Chapter 6

6. Cache Management and Replacement

6.1 Cache Memory Organization

The storage resources in various computing systems are organized based on different levels of hierarchy as shown in Fig. 6.1. This is termed as memory hierarchy [37]. They are of different kinds and sizes. The top level of the hierarchy is the fastest (least access latency) memory module present in the processing unit. But it has the smallest size. The lower levels of memory hierarchy have increasing sizes and correspondingly decreasing access latencies. When data is requested by an application of a client, the memory at highest level in CPU services the request. In a mobile environment, the request is translated to the mobile client cache then followed by disk present in the client and finally towards the server cache followed by disk of server. In push environment, broadcast behaves as an intermediate level between the client and the server. If an application fails to locate a page locally in its cache then it searches in disk. If it is still not available there then it searches in the broadcast multi-level storage to fulfill the request. If the page is not available, then an explicit request is given through the uplink channel.
**Issues in Cache Management**

We begin this section of the chapter by discussing the issues of client resources namely Client Cache and managing it in a multicast data dissemination environment. Caching is a popular strategy in mobile environments to improve performance and scalability. Several advantages of Caching are system performance enhancement due to the data storage done very close to the application (client cache) and minimizing request response time. Two forms of Caching are discussed below:

(i) *on-demand Caching*(ODC) – If the data items are brought into the cache whenever there is a request from the client then it is referred to as ODC. There is always a cache miss in this method while the items are first accessed.

(ii) *PreCaching*(PC) – In this form the data items are prefetched into the cache well in advance in an anticipation that they may be required in near future. The pre-caching of data items is a good method to access data with almost no delay. But it has the limitation of over-utilizing the cache that may appear a burden on client in the form of wastage of cache space. The most significant feature of a caching method is its selection of page for replacement to make a room for the new data-item.
Fig. 6.1 Memory Hierarchy
6.2 Semantic Neural Network based Cache Replacement Policy

The development of high speed wireless network and the increasing use of portable wireless devices have led to the development of mobile computing. The ability to move and know your own location has given rise to services known as Location Information Services. A Location based data is the data whose value depends on the current location of the mobile user[38]. The queries used to process these kind of data is known as Location based queries. Location based querying is gaining significant importance in mobile computing and communication systems. Mobile users in wireless communications normally face several difficulties like low bandwidth, frequent network disconnection and they resort to most common solution namely data caching [37]. When the mobile user cache gets full then data items from cache has to be removed to accommodate new items [24]. This process is known as cache replacement [23].

We propose a hybrid cache replacement scheme that supports both the temporal and location querying. It is based on Semantic data caching. The basic idea of semantic caching is that the information about the data should be stored along with the data in the cache[3]. For improving the response time of query, several semantic caching models have been proposed.

Whenever a new location-based-query is generated, the system checks whether it can be answered with the information available from
within the cache. If yes, then the answer is then and there. If no, then if the query is partially answerable then the only a trimmed part of query is sent to the server. Semantic caching this way helps to reduce the network load by decreasing the amount of data that is transferred over the network.

In this chapter we study a semantic caching model that is suitable for the client population within a network to improve the response time of the query and thus minimize the traffic on network and also the response time. If the clients on the network run the CRP that is proposed in this section it can be proved that the performance is increased as shown in the concluding part of this chapter. The definition of semantic segment that was proposed previously is enhanced to accommodate a new dimension Segment Frequency (Sn) and time factor corresponding to the predicted update interval (PUI) for the data items present in the cache. These factors play an important role when the decision has to be taken for which item has to be replaced in cache to accommodate a new item. This cache replacement policy also predicts the clients future movement based on velocity. In this chapter we discuss the existing cache replacement polices like Furthest Away Replacement (FAR) where the data item that is furthest is away from cache is removed first form cache and a policy that is an improvement of FAR-CRP known as RBF-FAR. RBF_FAR predicts the client’s future location using a RBF(Radial Basis Far) Network[37]. Motivated by these CRPs we have
given a new scheme which an improvement on existing RBF-FAR scheme known as Enhanced RBF-FAR (SNN-CRP).

