Collaborative Spatial Data Sharing Among Mobile Lightweight Devices

Zhiyong Huang[1,3], Christian S. Jensen[2], Hua Lu[1,2], and Beng Chin Ooi[1]

[1] School of Computing, National University of Singapore, Singapore
[2] Department of Computer Science, Aalborg University, Denmark
[3] Institute for Infocomm Research, Singapore

Abstract. Mobile devices are increasingly being equipped with wireless peer-to-peer (P2P) networking interfaces, rendering the sharing of data among mobile devices feasible and beneficial. In comparison to the traditional client/server wireless channel, the P2P channels have considerably higher bandwidth. Motivated by these observations, we propose a collaborative spatial data sharing scheme that exploits the P2P capabilities of mobile devices. Using carefully maintained routing tables, this scheme enables mobile devices not only to use their local storage for query processing, but also to collaborate with nearby mobile peers to exploit their data. This scheme is capable of reducing the cost of the communication between mobile clients and the server as well as the query response time. The paper details the design of the data sharing scheme, including its routing table maintenance, query processing and update handling. An analytical cost model sensitive to user mobility is proposed to guide the storage content replacement and routing table maintenance. The results of extensive simulation studies based on an implementation of the scheme demonstrate that the scheme is efficient in processing location dependent queries and is robust to data updates.

1 Introduction

In step with the continued advances in computing electronics and wireless networking technologies, the mobile computing is gaining in prominence. In mobile computing, users equipped with portable devices such as mobile phones and PDAs, termed mobile clients, may issue local queries to learn about their geographic surroundings. For example, mobile services may enable tourists to learn about near-by attractions and may inform shoppers about near-by sales. Traditionally, the data provided by such services are stored in a central database. By means of a point-to-point wireless communication channel, the mobile clients may communicate with an application server that accesses the data and is responsible for the processing of queries.

To reduce the client/server (C/S) communication cost, techniques have been proposed that use client storage to cache results of previous queries and then use these data for answering new queries either fully or, more often, partially [3, 11, 18]. These techniques almost completely rely on the C/S architecture, with little or no direct communication and collaboration among the mobile devices.

This sole reliance on the C/S architecture fails to take advantage of the new wireless peer-to-peer (P2P) communication capabilities of modern mobile devices. The wireless

P2P channels have considerably more bandwidth than do traditional wireless C/S channels [15]. Moreover, as adjacent mobile devices are likely to issue queries whose results overlap, it is becoming possible, and potentially attractive, for mobile devices to share data in P2P fashion.

In Figure 1, for example, mobile devices M_1 and M_2 have issued queries and have already stored locally data that correspond to the rectangles they belong to. Then a third

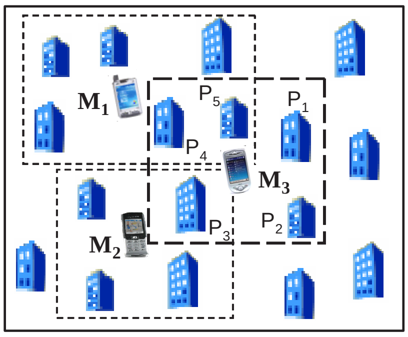


Fig. 1. Use Peer Storage to Answer Query

device M_3 issues a query for data corresponding to its rectangle, i.e., data corresponding to P_1 to P_5. Only the data pertaining to P_1 and P_2 are in M_3's local storage.

Using traditional techniques, M_3 must access the remote wireless server to obtain data for P_3, P_4, and P_5. In contrast, with P2P wireless communication M_3 can obtain data for P_4 and P_5 from M_1, and data for P_3 from M_2. This reduces the query response time significantly because the P2P bandwidth is much higher than the C/S bandwidth.

With the objective of exploiting the much faster wireless P2P channels, we propose a new collaborative data sharing scheme. An underlying grid-based structure is used for managing the data stored on the server and those portions of the data distributed among the mobile devices. The entire data space is partitioned by the grid, with each cell being a basic unit of data storage. The grid information is organized as a string, each bit of which indicates whether the corresponding grid cell contains any data or not. By broadcasting this space-saving string, the server is able to give clients global knowledge of its data, and to efficiently notify them of updates.

