Volume: 05, Issue: 03, Pages:1928-1938 (2013) ISSN : 0975-0290

Data Retrieval Mechanisms in Mobile Environments by Using Spatial Queries

C. Gopala Krishnan
Assistant Professor Computer Science and Engineering
Email:gopckrish@yahoo.com
R.Suceendrakumar
Department of Computer Science, Sree sastha institute of Engg, Tamil Nadu
kumarcsse@gmail.com

---ABSTRACT---
This work discusses a novel hybrid replacement policy to adopt in the design of our cache. According to our hybrid caching strategy, the results of the most frequently accessed queries are maintained in a static cache of fixed size, which is completely rebuilt at fixed time intervals. Only the queries that cannot be satisfied by the static cache compete for the use of a dynamic cache. Our hybrid cache represents an effective and fast way to address both *recency*, and *frequency* of occurrences criteria. While the static cache maintains results of queries that are globally frequent, a simple and fast policy like LRU, which only takes into account query reference recency,could be adopted for the dynamic cache. This novel representation and the associated creation algorithm result in more effective EVRs of window queries. In addition, due to the distinct characteristics, a separate index structures, namely EVR-tree and grid index, for NN queries and window queries, respectively are used. To further increase efficiency, an algorithms to exploit the results of NN queries to aid grid index growth, benefiting EWV creation of window queries is developed. Similarly, the grid index is utilized to support NN query answering and EVR updating. Several experiments for performance evaluation are performed. The experimental results show that the proposed approach significantly outperforms the existing proxy-based approaches.

Keywords- **Ad hoc networks, Cooperative cache, Cache management, Cache replacement policy, Simulations. Nearest neighbor query, window query, spatial query processing, location-based service,**

Date of Submission: November 7, 2013 Date of Acceptance: December 17, 2013
----

1. INTRODUCTION

Broadcast query processing is becoming an integral part of many new mobile applications. Recently, there has been a growing interest in the use of broadcast queries (BQs), which represent a set of queries that retrieve information based on mobile users' current locations [1], [2]. Novel query processing techniques must be devised to handle the following new challenges:

1. **Mobile Query Semantics** In a mobile environment, a typical BQ is of the form "*find the top-three nearest hospitals*." The result of the query depends on the location of its requester.
Caching and sharing of query results must take into consideration the location of the query issuer

2. **Heavy server load.** The database resides in a centralized server, which typically serves a large mobile user community through wireless communication. Consequently, bandwidth constraints and scalability become the two most important design concerns of BQ algorithms [2].

3. **Query promptness and accuracy.** Due to user mobility, answer to an BQ will lose their relevancy if there is a long delay in query processing or in communication. For example, answers to the query "*find the top-three nearest hospitals*" received after 5 minutes of high-speed driving will become meaningless. Instead, a prompt, albeit approximate, top-three nearest hospitals, may serve the user much better. This is an importance issue, as a long latency in a high workload wireless environment is not unusual.

The wireless environment and the communication constraints play an important role in determining the strategy for processing BQs. In the simplest approach, a user establishes a point-to-point communication with the server so that the user queries can be answered on demand. However, this approach suffers from several drawbacks. First, it may not scale to very large user populations. Second, to communicate with the server, a client must most likely use a fee-based cellular-type network to achieve a reasonable operating range. Third, users must reveal their current location and send it to the server, which may be undesirable for privacy reasons [12]. A more advanced solution is the wireless broadcast model [1], [9], [18]. It can support an almost-unlimited number of mobile hosts (MHs) over a large geographical area with a single transmitter. With the broadcast model, MHs do not submit queries. Instead, they tune in to the broadcast channel for information that they desire. Hence, the user's location is not revealed. One of the limitations of the broadcast model is that it restricts data access to be sequential. Queries can only be fulfilled after all the required on-air data arrives. This is why in some cases, a 5-minute delay to the query "find the top-three nearest hospitals" would not be unusual.

Alleviating this limitation, we propose a scalable low-latency approach for processing BQs in broadcast environments. Our approach leverages ad hoc networks to share information among mobile clients in a peer-to -peer (P2P) manner [10], [11].

The rationale behind our approach is based on the following observations:

- As mentioned previously, when a mobile user launches a nearest neighbor (NN) query, in many situations, the user would prefer an approximate result that arrives with a short response time rather than an accurate result with a long latency.
- The results of spatial queries often exhibit spatial locality. For example, if two MHs are close to each other, the result sets of their spatial queries may overlap significantly. Query results of a mobile peer are valuable for two reasons: 1) they can be used to answer queries of the current MH directly and 2) they can be used to dramatically reduce the latency for the current MH relative to on-air information.
- P2P approach can be valuable for applications where the response time is an important concern. Through mobile cooperative caching[5] of the result sets,query results can be efficiently shared among mobile clients.

BQs concentrate on two common types of Query searches, namely, kNN queries and window
queries (WQs). The contributions of existing study are given as follows:

1. Identify certain characteristics of BQs that enable the development of effective sharing methods in broadcast environments. We introduce a set of algorithms that verify whether data received from neighboring
 clients are complete, partial, or irrelevant
 answers to the posed query.

2. Utilize a P2P-based sharing method to improve the current approaches in answering on-air kNN queries and WQs.

3. Evaluate BQ approach through a probabilistic analysis of the hit ratio in sharing. In addition, through extensive simulation experiments, we evaluate the benefits of our approach with different parameter sets.

2. WIRELESS DATA BROADCAST

In general, there are two approaches for mobile data access. One is the *on-demand access model*, and the other is the *wireless broadcast model*. For the on-demand access model, point-to-point connections are established between the server and the mobile clients, and the server processes queries that the clients submit on demand. For the wireless broadcast model, the server repeatedly broadcasts all the information in wireless channels, and the clients are responsible for filtering the information. The advantage of the broadcast model over the on-demand model is that it is a scalable approach. However, the broadcast model has large latency, as clients have to wait for the next broadcast cycle broadcasting cycle. If a client misses the packets that it needs, it has to wait for the next broadcast cycle.

