Distributed Caching of Multi-dimensional Data in Mobile Environments

Bin Liu[†] Wang-Chien Lee[§] Dik Lun Lee[†]
[†]Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Hong Kong
{liubin, dlee}@cs.ust.hk
[§]Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
wlee@cse.psu.edu

ABSTRACT
Caching has been an important technique for saving network traffic and reducing response time, especially in mobile environments where bandwidth is often a scarce resource. In this paper, we propose a novel approach for caching multidimensional data in a cluster of mobile devices. In particular, we focus on the most common types of multi-dimensional queries, namely range and k-nearest neighbor queries, by computing a cacheable region for every query, caching the result at the client, and indexing it in an R*-tree at the cluster gateway. Subsequent queries are first issued to the R*-tree and only remainder queries or queries that cannot be guaranteed exact answers are sent to the remote data server. To the best of our knowledge, our work is the first to study caching results from complex multi-dimensional queries (e.g., kNN query) and propose to build an R*-tree on previously fetched query results in a cluster of mobile devices. Rigorous experiments show that our approach significantly reduces network traffic and response time.

Categories and Subject Descriptors
H.2.4 [**Information Systems**]: Database Management –*Systems*

General Terms
Algorithms, Experimentation

Keywords
Caching, Mobile Data Management, Multi-dimensional Database

1. INTRODUCTION
Mobile computing has been flourishing with the maturity of wireless link standards (e.g., IEEE802.11b/g), evident from the increasing number of home appliances going wireless (e.g., laptop, PDA, mobile phone). While mobile hosts can interconnect with wide-band connections (e.g., IEEE802.11g provides up to 54 Mbps data rate in a wireless local area network), they often face a situation where there is only a narrow-band link to the Internet from the gateway. For example, many family users share one 56 Kbps phone line connection. Moreover, in some hard-to-reach areas or major disaster sites, wireless networks can be immediately deployed but the connection to Internet or remote servers is constrained. This situation is depicted in Fig. 1.1. The problem becomes more severe as every laptop out of factory is equipped with built-in wireless access capability while Internet infrastructure develops slowly. In this case, when clients submit queries to Internet databases (e.g., online real estate information), the performance is hold back by inadequate Internet connection despite abundant intra-network bandwidth.

In order to save traffic from/to the Internet and speed up querying of Internet databases, research community has developed semantic caching schemes [2, 5, 8, 6]. The basic idea is that mobile clients maintain in their caches both semantic descriptions (semantic regions) and the results of previous queries. New queries that can be answered with locally cached results are not sent to the server, and queries that can be partially answered are trimmed and only a *remainder query* is sent. As such, much network traffic to the server can be reduced. Semantic caching has been shown to have desirable performance in mobile environments compared to page caching or tuple caching [2]. Though there are abundant research results in semantic caching, no study has been conducted in an environment where local connection is more efficient and of lower cost than connection to the Internet, which is a reality in many occasions.

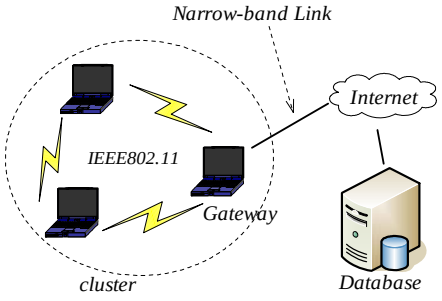


Figure 1.1: Network model

This paper proposes a comprehensive framework for caching multi-dimensional data in a cluster of mobile devices, which is especially suited for, but not limited to, the aforementioned environment. We focus on caching results from the most common types of multi-dimensional queries, namely range and k-nearest neighbor (*k*NN) queries, and make three important and original contributions. We first propose an innovative distributed caching mechanism which indexes at the gateway *cacheable regions* (the

MDM 2005 05 Ayia Napa Cyprus

regions in the semantic space that we can obtain complete knowledge from previous query results). The actual data, however, are cached for a period (*lifespan*) at the clients that issue the queries. Second, we propose to build CR-tree, a special R*-tree, at the gateway to index *cacheable regions* together with their *lifespan*. We also discuss in detail the construction and maintenance of a CR-tree. Our third contribution is query processing algorithms for range and *k*NN queries that take advantage of the local caches distributed in the cluster of mobile devices and significantly save traffic from the remote database. The effectiveness of our contributions is validated by rigorous experiments with real datasets.