To facilitate the retrieval of peer data, each mobile device maintains a routing table. A routing entry captures which neighbor peer to contact for data pertaining to a specific grid cell, and how many hops to reach that peer. With routing tables, search among peers becomes directed, which contrasts to the blind flooding. This not only speeds up data search but also reduces communication messages. The routing table on each device is dynamically updated when its neighboring peers' storage contents change. Such changes are broadcast locally.

To answer a location-based spatial query with collaborative data sharing, a mobile device first checks its own storage, identifying those grid cells overlapping the query for which data is not available locally. Then it checks each cell of this kind in its routing table. If a relevant routing entry is found, a data request is sent to the corresponding peer via the fast P2P channel. Only the data not obtained this way is subsequently requested from the server via the C/S channel. Upon receiving data as needed, a mobile device conducts a refinement to reach the exact query result, and places the data in its storage.

To best utilize the limited device storage, a probability-based predictive cost model is proposed for storage replacement and routing table maintenance. Taking into consideration the predicted movement of each device, this model gives priority to the data in grid cells that have the highest probabilities of being reused in the future. By retaining relevant data in device storage, this model reduces the communication between devices and the remote wireless server.

To efficiently handle data updates, the server notifies clients of updates using a compact data format. Upon receiving a notification, a client can easily identify and evict invalidated data.

This paper makes the following contributions: First, it recognizes the discrepancy between previous techniques and current mobile environments, and accordingly proposes a collaborative data sharing framework for location-based spatial queries in such environments. Second, it proposes a probability-based predictive cost model that guides both storage replacement and routing table maintenance. Third, it proposes query processing strategies for mobile devices within the framework. Fourth, it discusses how to accommodate data updates within the framework. Fifth, it conducts extensive experiments to confirm that the paper's proposals are efficient and robust.

One might advocate a wireless C/S architecture based on technologies such as an IEEE 802.11 network, as an alternative to our assumed setting. However, such alternative technologies are not widely deployed, but are limited to very short ranges. In contrast, cellular networks are widely available and have very large numbers of users; these thus provide a huge space for our assumed setting [15].

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the system framework. Section 4 details the collaborative data sharing scheme and the cost model. Section 5 describes the relevant query and update processing techniques. Section 6 reports the experimental study results. Finally, Section 7 concludes.

2 Related Work

To reduce the communication between client and server via low-band wireless channels and shorten the query response time, caching techniques have been proposed for mobile environments [3]. Most such techniques are based on semantic caching [9], where semantic descriptions of previous queries as well as their corresponding results are cached. Subsequent queries are fully or partially answered using the cached results by matching the semantic descriptions. Semantic-based clustering [18] and Voronoi diagrams [23] are employed to organize semantic contents cached on mobile devices. However, these semantic caching schemes only support homogeneous queries. To remedy this, Hu et al. [11] propose a proactive technique that stores on the mobile clients both query results and the R-tree index nodes accessed during query processing. This technique may result in substantial device-side space use and processing costs. Later, Lee et al. [13] propose to cache both concrete data objects of interest and complementary regions at a coarse granularity. Inter-device collaboration is not addressed in any of the above works. Deviating from the previously dominant C/S architecture, Liu et al. [14] propose to cluster mobile devices via wide-band wireless links. In each cluster, one mobile device acts as a gateway that is connected to the Internet via a narrow-band link and forwards queries/results from/to its cluster. This work does not address device-side cache organization, and data are not passed directly among mobile devices, but via the gateways.

Assuming a wireless broadcast environment, Hara [10] proposes several caching strategies that enable clients to cooperate on selecting broadcast items to cache. By utilizing the multi-hops routing in an ad hoc network, Yin and Cao [22] propose that a device caches either the requested data or the path information to get the data, when it is forwarding a query result. Our work differs from those works in several ways. First, we set our problem in a hybrid system architecture (see Section 3.1 for details), instead of assuming a traditional wireless C/S environment or a pure ad hoc network. Second, we are specifically interested in location dependent queries requiring spatial data. Third, we take advantage of client movements in cache management on devices by using a specific model. Recently, Chow et al. [8] address cooperative caching in a mobile environment similar to our setting, whereas the authors are focused on grouping mobile devices with similar mobility patterns in order to improve cache performance.

