Chapter 4

Continuous Access of Broadcast Data Using Artificial Pointers

4.1 Introduction

Chapter 3 described the various desirable properties of broadcast program. In this chapter, the cost metrics of indexed broadcast is explored quantitatively using an analytical model. In particular, the performance along with the robustness of access method is being investigated in this chapter. This performance is studied for the client when no data miss happens but also in case when there is a data miss. The caching at the client in the form of artificial pointer is introduced which raises a number of new issues. The results highlight the drawbacks of standard results derived earlier by not resorting to the caching of an offset value in the form
of an *artificial pointer*. The performance of modified progression an optimum cost based algorithm, is then studied.

There are other major issues being addressed in this chapter:

- **Variant and Invariant Updates.** What are the performance benefits of filtering the Variant and Invariant bits of a version number over the traditional updates in the real-time systems? In addition, how sensitive is a particular client’s responsiveness to variation in the server broadcast?

- **Types of User.** User in-system caches the offset value for a particular record. This caching raises two new issues:
  
  - What is the impact of the cache at the client on the organization of the broadcast?
  
  - What is the performance improvement due to caching (*Continuous Process*)?

The results in this chapter are based upon some assumptions. These assumptions supplement those presented in the previous chapter and are as follows:

- **EPR Scheme.** Data file is associated with a fully balanced index tree to be broadcast by an EPR scheme.

- **Size of broadcast.** Size of broadcast (due to update of data) remains same but the structure of broadcast may have changed.

- **Data miss.** A situation of multiple data misses, which may happen in the progression and modified progression method is ignored.
Packet error. Due to its properties of equal probability for an error to occur in each position and independence of errors, we consider packet errors as interrupts in formulating our results.

The main contribution of this chapter is the identification of the types of updates in a broadcast data. Based upon the type of updates, various continuous processes to deal with the updates for one group of users are proposed. These algorithms are aimed at the reduction of the tuning time, as they require minimum number of probes. 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 chapter is as follows: in the next Section, the version number of data file is identified, which depends upon two types of updates. Based upon the types of update of the broadcast data and the users capability of keeping the previous search result, these users are classified into two groups. Various algorithms in Continuous Processes for one set of users are discussed in Section 4.3. Section 4.4 exhibits the different access methods dealing with the updates of broadcast data. In Section 4.5, we give the analytical expressions to compare the different access methods. In Section 4.6, we examine the performance comparison of the access methods by carrying out the series of evaluations. Finally, conclusion is given in Section 4.7.

4.2 Updates of Broadcast Data

This section addresses the two types of updates of the broadcast. These updates (stream of bits) are used to decide whether the previous location information should be applied to the current broadcast or a re-probe should take place. We
also observe that these updates work even when the access sequence is encountered with the access failures.

In a search range mechanism [LC00], search range dynamically record 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 that 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, then the search range may become invalid for the new version.

Therefore, in their approach, an update 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 considered are as The Company Name, Previous closing price, Today's opening price, and Current price (To shorten the data size, any description of variables except for company name can be discarded. The professionals and stock dealers can interpret the values of the variables of different companies). Table 4.1a) shows that the stock price of a company X retrieved at 12.45 P.M. from a broadcast associated with the version no. 000001.
Considering the prices as given in Table 4.1b) of company X, an investor (user) places an instruction for the sale of 500 shares immediately after verifying the same version number 0000001 of the next broadcast cycle. In this deal, he intends to receive $750 less than the estimated value $13375. An interesting aspect of this example is that the update of broadcast data as in Table 4.1b) 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 therefore is associated with this new broadcast cycle.

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 (same size and structure), which is a consequential of the replacement of data, is assigned the new version number. In a 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 ascertaining correctness about any update of data.

In our method, two types of updates influence the version change to the broadcast data, which occurs under the mobile environment are as follows:
Table 4.1 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.
• **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 either 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 $u$ invariant bits and $v$ variant bits as shown in Figure 4.1.

![Figure 4.1: Allocation of Variant and Invariant Bits in a Version Number](image)

In Table 4.1c) 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.
- $T_s$: Maximum disconnection time without mistaking the invariant update.
\( T_{d} \): Maximum disconnection time without mistaking the variant update.

\( T_{u} \): Maximum disconnection time without mistaking the version change.

The maximum number of invariant updates during the period \( T_{s} \) is \( \lceil T_{s} / T_{u} \rceil \). Therefore, for \( u \) and \( T_{s} \) values the following condition holds true

\[
\left\lceil \frac{T_{s}}{T_{u}} \right\rceil \leq 2^{u} - 1 \quad \Rightarrow \quad u \geq \log_{2} \left\{ \left\lceil \frac{T_{s}}{T_{u}} \right\rceil + 1 \right\},
\]

Similarly

\[
\left\lceil \frac{T_{d}}{T_{u}} \right\rceil \leq 2^{v} - 1 \quad \Rightarrow \quad v \geq \log_{2} \left\{ \left\lceil \frac{T_{d}}{T_{u}} \right\rceil + 1 \right\},
\]

If \( u = 3 \) and \( v = 6 \), then the maximum disconnection time without mistaking SSS update \( T_{s} \) and DSS update \( T_{d} \) are \( 7T_{u} \) and \( 63T_{u} \) respectively. Assuming \( T_{u} \) is one minute, then \( T_{s} \) and \( T_{d} \) 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.