To facilitate information retrieval on wireless broadcast channels, the server usually transmits an index structure, along with data objects. A well-known broadcast index structure is the (1, m)[9] indexing allocation method . As we can see in Fig. 1, the whole index is broadcast preceding every 1/m fraction of the data file. Because the index is available m times in one cycle, it allows a mobile client easy access to the index so that it can predict the arrival time of its desired data in a timely manner, and once it knows the arrival time, it only needs to tune into the broadcast channel when the data bucket arrives. This mechanism is important for battery-based devices.

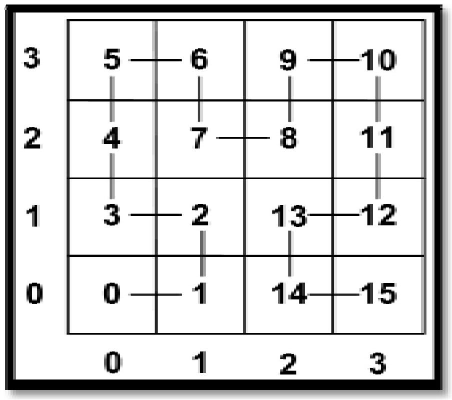


Fig. 1. The Hilbert-curve-based index structure. The numbers represent index values.

Thus, the general access protocol for retrieving data on a wireless broadcast channel involves three main steps[15] :

- The initial probe. A client tunes in to the broadcast channel and determines when the next index segment will be broadcast.
- Index search. The client accesses a sequence of pointers in the index segment to figure out when to tune in to the broadcast channel to retrieve the required data.
- Data retrieval. The client tunes in to the channel when packets containing the required data arrive and then downloads all the required information.

However, nearly all the existing spatial access methods are designed for databases with random access disks.These existing techniques cannot be used effectively in a wireless broadcast environment, where only sequential data access is supported. Zheng et al. [19] proposed indexing the spatial data on the server by a space-filling curve. The Hilbert curve [20] was chosen for this purpose because of its superior locality. The index values of the data packets represent the order in which these data packets are
broadcast. For example, the Hilbert curve in Fig. 1 tries grouping data of close values so that they can be accessed within a short interval when they are broadcast sequentially. The MHs use on-air search algorithms [19] to answer LBSQs (kNN and WQs) over data that arrives in the order prescribed by the Hilbert curve.

2.1 SPATIAL QUERIES

Existing work focus on two common types of spatial queries, namely, kNNqueries and WQs[17]. With R-tree-based [15] spatial indices, depth-first search (DFS) and best first search (BFS)[8] have been the prevalent branch-and-bound techniques for processing NN queries. The DFS method recursively expands the index nodes for searching NN candidates. At each newly visited nonleaf node, DFS computes the ordering metrics for all its child nodes and applies pruning strategies to remove unnecessary branches. When a leaf node is reached, the data objects are retrieved, and the NN candidates are updated. Comparatively, the BFS technique utilizes a priority queue to store nodes to be explored through the search process. The nodes in the queue are sorted according to their minimum distance (MINDIST) to the query point. During the search process, BFS repeatedly dequeues the top entry in the queue and enqueues its child nodes with their MINDIST into the queue. When a data entry is dequeued, it is inserted into the result set. For WQs that find objects within a specified area, the R-tree families[3],[16] provide efficient access to disk-based databases. Basically, an R-tree structure groups objects close to each other into a minimum bounding rectangle (MBR), and a range query only visits MBRs that overlap with the query area.

2.2 COOPERATIVE CACHING

Caching is a key technique to improve data retrieval performance in widely distributed environments Hara and Madria proposed three data replica allocation methods in ad hoc networks by considering the access frequency from MHs to each data item and the status of the network connection With the increasing deployment of new P2P wireless communication technologies, P2P cooperative caching becomes an effective sharing alternative With this technique,[4],[7].MHs communicate with neighboring peers in an ad hoc manner for information sharing instead of relying solely on the communication between remote information sources .

P2P cooperative caching can bring about several distinctive benefits to a mobile system: improved access latency, reduced server workload, and alleviated point-to-point channel congestion. In this research, we leverage the P2P caching technique to alleviate the inherent access latency limitation in wireless broadcast environments.

2.3 SHARING-BASED NN QUERIES

Fig. 2 shows an example of an on-air kNN query based on the Hilbert curve index structure [19]. At first, by scanning the on-air index, the k-nearest object to the query point is found, and a minimal circle centered at q and containing all those k objects is constructed. The MBR of that circle, enclosing at least k objects, serves as the search range. Consequently, q has to receive the data packets that covers the MBR from the broadcast channel for retrieving its k-nearest objects. As shown in Fig. 2, the related packets span a long segment in the index sequence, that is, between 5 and 58, which will require a long retrieval time. The other problem with this search algorithm is that the indexing information has to be replicated in the broadcast cycle to enable twice scanning. The first scan is for deciding the kNN search range, and the second scan is for retrieving k objects based on the search range [19].
Therefore, the Sharing-Based Nearest Neighbor (SBNN) query approach to improve the preceding on-air kNN query algorithm. The SBNN algorithm attempts to verify the validity of k objects by processing results obtained from several peers..

3. OVERVIEW

The wireless data broadcast model has good scalability for supporting an almost-unlimited number of clients[9] .Its main limitation lies in its sequential data access: the access latency becomes longer as the number of data items increases. If we can provide (approximate) answers to spatial queries before the arrival of related data packets, we will overcome the limitation of the broadcast model.

A novel component in the methodology is a verification algorithm that verifies whether a data item from neighboring peers is part of the solution set to a spatial query. Even if the verified results constitute only part of the solution set, in which case the query client needs to wait for the required data packets to get the remaining answers, the partial answer can be utilized by many applications that do not need exact solutions but require a short response time (for example, the query “What are the top-three nearest hospitals?” issued by a motorist on a highway).