Next, data management in mobile ad hoc networking (MANETs) [4] or mobile P2P environments has attracted significant research interest. Kortuem et al. [12] discuss scenarios where encountering mobile devices may exchange information, together with challenges faced by mobile ad hoc information systems. At a conceptual level, Xu et al. [21] discuss extensive data management topics in mobile P2P networks, including data modeling and data dissemination specific to mobile peers. Within a hierarchical mobile P2P environment with wireless cells at the bottom and fixed networks at the top, Budiarto et al. [7] discuss mobile data replication strategies in the fixed network. To avoid message flooding and improve search hit rates, Lindemann et al. [16] propose a distributed document search service that helps access results cached in peer devices by locally broadcasting queries and response messages.

3 System Framework

3.1 System Architecture

In a traditional, mobile C/S environment, one or more wireless application servers store data and process queries from mobile clients within their corresponding coverage. This setting is being shifted as mobile devices are increasingly being equipped with wireless ad-hoc networking capabilities such as infrared, Bluetooth, or even Wi-Fi. The ad-hoc networking channel usually has much higher bandwidth (1-11Mbps for IEEE 802.11b, and up to 54Mbps for IEEE 802.11a and 802.11g) than the traditional C/S channel (38.6 Kbps to 2.4 Mbps) [15]. This makes it possible and potentially attractive for the mobile devices to share their local data with their peers. The architecture with both C/S and P2P communication, as shown in Figure 2, we call a *hybrid* architecture. In this paper, we limit our problem to the extent of a single wireless application server's coverage.

For this as well as the traditional C/S architecture, a key objective is to reduce the communication between client and server, as the bandwidth of the wireless C/S channel is low. In the traditional C/S architecture, local storage is the only resource that a mobile device can attempt to exploit. In the hybrid architecture, the high bandwidth channels

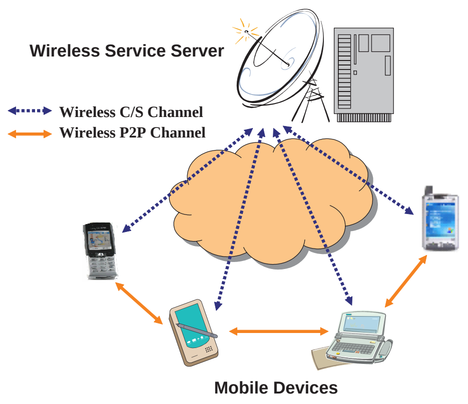


Fig. 2. Hybrid Mobile System Architecture

among mobile devices are an important resource that can be utilized. By allowing mobile devices to share their data via the P2P wireless channels, the communication between a mobile device and the server is expected to be reduced. We proceed to discuss how to achieve this reduction by using an appropriate data sharing scheme.

3.2 Collaborative Data Indexing Requirements

The spatial data on the server are usually indexed by some spatial indexes in order to facilitate efficient spatial-query processing. For the server alone, which has ample storage space and computing power, different spatial indexes are applicable, and their performance differences are not expected to be significant in comparison to as the considerable wireless communication cost.

The situation becomes complex when we intend to store and reuse data from the server on resource-constrained mobile devices. If a spatial query cannot be fully answered by reusing data stored locally, its unanswered portion will be sent to other mobile peers or to the server. This query portion can be processed in a direct way, without any transformation or adjustment, if the organizations of data on the mobile devices and on the server share some basic characteristics. Therefore, it will be beneficial if the index used on server also can be used on the mobile devices.

Since mobile devices usually have limited storage space and computing power, we must be careful to choose an appropriate spatial index that can be shared among the devices and the server. The limited storage makes a spatial index with little storage overhead attractive. The limited computing power renders it important that the operations to be performed on the spatial index are simple yet efficient.

3.3 Collaborative Indexing

Because the R-tree and its variants involve comparatively complex operations and consume extra space for internal nodes, we do not use them on the resource-constrained mobile devices. Rather, we use the simple yet effective grid file [17] as the spatial index in our system. A grid file allows economically storing on each mobile device a summary of the data indexed on the sever, enabling the devices to maintain knowledge of the server data. The entire data space is partitioned by a H (rows) by W (columns) uniform grid, yielding a total of $H \cdot W$ grid cells. The server holds a grid directory, which contains the extent ($left$, $right$, top, $bottom$) of the data space and a linear array representing all grid cells in row major order. Each cell has a pointer to the disk page storing those data points.

