence, when the user makes first probe at 64 and his desired record is in data bucket 22, then the access sequence is given by: 64, *ninth_R*, *first_R*, *first_*a_{11}, a_{23}, a_{38}, and 22. Here in this case data miss occurs and the next broadcast was required.

2.3 Adaptive Access Method

An adaptive access method based on distributed indexing scheme was proposed by Lo and Chen in [24] that has a feature of tolerance of the access failure. This follows two principles: a) it keeps the previous search result before the occurrence of a failure, b) continue an unfinished search as soon as possible. Under an EPR scheme, it has already been observed that the position (i.e., level number in index tree) of the replicated index bucket in all the broadcast segments remains the same as given in Figure 2b. This feature enables us to continue the search by using these replicates instead of waiting for the next broadcast cycle when a replicated bucket fails to be retrieved. In this method, the position of the segment containing the previous replicate in a broadcast cycle is recorded, which can be utilized to determine whether there is a replicate still available after a certain bucket in the current broadcast cycle. A mechanism called *search range* (an area which contains the data buckets where the desired record may exist) was given by Lo and Chen [24] to achieve this goal. The search range may be denoted by $[L, U]$, where position of the next bucket to visit in the access sequence is denoted by sequence number L and the upper boundary of the segment containing the last replicate of a replicated index bucket is denoted by U. A suitable action whether to continue or to wait can be decided when the current downloaded bucket is at, or before/after the location of the expected bucket. A sequence number U is used to verify the availability of replicated index bucket. For the initial probe, the search range is taken as an entire broadcast and is updated by each inspection of index bucket during the search. Assume V denotes the sequence number of the next bucket to visit induced by the current bucket. Then

$L \longrightarrow^{\text{is changed to the sequence number}} V$, and similarly

$U \longrightarrow$ the sequence number of the last bucket in the segment containing the last replicate of V^{th} bucket, the V^{th} bucket is a replicated index bucket.

Otherwise U requires no alteration in its sequence number.

2.4 Artificial Offset Value

In this section we modify the search range mechanism [24] to find an offset value and a formal procedure for obtaining artificial offset value is being given. The offset of the desired data bucket from the first bucket can be computed using the sequence number. This artificial offset value temporarily stored in the user's portable computer can be used to retrieve data buckets. Assume that one index bucket B is downloaded and the index information of the index bucket are represented by a sequence of (P_i, k_i). If $\mathit{OFFSET}(P_i)$ denotes the offset value for each P_i and if a search key is guided by the index pointer P_i of the index bucket B, then the new search range $[L, U]$ can be obtained by rule 1, where N is the position of index bucket B within a broadcast segment, i.e., the level number of the index bucket B in the index tree.

Rule 1.

- $L = S(B) + OFFSET(P_i)$
- **If** ((B is a replicated index bucket)
  and (P_i is not the last index pointer)) **then**
  $U = S(B) + \mathit{OFFSET}(P_{i+1}) - (N + 1)$.

Pointer to a specific/target bucket T from the first bucket within a broadcast can therefore easily be computed. This offset value provided by specifying the offset of a bucket T is known as Temporary (Artificial) Offset Value (TOV), which can be used to retrieve the some information again from any bucket if accessed within the same time limit and provided the size and structure of the broadcast is not altered due to any data modification. This temporary offset value P is stored temporarily in the memory of the portable computer; therefore it should be applied carefully to find the relative distance from the first bucket. The TOV changes its value when any search takes place in an index tree to find a bucket. According to Figure 2a, an artificial pointer is used to locate the same bucket; this temporary path in the index tree drawn in dotted lines is mapped to the access sequence in the broadcast cycle, i.e., pointer to a specific bucket T (target bucket) from the current bucket in a broadcast can be computed by Rule 2.

Rule 2.

- $Q = P - S(B)$; where P is a TOV.

The search range $[L, U]$ is modified using Rule 3, when probing an index bucket B of the broadcast discovers a data miss. This means the desired record do as not fall after an index bucket B.

Rule 3.

- $U = S(B) - 1$.

2.4.1 Example of Search Range

Before we proceed further, let us now demonstrate how to update the search range during the data search. Consider an access sequence as given in example 2.2.1 (data miss case) and the corresponding changes of search range as shown in Figure 3.

$$64 \rightarrow \text{Ninth\_R} \rightarrow \text{First\_R} \rightarrow \text{First\_}\,a_{11} \rightarrow a_{23} \rightarrow a_{38} \rightarrow \text{match } 22$$

(*a*)

$$[0, \text{max\_seq\_no}] \rightarrow [0, \text{S}(71)] \rightarrow [\text{S}(\text{First\_}\,a_{11}), \text{S}(26)] \rightarrow [\text{S}(a_{23}), \text{S}(26)] \rightarrow [\text{S}(a_{38}), \text{S}(26)] \rightarrow [\text{S}(22), \text{S}(26)] \rightarrow \text{Match } 22$$

(b)

Fig. 3. Search range alterations: a) An access sequence; b) analogous search ranges

For the initial probe, the search range is set as [0, *Max_seq_no*], where *Max_seq_no* denotes that the upper boundary is unknown at present. A data miss is discovered at the bucket *ninth_R*, which means the desired record has already passed. Therefore, U is changed to $S(71)$ [the previous bucket of *ninth_R* using rule 3]. At the bucket *first_R*, the next bucket to visit is induced to be *first_a*$_{11}$, whereas the second pointer leads to *first_a*$_{12}$. Then using rule 1, $U = S(\textit{first_R}) + \textit{OFFSET}($the second pointer of $\textit{first\_R}) - (1+1) = S(26)$, where $N = 1$ is the level number of the index bucket R.

Alternatively, $U \rightarrow$ Sequence number of the last bucket in the segment containing the last replicate of a_{11}, i.e., $U \rightarrow S(26)$. At bucket *first_a*$_{11}$, the next bucket to visit is a_{23}. The pointer pointing to the bucket a_{23} is the last index pointer of *first_a*$_{11}$. Therefore, U requires no alternation. The search range becomes $[S(a_{23}), S(26)]$. Now for the subsequent non-replicated index, only L requires to be changed.

2.5 Progression Method

We now present a progression method which is based on the search range mechanism by storing the offset of the bucket from the beginning of the broadcast, i.e., *bucket_id* of the target bucket T. This provides a temporary offset value (TOV) of the target bucket T, provided it is used for the data modified with SSS updates only. The TOV can also be used to compute the offset value of the target bucket T when the current bucket is not the first bucket. This detailed procedure (whose access flow is incorporated in a flowchart of continuous process) is based upon the mechanism of search range, and is presented in the following.

The search range $[L, U]$, and N are initialized as $[0, \mathit{Max\_seq\_no.}]$, and 1, respectively. As we stated earlier, the packet errors in a wireless communication system occur more frequently, it becomes necessary to keep away from a situation where access procedure may enter into a long loop due to high frequency of packet errors. Therefore, in addition, a threshold[4] (*Max_Interrupts*) can be defined and incorporated into an access flow by limiting the maximum number of interrupts to be met in the access.

1. Assume that a bucket B is downloaded after tuning to a broadcast channel. Based upon the comparison result of the sequence number of bucket B and $[L, U]$, one of the following actions takes place.

```
(a) If ($S(B) \leq L$) then /* next bucket is just downloaded or yet to arrive */
     wait for the bucket with the sequence number in the current broadcast cycle
    under the less than condition, i.e., To_$L$.
(b) If ($S(B) \geq U$) then /* next bucket to visit has passed in the current broad-
    cast cycle */
     wait for the bucket with the sequence number $L$ in the next broadcast.
(c) If ($L < S(B) < U$) then /* next bucket to visit has passed but an index
    replicate may be available */
    begin
     If (replicate available within search range $[L, U]$) then
     wait for available replicate from next broadcast segment and download it
      Shift $N$. /* to the level number of downloaded replicated index bucket */
     Set Flag to check data miss, if any.
```

[4] Another threshold can also be defined by limiting the maximum duration for getting the desired data record.

```
    else
        wait for the bucket with the sequence number $L$ in the next broadcast
        cycle(Next_$L$).
    end
```

When the *check_flag* is on, wait for the first bucket of the next broadcast, and decrease the value of N to the level number after downloading the replicated bucket and *Decrease_U*.