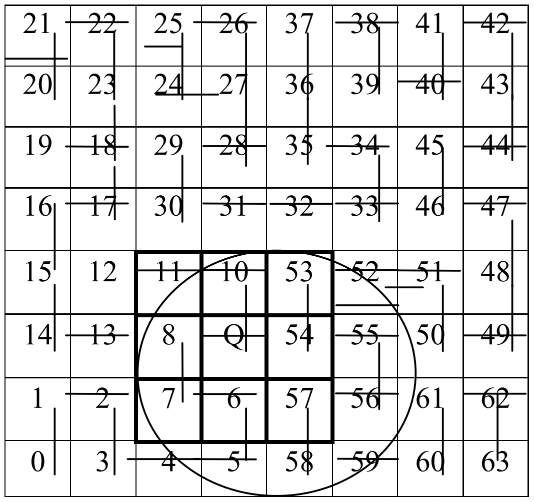


Fig. 2. An on-air kNN query example. The numbers represent index values

3.1NEAREST NEIGHBOR VERIFICATION

When an MH q executes SBNN, it first broadcasts a request to all its single-hop peers for their cached spatial data. Each peer that receives the request returns the verified region MBR and the cached POIs to q. Then, q combines the verified regions of all the replying peers, each bounded by its MBR, into a merged verified region MVR (the polygon in Fig. 3). The merging process is carried out by the MapOverlay algorithm [6] (line 4 in Algorithm 1). The core of SBNN is the NN verification (NNV) method, whose objective is to verify whether a POI oi obtained from peers is a valid (that is, the top-k) NN of the MH q.

Let IP denote the data collected by q from j peers pi,… pj. Consequently, the merged verified region MVR can be represented as

MVR =p1.VR U p2.VR U ….. U pj VR

Suppose that the boundary of MVR consists of k edges, IE ={e1,e2, . . . ek}, and there are l POIs,

={o1, o2,…..ol} inside the MVR. Let es ϵ IE be the edge that has the shortest distance to q. An example is given in Fig. 3, where k=10, and e1 has the shortest distance to q.

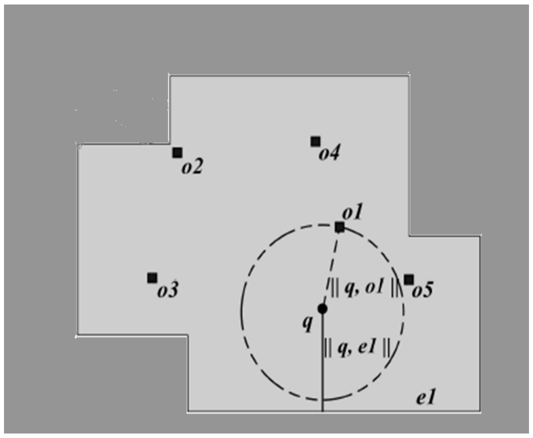


Fig. 3. Because e1 has the shortest distance to q and ||q,o1||<=||q,e1||,POI is verified as a valid NN of MHq.

The NNV method uses a heap H to maintain the entries of verified and unverified POIs.Initially, H is empty. The NNV method inserts POIs to H as it verifies objects from MHs in the vicinity of q. The heap H maintains the POIs in ascending order in terms of their euclidean distances to q. nverified objects are kept in H only if the number of verified objects is lower than what was requested by the query. The NNV method is formalized in Algorithm 1. Since the verified-region merging process dominates the algorithm complexity, the NNV method can be computed in O(n log n+i log n) time,
where n is the total edge number of the two merged polygons, and i is the number of intersection points.

Algorithm : NNV(q,H,k)
```
1:  P← peer nodes responding to the query request
    issued from q.
2:  MVR ←Φ
3:  for ∀p∈ IP do
4:  MVR U= p.V R and   U p.
5:  end for
6:  ∀oi ϵ   , sort according to ||q, oi||
7:  Compute ||q,es||, where edge es has the shortest
    distance to q among all the edges of MVR
8:  i= 1
9:  while |H| < k and i <= |   | do
10:     if ||q, oi||<=||q,es|| then
11:         H.verified U = oi
12:     else
13:         H.verified U = oi
14:         i++
15:     end if
16: end while
17: return H
```

If k elements in H are all verified by NNV, the kNN query is fulfilled. There will be cases when the NNVmethod cannot fulfill a kNN query. Hence, a set that contains unverified elements is returned. If the response time is critical, a user may agree to accept a kNN data set with unverified elements, where the objects are not guaranteed to be the top kNNs.However, the correctness of these approximate results can be estimated and will be discussed in the next section. If the result quality is the most important concern, the client has to wait until it receives all the required data packets from the broadcast channel. Nevertheless, the partial results in H can be used to decrease the required data packets and thus speed up the on-air data collection.

3.2 APPROXIMATE NEAREST NEIGHBOR

We calculate the probability that the unverified ith NN o of a query point q is actually the true ith NN of q. The reason that o cannot be verified is because there is a region that is not covered by q's neighboring peers. As long as a POI exists in the region, o cannot be q's ith NN. We denote such a region as o's unverified region. We assume that the POIs are Poisson distributed in our environment based on our experiments of several common POI types (gas stations, grocery stores, etc.) with chi-square tests [13], [14]. The probability of finding another POI in the unverified region Ui of an unverified POI oi can be calculated with respect to the area of Ui. We formulate the correctness of an unverified POI based on the probability model

3.3 NEAREST-NEIGHBOR QUERY PROCESSING

When receiving an NN query, a proxy first attempts to answer the query with the EVR-tree and the grid index. If the proxy cannot answer the query, it will submit one or two 2NN queries to the LBS server. The rationale of extending the received NN query into 2NN queries is that the distance between the nearest and second nearest objectsis useful for building the EVR of the nearest object based on the theoretical result of [26].1 By exploiting the objects returned by the LBS server, the proxy extends an existing or creates a new EVR. Besides, the returned objects are utilized to aid grid index growth, which will facilitate window query processing. For each NN query, the proxy provides the mobile client not only the answer object (the nearest object) but also an EVR for helping the client avoid subsequent queries. Note that these 2NN queries initially cause slightly heavier load on the LBS server, but they lead the proxy to provide effective EVRs for mobile clients efficiently. With the EVRs, the number of queries submitted by the clients and the load on the LBS server are reduced greatly. Note that given an object p, the EVR of p is denoted by EVR(p). The processing steps executed by the proxy are as follows:

Step 1. The proxy checks whether the NN query the location (xq, yq) is in any EVR of the EVR-tree by performing the general R-tree search operation since EVR-tree is an R-tree (or its variant). If so, go to Step 7.