4.2.1 Artificial Offset Value

In this section, we modify the search range mechanism [LC00] 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 $(P_i, K_i)$, where $P_i$ and $K_i$ are the pointers and key values respectively. If $OFFSET(P_i)$ 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 $B$ i.e., offset is a pointer to the bucket that contains the record identified by key value, 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., 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 index pointer)) then
\[ U = S(B) + OFFSET(P_{i+1}) - (N + 1) \]

Pointer to a specific bucket \( T \) from the first bucket within a broadcast can be computed by rule 2. This offset value provided by specifying the offset of a bucket \( T \) is known as Temporary (Artificial) Offset Value (abbreviated as \( TOV \)), which can be used to retrieve the same information again from the first bucket if accessed within the some time limit and provided the size or 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 3.3, 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 as Rule 2.

**Rule 2.**

- \[ S(B) = S(B) + OFFSET(P_i) \]
- \[ P = S(B) \]

Similarly, pointer to a specific bucket \( T \) (target bucket) from the current bucket in a broadcast can be computed by rule 3.

**Rule 3.**

- \[ Q = P - S(B); \text{ where } P \text{ is } TOV \text{ as obtained by rule 2.} \]

The search range \([L, U]\) is modified using Rule 4, when probing an index bucket \( B \) of the broadcast discovers a data miss. This means the desired record does not fall after an index bucket \( B \).
Rule 4.

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

4.2.2 Users in-system

A client, who was in listening mode goes into doze mode after successfully accessing the data and tune in again at the first replicated index bucket of the next broadcast cycle or tune it 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.

Figure 4.2: Types of Users in a broadcast
A client is said to be *in-system* if it has a) retrieved the desired record or probed a bucket in its access sequence. b) Computed the temporary offset value in its previous or current search. c) 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 \( \min \{ 2^u - 1, 2^v - 1 \} \), where \( u \) and \( v \) are the SSS and DSS bits respectively. When a client is in-system, it has knowledge of 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 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 4.2 is also due to the disconnection time, this disconnection time is the elective nature of the client:- any disconnection when the user is in-system is a planned failure (elective failure is used to distinguish planned failure and failure), which can be anticipated and prepared.

In the next Section, we present an improved progression method and the various Continuous Processes. These continuous processes are based upon the artificial pointer when the broadcast is updated with the SSS bits.

4.3 Continuous Processes

This section starts with the improvised progression method, which can be used by the new users for the initial search. The search range of the progression method
provides the basis for the various continual processes for the users, which remain in the system.

4.3.1 Progression Method

We can improve upon the progression method [LC00] 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 if the data is 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) as shown in Figure 4.3, which is based upon the mechanism of search range, is presented in the following. The search range \([L, U]\), and \(N\) are initialized as \([0, \text{Max\_Seq\_No.}]\), and 1 respectively. In addition, the wireless communication is prone to the packet errors, which occurs more frequently. To keep away from a situation where access procedure may enter into a long loop due to high frequency of packet errors, a threshold (Max\_Interrupts) in addition can be defined and incorporated into an access flow by limiting the maximum number of interrupts to be met in the access. Another threshold can also be defined by limiting the maximum duration for getting the desired data record but this situation is ignored just to simplify the algorithm.

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 can take 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** \((S(B) \geq L)\) **then** /* next bucket to visit has passed in the current broadcast 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. */

If (replicate available within search range \([L, U]\)) **then**

wait for the available replicate and download it.

Shift \(N\).

Set Flag to check data miss, if any.

**Else**

wait for the bucket with the sequence number \(L\) in the next broadcast cycle (Next_\(L\)).

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. Again if \(S(B)\) is the sequence number of the current downloaded bucket \(B\), which was retrieved after any of the actions stated above.

If (Next bucket to visit is induced to be a replicated index bucket) **then**
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. */

Else

Increase_L. /* L is changed to the sequence number of the next bucket & U need not be changed. */

Increase the value of N. If the tuple present in the first bucket of each broadcast segment, when the check_flag is on, wait for the first bucket of the next broadcast, and decrease the value of U (Decrease_U).

3. Compute the temporary offset value P of the target bucket T from the first bucket to get the desired record.

\[ S(B) = S(B) + OFFSET(P_i) \]

Where \( P_i \)'s are the pointers and

\[ OFFSET(P_i): \text{offset value for each } P_i. \]

\[ P = S(B). \]

The value of 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. The value of P vanishes/modifies when any new search for a bucket takes place or the disconnection time exceed its limit i.e., where all the versions due to either DSS bits or SSS bits are exhausted.

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 desired record and wastage of battery power.
Figure 4.3: Continuous Access of Data in Modified Progression Method

Diagram shows a flowchart with decision points and actions:

1. **Begin**
2. Start/resume/continue access
3. Download a desired bucket
4. Whether to retain same variable?
   - **YES**
     - Time limit?
       - **YES**
         - Check boundary 0 ≤ cur_loc ≤ P, cur_loc ≥ P
           - Update [L, U], to_L
           - Re-initialize [L, U]
       - **NO**
         - Increase L, Decrease U
         - Wait & get the designated bucket
         - Next L, Decrease U
         - L → l
         - Index replicate?
           - **YES**
             - Update [L, U], to_L
           - **NO**
             - Index replicate?
               - **YES**
                 - Update [L, U], to_L
               - **NO**
         - Data bucket?
           - **YES**
             - Data miss?
               - **YES**
                 - Match search key?
                   - **YES**
                     - Check version bits change?
                       - **YES**
                         - Records still valid
                       - **NO**
                         - Update [L, U], to_L
                       - **YES**
                         - Replicate index
                     - **NO**
                       - Update [L, U], to_L
                   - **NO**
                     - Check change in DSS bits?
                       - **YES**
                         - Replicate index
                       - **NO**
                         - Update [L, U], to_L
               - **NO**
                 - Continue retrieval?
                   - **YES**
                     - End
                   - **NO**
                     - Within disconnection time limit?
                       - **YES**
                         - Re-initialize [L, U]
                       - **NO**
                         - End
             - **NO**
               - Time check?
                 - **YES**
                   - End
                 - **NO**
                   - Wait & get the designated bucket
         - **NO**
           - Data bucket?
             - **YES**
               - Data miss?
                 - **YES**
                   - Match search key?
                     - **YES**
                       - Check version bits change?
                         - **YES**
                           - Records still valid
                         - **NO**
                           - Update [L, U], to_L
                         - **YES**
                           - Replicate index
                       - **NO**
                         - Update [L, U], to_L
                     - **NO**
                       - Check change in DSS bits?
                         - **YES**
                           - Replicate index
                         - **NO**
                           - Update [L, U], to_L
                 - **NO**
                   - Continue retrieval?
                     - **YES**
                       - End
                     - **NO**
                       - Within disconnection time limit?
                         - **YES**
                           - Re-initialize [L, U]
                         - **NO**
                           - End
             - **NO**
               - Data bucket?
                 - **YES**
                   - Data miss?
                     - **YES**
                       - Match search key?
                         - **YES**
                           - Check version bits change?
                             - **YES**
                               - Records still valid
                             - **NO**
                               - Update [L, U], to_L
                             - **YES**
                               - Replicate index
                           - **NO**
                             - Update [L, U], to_L
                         - **NO**
                           - Check change in DSS bits?
                             - **YES**
                               - Replicate index
                             - **NO**
                               - Update [L, U], to_L
                       - **NO**
                         - Continue retrieval?
                           - **YES**
                             - End
                           - **NO**
                             - Within disconnection time limit?
                               - **YES**
                                 - Re-initialize [L, U]
                               - **NO**
                                 - End
                      - **NO**
                        - Time check?
                          - **YES**
                            - End
                          - **NO**
                            - Update [L, U], to_L
                      - **NO**
                        - Update [L, U], to_L
                      - **NO**
                        - Increase L, to_L
                - **NO**
                  - Time limit?
                    - **YES**
                      - Check boundary 0 ≤ cur_loc ≤ P, cur_loc ≥ P
                        - Update [L, U], to_L
                        - Re-initialize [L, U]
                    - **NO**
                      - Increase L, to_L

Diagram notes:
- (i) S(B) = S(B) + OFFSET(P)
- (ii) P = S(B)
4.3.2 Continuous Retrieval Algorithms

Our main objective of the continuous process is to provide the retrieval/resumption of the target bucket from the following broadcasts using search range method when the user has already successfully probed and accessed the desired record. This continual process is particularly useful for the users those are in-system. We propose the following ways to continue its process of retrieval of data records. If the user

I. Reacquire the same search result from the next broadcast using the previously stored range data (i.e., wish to retain the updated information of the same variables). In this 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.

II. Disconnects for a certain time and tunes in again, either anywhere in the next broadcast (except at the beginning) 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 termed as continuous process after disconnection.

III. When the user altogether goes for the new search.

4.3.2.1 Continuous Process without Disconnection

In this case, the user after successfully retrieval of the desired record does not disconnect from the broadcast and is willing to reacquire the same search result as shown in Figure 4.4. The detailed procedure is given as follows:
**Procedure**: Continuous Access1 (Without Disconnection from Broadcast)

**Begin**
1. Initialize $L = 0$.
2. Wait for the arrival of the bucket with the sequence number $L$ in the next broadcast cycle.
3. **If** ($\text{Disconnection time} < \text{Limit}$) **then**
4. **If** (change in version bits) **then**
5. **If** (change in DSS bits) **then**
6. go to step 18.
7. **Else**
8. $L = S(B) + \text{OFFSET}(P-S(B))$.
9. go to step 20.
10. **End If.**
11. **Else**
12. No change in data since the previous search.
14. **End If.**
15. **Else**
16. go to step 18.
17. **End If.**
18. Apply progression method for fresh search and obtain $TOV \cdot P$.
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 search key 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.**
Figure 4.4: Continuous Access of data without disconnection in Progression Method
Figure 4.5: Continuous Access of data after disconnection in Progression Method

[Diagram flowchart showing the process of continuous access after disconnection]
4.3.2.2 Continuous Process after Disconnection

In the following, we demonstrate the procedure in detail to access the updated data of the variable, when a client enters into a broadcast within a disconnection time. This is as follows:

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 next bucket in the range $[0,P]$. Otherwise, wait for the arrival of the bucket with sequence number $L$ 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), the offset of the target bucket $T$ from the current bucket is computed as a difference between the bucket $id$ of $T$ and the bucket $id$ of current bucket. Since 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 $P - Q$, where $Q$ is the sequence number of the current bucket as displayed in Figure 4.5 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 2 (When disconnection is within limit)

**Begin**

1. **If** $(S(B) \geq P)$ **then**
2 Wait for the arrival of the bucket with the
    Sequence number \( L \) in the next broadcast cycle.

3 \textbf{Else If} (Replicated index bucket available in the range \([0,P]\)) \textbf{then}
   
4 tune to the first bucket of the next broadcast segment in the range
   \([0,P]\)

5 shift \( N \)

6 \textbf{Else}

7 increase \( L \)

8 wait for the arrival of the next bucket

9 increase \( N \)

10 \textbf{End If}.

\textbf{End}.