2. Clients may go into doze mode and tune in at the broadcast to retrieve the expected bucket containing the data record. Sequence of index pointers P_i is to be followed to retrieve the data buckets containing the data record.

```
If (Next bucket to visit is induced to be a replicated index bucket) then
begin
    Update_$LU$.
    /* $L$ is changed to the sequence number of the next bucket and $U$ is changed to
    the sequence number of the last data bucket in the segment containing the last
    replicate of the next bucket */
    Increase $N$ by 1.
else
    Increase_$L$.
    /* $L$ is changed to the sequence number of the next bucket & $U$ need not be
    changed */
end.
```

3. Compute the temporary offset value P of the target bucket T.
4. Always modify the values of $[L, U]$, N, Max_seq_no, and P according to an action as given above. Any modification, which is done partially, leads to a false search of the desired record and wastage of battery power.

3 UPDATES OF BROADCAST DATA

In this section, we present the two types of updates. These updates can be applied to infer whether the previous loaction (after retaining the previous search result) results in substantial saving of tuning time. These updates work even when the access sequence is encountered with the access failures.

In a search range mechanism [24], search range dynamically records the range of sequence number of buckets where the desired records of data items may exist in the broadcast. In their approach, the previous search range, which provides the location information of records, can be applied only to the same version of the broadcast data. In this case where the replacement of data due to insertion, deletion or modification that changes the size or structure of the broadcast is given the new version number, the search range then becomes invalid for the new version. Therefore, in their approach, an update that does not constitute any change in size and structure of data in a broadcast is assigned the same version number. This situation is worse as the mechanism provided failed to detect an update of data. Consider, for example, a broadcast, which includes the buckets with the information of the stock prices during the business hours of the stock market. Frequent transactions of scripts and varying prices during the business hours require equally frequent updating of broadcast data. For a script of a particular company, four variables[5] considered are the Company name, Previous closing price, Today's opening price, and Current price. Table 1a) shows the stock price of a company X retrieved at 12.45 P.M. from a broadcast is associated with the version No. 000001.

Considering the prices as given in Table 1b) of company X, an investor (user) places an instruction for the sale of 500 shares immediately after verifying the same version number 000001 of the next broadcast cycle. In this deal, he intends to receive $ 750 less than the estimated value of $ 13375. An interesting aspect of this example is that the update of broadcast data as in Table 1b) could not be established by mere inspection of version number. Since the type of update (changes in the values only) by which the size and structure of broadcast data is not altered, the same version number is associated with this new broadcast cycle.

Company Name	X
Previous Closing Price	$ 25.75
Today's Opening Price	$ 26.75
Current Price	$ 26.75

Version number 000001

a)

Company Name	X
Previous Closing Price	$ 25.75
Today's Opening Price	$ 26.75
Current Price	$ 25.25

Version number 000001

b)

Company Name	X
Previous Closing Price	$ 25.75
Today's Opening Price	$ 26.75
Current Price	$ 25.75

Version number 000010

c)

Table 1. a) Variable X with initial version number, b) Update without change in version number, and c) Variable X in a broadcast with change in SSS bits of version number

[5] To shorten the data size, any description of variables except for company name can be discarded. The professional and stock dealers can interpret the values of the variables of different companies.

In the kind of updates that do not change the size and structure of the broadcast data, the decision criteria can be thought of as two ways: (1) The subsequent broadcast cycle of same size and structure is assigned the same version bits. As seen in the above example, this violates the correctness as the user fails to detect the updates of data and is left to deal only with the obsolete data that may result in wrong analysis, conclusions and decisions. (2) The next broadcast (the same size and structure), which is a consequence of the replacement of data, is assigned the new version number. In this situation any old search range is discarded and re-probe takes place. This seriously affects the efficient retrieval of data and may result in a substantial increase in the latency and the tuning time.

Hence neither the same version number nor the next version number to the broadcast when any replacement of data that does not constitute any change in its size and structure can be used to ascertain correctness about any update of data.

In our method, the two types of updates influencing the version change to the broadcast data, which occurs under the mobile environment are as follows

Invariant Updates: When there is no change in the size and structure of the broadcast data due to data deletion, insertion or modification, i.e., Same Size and Structure Update (SSS Update)

Variant Updates: When the size or structure of the broadcast data is changed, i.e., Distinct Size or Structure Update (DSS Update).

An approach to detect two types of updates (Invariant and Variant) in a broadcast is devised by splitting the version bits into invariant bits and variant bits as shown in Figure 4.

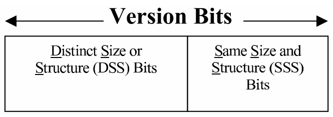

Version Bits
Distinct Size or Structure (DSS) Bits
Same Size and Structure (SSS) Bits

Fig. 4. Allocation of variant and invariant bits in a version number

In Table 1c) the update of broadcast is depicted by the change in only SSS bits (last two bits) of the version number. Now the simple formulae are derived to decide the size of invariant and variant bits that influence the version bits of the broadcast by giving the maximal disconnection time at which the version change will not be mistaken.

- T_c: Time for a broadcast cycle.
- T_u: Minimum time interval between two updates of broadcast data.
- T_{ss}: Maximum disconnection time without mistaking the invariant update.
- T_{ds}: Maximum disconnection time without mistaking the variant update.
- T_d: Maximum disconnection time without mistaking the version change.

The maximum number of invariant updates during the period T_{ss} is $\lceil T_{ss}/T_u \rceil$. Therefore for β and T_{ss} values the following condition holds true

$$\left\lceil \frac{T_{ss}}{T_u} \right\rceil \leq 2^{\beta} - 1 \Longrightarrow \beta \geq \log_2 \left\{ \left\lceil \frac{T_{ss}}{T_u} \right\rceil + 1 \right\},$$

similarly

$$\left\lceil \frac{T_{ds}}{T_u} \right\rceil \leq 2^{\alpha} - 1 \Longrightarrow \alpha \geq \log_2 \left\{ \left\lceil \frac{T_{ds}}{T_u} \right\rceil + 1 \right\}.$$

If $\beta = 3$ and $\alpha = 6$, then the maximum disconnection time without mistaking SSS update T_{ss} and DSS update T_{ds} is $7T_u$ and $63T_u$ respectively. Assuming T_u is one minute, then T_{ss} and T_{ds} are 7 and 63 minutes, respectively.

This approach is more advantageous as it detects both SSS and DSS updates which affect the version number of a broadcast. That is, the version number of the successive broadcasts depends upon both the types of updates such that decision can be arrived whether the old search range can be discarded and the re-probe is applied. The previous search range is used to record the related location information in the broadcast data, when the successive broadcast is not changed or encountered with the SSS update only. Hence simply by waiting for the arrival of the bucket we can retrieve the newly inserted records of the form of SSS update using the same sequence number in the broadcast cycles.

Thus the retrieval of a record without any re-probe leads to a substantial saving in the tuning time.

3.1 Users in System

A client who was in listening mode goes into doze mode after successfully accessing the data and tunes in again at the first replicated index bucket of the next broadcast cycle, or tunes in again into a broadcast within disconnection time. This disconnection time is a planned failure when mobile terminal is switched off as a power saving measure.

The whole idea behind a planned failure is to let user cache the previous search result and use it judiciously. For this, we define a term user *in-system.*

A client is said to be *in-system* if it has retrieved the desired record or probed a bucket in its access sequence, computed the temporary offset value in its previous or current search and kept this offset value to resume its unfinished search within some specified time say disconnection time such that the user can not be mistaken by the version changes due to the SSS bits.