Step 2. If not, the proxy attempts to answer query with the grid index. If the two nearest objects, say p1and p2, are found in the grid index,2 the proxy looks up the EVR-tree to see whether p1 is located in any
existing EVR. If so, go to step 6.

Step 3. The proxy extends the received NN query into a 2NN query with the same query location(xq, yq) and submits the 2NN query to the LBS server. Let p1 and p2 be the nearest and the second nearest objects to q, respectively. When receiving p1and p2 from the LBS server, the proxy searches the EVR-tree for EVR(p1). Meanwhile, the proxy runs algorithm GridIndexUpdate3 to update the grid index based on q, p1, and p2. If EV R(p1) is found ,go to Step 6.

Step 4. The proxy generates another 2NN query with query location (x1; y1) where (x1, y1) is the location of p1. Obviously, the nearest object (x1, y1) is p1.Let the second nearest object to ðx1; y1Þ be p3. The proxy runs algorithm EVR-Creation4 to create the EVR of p1 based on p1, p2, and p3. At the same time, the proxy runs algorithm Grid Index Update to update the grid index based on p1 and p3.

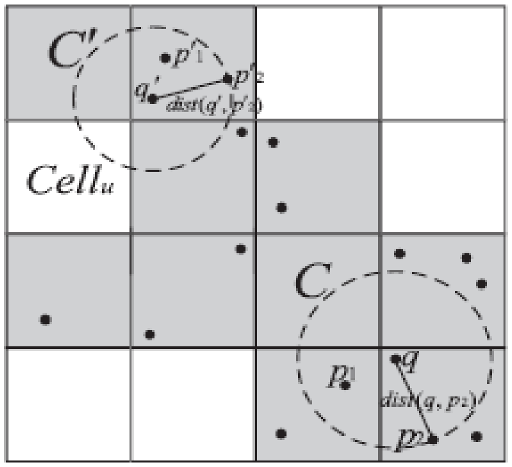
Step 5. The proxy inserts p1 and EVRðp1Þ into the object cache and the EVR-tree, respectively. Go to Step 7.

Step 6. With q, p1, and p2, the existing EV Rðp1Þ is extended using algorithm EVR-Extension.5 The updated EV R(p1) is reinserted into the EVR-tree.

Step 7. The proxy returns the answer object p1 and the corresponding EVR(p1) to the mobile client. It is worth mentioning that 1) the grid index for window query processing is helpful for resolving NN queries as well as extending existing EVRs, and 2) the returned answer objects of NN queries benefit grid index growth.

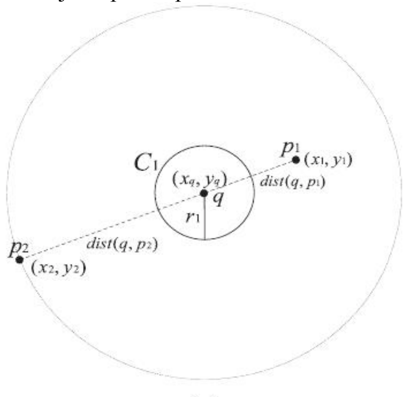
3.4 Resolving NN Queries by Grid Index

When the proxy cannot answer an NN query by the EVR tree, it attempts to exploit the grid index and cached objects to resolve the query. With the grid index, the proxy examines whether the query point q lies in a fully cached cell. If so, the proxy retrieves the nearest object p1 and the second nearest object p2 to q via the grid index. Besides p1, the proxy retrieves p2 since we intend to update EV R(p1) in case that EVR(p1) exists. With p1 and p2, the proxy calculates dist. (q, p2) and creates a circle C centered at q with dist.(q,p2) as the radius. Note that dist(p, q) denotes the Euclidean distance between any two given points p and q. If the circle C does not overlap any un cached cell, p1 and p2 can be guaranteed to be the real nearest and second nearest objects to q, as shown in Fig. 4. The proxy proceeds to examine whether p1 is located in any EVR of the EVR-tree. If EVRp1 is found, the proxy updates EV R(p1) by algorithm EVR-Extension with q, p1, and p2. Finally, the proxy returns p1 together with the updated EVR(p1) to the query client .Note that, if any un cached cell is involved, the NN query cannot be resolved by the grid index. For example, in Fig. 4, the circle C', centered at q' and with radius as dist.(q',p'),overlaps an un cached cell . An object may be located in the overlapping region of Cell and C' such that p_2 is not the real second nearest object q

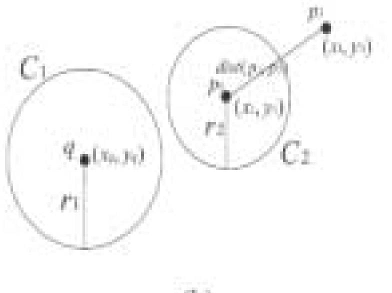


3.4 Updating Grid Index by Results of NN Queries

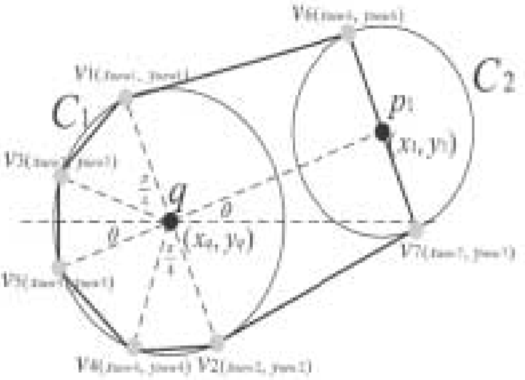
In addition to EVR creation, we exploit the answer objectsp_1 and p_2 to speed up grid index growth. We observe that concerning the 2NN query with query point q, there exist only two objects p_1 and p_2 within the circle C centered at q