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

Now we provide two representative examples (one each with data miss
and without any data miss) showing that lesser number of probes are required in a
continuous algorithms for the users those are in-system. These algorithms also
work appropriate, even if the occurrence of frequent access failures is caused by
various communication noises.

The examples shown in Table 4.2 convey that the artificial offset value
used to reacquire the updated record from the same search range performs quite
well in the continuous process. This artificial offset value works well with the new

89
<table>
<thead>
<tr>
<th>Type of User</th>
<th>An Access Sequence</th>
<th>The Corresponding Search Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh Search for new user</td>
<td>32→ Fifth_R→ First_a₃→ b₈→ c₂₄→ 66</td>
<td>[0, max_seq_no]→ [S(First_a₃), S(80)]→ [S(b₈), S(80)]→ [S(c₂₄), S(80)]→ [S(66), S(80)]→ Match 66</td>
</tr>
<tr>
<td>When the User in system reacquire the same search result in the next broadcast without disconnection.</td>
<td>66 (downloaded bucket)→First_R→ 66</td>
<td>[0, S(80)]→ Match 66</td>
</tr>
<tr>
<td>When the user in system reacquire the same search result after disconnection and if cur_loc &lt; P.</td>
<td>18→ Fourth_R→ 66</td>
<td>[0, max_seq_no]→ [S(Fourth_R), S(80)]→ [S(66), S(80)]→ Match 66</td>
</tr>
<tr>
<td>When the user in system reacquire the same search result after disconnection and if cur_loc &gt; P.</td>
<td>69→ Ninth_R→ First_R→ 66</td>
<td>[0, max_seq_no]→ [0, S(71)]→ [S(66), S(71)]→ Match 66</td>
</tr>
<tr>
<td>Fresh Search for new user</td>
<td>64→ Ninth_R→ First_a₁→ b₂→ c₆→ match 16</td>
<td>[0, max_seq_no]→ [0, S(71)]→ [S(First_a₁), S(26)]→ [S(b₂), S(26)]→ [S(c₆), S(26)]→ [S(16), S(26)]→ Match 16</td>
</tr>
<tr>
<td>When the User in system reacquire the same search result in the next broadcast without disconnection.</td>
<td>16 (downloaded bucket)→First_R→ 16</td>
<td>[0, S(26)]→ [S(16), S(26)]→ Match 16</td>
</tr>
<tr>
<td>When the user in system reacquire the same search result after disconnection and if cur_loc &lt; P.</td>
<td>6→ Second_R→ 16</td>
<td>[0, max_seq_no]→ [S(Second_R), S(80)]→ [S(16), S(80)]→ Match 16</td>
</tr>
<tr>
<td>When the user in system reacquire the same search result after disconnection and if cur_loc &gt; P.</td>
<td>64→ Ninth_R→ First_R→ 16</td>
<td>[0, max_seq_no]→ [0, S(71)]→ [S(16), S(71)]→ Match 16</td>
</tr>
</tbody>
</table>

Table 4.2: Search range changes for new user and the user in system
broadcast which undergoes the process of modification of data values without disturbing the data size or structure of the broadcast i.e., the broadcast with a different version number due to change in the SSS bits for the new broadcast.

4.4 Access Methods for Handling the Updates

In this section, the different methods are being presented to deal with the updates of broadcast data and the access failures. We make use of the access sequence 64, Ninth_R, First_R, First_a, b_2, c_6, and 16 to elucidate the revival activities when the bucket b_2 contains a packet error and is abandoned.

Reaccess: In a reaccess method, a user reaccess the bucket affected by interrupts. Each affected bucket is reaccessed after verifying the version bits from the first replicated bucket of the next broadcast. If there is no change or change in SSS bits is found then the affected bucket is reaccessed, otherwise the user is regarded as a new user and has to access the broadcast data right from the beginning. In our example, since the bucket b_2 is reaccessed 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, b_2, First_R, b_2, c_6, and 16. 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, b_2, First_R, First_R, b_2, c_6, and 16. The defect of this method is that the waiting time for reaccessing of the affected bucket of the next broadcast cycle is very long and is equal to the time for the broadcast cycle.
Modified Reaccess: In modified reaccess method, reaccess of the affected bucket takes place only after verifying the version bits from any bucket of the next broadcast. Similarly, reaccess the bucket $b_2$ after verifying the version number of the next broadcast in second attempt (i.e., when $First_R$ or any other bucket also has a packet error), the access sequence changed to 64, $Ninth_R$, $First_R$, $First_{a_1}$, $b_2$, $First_R$, $First_{a_1}$, $b_2$, $c_6$, and 16.

Although this method gives an improvised result, yet suffers from 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 that appears in the successive broadcast segment and at the same position in these segments from the same broadcast. Each of the remaining affected buckets is reaccessed after confirmation of the version number of the next broadcast from the replicated index bucket. Consider, a bucket $b_2$ contains a packet error, then the changed access sequence becomes 64, $Ninth_R$, $First_R$, $First_{a_1}$, $b_2$, $First_R$, $b_2$, $c_6$, and 16. 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 appears in the next segment but at the same position, the access sequence is given by 64, $Ninth_R$, $First_R$, $First_{a_1}$, $b_2$, $First_R$, $Second_R$, $b_2$, $c_6$, and 16.

Modified Progression: In a Modified Progression, the version number confirmation takes place at any of the bucket in the next broadcast. The changed access sequence when downloaded bucket $b_2$ contains a packet error and
confirmation of version number is completed in second attempt, then becomes 64, Ninth_R, First_R, First_a_1, b_2, First_R, First_a_1, b_2, c_6, and 16. Since, in a 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 required for downloading a bucket for confirmation is not as long as it is needed in the progression method.