The summary of the server data indexed by a grid is organized in a compact format and broadcast to all mobile devices within the server's coverage. For each grid cell, one bit is used to indicate if it contains any data points: 0 for an empty cell and 1 otherwise. This way, a total of $\lceil (H \cdot W)/8 \rceil$ bytes are needed to represent all cells in row major order starting from the top left. In addition, the values of H, W, and the extent coordinates are necessary for a mobile device to perform basic grid indexing operations.

For H and W, we use a byte to represent either of them and thus can accommodate up to 64K grid cells, which usually is enough. For the extent coordinates we use floats, which need 4 bytes each. Therefore, every mobile device needs $\lceil (H \cdot W)/8 \rceil + 18$ bytes to hold the global grid summary. That information is organized into a byte string called a *grid string*, which sequentially contains $left$, $right$, top, $bottom$, H, W and bytes for all grid cells. The ending bits in the last byte are not used if the number of grid cells is fewer than 8.

An example is shown in Figure 3. In the upper part of this example, the region of

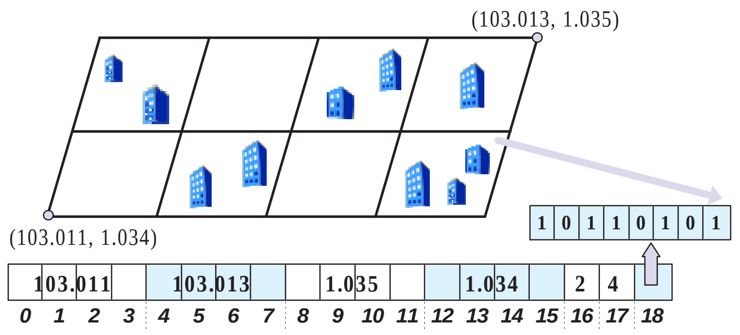


Fig. 3. Example Grid String

interest is partitioned into 2 rows and 4 column, i.e., 8 cells. In the lower part, the grid string is shown with 19 bytes totally. The bits in the last byte are used to indicate the validity of each grid cell, starting from the top-left one sequentially in row major order.

4 Collaborative Data Sharing Scheme

4.1 Storage Scheme for Mobile Devices

Every mobile device receives the grid string from the data server when it enters the coverage area of the server. This string is kept locally and used when processing spatial queries. For a given spatial query, the values of H, W, and the extent coordinates are used to determine which grid cells the query concerns. Then, the relevant bits in the grid string are used to identify the cells that actually contain data points. Only those cells are considered subsequently.

Without loss of generality, we suppose that every point has n attributes, in addition to the spatial coordinates of float type, and that these attributes occupy A_n bytes in total. Out of its total storage space, a mobile device M_i is allowed to use S_i bytes for data points. This portion is divided into N_i equal-sized contiguous slots, each of which can contain s_i points. In other words, $S_i = N_i \cdot s_i \cdot (8 + A_n)$.

Two data structures are used for this storage portion. The first, $StoredCells$, is a list of records of the format $\langle index, count, slot\_IDs\rangle$, where $index$ is the index of a cell stored, $count$ is the count of points in that cell, and $slot\_IDs$ holds the IDs of the slots that contain the cell's points. To facilitate search for grid cells, $StoredCells$ is maintained in ascending order of $index$. The second, $FreeSlots$, is a list of the IDs of unused storage slots. Initially $FreeSlots$ contains all storage slots and $StoredCells$ is empty.

As the grid parameters H and W each occupy a byte, a 2-byte short integer is enough for $index$. Assume a grid cell contains at most G_m points. Then $\lceil G_m/256\rceil$ bytes are needed for $count$. A slot ID needs $\lceil N_i/256\rceil$ bytes, while a grid cell needs at most $\lceil G_m/s_i\rceil$ storage slots. As a result, a stored cell needs at most $2 + \lceil G_m/256\rceil + \lceil (G_m \cdot N_i)/(256 \cdot s_i)\rceil$ bytes.