(a)



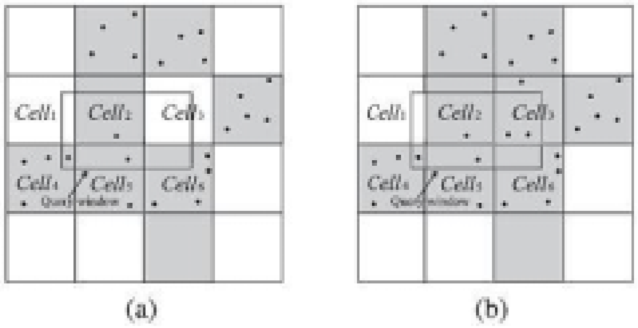
(b)



with dist.(q, p2) as the radius. Based on this observation, we attempt to take advantage of the answer objects and algorithm Grid Index Update to mark un cached cells as fully cached cells as follows: When the proxy receives p1 and p2, it uses the circle C to determine all totally overlapped cells of circle C, as shown in Fig. 6. A cell with respect to the circle C is referred to as a totally overlapped cell if the entire cell is within C. If a totally overlapped cell is uncached, this cell becomes fully cached, as the Celli in Fig. 6. By doing so, the proxy can accelerate grid index growth, resulting in the creation of more effective EWVs of window queries.

3.5 WINDOW QUERY PROCESSING

On receiving a window query, the proxy first checks whether all the answer objects of the query are available in the object cache by looking up the grid index. Answer objects are the objects within the query window. According to the grid



(a) Un cached Cell3. (b) Fully cached Cell3.

index, the proxy can know whether the grid cells overlapping the query window are fully cached. If a grid cell overlapping the query window is fully cached (e.g., $Cell_2$, $Cell_4$, $Cell_5$, and $Cell_6$ in Fig. 9a), the answer objects within the overlapping region can be directly retrieved from the object cache. Otherwise, the proxy uses the overlapping region of each un cached cell to generate a new window query (e.g., the overlapping regions of $Cell_1$ and $Cell_3$ in Fig. 9a). Then, the proxy submits the new window query to request the required answer objects from the LBS server. However, to accelerate fully cached cell growth, the proxy utilizes the corresponding entire cell as the new query window instead of the overlapping region if 1) the ratio of the region size to the cell size is above the predefined threshold, or 2) the number of interested queries in this cell exceeds the predefined threshold. For example, in Fig. 9a, the ratio of the overlapping region size to $Cell_3$ size is assumed to exceed the predefined threshold. The proxy sends a new window query with $Cell_3$ as the query window to the LBS server. Next, the proxy marks $Cell_3$ as fully cached when it receives all data objects in $Cell_3$ from the server, as shown in Fig. 9b. After obtaining the objects from the server, the proxy has all the answer objects. Finally, the proxy returns the answer objects together with the corresponding EWV to the query client. Note that these new queries used to accelerate grid index grow increase the load on the LBS server initially, but they lead to effective EWV creation, significantly reducing the number of subsequent queries submitted by clients

The processing steps executed by the proxy are as follows:

Step 1. The proxy identifies the overlapping cells based on the width w, the length l, and the center(xq, yq) of the query window.

Step 2. The proxy examines which of the overlapping cells being fully cached, searches the grid index for the object pointers within these overlapping fully cached cells, and then retrieves the answer objects from the object cache.

Step 3. For each unresolved overlapping region, the proxy sends a new query with either the region or the corresponding entire uncached cell as the query window to the LBS server.

Step 4. After receiving the objects from the LBS server, the proxy creates the EWV by algorithm EWV-Creation.6 Meanwhile, if all the objects in one cell are received, the proxy marks the cell in the grid index as fully cached and stores the objects into the object cache.

Step 5. The proxy returns the answer objects along with the corresponding EWV to the query client.

- **EWV Creation**
- **Extension to Range Queries**

4. PROPOSED BASIC COOPERATIVE CACHE SCHEMES

In this section, we propose two basic cooperative cache schemes and analyze their performance.

4.1 SYSTEM MODEL

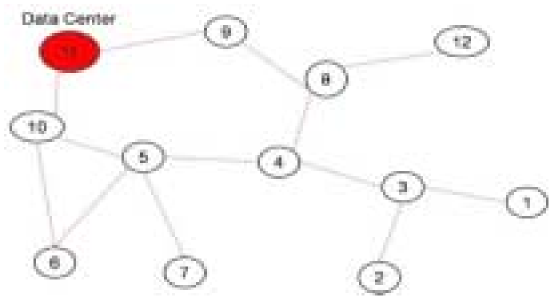


Fig 4.1 System Model

Fig 4.1 shows part of an ad hoc network. Some nodes in the ad hoc network may have wireless interfaces to connect to the wireless infrastructure such as wireless LAN or cellular networks. Suppose node N11 is a data source (center), which contains a database of n items d1; d2; . . . ; dn. Note that N11 may be a node connecting to the wired network which has the database. In ad hoc networks, a data request is forwarded hop-by hop until it reaches the data center and then the data center sends the requested data back. Various routing algorithms have been designed to route messages in ad hoc networks. To reduce the bandwidth consumption and the query delay, the number of hops between the data center and the requester should be as small as possible. Although routing protocols can be used to achieve this goal, there is a limitation on how much they can achieve. In the following, we propose two basic cooperative caching schemes: **Cache Data and Cache Path.**

4.1.1 CACHE THE DATA

In CacheData, the node caches a passing-by data item di locally when it finds that di is popular, i.e., there were many requests for di, or it has enough free cache space. For example, in Fig.4.1, both N6 and N7 request di through N5, N5 knows that di is popular and caches it locally. Future requests by N3, N4, or N5 can be served by N5 directly. Since CacheData needs extra space to save the data, it should be used prudently. Suppose the data center receives several requests for di forwarded by N3. Nodes along the path N3 _N4 _ N5 may all think that di is a popular item and should be cached. However, it wastes a large amount of cache space if three of them all cache di.