Thus the maximum time for a client to be in system is $\min\{2^{\alpha} - 1, 2^{\beta} - 1\}$, where α and β are the DSS and SSS bits respectively. When a client is in system it has knowledge of the version number and temporary offset value P of the probed bucket or a target bucket T. This previous search result can be used to obtain the possible location of the required data in a broadcast channel and to continue an unfinished search. On the other side, if a user has switched off its terminal for

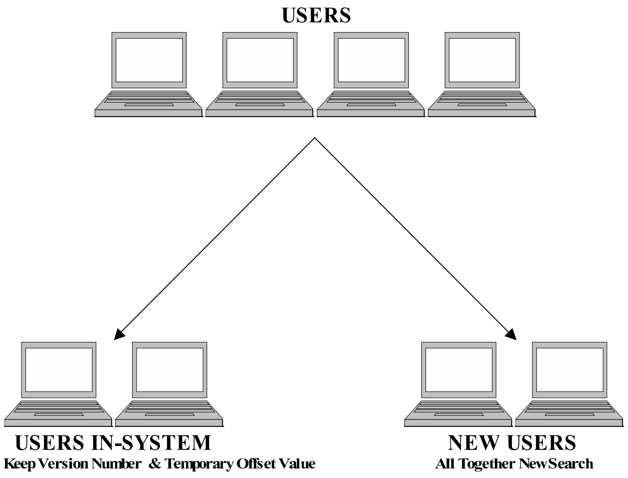


Fig. 5. Types of users in a broadcast

a time; larger than the disconnection time then it should be regarded as a new user. The distinction between user in system and new user as shown in Figure 5 is also due to the disconnection time, this disconnection time is the elective[6] nature of the client: any disconnection when the user is in system is a planned failure, which can be anticipated and prepared.

In the next section, various continuous processes are presented. These processes are based upon this artificial pointer when the broadcast is updated with the SSS bits and when the user is in-system.

4 CONTINUOUS PROCESSES

Our main objective of the continuous process is to provide/resumption of the target bucket from the following broadcasts using search range mechanism. This is true when the user has already successfully probed and accessed the desired record. This continuous process is particularly useful for the users who are in-system and wish to retrieve the updated information of the same variable. The temporary offset value P can be used for continuous search provided it is used within disconnection time and when the data modification is due to SSS updates only. This value of P vanishes/modifies, when any new search for the buckets takes place or the disconnection time exceeds its limit, i.e., where all the versions due to either DSS or SSS bits are exhausted.

[6] The term elective was coined and used by Daniel Duchamp, Columbia University. Tomasz Imielinski and B. R. Badrinath also used this term to distinguish planned failure and failure.

We now propose the following ways to continue the process of data records retrieval. If the user

1. re-acquires the same search result from the next broadcast using the previously stored range data (i.e., wishes to retain the updated information of the same variables). In this case the user goes into doze mode and tunes in again to the beginning of the next broadcast. We define this as a *continuous process without disconnection.*
2. disconnects for a certain time (elective disconnection) and tunes in again either anywhere (except at the beginning) of the next broadcast or at any point in any of the subsequent broadcasts. It is to retain the updated information of the same variables, provided the user remains in the system. This is referred to as *continuous process after disconnection.*
3. goes for the new search.

4.1 Continuous Process without Disconnection

In this case, after successfull retrieval of the desired record the user does not disconnect from the broadcast (but goes into doze mode) and is willing to re-acquire the same search result as shown in Figure 6. The detailed procedure is as follows:

Procedure: Continuous Access (Without Disconnection from Broadcast)

```
Begin
 1. Initialize $L = 0$.
 2. go into doze mode and wake up on arrival (Wait for the arrival) of the bucket
    with the sequence number $L$ in the next broadcast cycle.
 3. If ($Disconnection\ time < Limit$) then
 4.    If (change in version bits) then
 5.       If (change in $DSS$ bits) then
 6.          go to step 18.
 7.       Else
 8.          $L = S(B) + OFFSET(P - S(B))$.
 9.          go to step 20.
10.       End If.
11.    Else
12.       No change in data since the previous search.
13.       go to step 26.
14.    End If.
15. Else
16.    go to step 18.
17. End If.
18. Apply progression method for fresh search and obtain TOV '$P$'.
19. go to step 26.
```

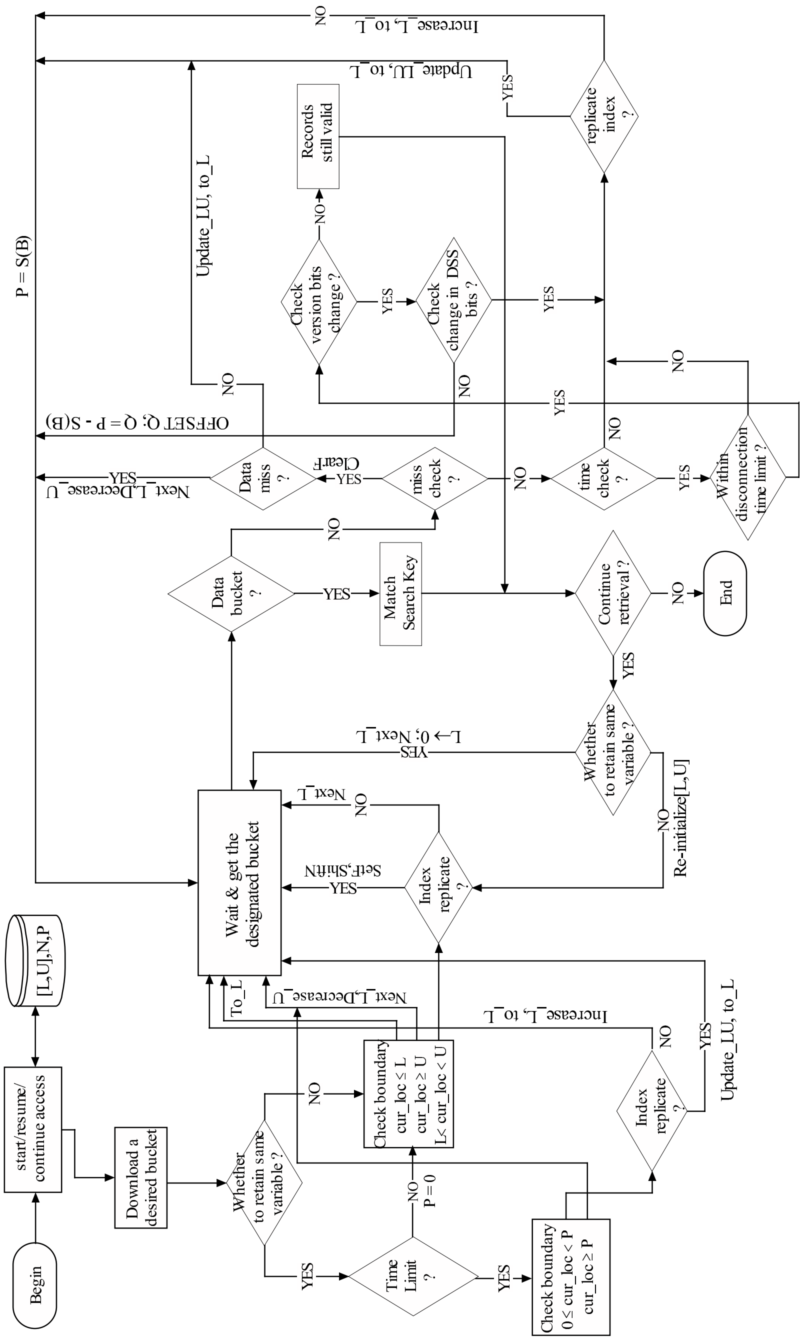


Fig. 6. Progression method and continuous access of data without disconnection in modified progression method

```
20. go into doze mode and wake up to retrieve the expected bucket
    by following the offset value.
21. **If** (New bucket downloaded is data bucket) **then**
22.     match the value of the attribute and download the desired record.
23. **Else**
24.     go to step 18.
25. **End If**.
26. go to step 1 when wish to continue the same track.
**End**.
```

4.2 Continuous Process after Disconnection

In the following, we describe the procedure to access the update of the data of the variable when a client enters into a broadcast within a disconnection time. Tune to a channel and record the sequence number of the downloaded bucket B. Check whether $S(B) \geq P$, if it is not, wait for the arrival of the either first bucket of the next broadcast segment (if any) or of the target bucket in the range $[S(B), P[$. Otherwise, wait for the arrival of the first bucket in the next broadcast cycle. In case the current downloaded bucket is replicated index bucket or non-replicated index bucket (except the last non-replicated index bucket in the given range), then the offset of the target bucket T from the current bucket is computed as the difference between the *bucket_id* of T and the *bucket_id* of current bucket. Buckets in each broadcast cycle are assigned unsigned integer numbers beginning from 0 and *bucket_id* of current bucket gives the offset of this bucket from the beginning of the broadcast. Therefore the offset of the target bucket T from the current bucket is $P - P^*$, where P^* is the sequence number of the current bucket, i.e., offset value from the beginning of the broadcast.