The rest of the paper is organized as follows. Section 2 surveys related work on semantic caching and R*-tree, due to their immediate relevance to our techniques. Section 3 provides an overview of the system, and Section 4 discusses cache management issues. Section 5 describes query processing algorithms of range and kNN query, and Section 6 experimentally validates the efficiency of proposed techniques. Section 7 concludes the paper with future work.

2. RELATED WORK

Semantic caching scheme was initially proposed by Dar et al. [2] for client-server environments. In this scheme, the client maintains a semantic description (e.g., $Salary > 50 \wedge Age \leq 30$) of data in its cache, and manage the cache access information and perform replacement at the unit of *semantic regions*. By checking semantic constrains, the client can decide whether a *selection* query can be locally answered or partially answered. The data needed from the server are specified as a *remainder query*. Semantic caching scheme in managing *location-dependent* information in mobile computing is studied in [5, 8]. Ren et al [6] thoroughly studied processing of selection queries, especially on forming *remainder queries*.

3. SYSTEM OVERVIEW

This section provides an overview of the proposed caching structure, before we proceed with details in subsequent sections. As shown in Fig.1.1, our work targets on a realistic environment where (i) computers (mobile clients) within a cluster interconnect via wide-band connection (e.g., IEEE802.11b/g) and (ii) clients in each cluster connect to the Internet through a narrow-band link from the gateway. This model is especially realistic with the popularity of wireless networking (most new notebook computers are equipped with a wireless LAN card). As the R*-tree is the dominant index for multi-dimensional data (implemented in *PostgreSQL*, *Oracle*, *IBM Informix* and *DB2*, etc), we assume that each client indexes its cached data using an R*-tree.

In this network, clients issue multi-dimensional queries to an Internet database server, which stores multi-dimensional data (e.g., online real estate information). In order to save traffic between the Internet and the cluster, clients cache previously fetched query results inside the cluster and send to the database only the part of the query that cannot be satisfied locally. Towards this, the gateway acts as a local organizer for maintaining cache information and local query processing. Note that, in our model the gateway is not responsible for storing all previous results for caching, since it might have limited resources. Instead, clients who issue queries are delegated to store fetched results for a period, since it is likely that they need those results again as well. For range and *k*NN queries, the gateway maintains records on previously fetched query results by indexing *cacheable regions* (elaborated in section 4.1) in an R*-tree variant, Cache R-tree (abbreviated as CR-tree), which indexes cacheable regions and the time they are admitted to cache. The cache can then be used to process queries. For example, upon receiving a range query, the gateway first checks whether the query can be locally satisfied with cached results. In particular, the gateway issues the same query to its CR-tree and decides whether the query window intersects any previously cached regions. There are three possible outcomes: (i) the query window is fully contained in cache and therefore the query is answered locally; (ii) query window partially intersects cached regions, and the query is decomposed into parts that can/cannot be locally answered; (iii) query window does not intersect with cached regions. In the latter two cases, the part of query window that cannot be locally answered is sent to the server as a *remainder query* and results are cached when returned. For the part of the query that can be locally answered, results are fetched by the gateway from corresponding clients that host the data. Afterwards, the combined results are sent to query issuer. An alternative scheme that reduces the workload of the gateway is to send the query issuer the address of clients that contain possible results. Since this detail does not affect our caching efficiency, we choose the previous scheme. By caching query results locally in a cluster, many queries can be answered completely or partially with local data with cheap communication cost. As such, much communication to the remote server can be avoided.

The above description serves as a high-level picture of our framework, omitting two key components: cache management and query processing algorithms, which will be discussed in detail in subsequent sections.

4. CACHE MANAGEMENT

Cache management involves computing *cacheable regions* from queries and indexing of those regions in the CR-tree. We will also discuss the construction and maintenance of the CR-tree.

4.1 Cache Admission Control

Clients submit various types of multi-dimensional queries (e.g., range queries, *k*-nearest neighbor queries) through the gateway to the database server. From the query results sent by the server, we can obtain complete knowledge about a certain region (*cacheable region*) in the semantic space of remote data. The region can be inserted into the cache, and the actual results are stored by the client that issues the query. To illustrate, consider examples in Fig. 4.1 (range query) and Fig. 4.2 (*k*NN query).