To avoid this, a conservative rule should be followed: A node does not cache the data if all requests for the data are from the same node. As in the previous example, all requests received by N5 are from N4, which in turn are from N3. With the new rule, N4 and N5 do not cache di. If the requests received by N3 are from different nodes such as N1 and N2, N3 will cache the data. If the requests all come from N1, N3 will not cache the data, but N1 will cache it. Certainly, if N5 receives requests for di from N6 and N7 later, it may also cache di. Note that di is at least cached at the requesting node, which can use it to serve the next query.

This conservative rule is designed to reduce the cache Space requirement. In some situations, e.g., when the cache size is very large or for some particular data that are interested by most nodes, the conservative rule may decrease the cache performance because data are not cached at every intermediate node.However, in mobile networks, nodes usually have limited cache spaces and we do not assume that some data are interested by all nodes. Therefore, the conservative rule is adopted in this paper.

4.1.2 CACHE THE DATA PATH

The idea of CachePath can be explained by using Fig. 4.1. Suppose node N1 has requested a data item di from N11. When N3 forwards the data di back to N1, N3 knows that N1 has a copy of di. Later, if N2 requests di, N3 knows that the data center N11 is three hops away whereas N1 is only one hop away. Thus, N3 forwards the request to N1 instead of N4. Note that many routing algorithms (such as AODV[and DSR provide the hop count information between the source and destination. By caching the data path for each data item, bandwidth and the query delay can be reduced since the data can be obtained through fewer number of hops. However, recording the map between data items and caching nodes increases routing overhead.

In the following, we propose some optimization techniques. When saving the path information, a node need not save all the node information along the path. Instead, it can save only the destination node information, as the path from current router to the destination can be found by the underlying routing algorithm. In CachePath, a node does not need to record the path information of all passing-by data.

For example, when di flows from N11 to destination node N1 along the path N5 _ N4 _ N3, N4 and N5 need not cache the path information of di since N4 and N5 are closer to the data center than the caching node N1. Thus, a node only needs to record the data path when it is closer to the caching node than the data center. Due to mobility, the node which caches the data may move. The cached data may be replaced due to the cache size limitation. As a result, the node which modified the route should reroute the request to the original data center after it finds out the problem. Thus, the cached path may not be reliable and using it may adversely increase the overhead. To deal with this issue, a node Ni caches the data path only when the caching node, say Nj, is very close. system tuning threshold, called TH, when CachePath is used.

5. HYBRID CACHING SCHEME

The performance analysis showed that Cache Path and Cache Data can significantly improve the system performance. We also found that Cache Path performs better in some situations such as small cache size or low data update rate, while CacheData performs better in other situations.

5.1 HYBRID ALGORITHM

To further improve the performance, we propose a hybrid scheme Hybrid Cache to take advantage of Cache Data and Cache Path while avoiding their weaknesses. Specifically, when a node forwards a data item, it caches the data or path based on some criteria. These criteria include the data item size si, the TTL time TTLi, and the Hsave. For a data item di, the following heuristics are used to decide whether to cache data or path

- If si is small, CacheData should be adopted because the data item only needs a very small part of the cache; otherwise, CachePath should be adopted to save cache space. The threshold value for data size is denoted as Ts.
- If TTLi is small, CachePath is not a good choice because the data item may be invalid soon. Using CachePath may result in chasing the wrong path and end up with resending the query to the data center. Thus, Cache Data should be used in this situation. If TTLi is large, CachePath should be adopted. The threshold value for TTL is a system tuning parameter and denoted as Tttl.
- If H save is large, Cache Path is a good choice because it can save a large number of hops; otherwise, Cache Data should be adopted to improve the performance if there is enough empty space in the cache. We adopt the threshold value TH used in Cache Path as the threshold value.

These threshold values should be set carefully as they may affect the system performance. Their effects and how to set them are studied through simulations

ALGORITHM (Hybrid Cache)

When a data item di arrives:

```
if (di is the requested data by the current node)      then
 cache data item di;
        return;
If (an old version of di is in the cache)then
        Update the cached copy;
Else if (si <Ts or there is an invalid copy in the cache Or
there is  a cached path for di ) then
                Cache data item di;
 Else if (Hsave  > Th and TTLi  > Tttl )then
               Cache  the Path of di;
```

Si – size of data item   H →Hop
Ts,Th – Thresholds TTL –Time to Live

When a request for data item di arrives:

```
If(there is a valid copy in cache ) then send di to the requester ;
Else if (there is a valid path for di in the cache)
        then
Forward the request to the caching node;
        Else
Forward the request to the data center.
```

Tttl –system tuning Parameter (Threshold value)

5.1 Hybrid Cache Algorithm

Section 5.1 shows the algorithm that applies these heuristics in Hybrid Cache. In our design, caching a data path only needs to save a node id in the cache. This overhead is very small. Therefore, in Hybrid Cache, when a data item di needs to be cached using Cache Data, the path for di is also cached. Later, if the cache replacement algorithm decides to remove di, it removes the cached data while keeping the path for di. From some point of view, Cache Data degrades to Cache-Path for di. Similarly, Cache Path can be upgraded to CacheData again when di passes by.

6. PERFORMANCE EVALUATION

The performance evaluation includes three sections. The simulation model is given in Section 6.1. In Section 6.2, simulated environments and screenshots are shown. Section 6.3 shows the performance graph of cache data,cache path and hybrid cache from the analytical results.

6.1 THE SIMULATION MODEL

The simulation is based on ns-2 with Cygwin Environment.. In our simulation, AODV (Ad Hoc On Demand Distance Vector) used as an underlying routing algorithm, Since it is better than DSDV and I-DSDV(Improved Direct sequence Distance Vector) .The aforesaid fact is justified by following results.