When the current downloaded bucket is the last non-replicated index bucket, or the data bucket then simply wait for the arrival of data bucket and match the search key to retrieve the data records.

Procedure: Continuous Access (When disconnection is within limit)

```
**Begin**
1. **If** ($S(B) \geq P$) **then**
2.     Wait for the arrival of the first bucket from the next broadcast cycle.
3.     Get pointer to the target bucket using artificial pointer 'TOV'.
4.     Download target bucket.
5. **Else If** (Replicated index bucket available in the range $[S(B), P[$) **then**
       /* $S(P) < P$ */
6.     tune to the first bucket of the next broadcast segment
        in the range $[S(B), P[$.
7.     get pointer to the target bucket using $Q = P - P^*$.
       /* rule 2, $Q = P - S(B)$ */
```

```
 8.     wake up to retrieve the required data from the target bucket.
 9.  Else
10.     wait for the arrival of target bucket and retrieve the required data.
11.  End If.
End.
```

The user who has already retrieved the desired record from the bucket with sequence number $S(B)$ in the just concluded search and who wishes to access a different data record without any disconnection from the channel can do so by a new search using the progression method.

Now we provide two representative examples (with and without data miss) showing the lower number of probes necessary to retrieve the required information in a continuous algorithm for the users who are in system. These algorithms are still applicable in case of the occurrence of frequent access failures due to various communication noises.

The examples shown in Table 2 reveal that the artificial offset value used to re-acquire the updated record from the same search range performs quite well in the continuous process. This artificial offset value works well with the new broadcast which undergoes the process of modification of data values without disturbing the data size and structure of the broadcast, i.e., the different version number due to change in the SSS bits for the new broadcast.

5 ACCESS METHODS FOR HANDLING THE BROADCAST DATA UPDATES

In this section, the divergent methods are presented to deal with the updates of broadcast data as well as the access failures. We make use of the access sequence 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, a_{38}, and 22 to elucidate the revival activities when the bucket a_{23} contains a packet error and is therefore abandoned.

Re-access: In the re-access method as given in [24], a user re-accesses the bucket affected by interrupts. Each affected bucket is re-accessed after verifying the version bits from the first replicated bucket of the next broadcast. If no change or change in SSS bits is found, then the affected bucket is re-accessed; otherwise the user is regarded as new user and accesses the broadcast data right from the beginning. In our example, since the bucket a_{23} is re-accessed after verifying the version number of the following broadcast cycle from the first replicated bucket *First_R*, the access sequence is changed to 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, a_{23}, a_{38}, and 22. Again, if the bucket *First_R* for verifying the version number is also affected by interrupt, then the access sequence is changed to 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, *First_*a_{11}, *First_R*, a_{23}, a_{38}, and 22. The defect of this method is that the waiting time for reaccessing of the affected bucket in the next broadcast cycle is very long and is equal to the time for the broadcast cycle.

	Type of User	An Access Sequence	The Corresponding Search Range
← No Data Miss →	Fresh Search for new user	32→ Fifth_R→ Second_a_{12}→ a_{26} → $a_{3,17}$→ 48	[0, max_seq_no] → [S(Second_a_{12}), S(53)] → [S(a_{26}), S(53)] → [S($a_{3,17}$), S(53)] → [S(48), S(53)] → Match 48
	When the User in system reacquire the same search result in the next broadcast without disconnection.	48 (downloaded bucket) → First_R → 48	[0, S(53)] → [S(48), S(53)] → Match 48
	When the user in system reacquire the same search result after disconnection and if cur_loc < P.	18→ Fourth_R→ 48	[0, max_seq_no] → [S(Fourth_R), S(53)] → [S(48), S(53)] → Match 48
	When the user in system reacquire the same search result after disconnection and if cur_loc ≥ P.	66→ First_R→ 48	[0, max_seq_no] → [0, S(71)] → [S(48), S(53)] → Match 48
← Data Miss →	Fresh Search for new user	64→ Ninth_R → First_R→ First_ a_{11} → a_{23}→ a_{38}→ match 22	[0, max_seq_no] → [0, S(71)] → [S(First_ a_{11}), S(26)] → [S(a_{23}), S(26)] → [S(a_{38}), S(26)] → [S(22), S(26)] → Match 22
	When the User in system reacquire the same search result in the next broadcast without disconnection	22 (downloaded bucket) →First_R→ 22	[0, S(26)] → [S(22), S(26)] → Match 22
	When the user in system reacquire the same search result after disconnection and if cur_loc < P.	6→ Second_R→ 22	[0, max_seq_no] → [S(Second_R), S(26)] → [S(22), S(26)] → Match 22
	When the user in system reacquire the same search result after disconnection and if cur_loc ≥ P.	64→ Ninth_R→ First_R→ 22	[0, max_seq_no] → [0, S(26)] → [S(22), S(26)] → Match 22

Table 2. Search range changes for new user and the user in system

Modified Re-access: In modified re-access method, affected bucket are re-accessed only after verifying the version bits from any bucket of the next broadcast. Similarly, in re-access of the bucket a_{23} after verifying the version number of the next broadcast in the second attempt (i.e., when *First_R* or any other bucket also has a packet error), the access sequence changed to 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, *First_*a_{11}, a_{23}, a_{38}, and 22. In this method, 3 broadcasts are used instead of 4 as seen in re-access method.

This method gives an improvised result in terms of number of broadcasts used, but suffers with the drawback that it also uses the next broadcast for version verification.

Progression Method: In this method, if the downloaded bucket affected by a packet error is a replicated index bucket, it can be recovered by retrieving the replicated bucket appears in the successive broadcast segment and at the same position in these segments from the same broadcast. Each of remaining affected buckets is re-accessed after confirmation of the version number of the next broadcast from the replicated index bucket. Consider a bucket a_{23} contains a packet error; the changed access sequence becomes 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, a_{23}, a_{38}, and 22. Again, if the bucket required for verification of the version number is also afflicted by the packet error, then it can be confirmed from the replicated bucket appearing in the next segment but at the same position, the access sequence is given by 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, *First_*a_{11}, *Second_R*, a_{23}, a_{38}, and 22. In this case bucket *First_*a_{11} is used to get the pointer to the next available replicate of *First_R*.

Modified Progression: In modified progression, the confirmation of version number takes place at any of the bucket of the next broadcast. The changed access sequence when downloaded bucket a_{23} contains a packet error and confirmation of version number is completed in the second attempt; then the access sequence becomes 64, *Ninth_R*, *First_R*, *First_*a_{11}, a_{23}, *First_R*, *First_*a_{11}, a_{23}, a_{38}, and 22. Since, in modified progression, there is no restriction on the type of bucket for confirmation of version bits of the next broadcast, the waiting time and the tuning time require to download a bucket for confirmation is not as long as in the progression method. This results in an improvised cost in terms of tuning time.

6 PERFORMANCE COMPARISONS

In this section, we address the important parameters, which affect the performance of the different access methods. Specifically, we investigate the performance of the access methods (re-access, modified re-access, progression, and modified progression) by evaluating the cost of accessing the broadcast data in the subsequent broadcasts with respect to the number of levels of replicated index buckets from the top of an index tree (r) and to the probability of the downloaded bucket containing an error (q). In our performance analysis updates are performed and the update (if any) from the following broadcasts is identified by the version bits of the broadcast data. Therefore a broad parameter 'number of broadcasts used' may measure and replace a metric 'access time'. In the following, we list parameters to be used in obtaining the analytical expressions for performance comparisons.

Parameters: Consider a broadcast under an EPR scheme using a fully balanced index tree where the smallest unit for measuring the broadcast data is bucket.