4.1.1 Range Query

After the gateway sends a range query with query window q (dashed rectangle in Figure 4.1) to the database server (on its client's behalf) and retrieves results, the *complete* knowledge of region q at the server is available to the gateway. Namely, all data objects in region q at remote server have been retrieved. Therefore, q can be indexed by the gateway as a cache item. The actual results (object a and b) form a smaller MBR, and they are stored

by the client which issues the query. For *n*-dimensional data (*n*≥3), the *cacheable region* is also the query window.

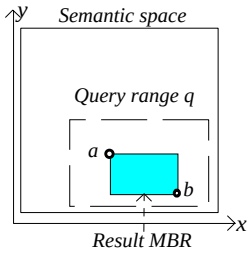
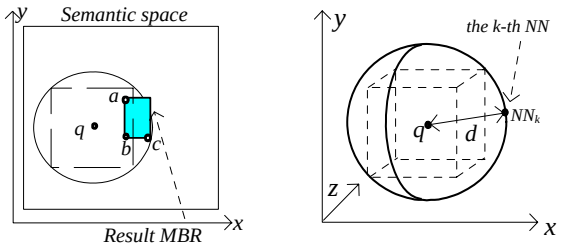


Figure 4.1: Cacheable region of a range query

4.1.2 kNN Query

The answer to a *k*NN query can be indexed since we can as well grasp the complete knowledge about certain region n in the database. Figure 4.2a illustrates a 3-NN query in 2D space with query point *q* and results *a*, *b*, and *c*. From the query results, we know that there is no *other* object in the circular region centered at *q* with radius |*qc*|, where |*qc*| is the distance from query point to the *k*-th neighbor (or the farthest neighbor among all results). Though we have complete knowledge about the circular region, it is inconvenient to index such a region in an R*-tree. In 2D space, we can use a Conservative Bounding Rectangle (CBR), which is the largest square bounded by the rectangular region (shown as a dashed square). *k*NN in n-dimensional space (n≥3) is similar, as demonstrated by an exemplary *k*NN query in 3D space (Fig. 4.2b). Denote the *farthest* nearest neighbor among results as NN_k. We are guaranteed no other object in the sphere centered at *q* with radius *d*, where *d* is distance from *q* to NN_k. We have a Conservative Bounding Box (CBB), which is the largest cube enclosed by the sphere. The CBB is inserted into the cache since we have complete knowledge about data it covers. The side length of the cube is 2*d*/*sqrt*(3), and it is straightforward to extend to *n*-dimensional case where side length of the CBB is 2*d*/*sqrt*(*n*).



(a) 2D kNN query (b) 3D *k*NN query

Figure 4.2: Cacheable region of a *k*NN query

With aforementioned methods, a gateway can decide the region possible to cache. If the cache space is not full, the region can be admitted to the cache together with information about the client that actually caches the data. However, since storage space is usually limited in mobile devices, clients only serve their cached data for only a period. We name this period *lifespan*, and denote it as *T*. The *lifespan* can be negotiated between a client and the gateway when they first connect, but we assume the same *lifespan* for all cache items (for simplicity). *T* can also be set as the expected time interval after which remote data are updated in order to automatically expunge obsolete cache entries. As such, the gateway inserts the *cacheable region* together with the address of the client hosting the actual data and the admission time into the cache. This cache item will expire after time *T* and has to be expunged. If the cache is full, the gateway has to victimize some existing cache items.

4.2 Cache Organization

We propose to use the R*-tree, the dominant spatial index, as our cache structure. As we maintain more information than a traditional R-tree and data nodes are distributed in clients, we name our R-tree variant Cache R-tree (CR-tree). We will discuss construction and maintenance of CR-tree.