4.5 Performance Comparisons and Impact of Updates

In our numerical studies, we address many significant parameters that have an influence on the performance of the various access methods. In particular, we examine the efficiency of the access methods (reaccess, modified reaccess, progression, and modified progression) by estimating the cost of accessing the broadcast data with respect to the number of levels of replicated index buckets from the top of an index tree \( r \) and the probability of the downloaded bucket containing an error \( q \). In our performance analysis, updates are performed on the broadcast data and the user examines the version of the following broadcasts (if required for the data access) before resuming the search. This model also addresses the weakness of the methods while dealing with the updates. 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:**

Let us assume a broadcast under an \( EPR \) scheme with 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.
- $r$: Number of top levels of an index tree which constitute the replicated part.
- $SEGMENT$: The size of the broadcast segment.
- $INDEX$: The size of an index of a broadcast cycle.
- $BCAST$: 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 and $BCAST$ is the product of the number of non-replicated index buckets and the size of the broadcast segment. Therefore

$$SEGMENT = r + \frac{M^{l-r+1} - 1}{M - 1}, \text{ and}$$

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

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 are performed on the broadcast data and to simplify the analysis, the size of the broadcast cycle ($BCAST$) for each broadcast is same, but the structure may be different i.e., $BCAST_i = BCAST \forall i$.

- Multiple Data misses may happen in the access methods, this situation is ignored just to simplify the procedure.

- Since the packet error has some inheritance properties: Occurrence of an error in each position has an 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 following subsection, we develop an analytical model using an access graph, which provides us the estimation of the cost of accessing the broadcast data.

4.5.1 Analytical Expression for Dealing with Updates

An access graph in the form of a state transition diagram is being 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)[LC00]. 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 an increase
in the cost of accessing broadcast data in terms of access time and tuning time. 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 contains packet error. This downloaded bucket may be a replicated bucket (when its replicate is not available in the current broadcast), non-replicated bucket or a data bucket. 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) 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 to access the broadcast data. The boundary of both the tuning time and the number of broadcasts used (herein referred as Tuning and Broadcasts respectively) are being obtained. These metrics, the tuning time and the broadcasts for the Ideal access graph can be obtained by the following relations:

\[
\text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 2, \text{ and }
\]

Broadcasts = 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 the following
relations can measure the cost of broadcast data for the Ideal access graph (without verification of version bits of the next broadcast).

\[ \text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 3, \text{ and} \]

\[ \text{Broadcasts} = 2. \]

The other cost parameters \textit{additional tuning}, \textit{tuning ratio}, \textit{additional broadcasts}, and \textit{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. \textit{The additional tuning (broadcasts) is the cost required to reaccess the same data bucket or the bucket at the same location from the next broadcast, and 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 \textit{ERR}, then it can be obtained using a parameter \( q \) as follows:

\[
x(\text{number of errors}): \quad 1 \quad 2 \quad 3 \quad ... \\
\text{Probability } p(x) : \quad q \quad q^2 \quad q^3 \quad ... \\
\text{ERR} = E(X) = \sum_{x=1}^{\infty} x.p(x) = \sum_{x=1}^{\infty} x.q^x = q.p^{-2}
\]

The tuning and broadcasts can be divided into two parts a) false part, and b) correct part. In a process of finding the cost of broadcast data, the dotted lines
specified above the solid lines give the Broadcasts used, as shown in an access
graph in Figure 4.6.

4.5.1.1 Reaccess

In the reaccess, there exist two kinds of access graphs as shown in Figure 4.6a),
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:

![Diagram](image)

Figure 4.6: a) Access graph for reaccess method, b) Access graph for
modified reaccess method, and c) Access graph for modified
progression.
Type 1: $ERR$

Type 2: $ERR.(1 + n_c.ERR + 1)$

Where $n_c$ is the cost of reaccessing the same bucket in the following broadcast cycle? A relation can compute the cost of false tuning part is

$$\text{False Tuning} = ERR.1 + ERR.(1 + n_c.ERR + 1) + ... + ERR.(1 + n_c.ERR + 1)$$

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

The Tuning and the Broadcasts, when $n_c = 1$, are given by

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

$$= ERR.[1 + (L + 1).(ERR + 2)] + (L + 3)$$

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

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

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

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

The other cost parameters tuning ratio, and broadcasts ratio are given by

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

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

4.5.1.2 Modified Reaccess

In this method, verification of version bits takes place at any of the bucket; accordingly, the modified access graphs are shown in Figure 4.6b).
The cost parameters in this case are given by

\[
\text{Tuning} = ERR.1 + ERR.(ERR + 2) + \ldots + ERR.(ERR + 2) + (L + 2)
\]

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

Broadcasts = 1 + (L + 1).ERR

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

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

Broadcasts = 2 + (L + 2).ERR

Tuning Ratio = \( \frac{(ERR + 1)(ERR + 2)}{ERR.[1 + (L + 1).(ERR + 2)] + (L + 2)} \), and

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

### 4.5.1.3 Progression and Modified Progression

With an objective to minimize the cost of accessing a 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 \( n_i \) be the cost of finding the next index replicate at level \( i \) of the index tree, then for the progression method

\[
\text{Tuning} = ERR.1 + \sum_{i=1}^{L} ERR.(1 + ERR + n_i) + n_c (L - r + 1).ERR + \sum_{i=r+1}^{L} ERR.[ERR.(1 + ERR + n_i) + 1] + (L + 2)
\]