Packet Delivery Fraction of AODV

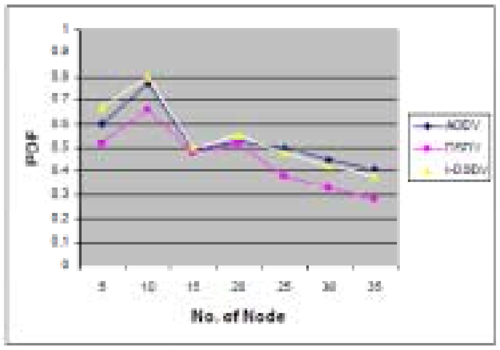


Figure 6.1 a)

From the Figure 6.1a) it is shown that AODV perform better when the number of nodes increases because nodes become more stationary will lead to more stable path from source to destination DSDV performance dropped as number of nodes increase because more packets dropped due to link breaks. I-DSDV is better than DSDV especially when the number of nodes is between 20 and 35. I-DSDV improved the PDF since it find new route to destination when link breaks existed.

End to End Delay of AODV

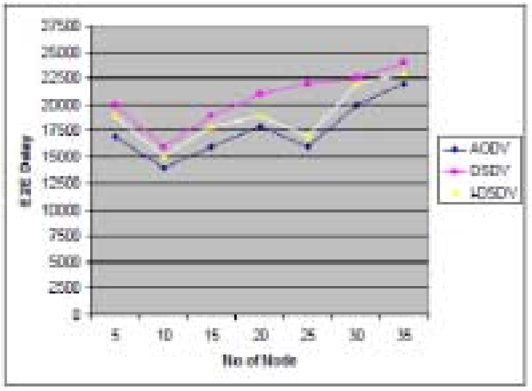


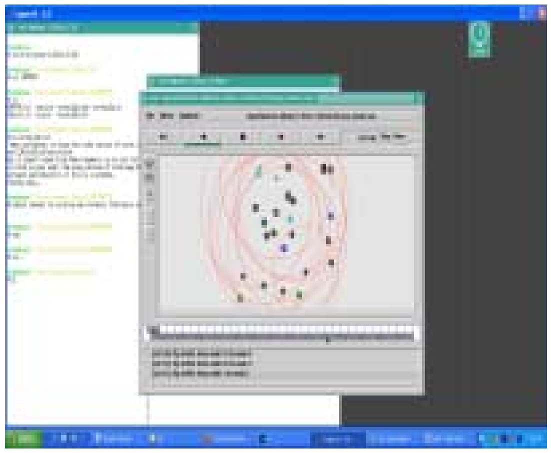
Figure 6.1 b)

From the Figure 6.1 b) it is clear that AODV didn’t produce so much delay even the number of nodes increased. It is better than other two protocols. The performance of I-DSDV is slight better than DSDV especially when the number of nodes between 15 and 30. It shows that, the I-DSDV protocol improved the DSDV but slightly lower than AODV when the nodes is higher.We assume that the wireless bandwidth is 2 Mb/s, and the radio range is 200m to 250m.

6.2 SIMULATION

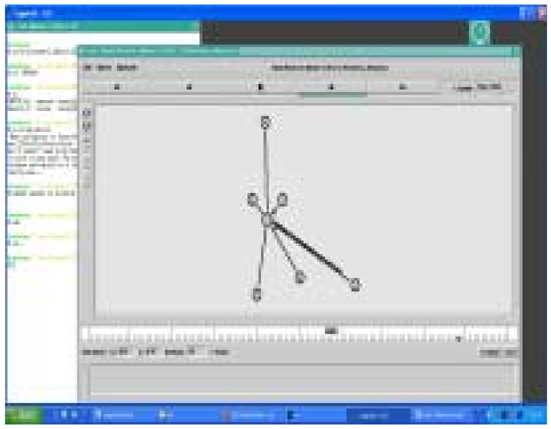
Experiments were run using different workloads and system settings. The design of wireless scenario is shown in the Fig 6.2 (a).In this caching of any node and caching of path is shown. In Figure 6.2(b) and (c)the system model of the ad hoc network is shown.In this totally seven nodes are shown.In that one node acting as as a Data Center and remaining nodes have peer to peer connection.

As per the Hybrid cache algorithm Cache Path and Cache Data have been done with the help of ns2 simulator. Caching of data is done when the node 0 requests the data from the data center.If the request is again from the node 0 , the node 0 itself analyze for the data and get the results .If some other node in this ad hoc network request for the same data again ,a node which has the path id of the node1 will respond for the request.

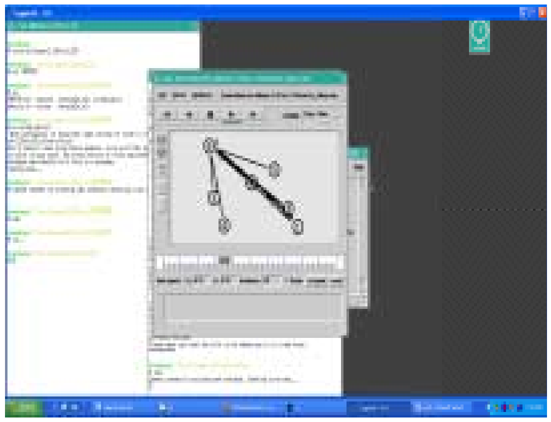


.