In this chapter, we first introduce the semantic caching model. This model uses the select and projects queries on a Database D, which has Relations \( R_1, R_2, \ldots, R_n \), i.e., \( D = \{ R_i, 1 \leq i \leq n \} \) [38]. An Attribute set A is also defined where \( A = UAR_i, 1 \leq i \leq n \) [38]. With this, we define the term semantic segment and query.

**Semantic Segment:** Given a database \( DB = \{ R_i, 1 \leq i \leq n \} \) and its attribute set \( A = UAR_i, 1 \leq i \leq n \), a semantic segment \( S \) [38], is a tuple \(<SR, SA, SP, S_m, Sc, ST>\), where \( Sc = (\Pi (SA)) \sigma SP (SR) \), \( SR \in DB \), \( SA \in ASR \) and \( SP \) indicates the select condition that the tuple in \( S \) satisfy, \( SC \) represents the actual content of \( S \). The \( S_m \) represents the MU user coordinate that helps to predict future location of moving objects and used to new replacement policy. \( ST \) is the timestamp for semantic segment.

**Query:** A Query \( Q \) is a semantic segment, \(<QR, QA, QP, Qm, QC>\) [37]. The Parameter \( ST \) is removed as this parameter is only used for taking replacement decision.

The storage of segments in the memory is done by a technique called paging [23]. Every page has a segment which connects to other pages and has a variable which holds the address and refers to the starting first page. An index is used to structure the cache memory in an organized manner which maintains the semantic descriptions and
storage information for each segment in the cache. An example structure of index can be shown by taking two relations: Hospital (Hno, Hname, Hx, Hy), Petrol Pump ((Pno, Pname, Px, Py). We assume that the mobile user gives queries at different locations on his way. The current location of mobile user is designated using M(x, y). In the following section we consider the case of two queries Q1, Q2 respectively,

Q1 - MU (30, 40): Show all hospitals within 25 km.
Q2 - MU (10, 30): Show all Petrol pumps within 7 km.

<table>
<thead>
<tr>
<th>S</th>
<th>SR</th>
<th>SP</th>
<th>Sm</th>
<th>Sc</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Hospital</td>
<td>(MUx-25 &lt;= Hx &lt;= MUx+25) ^ (MUy-25 &lt;= Hy &lt;= MUy+25)</td>
<td>(30, 40)</td>
<td>2</td>
<td>T1</td>
</tr>
<tr>
<td>S2</td>
<td>Petrol Pumps</td>
<td>(MUx-7 &lt;= Px &lt;= MUx+7) ^ (MUy-7 &lt;= Py &lt;= MUy+7)</td>
<td>(10, 30)</td>
<td>5</td>
<td>T2</td>
</tr>
</tbody>
</table>

**Table 6.1 An Example of semantic cache index**

### 6.2.1 Query Processing

Whenever a query is issued, we first check whether it can be answered by the cache. If yes, then the results are then and there. If the query can only be partially answered, we trim the original query by removing the presently answered parts and send it to the server for processing. The issue related processing query is the development of semantic segment. To prevent repeated data we cannot keep S and Q in
the cache. Whenever a query is issued, if it overlaps a semantic segment in cache different techniques can be used to evaluate them. The no coalescing approach generates three disjoint parts: $(Q \cap S)$, $(S \setminus Q)$ and $(\neg S \setminus Q)$ [24]. Problem with this approach is that it may result in a many smaller segments. Complete coalescing approach is better for small queries but not for replacement [38]. The “partial coalescence in query” technique is the best [23]. The segment is decomposed in two parts: the overlapped part and the no overlapped part [37]. In this paper, we use the “partial coalescence in query” technique [37].

6.2.2 Cache Replacement Policies

In this chapter we will discuss the existing replacement policies and the existing prediction architecture.