4.2.1 Construction of CR-tree

Whenever query results are retrieved from the server, the gateway computes the *cacheable region* using methods in section 4.1. If the cache is not full, the region, together with the address of the client hosting the actual data and the current timestamp, is inserted into the CR-tree. In order to facilitate cache replacement, we also maintain access count for each entry after it is accessed. As such, each entry in the leaf node of the CR-tree initially has the following information: (i) MBR, (ii) address of the client caching data in the MBR, (iii) insertion time, and (iv) access count (initially zero). An internal node, however, keeps a timestamp as the insertion time of its newest descendant. This would enable the gateway to discover whether all nodes in a sub-tree have expired, at the price of maintaining extra information (timestamp) at internal nodes. Being able to discover sub-tree expiration has two advantages: (i) query processing can stop descending the sub-tree earlier (more elaboration in section 5), and (ii) we can easily expunge a sub-tree if it expires. As such, much I/O cost is saved at a small cost in space (maintaining timestamp information does not incur extra I/O).

The insertion and deletion algorithms of the CR-tree are essentially the same as in [1] with some minor difference. Existing entries in a leaf node may be expunged earlier (before their *lifespan* expire) if they have much overlap with an incoming entry and their lifespan will soon expire. The rationale behind this is to save space in an early stage thus entries covering fresh regions can be accommodated later. However, this strategy may overload clients serving hot data since data replication is reduced. In order to remedy this, we take into account the access count of an entry. If an entry has been frequently accessed, it will not be expunged earlier even if there is an incoming entry covering the same region since they cover a *hot* region.

We will illustrate using an example. Fig. 4.3 shows *cacheable regions* from clients *C*2, *C*3, *C*4, and *C*5, and R*-tree nodes they formed in 2D space (assuming a node capacity of 2 entries), while Fig. 4.4 shows the corresponding CR-tree maintained at the gateway. For simplicity, we have omitted the access count information in leaf entries. To illustrate from scratch, assume that the initial cache is empty and client *C*2 issues a query (*range* query or *KNN* query). A *cacheable region* is formed and the MBR bounds result objects *a* and *b*. An entry, *E*7, which contains the MBR, address of *C*2, insertion time (0), and access count (0), is inserted into the empty R*-tree at the gateway. Similarly, another entry *E*9 is created at timestamp 1 when client *C*4 fetches objects *e* and *f*. Internal nodes (e.g., *E*3) take the latest timestamp among all its descendants (e.g., 2 for *E*3). In this fashion, we can build up the CR-tree. Note that the leaf level of the CR-tree contains no

actual data from the database server. Instead, it contains the addresses of clients which actually cache the corresponding MBRs and data objects (e.g., leaf entry $E7$ stores address of $C2$). Details on maintenance of CR-tree is available in the full version [4].

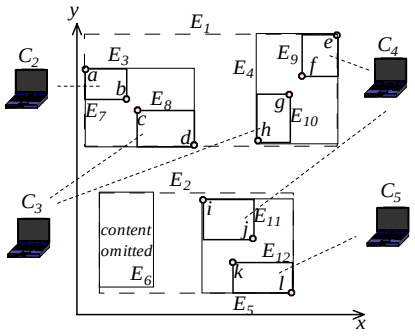


Figure 4.3: Cached regions

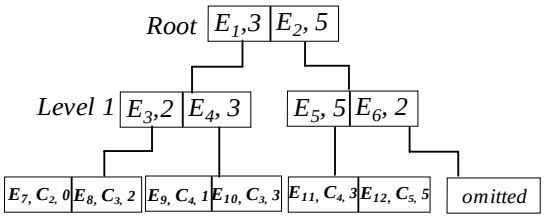


Figure 4.4: Corresponding CR-tree

5. QUERY PROCESSING

Upon receiving a multi-dimensional query to the database server from a client, the gateway checks the CR-tree for cached results and sends remainder queries to the server if necessary. In this section, we will discuss processing algorithms for the two types of queries of most importance, range and *k*NN queries.

5.1 Range Query Processing

Given a query window q at timestamp t, the gateway queries its CR-tree for entries intersecting q. The processing algorithm is essentially the same as in Section 2.1 except that we also check timestamp of a node during traversal. If a node has timestamp smaller than difference between t and *lifespan*, it is not be further explored since it has expired. When an entry with MBR intersecting q is found, the intersection region is sent to the client that actually hosts the data (denote this client as Ch). Client C_h checks its local cache for qualified results in the region and returns them to the gateway. The region that finds no intersection in the CR-tree is decomposed to remainder queries and sent to the database. The decomposition algorithm has been thoroughly studied by Ren et al [6] and therefore omitted here.