For \( n_i = 1 \) and \( n_c = 1 \),

\[
\text{Tuning} = ERR.1 + \sum_{i=1}^{L} ERR.(ERR + 2) + (L - r + 1).ERR + \sum_{i=r+1}^{L} ERR.[ERR.(ERR + 2) + 1] + (L + 2)
\]
Broadcasts = 1 + (\textit{L} - r + 1).\textit{ERR}

For Continuous access

\[
\text{Tuning} = \text{ERR}.1 + \sum_{i=1}^{L} \text{ERR}.(\text{ERR} + 2) + (\text{L} - r + 2).\text{ERR} + 
\sum_{i=r+1}^{L} \text{ERR}.[\text{ERR}.(\text{ERR} + 2) + 1] + 
(\text{ERR} + 1).[\text{ERR}.(\text{ERR} + 2) + 1] + (\text{L} + 3)
\]

Broadcasts = 2 + (\text{L} - r + 2).\text{ERR}

\text{Tuning Ratio} = 

\[
\frac{\text{ERR} + (\text{ERR} + 1)^3 + 1}{\text{ERR}.1 + r.\text{ERR}.(\text{ERR} + 2) + (\text{L} - r + 1).\text{ERR}.[(\text{ERR} + 1)^2 + 1] + (\text{L} + 2)}, \text{ and}
\]

\text{Broadcast Ratio} = 

\[
\frac{(\text{ERR} + 1)}{1 + (\text{L} - r + 1).\text{ERR}}
\]

In modified progression, the three types of access graphs one each for the initial probe, the replicated index and the non-replicated indexes are shown in Figure 4.6c). Then the cost factors are

\[
\text{Tuning} = \text{ERR}.1 + \sum_{i=1}^{L} \text{ERR}.(1 + \text{ERR} + \text{n}_i) + \text{nc}.(\text{L} - r + 1).\text{ERR} + 
\sum_{i=r+1}^{L} \text{ERR}.(1 + \text{ERR}) + (\text{L} + 2)
\]

Again when \text{n}_i = 1 \text{ and } \text{nc} = 1,

\[
\text{Tuning} = \text{ERR}.1 + \sum_{i=1}^{L} \text{ERR}.(\text{ERR} + 2) + (\text{L} - r + 1).\text{ERR} + 
\sum_{i=r+1}^{L} \text{ERR}.(1 + \text{ERR}) + (\text{L} + 2)
\]

Broadcasts = 1 + (\text{L} - r + 1).\text{ERR}
For continuous access

\[
\text{Tuning} = ERR.1 + \sum_{i=1}^{t+1} ERR.(ERR + 2) + (L - r + 2).ERR + \\
\sum_{i=r+1}^{t+1} ERR.(1 + ERR) + (ERR + 1)^2 + (L + 3)
\]

\[
\text{Broadcasts} = 2 + (L - r + 2).ERR
\]

\[
\frac{1 + ERR + (ERR + 1)^2}{ERR + r.ERR.(ERR + 2) + (L - r + 1).ERR.(ERR + 2) + (L + 2)^2}
\]

\[
\frac{(ERR + 1)}{1 + (L - r + 1).ERR}
\]

4.5.2 **Analytical Model (for Data Miss Cases)**

Similarly, in case of data miss, the process of searching the broadcast data, we use again an access graph in the form of a state transition diagram. Again, it has two types of edges, solid lines (transition with correct access) and dotted lines (transition with false access). The labels on each edge indicate transition cost in buckets for evaluating the tuning/access time. Updates also results in the increase in the cost of accessing broadcast data in terms of access time and tuning time. This is due to the checking of version bits of the successive broadcast cycles. These successive broadcasts are used when the downloaded bucket contains packet error. Therefore, the cost in terms of ‘number of broadcasts used’ and b) the tuning time can measure the accessing of the broadcast data using the buckets from the successive broadcasts.

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 three additional buckets. The additional buckets are a) the first probe bucket, b) the result data bucket, and c) the first bucket of the broadcast data when a data miss happens). Similarly, the buckets from the two broadcasts are used for accessing the broadcast data. The boundary of both the tuning time and the number of broadcasts used (herein referred as Tuning and Broadcasts respectively). The following relations can obtain these metrics, the tuning and the broadcasts for the Ideal access graph are

\[
\text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 3, \text{ and}
\]

\[
\text{Broadcasts} = 2.
\]

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

\[
\text{Tuning} = \sum_{i=1}^{L+1} s_i = L + 4, \text{ and}
\]

\[
\text{Broadcasts} = 3.
\]

We also define other cost parameters additional tuning, tuning ratio, additional broadcasts, and broadcasts ratio, which 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.

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
\[
ERR = E(X) = \sum_{x=1}^{\infty} x.p(x) = \sum_{x=1}^{\infty} x.q^x = q.p^{-2}
\]

In an error-prone situation, both the cost parameters, can be divided into two parts a) false part, and b) correct part. In a process of finding the cost of broadcast data, the dotted lines specified above the solid lines give the broadcasts used and the checking of data miss is performed on the thick-bordered circular node as shown in an access graph in Figure 4.7.