Furthest Away Replacement (FAR) policy is an existing policy which replaces the items which is furthest away from mobile client’s present location. It also predicts the mobile client’s future location using velocity. However, the FAR has its own problems which can be demonstrated by the following figure.
In the figure above, Seg-1, Seg-2, Seg-3, Seg-4, are semantic segments. Seg-5 and Seg-6 are assuming to be future semantic segments. We assume that, the Present timestamp is T4 of Seg-4, and T1<T2<T3<T4. Seg-5 or Seg-6 will be the semantic segment accessed if a new query is issued. When FAR algorithm is used, in Seg-4, the result that we get should be stored in cache but the space is not enough to store the result. Hence we have to remove an item from cache to accommodate the new item. Using FAR algorithm, the future location would be Seg-5 based on current velocity of Seg-4. So FAR replacement policy will remove Seg-1. But, if the mobile user changes its direction to Seg-6 half way then the next query will be generated on Seg-6 rather than Seg-5. Hence the Seg-1 which was removed was useful and should not have been removed. This was the result of inaccurate prediction by FAR algorithm.

In this section a hybrid CRP that uses new prediction model is developed. It uses RBF-Network to predict mobile user’s future location
[23] so that the mobile client take the user’s location along with predicted update time interval into account while performing the replacement. The RBFNN is self learning model which takes the current location of mobile user $M(x, y)$ using the $S_m$ of Semantic segment and the timestamp $S_t$ as input to the prediction architecture which gives the future location of mobile user $M_f(x, y)$. The algorithm was called RBF-FAR [38]. It uses RBF-Network to predict future location and based on that FAR replacement policy is used to replace the items in cache. The prediction architecture can be illustrated through the figure below.

![Prediction Architecture](image)

**Fig 6.3 Prediction Architecture**

The whole description of how the RBF-FAR Replacement policy works is given by the following algorithm [38].
1: Algorithm RBF-FAR(C, M)
2: {
3: In-Direction ← NULL;
4: Out-Direction ← NULL;
5: Call RBFNN to predicate location;
6: Mfl= (x-predicated, y-predicated);
7: for every segment seg in C
8: {
9: if Distance(segL, MfL) ≤ Distance(segL, ML)
10: then In-Direction ← In-Direction + {seg};
11: else Out-Direction ← Out-Direction + {seg};
12: }
13: while (Out-Direction != Empty)
14: {
15: seg ← the segment in Out-Direction which is the furthest from M;
16: discard seg from C;
17: remove seg from Out-Direction;
18: add free space;
19: if (free space is enough)
20: return (Success);
21: }

while ( In-Direction != Empty )
{
    seg ← the segment in In-Direction which is the furthest from M;
    discard seg from C;
    remove seg from In-Direction;
    add free space;
    if ( free space is enough )
        return (Success);
}
return (Fail);

The RBF-FAR algorithm should be trained for good performance. However if the training data is not sufficient then the RBFNN will give a lower performance [11].

6.2.3 **ERBF-FAR Scheme**

The existing RBF-FAR scheme needs to be trained for good performances and requires good amount of training data to perform well. The proposed ERBF-FAR scheme extends the RBF-FAR scheme for acquiring good performance. A new dimension $S_f$ is added to the existing semantic segment which is used for cache replacement decision. The
future location of the neighboring mobile users is given as input to the RBF-Network which improves the quality of prediction. The improvements in proposed scheme are discussed one by one in this chapter.

**Semantic Segment:** Given a database $D = \{R_i, 1 \leq i \leq n\}$ and its attribute set $A = \cup R_i, 1 \leq i \leq n$, a semantic segment, $S$, is a tuple $<SR, SA, SP, S_m, SC, ST, S_f>$, where $Sc = \Pi SAoSp(SR)$, $SR \in D$, $SA \in ASR$ and $SP$ indicates the select condition that the tuple in $S$ satisfy, $SC$ represents the actual content of $S$. The $S_m$ represents the MU user coordinate that helps to predict future locations of moving objects and used to new replacement policy. $ST$ is the timestamp for semantic segment; $S_f$ is the frequency of the semantic segment.