6.2 a) Wireless scenario

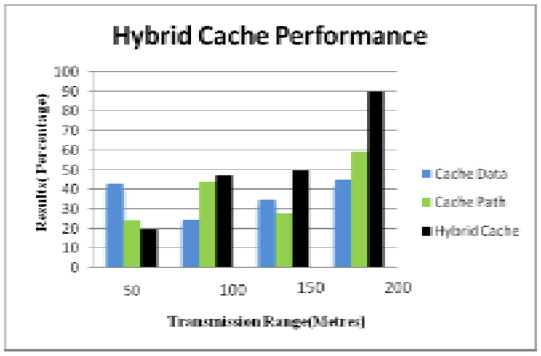


6.2 b) Ad Hoc Model



6.2 c) Request of node 1 and Response from Data center (node 0)

6.3 PERFORMANCE GRAPH



6.3 Transmission Range Experiments

The graph 6.3 shows the comparison graph of cache data ,cache path and hybrid cache.Comparison has been done by taking transmission Range as a parameter.From the Analytical results we can understand that hybrid cache methodology gives the gradual growth while increasing transmission range. Hybrid cache is better than cache data and cache path.Since In this scenario we have considered cache data and cache path. We have considered cache data whenever it is essential (depending upon threshold level and size of a data) and considered cache path(depending upon threshold of Time to live of a data item).

7. CONCLUSION AND FUTURE WORK

This paper presented a novel approach for reducing the spatial query access latency by leveraging results from nearby peers by applying Hybrid cache Mechanism. Specifically, we proposed three schemes: Cache Path, Cache Data, and Hybrid Cache. In Cache Data, intermediate nodes cache the data to serve future requests instead of fetching data from the data center. In CachePath, mobile nodes cache the data path and use it to redirect future requests to the nearby node which has the data instead of the faraway data center. Hybrid Cache takes advantage of CacheData and CachePath while avoiding their weaknesses. Cache Replacement policies are also studied to further improve the cache performance. Simulation results showed that the proposed schemes can significantly reduce the query delay when compared to SimpleCache and significantly reduce the message complexity when compared to FloodCache The experiment results indicate that our method can reduce the access to the wireless broadcast channel by a significant amount, for example, up to 90 percent.

REFERENCES

[1] S.Acharya, R.Alonso,M.J. Franklin, and S.B. Zdonik, “BroadcastDisks: Data Management for Asymmetric Communications Environments,” Proc.ACM SIGMOD’95, pp.199-210, 1995.

[2] D.Barbara´ , “Mobile Computing and Databases: A Survey,”IEEE Trans. Knowledge and Data Eng., vol. 11, no. 1, pp. 108-117, Jan./Feb. 1999.

[3] N.Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD '90, pp. 322-331, 1990.

[4] C. Y. Chow, H. Va Leong, and A. Chan, "Peer-to-Peer Cooperative Caching in Mobile Environment," Proc. 24th IEEE Int'l Conf. Distributed Computing Systems Workshops (ICDCSW '04), pp. 528-533, 2004.

[5] C.-Y. Chow, H. Va Leong, and A.T.S. Chan, "Distributed Group- Based Cooperative Caching in a Mobile Broadcast Environment," Mobile Data Management, pp. 97-106, 2005.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry Algorithms and Applications, second ed. Springer, 2000.

[7] T.Hara, "Cooperative Caching by Mobile Clients in Push-Based Information Systems," Proc. 11th ACM Int'lConf.Information and Knowledge Management, pp.186-193, 2002.

[8] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," ACM Trans. Database Systems, vol. 24, no. 2, pp. 265-318, 1999.

[9] T.Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air: Organization and Access," IEEE Trans. Knowledge and Data Eng., vol. 9, no. 3, pp. 353-372, May-June 1997.

[10] W.-S. Ku and R. Zimmermann, "Location-Based Spatial Queries with Data Sharing in Mobile Environments," Proc. 22nd IEEE Int'l Conf. Data Eng. (ICDE '06) Workshops, p.140, 2006.

[11] W.-S. Ku, R. Zimmermann, and H. Wang, "Location-Based Spatial Queries with Data Sharing in Wireless Broadcast Environments," Proc. 23rd IEEE Int'l Conf. Data Eng. (ICDE), 2007.

[12] M.F. Mabel, C.-Y. Chow, and W.G. Aref, "The New Casper: Query Processing for Location Services without Compromising Privacy," Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06), pp. 763-774, 2006.

[13] D.C. Montgomery and G.C. Runger, Applied Statistics and Probability for Engineers, second ed. John Wiley & Sons, 1998.

[14] J.L. Ringuest, "A Chi-Square Statistic for Validating Simulation- Generated Responses," Computers and Operations Research, vol. 13,no. 4, pp. 379-385, 1986.

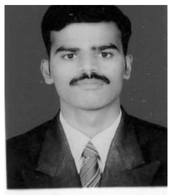
[15] .N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," Proc. ACM SIGMOD '95, pp. 71-79, 1995.

[16] T.K. Sells, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi- Dimensional Objects," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '87), pp. 507-518, 1987.

[17] W.-S. Ku, R. Zimmermann, and H. Wang,"Location-Based Spatial Query Processing in Wireless Broadcast Environments," IEEE Trans. Mobile Computing,vol. 7, no.6, pp. 778-791, June .2008.

[18] B. Zheng and D. Lun Lee, "Information Dissemination via Wireless Broadcast," Comm. ACM, vol. 48, no. 5, pp. 105-110, 2005.

Authors Biography



C.Gopala Krishnan received his B.E. degree in Computer science Engg in 2002 from Madurai Kama Raj University and M.E. degree in Computer Science and Engineering in 2010 from Anna University Tirunelveli, Tirunelveli, He is currently working as Assistant Professor in the Department of Computer Science and Engineering at Sree sastha institute of Engg,India. His areas of interest are Mobile computing, operating systems and Computer Networks.Currently he is doing research on Mobile Computing and Spatial Queries He has published many papers in national and International journals.He is the lifetime member of ISTE and IEEE.

R.Suceendrakumar received his B.E. degree in Computer science and Software Engg in 2012 from vinayaka mission University and post graduate degree in Computer Science and Engineering from Anna University Chennai, India. His areas of interest are Mobile computing, operating systems and Computer Networks. He has presented many papers in national conferences in various fields. As part of this paper, he is working on developing communication protocols for wireless networks—protocols optimized for wireless and mobility that can support file and database access. He is also investigating operating systems support for mobile hosts .He is a member of ISTE.