- L: The depth of the index tree excluding the set of data buckets.
- M: Outgoing pointers of each index (replicated/ non-replicated) bucket.
- r: The number of top levels of an index tree which constitute the replicated part.
- $INDEX$: Size of an index of a broadcast cycle.
- $DATA$: Size of the data in a broadcast.
- $SEGMENT$: Size of a broadcast segment.
- $BCAST$: The size of a broadcast cycle i.e., $BCAST = INDEX + DATA$.
- p: The probability of success i.e., the probability that a downloaded bucket does not contain an error.
- q: The probability that a downloaded bucket containing a packet error ($q = 1 - p$).

$SEGMENT$ is the sum of the number of buckets from the root of an index tree to the non-replicated index bucket and all buckets (including data buckets) of the subtree rooted at the non-replicated bucket. Therefore

$$SEGMENT = r + \frac{M^{L-r+1} - 1}{M - 1},$$

and $BCAST$ is the product of the number of non-replicated index buckets and the size of the broadcast segment

$$BCAST = rM^r + \frac{M^r(M^{L-r+1} - 1)}{M - 1}.$$

From the above, it is seen that the number of buckets of the subtree rooted at the non-replicated bucket and the size of the broadcast segment decreases as r grows, whereas the size of the broadcast cycle increases when r increases.

Assumptions:

- We consider an EPR scheme for a broadcast, where the data file is associated with the fully balanced index tree.
- Updates can be performed on the broadcast data; but, to simplify the analysis, the size of the broadcast cycle ($BCAST$) for each broadcast is the same, but the structure may be different (i.e., $BCAST_i = BCAST \; \forall i$).

- Data misses may happen in multiples in an access methods; this is ignored to simplify the procedure.
- Since the packet error has some inheritance properties: Occurrence of an error in each position has equal probability and independence of errors in different positions. Moreover, q is independent of the size of the $BCAST$; therefore we consider packet errors as interrupts in our analysis.

In the next subsection, we use an access graph to develop an analytical model, which helps in measuring the cost of accessing the broadcast data.

6.1 Analytical Expression for Dealing with Updates

An access graph in the form of a state transition diagram is used for illustrating the process of searching the broadcast data. It has two types of edges, solid lines (transition with correct access) and dotted lines (transition with false access) [24]. The labels on each edge indicate transition cost in buckets for evaluating the tuning/access time. In our performance analysis, the sensitivity of the broadcast data in successive cycles due to update of data results in the increase of the cost of accessing broadcast data in terms of access and tuning times. This is mainly due to the checking of version (SSS and DSS) bits of the successive broadcast cycles. These successive broadcasts are used when the downloaded bucket[7] contains packet error. Therefore the cost of accessing the broadcast data using the buckets from the successive broadcasts can be measured by a) the 'number of broadcasts used' instead of access time, and b) by the tuning time. This tuning time is proportional to the number of downloaded buckets.

Tuning Time = The number of visited states on the path from 0 to $L+1$.

The upper limit of the number of visited states on the path is the depth of the index and two additional buckets. The additional buckets are a) the first probe bucket, and b) the result data bucket (the third additional bucket is the first bucket of the broadcast data when a data miss happens). Similarly the buckets from the one broadcast (two broadcasts when data miss happens) are used for accessing the broadcast data. The boundary of both the tuning time and the number of broadcasts used (herein referred to as *Tuning* and *Broadcasts* respectively). These metrics, the tuning and the broadcasts for the normal access graph can be obtained by the relations

$$\text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 2,$$

and

$$\text{Broadcasts} = 1.$$

[7] This bucket may be a replicated bucket (when its replicate is not available in the current broadcast), non-replicated bucket or a data bucket.

Similarly, for continuous access of the bucket at the same location when broadcast is updated with the SSS bits and when the user is in system, the following relations can measure the cost of broadcast data for the Ideal access graph:

$$\text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 3,$$

and

$$\text{Broadcasts} = 2.$$

The other cost parameters *additional tuning*, *tuning ratio*, *additional broadcasts*, and *broadcasts ratio* are also defined to justify the importance of the continuous access of broadcast data in various methods; these are defined when the user is in system, and the next broadcast has the same version or is updated with SSS bits only. The additional tuning (broadcasts) is the cost required to re-access the same data bucket or the bucket at the same location from the next broadcast; the tuning (broadcasts) ratio is the ratio of additional tuning (broadcasts) to that of the tuning (broadcasts) for the successful data access from the beginning using the same access method.

In the following, we analyze the cost for the access methods in terms of tuning, broadcasts, tuning ratio, and broadcast ratio. If the average number of errors repeatedly occurring at the same bucket is denoted by *ERR*, then it can be obtained using a parameter q as follows:

$$x(\textit{number of errors}) : 1, 2, 3, \ldots$$

$$\textit{Probability } p(x) : q, q^2, q^3, \ldots$$

$$\textit{ERR} = E(X) = \sum_{x=1}^{\infty} x \cdot p(x) = \sum_{x=1}^{\infty} x \cdot q^x = \frac{q}{(1-q)^2}.$$

The tuning and broadcasts can be divided into two parts: a) the false part, and b) the correct part. In the process of finding the cost of broadcast data, the dotted lines above the solid lines give the broadcasts used (Figure 7).

6.1.1 Reaccess

In the re-access, there are two kinds of access graphs (Figure 7a). Type 1 indicates the first probe whereas Type 2 indicates the cases for replicated and non-replicated index. The recovering cost from all false accesses for both the types is as follows:

$$\text{Type 1: } \textit{ERR}$$

$$\text{Type 2: } \textit{ERR} \cdot (1 + n_c \cdot \textit{ERR} + 1),$$

where n_c is the cost of reaccessing the same bucket in the following broadcast cycle. The relation to compute the cost of false tuning part is as follows:

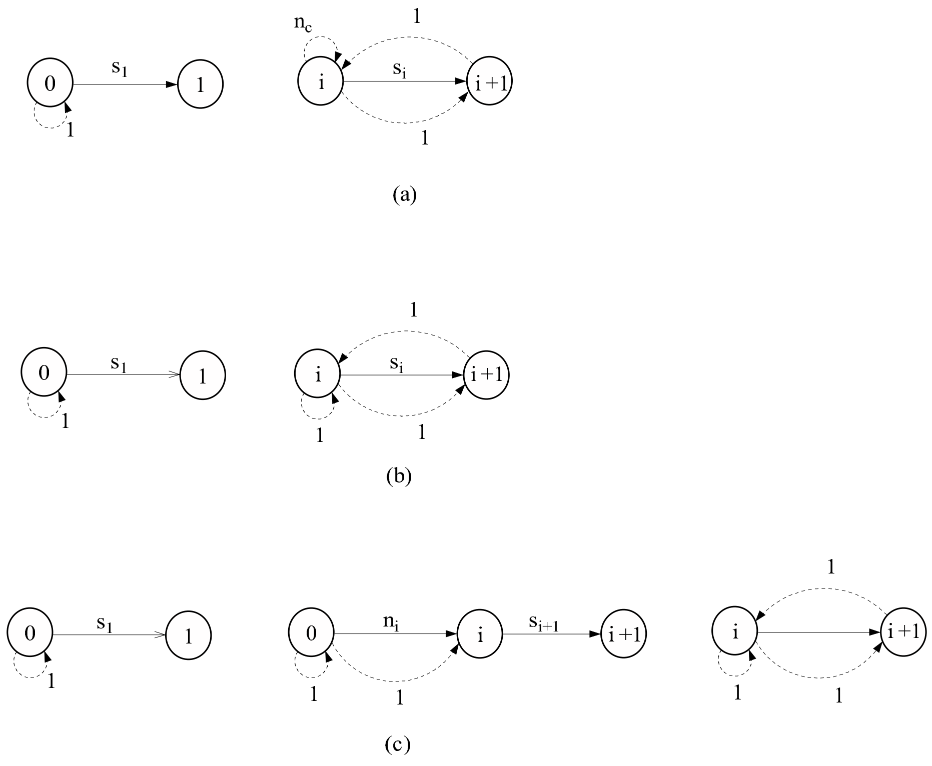


Fig. 7. a) Access graph for a reaccess method; b) Access graph for modified reaccess method; c) Access graph for modified progression