**Replacement Policy:** The cache replacement policy takes the newly proposed dimension $S_f$ into consideration for replacing an item from cache. The $S_f$ denotes number of time the semantic segment was accessed. Whenever a new query results are obtained then an item in the cache has to be replaced to accommodate new results if the cache is full. According to FAR the items that are furthest away from current location should be removed. However In the proposed scheme we take the average of value of $S_f$ of all semantic segment called $S_{favg}$. When the cache replacement decision is to be taken then we remove only the segment which has $S_f$ less than $S_{favg}$ and furthest away from current location. If a
data item is furthest away from current location but its $S_f$ is greater than or equal to average $S_f$ then its should be removed from cache. The immediate next segment which is less further but has $S_f$ less than $S_{favg}$ should be removed. If all the items in cache have $S_f$ greater then $S_{favg}$ then the item with lowest $S_f$ value should be removed.

![Fig 6.4 ERBF-FAR Cache Replacement Scheme](image)

Fig 6.4 ERBF-FAR Cache Replacement Scheme

In the above figure, a new query is issued at Seg-4 and the results are obtained. We assume the cache is full so we need to replace an item from cache. Seg-1 is the farthest away segment from Seg-4 but it has $S_f$ value 5 which is greater that average frequency 4. So the next farthest segment Seg-2 is removed as it has $S_f$ less than average frequency.

**Prediction Architecture:** The prediction architecture in the proposed system takes into account the future locations of neighboring mobile clients. The proposed prediction architecture is called Enhanced RBFNN (ERBFNN) which is an extension to RBFNN. The proposed prediction
architecture will give improved performance. A new mobile user will not have any training data based on which prediction can be done. So training data can be taken from neighboring mobile clients who have good training sets and future locations can be accurately predicted.

![Proposed Prediction Architecture](image)

**Fig 6.5 Proposed Prediction Architecture**

The whole description of how the ERBF-FAR Replacement policy works is given by the following algorithm.

1: Algorithm ERBF-FAR(C, M)

2: 

3: In-Direction ←NULL;

4: Out-Direction ←NULL;

5: Call ERBFNN to predicate location;
6: Mfl= (x-predicted, y-predicted);

7: for every segment seg in C

8: {

9: if Distance(segL, Mfl) ≤ Distance(segL, ML)
10: then In-Direction ← In-Direction +\{seg\};
11: else Out-Direction ← Out-Direction +\{seg\};
12: }

13: Favgi = average $S_f$ of all segments in In-Direction.
14: Favgo = average $S_f$ of all segments in Out-Direction.
15: while ( Out-Direction != Empty )
16: {

17: seg ← the segment in Out-Direction which is
18: the furthest from M;
19: while ( Out-Direction != Empty )
20: {

21: if(seg.$S_f$<Favgo)
22: {

23: discard seg from C;
24: remove seg from Out-Direction;
25: add free space;
26: if ( free space is enough )
27: return (Success);
28: }
29: }
30: }


29:   Else
30:   {
31:   Move to next furthest segment in Out-Direction
32:   Repeat 19.
33:   }
34:   }
35:   Seg ← segment with lowest $S_f$ in Out-Direction.
36:   Goto step 23
37:   }
38:   while ( In-Direction != Empty )
39:   {
40:   seg ← the segment in In-Direction furthest from $M$;
41:   while ( In-Direction != Empty )
42:   {
43:   if(seg.$S_f$<Favg)$
44:   {
45:   discard seg from C;
46:   remove seg from In-Direction;
47:   add free space;
48:   if ( free space is enough )
49:   return (Success);
50:   }
51:   Else
In this algorithm, if the mobile user moves in a well known location the replacement technique has a good performance. If user M moves to a new area, the FAR and RBF-FAR will be low effective whereas ERBF-FAR is more effective as it takes neighbors into consideration for future location prediction.