Fig. 5.1a shows an exemplary range query at timestamp 4 with query range shown in gray. Assume a *lifespan* 10 for all entries, entry $E8$ and $E11$ intersect the query region. This situation is magnified in Fig. 5.1b. Region $R1$ and $R2$ are cache hits and contain potential results, and therefore they are sent to client $C3$ and $C4$, respectively. The remainder in the query window is decomposed into three regions, $R3$, $R4$, and $R5$, and they are sent to the database server. When all results to the query are ready, they are combined and sent to the query issuer, and the query window is inserted to the CR-tree as a candidate cache item.

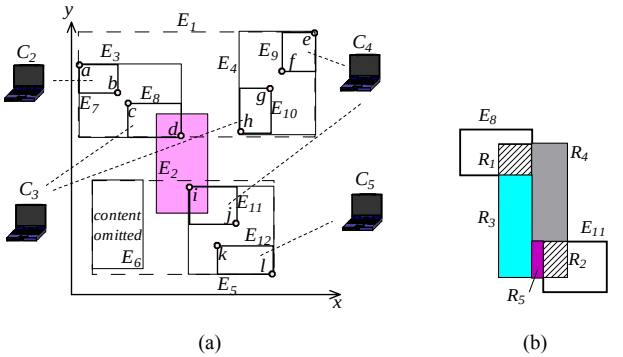


Figure 5.1: Range query and its remainder queries

5.2 *k*NN Query Processing

For *k*NN queries, since the gateway does not have the complete knowledge of data in the database, it is possible that locally identified nearest neighbors are actually false hits. This situation is illustrated in Fig 5.2, where a 2-NN query is issued and the query point is q. With only local data, object h and j are found to be candidate nearest neighbors. However, there is an object m in MBR B (shaded MBR in the lower right corner) in the database, and it is closer to q than j. This situation is alleviated as more regions are cached. In particular, if the circular region centered at the query point with radius D falls completely in cached regions, we are guaranteed of exact answer, where D is the distance from q to the farthest NN among results. This is the necessary and sufficient condition for the gateway to guarantee validity of locally identified results. As such, we provide a best-effort query processing for *k*NN queries. If no exact answers are available, the query is forwarded to the database server.

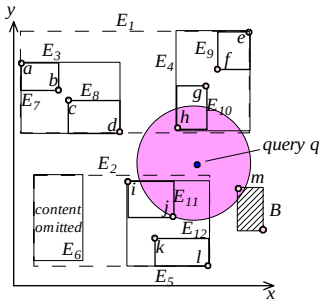


Figure 5.2: Exemplary kNN query

6. EXPERIMENTAL EVALUATION

In this section, we demonstrate the effectiveness of proposed techniques with extensive experiments.

6.1 Simulation Settings

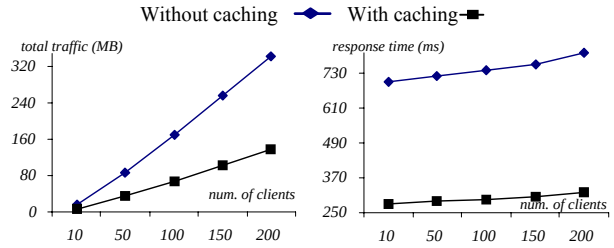
Experiments are performed on a PC with Pentium 4 2.4G CPU and 1GB RAM. We assume a cluster of mobile clients of size C

(variable). Mobile clients issue range and *k*NN queries via a gateway to an Internet database, which indexes a real dataset containing 1314*k* geographical locations (*X*, *Y* coordinates) in California, Los Angeles [7]. The values are normalized to range [0, 10k]. The dataset is indexed by an R*-tree [1] with node (disk page) size of 1k bytes, and the capacity (i.e., the maximum number of entries per node) is 50. Since clients are more interested in places with more data, we generate range and *k*NN queries distributed according to data distribution. Range queries are square regions of side length q_L, which varies from 200 to 1000.