$$FalseTuning = ERR \cdot 1 + ERR \cdot (1 + n_c \cdot ERR + 1) + \ldots + ERR \cdot (1 + n_c \cdot ERR + 1)$$

$$= ERR[1 + (L+1) \cdot (n_c \cdot ERR + 2)].$$

The tuning and the broadcasts, when $n_c = 1$, are given by

$$\text{Tuning} = ERR[1 + (L+1) \cdot (ERR + 2)] + \sum_{i=1}^{L+1} s_i$$

$$= ERR[1 + (L+1) \cdot (ERR + 2)] + (L+2)$$

$$\text{Broadcasts} = 1 + ERR \cdot (ERR + 1) \cdot (L+1).$$

Similarly, for continuous access of the bucket at the same location when broadcast is updated with the SSS bits and when the user is in system, then

$$\text{Tuning} = ERR[1 + (L+2) \cdot (ERR + 2)] + (ERR + 1) + (L+3)$$

$$\text{Broadcasts} = 2 + ERR \cdot (ERR + 1) \cdot (L + 2) + ERR.$$

The other cost parameters (*tuning ratio*, *broadcasts ratio*) are given as follows:

$$\text{Tuning Ratio} = \frac{(ERR + 1)(ERR + 2)}{ERR \cdot [1 + (L + 1) \cdot (ERR + 2)] + (L + 2)}$$

$$\text{Broadcast Ratio} = \frac{(ERR + 1)^2}{1 + ERR \cdot (ERR + 1)(L + 1)}.$$

6.1.2 Modified Re-access

In this method, verification of version bits takes place at any of the buckets; accordingly, the modified access graphs are shown in Figure 7b).

The cost in this case is given by

$$\text{Tuning} = ERR \cdot 1 + ERR \cdot (ERR + 2) + \ldots + ERR \cdot (ERR + 2) + (L + 2)$$

$$= ERR \cdot [1 + (L + 1) \cdot (ERR + 2)] + (L + 2)$$

$$\text{Broadcasts} = 1 + (L + 1) \cdot ERR.$$

Similarly, for continuous access, the cost of accessing the broadcast is

$$\text{Tuning} = ERR \cdot [1 + (L + 2) \cdot (ERR + 2)] + (ERR + 1) + (L + 3)$$

$$\text{Broadcasts} = 2 + (L + 2) \cdot ERR$$

$$\text{Tuning Ratio} = \frac{(ERR + 1)(ERR + 2)}{ERR \cdot [1 + (L + 1) \cdot (ERR + 2)] + (L + 2)},$$

and

$$\text{Broadcast Ratio} = \frac{(ERR + 1)}{1 + (L + 1) \cdot ERR}.$$

6.1.3 Progression and Modified Progression

With the objective to minimize the cost of accessing broadcast data, modified progression method (as in the modified reaccess) does not put any condition on the type of buckets for verification of version bits. If n_i is the cost of finding the next index replicate at level i of the index tree, then for the progression method

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (1 + ERR + n_i) + n_c \cdot (L - r + 1) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot [ERR \cdot (1 + ERR + n_i) + 1] + (L + 2).$$

For $n_i = 1$ and $n_c = 1$,

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (ERR + 2) + (L - r + 1) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot [ERR \cdot (ERR + 2) + 1] + (L + 2)$$

$$\text{Broadcast} = 1 + (L - r + 1) \cdot ERR.$$

For continuous access,

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (ERR + 2) + (L - r + 2) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot [ERR \cdot (ERR + 2) + 1]$$

$$+ (ERR + 1) \cdot [ERR \cdot (ERR + 2) + 1] + (L + 3))$$

$$\text{Broadcast} = 2 + (L - r + 2) \cdot ERR$$

$$\text{Tuning Ratio} = \frac{ERR + (ERR + 1)^3 + 1}{ERR \cdot 1 + r \cdot ERR \cdot (ERR + 2) + (L - r + 1) \cdot ERR[(ERR + 1)^2 + 1] + (L + 2)},$$

and

$$\text{Broadcast Ratio} = \frac{(ERR + 1)}{1 + (L - r + 1) \cdot ERR}.$$

In modified progression, the three types of access graphs (for the initial probe, the replicated index and the non-replicated indexes, respectively are shown in Figure 7c). Then the cost factors are

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (1 + ERR + n_i) + n_c \cdot (L - r + 1) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot (1 + ERR) + (L + 2).$$

Again, when $n_i = 1$ and $n_c = 1$,

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (ERR + 2) + (L - r + 1) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot (1 + ERR) + (L + 2)$$

$$\text{Broadcast} = 1 + (L - r + 1) \cdot ERR.$$

For continuous access,

$$\text{Tuning} = ERR \cdot 1 + \sum_{i=1}^{r} ERR \cdot (ERR + 2) + (L - r + 2) \cdot ERR$$

$$+ \sum_{i=r+1}^{L+1} ERR \cdot (1 + ERR) + (ERR + 1)^2 + (L + 3)$$

$$\text{Broadcast} = 2 + (L - r + 2) \cdot ERR$$

$$\text{Tuning Ratio} =$$

$$\frac{1 + ERR + (ERR + 1)^2}{ERR + r \cdot ERR \cdot (ERR + 2) + (L - r + 1) \cdot ERR \cdot (ERR + 2) + (L + 2)},$$

and

$$\text{Broadcast Ratio} = \frac{(ERR + 1)}{1 + (L - r + 1) \cdot ERR}.$$

6.2 Experimentation

Progression and modified progression methods have the capability of fault tolerance due to their characteristics of using the available replicates. In these, the replicated index buckets increase as r increases. This, however, increases the size of the broadcast cycle. We try to address the parameters affecting the performance of the access methods by performing a series of evaluations of the analytical expressions obtained earlier in this paper. To use the available replicate from the next segment, the error probability q should be contained in such a way that the condition $ERR \leq SEGMENT - 1$ is satisfied. The cost of ideal access of broadcast data without any access failure and data miss is also given for reference in our experiment. This ideal access is denoted by Ideal. To compare the different access methods, we set the different parameters for performance evaluations in the experiments as given below:

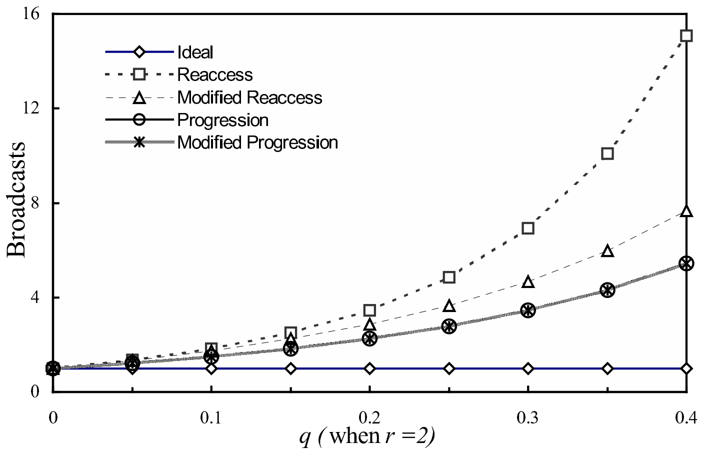
- R: Threshold in the form of maximum number of broadcasts when no change in DSS bits is assumed = 16.
- E: Average threshold without any change in the DSS bits $= R/2 = 8$.
- L: The depth of index tree excluding the set of data buckets = 5.
- r: The number of top levels of an index tree which constitute the replicate part $= 1 \sim 5$.
- q: The probability that a downloaded bucket contains a packet error $= 0.05 \sim 0.4$
- M: Outgoing pointers of an index bucket = 64.

Experiment 1. The effect of q for the number of broadcasts used to access the broadcast data is shown in Figure 8a). The progression and modified progression methods, which employ the index replicate, perform equally well when compared to other methods. The broadcasts rapidly grow up with higher error probabilities in a re-access method, because more errors cause more cycle waits. Modified re-access is slightly better, as it makes use of any bucket for version bits verification. In re-access method, the average threshold limit is reached and restarts take place after establishing the change in version bits of the broadcast, this also increases the access time in the form of broadcasts used. A user enters into a broadcast and searches for a desired record. He remains in system after the retrieval of data record, uses the just concluded search result to continue the retrieval of the same variable till the average threshold limit with respect to the number of broadcast ($E = 8$) where the broadcast is updated with the SSS bits. In this continuous retrieval of data, Figure 8b) shows the effect of q for the number of successes where the modified progression and progression methods give higher performance than the other two methods.