### 6.3. Experimental Results

The design of our simulation scenario consists of one server, one mobile user and a wireless channel between them to communicate. The mobile user issues queries and the server responds to the queries by maintaining a database containing information to serve the mobile user. We consider that the mobile client has one of the three cache
replacement policies i.e., FAR, RBF-FAR and ERBF-FAR. In the mobile client simulation scenario, we have LDQ which is generated on the basis of workload, A Semantic cache manager which manages the semantic cache memory and a semantic cache query processor which processes the queries. The system parameters are displayed in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUProcessor</td>
<td>Mobile user CPU speed</td>
<td>500(Mips)</td>
</tr>
<tr>
<td>MUCach</td>
<td>Mobile client cache size(KB)</td>
<td>512</td>
</tr>
<tr>
<td>BW</td>
<td>Wireless channel bandwidth</td>
<td>32 K</td>
</tr>
<tr>
<td>LenPage</td>
<td>Size of the data page(bytes)</td>
<td>4096</td>
</tr>
<tr>
<td>MsgFC</td>
<td>Fixed part protocol cost to send/receive message</td>
<td>10000</td>
</tr>
<tr>
<td>MsgPB</td>
<td>Size-dependent part protocol cost to S/R message</td>
<td>500</td>
</tr>
<tr>
<td>AttrSel</td>
<td>The attributes to be queried</td>
<td>AX,AY</td>
</tr>
<tr>
<td>QSel</td>
<td>Query selection</td>
<td>4 sets</td>
</tr>
</tbody>
</table>

**Table 6.2 Parameter Setting**

The working design of the simulation scenario made up of three relations. One relation is a 1000 set and other two relations are 2000 set. We take two important attributes AX and AY. AX is indexed and unclustered and AY is indexed and clustered. We use select queries to generate LDQ’s. We use location as predicate. AttrSel is used to specify
attributes upon which query is generated and QSel is used to specify the query.

The queries are generated using five random sets consisting of 1000 LDQ’s. In Each set, the starting 100 queries are used as warm-up data and the rest 900 queries are used as test data. The LDQ’s are well defined. The average time and network load is tested to show the effectiveness and efficiency of the model.

![Average response time vs well-defined paths](image)

**Fig 6.6 Average response time vs well-defined paths**
The experimental results show that the semantic cache has less average response time than the traditional cache. The ERBF-FAR gives a good average response time performance than others. The network load has increased in the proposed system as the communication overhead has increased due to high communication with neighboring mobile clients for accessing their future locations.

We also define LDQ’s where mobile user is not well defined which means that the mobile user moves to a new location every time which it has previously not visited. The ERBF-FAR gives a good performance as it incorporates the future location of neighboring mobile users which increases the quality of future location prediction. The proposed scheme gives a better average response time than other schemes and the network load is increased due to communication overhead. But still it’s sacrificed due good quality of prediction.
The main characteristics of a good cache replacement policy are that it will maintain a high cache hit ratio to improve the system performance. The ERBF-FAR shows an improved performance over other
cache replacement policy and gives a high cache hit ratio. The addition of factor $S_t$ to the semantic segment and its influence on the cache replacement decision has led to the increased cache hit ratio. The results are shown as following.

![Graph showing cache hit ratio vs. cache sizes](image)

**Fig 6.10 Cache hit ratio vs. cache sizes**

Hence the ERBF-FAR shows an improved performance and high cache hit ratio and is twice better than RBF-FAR and thrice better than FAR.

In this chapter, we have discussed various cache management issues for location dependent data. We also defined the existing semantic segment, query and semantic segment index. The existing cache replacement policies were also discussed. The FAR cache replacement policy replaces the furthest away item and uses velocity for prediction of future location. The RBF-FAR uses FAR policy for replacement and RBF-Network for prediction of future location. The Proposed ERBF-FAR
scheme proposes a new semantic segment which adds a new dimension Segment frequency ($S_i$) which is used for taking cache replacement decision. The prediction architecture is improved by adding future location of neighboring mobile users as input to the RBF-Network which gives an improved performance.

However, the introduction of future location of neighboring mobile users as increased the network load but has improved the quality of future location prediction. Hence, the Experimental results shows that, ERBF-FAR shows an improved performance over existing policies and gives better average response time and cache hit ratio.