4.5.2.1 Reaccess

In this method, there exist two kinds of access graphs as shown in Figure 4.7a), Type 1 specifies 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

Type 1: \(ERR\)

Type 2: \(ERR.(1 + n_c. ERR + 1)\)

Therefore, false tuning part (when \(n_c = 1\), is given by

\[
\text{False Tuning} = ERR.1 + ERR.(1 + n_c. ERR + 1) + \ldots + ERR.(1 + n_c. ERR + 1)
= ERR.[1 + (L + 2).(ERR + 2)]
\]

The Tuning and the Broadcasts are

\[
\text{Tuning} = ERR.1 + ERR.1 + (L + 1).(ERR + 2) + \sum_{i=1}^{L+1} s_i
= ERR.[2 + (L + 1).(ERR + 2)] + (L + 3)
\]

\[
\text{Broadcasts} = 2 + ERR[(ERR + 1).(L + 1) + 1]
\]

104
For continuous access of the bucket at the same location when broadcast is verified and found to be updated with the SSS bits only, and when the user is in-system, then

\[
Tuning = ERR \cdot [(2 + (L + 2) \cdot (ERR + 2)) + (ERR + 1) + (L + 4)]
\]

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

The other cost parameters tuning ratio, and broadcasts ratio are given by

\[
Tuning Ratio = \frac{(ERR + 1) \cdot (ERR + 2)}{ERR \cdot [2 + (L + 1) \cdot (ERR + 2)] + (L + 3)}, \text{ and}
\]
Broadcast Ratio = \( \frac{(ERR + 1)^2}{2 + ERR.(ERR + 1).(L + 1) + 1} \)

### 4.5.2.2 Modified Reaccess

In the modified method, confirmation of version bits takes place at any of the bucket; consequently, the modified access graphs are shown in Figure 4.7b).

The cost parameters in this case are as

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

\[
= ERR.\left[2 + (L + 1).(ERR + 2)\right] + (L + 3)
\]

Broadcasts = \(2 + (L + 1).ERR\)

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

\[
\text{Tuning} = ERR.\left[2 + (L + 2).(ERR + 2)\right] + (ERR + 1) + (L + 4)
\]

Broadcasts = \(3 + (L + 2).ERR\)

\[
\text{Tuning Ratio} = \frac{(ERR + 1).(ERR + 2)}{ERR.\left[2 + (L + 1).(ERR + 2)\right] + (L + 3)}, \text{ and}
\]

Broadcast Ratio = \(\frac{(ERR + 1)}{2 + (L + 1).ERR}\)

### 4.5.2.3 Progression and Modified Progression Methods

To reduce the cost of accessing a broadcast data in terms of tuning time, verification of version bits in modified progression method does not specify any condition on the type of buckets. If \(n\), be the cost of finding the next index replicate at level \(i\) of the index tree, then for the progression method
Tuning =

\[ ERR.1 + \sum_{i=1}^{r+1} ERR.(1 + ERR + n_i) + n_c (L - r + 1).ERR + \]

\[ \sum_{i=r+1}^{l+1} ERR.[ERR.(1 + ERR + n_i) + 1] + (L + 3) \]

When \( n_i = n_c = 1 \),

Tuning = \[ ERR.1 + \sum_{i=1}^{r+1} ERR.(ERR + 2) + (L - r + 1).ERR + \]

\[ \sum_{i=r+1}^{l+1} ERR.[ERR.(ERR + 2) + 1] + (L + 3) \]

Broadcasts = \( 2 + (L - r + 1).ERR \)

For Continuous access

Tuning = \[ ERR.1 + \sum_{i=1}^{r+1} ERR.(ERR + 2) + (L - r + 2).ERR + \]

\[ \sum_{i=r+1}^{l+1} ERR.[ERR.(ERR + 2) + 1] + \]

\[ (ERR + 1).[ERR.(ERR + 2) + 1] + (L + 4) \]

Broadcasts = \( 3 + (L - r + 2).ERR \)

\[ \frac{ERR + (ERR + 1)^3 + 1}{ERR.1 + (r + 1).ERR.(ERR + 2) + (L - r + 1).ERR.[(ERR + 1)^2 + 1] + (L + 3)} \]

Broadcast Ratio = \[ \frac{(ERR + 1)}{2 + (L - r + 1).ERR} \]

In modified progression, the three types of access graphs one each for, the initial probe, the replicated index and the non-replicated indexes are shown in Figure 4.7c). Then the cost parameters are
\[ \text{Tuning} = \]
\[ \text{ERR}.1 + \sum_{i=r+1}^{r+1} \text{ERR}.(1 + \text{ERR} + n_i) + n_c (L - r + 1).\text{ERR} + \]
\[ \sum_{i=r+1}^{r+1} \text{ERR}.(1 + \text{ERR}) + (L + 3) \]

Again, when \( n_i = n_c = 1 \),

\[ \text{Tuning} = \text{ERR}.1 + \sum_{i=r+1}^{r+1} \text{ERR}.(\text{ERR} + 2) + (L - r + 1).\text{ERR} + \]
\[ \sum_{i=r+1}^{r+1} \text{ERR}.(1 + \text{ERR}) + (L + 3) \]

\[ \text{Broadcasts} = 2 + (L - r + 1).\text{ERR} \]

For continuous access

\[ \text{Tuning} = \text{ERR}.1 + \sum_{i=r+1}^{r+1} \text{ERR}.(\text{ERR} + 2) + (L - r + 2).\text{ERR} + \]
\[ \sum_{i=r+1}^{r+1} \text{ERR}.(1 + \text{ERR}) + (\text{ERR} + 1)^2 + (L + 4) \]

\[ \text{Broadcasts} = 3 + (L - r + 2).\text{ERR} \]

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

\[ \text{Broadcast Ratio} = \frac{(\text{ERR} + 1)}{2 + (L - r + 1).\text{ERR}} \]

4.6 Experimentation

A 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 chapter. 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 (in both the data miss and no data miss cases), we set the different parameters for performance evaluations in the experiments as given below in Table 4.3.

Experiment1: (No Data Miss).