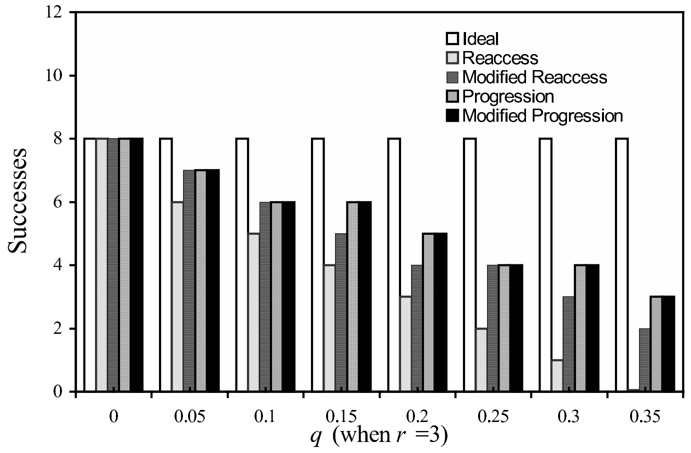
Experiment 2. In this experiment, the effect of q for the tuning time is shown in Figure 9a). Although tuning time in each case increases with q (except Ideal). The modified progression gives better results (though cycle wait becomes dominant and vital when q attains larger value) as it uses the index replicate in the same broadcast when the affected downloaded bucket is a replicated bucket, taking into consideration that modified progression uses less number of broadcasts (as given the previous experiment) and less number of buckets to be downloaded to complete the access of broadcast data. Our proposed method, i.e., modified progression, provides higher performance than the rest of the access methods. The effect of q for the average tuning time in a continuous process (when the user remains in system) is shown in Figure 9b). This average is computed when a user enters into a broadcast as a new user and remains there for ($R + E =$) 24 broadcasts. As observed in Figure 9b) the performance degrades more rapidly in all the access methods except the modified progression.

It is also observed (Figure 9c) in a continuous scheme the modified progression gives higher performance in terms of the average number of buckets (average tuning time) used to retrieve the desired information. It is observed that the average tuning time at the end of nR^{th} broadcast (n is a positive integer) is minimum, whereas it rises for the broadcasts $(nR+1)$, $(nR+2)$, $\ldots$, and $(nR+[b])$ where b is the number of broadcasts used to access the desired data record for the new user and $[b]$ is the maximum integer value contained in b and $b \leq R$. As seen in Figure 9c), average tuning time attains its maximum (for each cycle of R broadcasts) at the $(nR+[b])^{\text{th}}$ broadcast. When the user continues its access of data for a long time, i.e., when n is large, the average tuning time at $(nR + [b])^{\text{th}}$ broadcast converges to the average tuning time at the nR^{th} broadcast.

Experiment 3. The previous two experiments have revealed that the modified progression provides finer results in terms of tuning time and broadcasts with respect



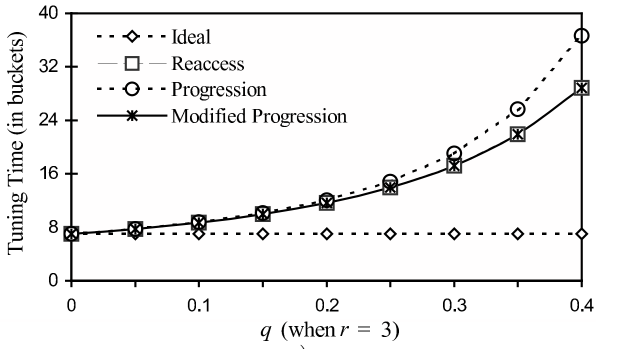
a)



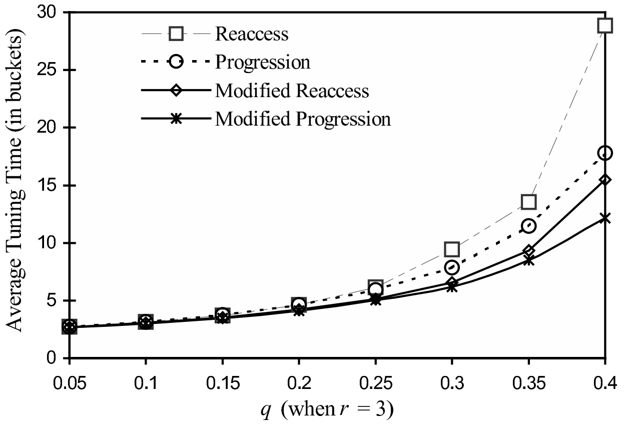
b)

Fig. 8. a) Broadcasts as a function of q (New User); b) Number of successes as a function of q in continuous process (User in System)

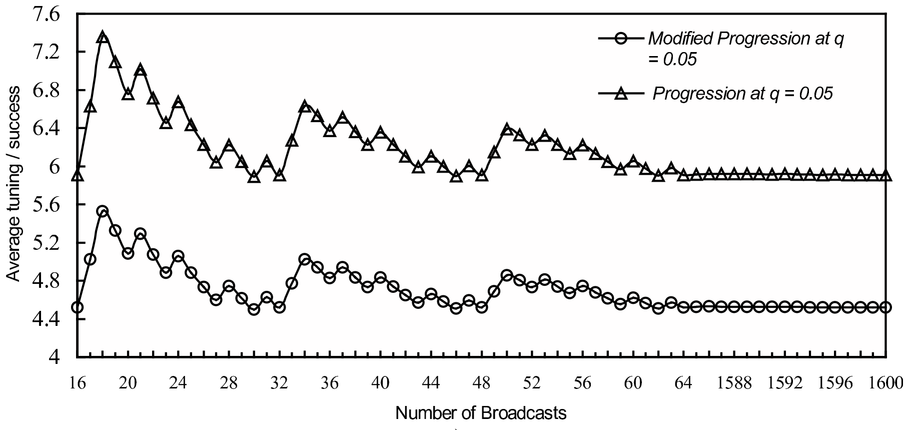
to q. Now in this experiment we consider the effect of r for modified progression method. As shown in Figure 10, we observe the effect of r for the broadcasts when $q = 0.1, 0.25$, and 0.4. The tolerance of modified method increases with r, and its performance in terms of broadcasts approaches to that of Ideal, i.e., it exploits less broadcasts to access the data. Another facet of an increase of r is that it also increases the $BCAST$; a false access on the non-replicated index involves a cycle wait before downloading a bucket from the next broadcast for testing the version number and resuming the search. As r increases, the cycle wait becomes longer, whereas



a)



b)



c)

Fig. 9. a) The tuning time as a function of q; b) Average tuning time as a function of q in continuous process; c) Comparison of average tuning w.r. to number of broadcasts in continuous access in a long term basis the number of non-replicated index buckets becomes smaller and more replicated buckets can be recovered from errors by employing their replicates so as to reduce the number of broadcasts used to access the broadcast data.