At the gateway, previous query regions are cached and indexed with a CR-tree of node size 1k bytes and capacity of 50. Gateway to Internet database communication channel is modeled as a FIFO queue with implementation from [3]. Capacity of the link between the gateway and database is 56 kbps. Queries at the server are delayed but not dropped if the link is overloaded. Internet connection overhead (e.g., packet header, etc) is not modeled, since our caching technique does not change the total number of messages between the cluster and database and therefore the overhead is the same.

6.2 Caching vs. No Caching

We simulate 60 minutes of querying to the database with an even mixture of range and 10-NN queries and vary the number of clients in a cluster. Figure 6.1a and 6.1b show the network traffic from the database and response time per query, respectively. With caching, both network traffic and response time are reduced. When the cluster size increases, more queries are submitted per unit time, and they introduce larger amount of Internet traffic and slightly increased response time.



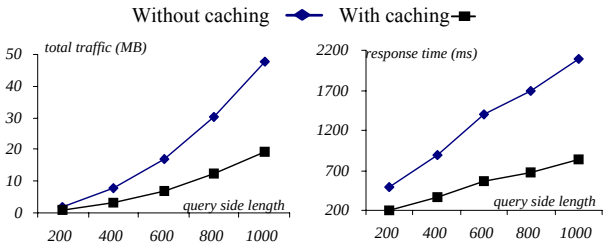
(a) Network traffic (b) Response time

Figure 6.1: Effectiveness of caching

6.3 Impact of Query Selectivity

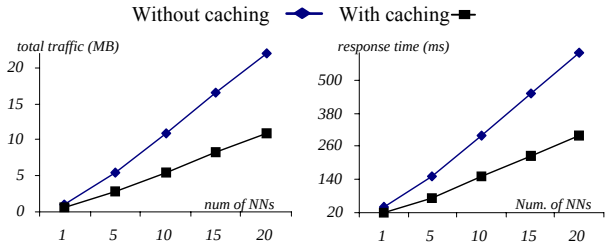
We measure performance with various side lengths of range queries, and various values for *k* of *k*NN queries. Larger side length or value for *k* means larger selectivity, and hence increased number of objects in query results. We simulate 60 minutes of querying with each type of query. Fig. 6.2 and Fig. 6.3 show the impact of q_L and *k*, respectively. While both Internet traffic and response time increase with q_L and *k*, caching techniques always provide much better performance than without caching.

More experiments are performed to evaluate the impact of query selectivity, available space, and *lifespan*. We also measure the maintenance cost of CR-tree at the gateway. Details of these experiments are available in the full version [4]. All experiments show that our system is robust and efficient in various situations.



(a) Network traffic (b) Response time

Figure 6.2: Impact of side length of range queries



(a) Network traffic (b) Response time

Figure 6.3: Impact of *k*

7. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel caching framework for clusters of mobile devices and make three important and original contributions. We first propose an innovative distributed caching mechanism that can significantly reduce Internet traffic at small costs of CPU and storage space. Second, we propose to build the CR-tree, a special R*-tree, to index previously fetched query results. Our third contribution is query processing algorithms for range and *k*NN queries that take advantage of local caches distributed in the cluster of mobile devices. The effectiveness of our contributions is validated by experiments with real datasets. We are currently developing analytical models for our framework, which will facilitate the optimization of the system.

REFERENCES

[1] Beckmann, N., Kriegel, H. P., Schneider, R., Seeger, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. *SIGMOD*, 1990.

[2] Dar, S. et al. Semantic Data Caching and Replacement. *VLDB*, 1996.

[3] Law, A., Kelton D. *Simulation Modeling and Analysis*. McGraw-Hill, 2000.

[4] Liu, B., Lee, W., Lee, D. *Distributed Caching of Multi-dimensional Data in Mobile Environments*. (available at http://ihome.ust.hk/~liubin).

[5] Ren, Q., Dunham, M. Using Semantic Caching to Manage Location Dependent Data in Mobile Computing. *Mobicom*, 2000.

[6] Ren, Q., Dunham, M., Kumar, V. Semantic Caching and Query Processing. *IEEE TKDE*, 15(1), 2003.

[7] http://www.census.gov/geo/www/tiger/

[8] Zheng, B., Lee, D. Semantic Caching in Location-Dependent Query Processing. *SSTD*, 2001.