The effect of $q$ for the number of broadcasts used to access the broadcast data are shown in Figure 4.8. The progression and modified progression method, which employ the index replicate, perform equally well as compare to other methods. The broadcasts rapidly grow up with higher error probabilities in a reaccess method because more errors cause more cycle waits. Modified reaccess is slightly better as it makes use of any bucket for version bits verification. In a reaccess method, the average threshold limit is reached and the client restarts the reaccess after establishing the change in the version bits of the broadcast. This also increases the access time in the form of broadcasts used. A user enters into a
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Default values</th>
<th>Default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>Threshold in the form of maximum number of broadcasts when no change in DSS bits is assumed</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>$E$</td>
<td>Average Threshold without any change in the DSS bits</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>$L$</td>
<td>Depth of the index tree excluding the data buckets</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$r$</td>
<td>The number of top levels of an index tree constitute the replicated part</td>
<td>1 ~ 5</td>
<td>1 ~ 5</td>
</tr>
<tr>
<td>$q$</td>
<td>The probability that a downloaded bucket contains a packet error</td>
<td>0.05 ~ 0.4</td>
<td>0.05 ~ 0.4</td>
</tr>
<tr>
<td>$M$</td>
<td>Outgoing pointers of an index bucket</td>
<td>64</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 4.3
Parameter Settings

broadcast and search for a desired record and remains in-system after the retrieval of data record. He uses the just concluded search result to continue the retrieval of 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 is reached. In this continuous retrieval of data, Figure 4.9 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: (Data Miss)

Similarly, the client uses the just concluded search result to continue the retrieval of same variable till the average threshold limit with respect to the number of broadcast ($E = 9$) is reached. In this continuous retrieval of data, Figure 4.10 a)
shows the effect of \( q \) for the number of successes in an average threshold limit \( (E = 9) \), where the modified progression and progression methods give higher performance than the other two methods.

**Figure 4.8:** Broadcasts as a function of \( q \) (New User).

**Figure 4.9:** Number of successes as a function of \( q \) in continuous process (User in-System)
Figure 4.10: a) Successes as a function of $q$ in Continuous process without disconnection, and b) Broadcasts as a function of $q$ for new user.

Experiment 3: (No Data Miss)

In this experiment, the effect of $q$ for the tuning time is shown in Figure 4.11. Although tuning time in each case increases as $q$ increases (except in the case of Ideal). The modified progression gives better results (though cycle wait become dominant and vital when $q$ attains larger value) as it uses the index replicate in the same broadcast when an affected downloaded bucket is a replicated bucket.
Taking into consideration that modified progression uses less number of broadcasts (as given in 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 4.12. This average is computed when a user enters into a broadcast as a new user and remains there for \((B+E=) 24\) broadcasts. As observed in Figure 4.12 the performance degrades more rapidly in all the access methods except in the modified progression.

![Figure 4.11: The tuning time as a function of \( q \).](image)

Experiment 4: (Data Miss)

The effect of \( q \) for the average tuning time in a continuous process (when the user remains in-system) is shown in Figure 4.13. This average is computed when a user enters into a broadcast as a new user and remains there for \((B+E=) 27\) broadcasts. As observed in Figure 4.13 the performance degrades more rapidly in all the access methods except in the modified progression. Let us assume that the user
avails all the $B (= 18)$ broadcasts of a version number (same $DSS$ bits but may have different $SSS$ bits) in a continuous scheme.

4.12: Average tuning time as a function of $q$ in continuous process.

Figure 4.13: Average tuning time as a function of $q$ in continuous access without disconnection.
It is observed from the Figure 4.14 that the average tuning time at the end of $nB^{th}$ broadcast ($N$ is a positive integer) is minimum whereas it rises for the broadcasts $(nB + 1), (nB + 2), \ldots, \text{and } (nB + [b])$, 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 $b$. As seen in Figure 4.14 average tuning time attains its maximum (for each cycle of $B$ broadcasts) at $(nB + [b])^{th}$ broadcast. When the user continues its access of data for long time i.e., when $N$ is large, the average tuning time at $(nB + [b])^{th}$ broadcast approaches to the average tuning time at the $nB^{th}$ broadcast.

![Graph showing average tuning time vs number of broadcasts]

**Figure 4.14: Comparison of average tuning with respect to number of broadcasts in continuous access**

**Experiment 5:**

In the previous four experiments, it has been seen that the modified progression provides finer results in terms of tuning time and broadcasts with respect to $q$. 

115
Now in this experiment, we consider the effect of $r$ for modified progression method. As shown in Figure 4.15 and 4.16, we observe the effect of $r$ for the broadcasts (in both the cases i.e., for data miss as well as for no data miss) at $q = 0.1, 0.25, \text{ and } 0.4$. The tolerance of modified method increases as $r$ increases and its performance in terms of broadcasts approaches to that of Ideal i.e., it exploits less number of 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 larger, whereas 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.

![Figure 4.15: The number of broadcasts as a function of $r$ (no data miss)](image-url)
4.7 Conclusion

In this chapter, we investigated the access methods for the indexed data broadcasting in an error-prone mobile environment. Four methods reaccess, modified reaccess, progression and modified progression, based upon the EPR scheme, are presented. In the 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 number of broadcasts used) and the tuning time. In the wireless medium, occurrence of access failures tends to affect the data. Therefore, we make use of the previous search result in a progression method and the index replicate available to shorten the time for recovering from an error. In this method, 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 only $SSS$ bits (and retain previous $DSS$ bits). We also demonstrated 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 of data so that the 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, 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 average tuning time.