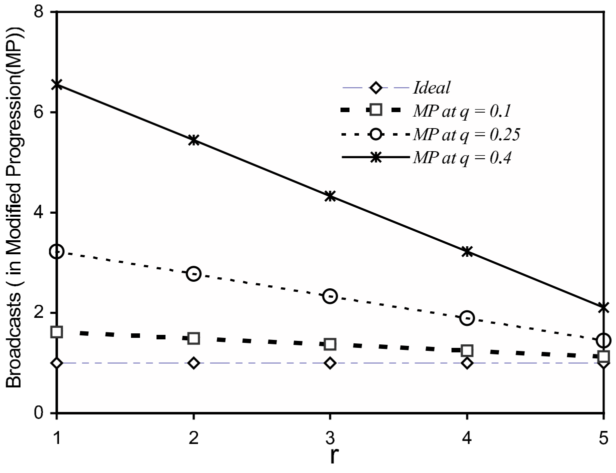


Fig. 10. The number of broadcasts as a function of r

7 CONCLUSIONS

In this paper, we investigated the access methods for the indexed data broadcasting in the mobile environment. Four methods (reaccess, modified reaccess, progression and modified progression, based upon the EPR scheme), are presented. In our numerical work, we have shown that our modified progression method gives best result for the cost of accessing broadcast data in terms of access time (here we measure it by the number of broadcast used) and the tuning time. In the wireless medium the occurrence of access failure tends to affect the data, and then we make use of the previous result and the index replicate available to shorten the time for recovering from an error in a progression method. In this the search range is used to record the location information in the broadcast data. This location information should be applied either to the same version of the broadcast data or the broadcast which is updated by SSS bits only (retaining previous DSS bits). We have also demonstrated that the modified progression method is more advantageous as it uses any kind of bucket to test the version number of the successive broadcasts. For retrieving the same data again or the updated data at the same location when no change happens in the DSS bits, the user in system incurs less additional cost in the form of tuning time and broadcasts. This bare minimum additional cost encourages the user to remain in system for continuous access so that updated information can be retrieved at less cost. This continuous access of broadcast data substantially saves the energy and may find applications in retrieving the stock, and weather information, etc. Taking into consideration all the experimental results, we conclude that our proposed modified progression method has the best performance as it employs the minimum number of broadcasts and a reasonable tuning time.

Acknowledgement

The authors would like to thank the anonymous reviewers for their useful suggestions that facilitated us to improve the paper to the present form.

REFERENCES

[1] Acharya, S.—Franklin, M.—Zdonik, S.: Dissemination-Based Data Delivery Using Broadcast Disks. IEEE Personal Comm., Vol. 2, 1995, No. 6, pp. 50–60.

[2] Acharya, S.—Franklin, M.—Zdonik, S.: Balancing Push and Pull for Data Broadcast. Proc. ACM SIGMOD Int'l Conf. Management of Data, Phoenix, Ariz., pp. 183–194, May 1997.

[3] Acharya, S.—Franklin, M.—Zdonik, S.: Disseminating Updates on Broadcast Disks. Proc. of VLDB Conf., pp. 354–365, India, Sep. 1996.

[4] Acharya, S.—Franklin, M.—Zdonik, S.: Prefetching from a Broadcast Disks. Proc. of 12th Int'l. Conf. of Data Engg., pp. 276–285, New Orleans, Feb. 1996.

[5] Acharya, S.—Franklin, M.—Zdonik, S.—Alonso, R.: Broadcast Disks: Data Management for Asymmetric Communication Environments. Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 199–210, San Jose, May 1995.

[6] Acharya, S.—Franklin, M.—Zdonik, S.—Alonso, R.: Disks on Air. Proc. ACM SIGMOD, 1994.

[7] Chen, M.—Yu, P. S.—Wu, K.: Indexed Sequential Data Broadcasting in Wireless Mobile Computing. Proc. 17th Int'l Conf. Distributed Computing Systems, Baltimore, pp. 124–131, May 1997.

[8] Cheriton, D.: Dissemination-Oriented Communication Systems. Technical Report, Stanford Univ., 1992.

[9] Dao, S.—Perry, B.: Information Dissemination in Hybrid Satellite/Terrestrial Networks. IEEE Data Eng. Bull., Vol. 19, 1996, No. 3, pp. 12–18.

[10] Duchamp, D.—Reynolds, N.: Measured Performance of a Wireless LAN. Columbia University, Oct. 1992.

[11] Erçetin, Ö.—Tassiulas, L.: Push-Based Information Delivery in Two Stage Satellite-Terrestrial Wireless Systems. IEEE Transaction on Computers, Vol. 50, 2001, No. 5, pp. 506–518.

[12] Franklin, M.—Zdonik, S.: Data in Your Face: Push Technology in Perspective. ACM SIGMOD, pp. 516–519, Seattle, June 1998.

[13] Franklin, M.—Zdonik, S.: Dissemination-Based Information Systems. IEEE Data Eng. Bull., Vol. 19, 1996, No. 3.

[14] Gifford, D. K.: Polychannel Systems for Mass Digital Communication. Comm. ACM, Vol. 33, 1990, No. 2.

[15] Herman, G. et al.: The Datacycle Architecture for Very Large High Throughput Database Systems. Proc. ACM SIGMOD Conf., San Fran., pp. 97–103, May 1987.

[16] Hughes Network Systems DirecPC, `http://www.direcpc.com`, 2001.

[17] Imielinski, T.—Badrinath, B. R.: Data Management for Mobile Computing. SIGMOD RECORD, Vol. 22, 1993, No. 1, pp. 34–39.

[18] Imielinski, T.—Badrinath, B. R.: Mobile Wireless Computing: Challenges in Data Management. Communications of ACM, Vol. 37, 1994, No. 10, pp. 18–28.

[19] Imielinski, T.—Viswanathan, S.: Adaptive Wireless Information Systems. Proceeding SIG Database Systems Conference, pp. 19–41, Tokyo, October 1994.

[20] Imielinski, T.—Viswanathan, S.—Badrinath, B. R.: Energy Efficient Indexing on Air. Proc. SIGMOD'94 Conf., pp. 25–36, May 1994.

[21] Imielinski, T.—Viswanathan, S.—Badrinath, B.: Data on Air: Organization and Access. IEEE Transactions of Data and Knowledge Engineering, Vol. 9, 1997, No. 3, pp. 353–372.

[22] Imielinski, T.—Viswanathan, S.—Badrinath, B.: Power Efficient Filtering of Data on Air. Proc. 4th Intl. Conf. of Extending Database Technology, pp. 245–258, Mar. 1994.

[23] Lee, W. C.—Lee, D. L.: Using Signature Techniques for Information Filtering in Wireless Mobile Environments. Distributed and Parallel Databases. Vol. 4, 1996, No. 3, pp. 205–227.

[24] Lo, Sh.-Ch.—Chen Arbee, L. P.: An Adaptive Access Method for Broadcast Data under an Error-Prone Mobile Environment. IEEE Transaction on Knowledge and Data Engineering. Vol. 12, 2000, No. 4, pp. 609–620.

[25] Marimba's Castanet, `http://www.marimba.com`, 2001.

[26] PointCast's webcasting, `http://www.pointcast.com`, 2001.

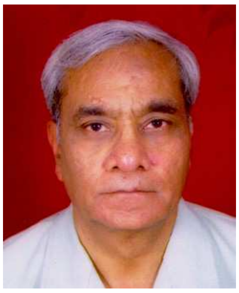
[27] Saxena, P. C.—Arora, I. J.: Cost Effective Indexed Data Broadcasting in Wireless Mobile Computing. International Journal of Information and Computing Science, Vol. 4, 2001, No. 1, pp. 27–38.

[28] Shekhar, S.—Fetterer, A.—Lui, D.: Genesis: An Approach to Data Dissemination in Advanced Traveler Information Systems. IEEE Data Eng. Bull., Vol. 19, 1996, No. 3, pp. 37–44.

[29] Teleglobe and AT & T Systems' Internet Delivery Service, `http://www.ats.com`, 2001.

[30] Wong, J. W.: Broadcast Delivery. Proc. IEEE, Vol. 76, 1998, No. 12, pp. 1566–1577.

[31] Zdonik, S.—Franklin, M.—Alonso, R.—Acharya, S.: Are 'Disks in the Air' Just Pie in the Sky? IEEE Workshop on Mobile Computing Systems and Applications, December 1994.



Prem Chandra Saxena is a professor of computer science in the School of Computer Science in the School of Computer and System Sciences at Jawaharlal Nehru University, New Delhi (India). He has supervised 13 Ph.D. students in the field of data communication, networking and distributed computing and database management. He has also guided 67 M.Tech dissertations. His research interests include DBMS, distributed computing and mobile computing.

Inder Jeet Arora is a reader in the Department of Statistics, PGDAV College, University of Delhi, New Delhi, India. He received his M.Sc. and M.Phil. degrees in statistics from the same university in 1982 and 1984, respectively, and his Ph.D. degree in computer science from School of Computer and Systems Sciences from Jawaharlal Nehru University in New Delhi, India in 2002. His research interests include mobile computing, data dissemination, mobile ad-hoc networks, and sensor networks.