When a new grid cell is to be stored locally, the first step is to determine how many slots are needed. Then we need to search the $FreeSlots$ to see if enough free slots are available. If so, free slots are selected to store the cell. $FreeSlots$ is modified, and new cell information is created in $StoredCells$ accordingly. If not enough free slots exist, some stored grid cells must be evicted to make room for the new one. We discuss this issue in Section 4.3. Next, we address how to share storage contents among mobile devices.

4.2 Sharing Data via Routing Table

A naive approach for a mobile device to retrieve data from peers is to send a message to all its neighbors, each of which then either returns the requested data if it has, or forwards the request to its own neighbors. This approach leads to blind flooding rather and incurs a large number of messages, which may yield long response time.

To facilitate the retrieval of data from peers, we maintain a routing table on each mobile device. A routing table entry on a device M_i is in the format of $\langle cell\_id, nb, hops\rangle$. It tells that a neighbor peer nb of M_i has data for cell $cell\_id$ in its local storage or routing table. And $hops$ is the count of hops from M_i to the peer, via intermediate peers, that actually stores the data for the cell.

When M_i stores a new cell or evicts a cell, it notifies its neighbors by sending out corresponding messages. Upon receiving a message from M_i invoked by storing $cell_j$, a mobile device M_k takes one of the following three actions: (1) ignores it because $cell_j$ is in local storage; (2) creates a new routing entry for it because $cell_j$ is in neither its storage nor routing table; (3) compares the length of the new route with the that for $cell_j$ currently in the routing table, changing the current one if the new route involves fewer hops. Device M_k can also forward the new route to its own neighbors, if (2) happens. Similar actions are to be taken by subsequent neighbors. Upon receiving a message from M_i invoked by evicting $cell_j$, a mobile device M_k removes the relevant entry from its routing table if it exists. If a removal occurs, M_k sends similar messages to its own neighbors who will take similar actions.

A maximum hops value is used as a system parameter to further limit the forwarding of maintenance messages in the last two situations described above. A message for either a new route or for an eviction will not be forwarded any further when it has traveled the maximum number of hops.

When a mobile device M_i detects a new neighbor M_k through wireless signaling, M_i organizes the contents of both its storage and routing table into routing entries and sends them to M_k. Device M_k then updates its own routing table by checking the incoming entries against its storage and routing table.

4.3 Management of Limited Device Storage

For storage constrained devices, we need to replace both storage contents and routing table entries when necessary. We propose a cost model to guide the replacement.

Cost Model In our storage scheme, two factors need to be considered when choosing one or more victims for replacement. One is that after selecting victim(s), there should be enough free space for the new grid cell. For a stored grid cell C_j, the count of data points it covers can be found in the $\mathit{StoredList}$, i.e., $count$. The other is that we should attempt to avoid eliminating grid cells that will be accessed in the (near) future. This can be achieved by taking into account a mobile device's predicted movement.

We use $prob(M_i, C_j, \Delta t)$ to denote the probability that a device M_i will access cell C_j in the time period $[t_c, t_c + \Delta t]$, where t_c is the current time. We thus look Δt time units into the future and predict how likely it is that M_i will need C_j in that time period. In Section 5.4, we discuss how to choose an appropriate value for Δt.

Probability $prob$ also depends on the movement of M_i and the queries it will issue. Although it is difficult to predict the query pattern of a mobile device, we expect a significant spatial locality for the queries issued by a mobile device. Therefore, the distance between a device M_i and a grid cell C_j is of importance. At any single time point, we consider the Manhattan distance between M_i's current position and the center of C_j. Following Tao et al. [20] and Brilingaitė and Jensen [5], we assume that each device is aware of its own motion pattern. In contrast to Dar et al. [9], we use an integral to represent the distance between M_i and C_j for the period $[t_c, t_c + \Delta t]$, as shown in Formula 1. An integral along time has been proven to be effective in dealing with changes covering a future time period [19].

$$\mathit{iDist}_M(M_i, C_j, \Delta t) = \int_{t_c}^{t_c+\Delta t} \mathit{dist}_M(M_i.pos(t), C_j)\,dt \tag{1}$$

Suppose in an n-dimensional space, c_k is the middle point of a grid cell on the k-th dimension, and $p_k(t)$ is the time-parameterized position of a mobile device on that dimension. Formula 1 can be developed as follows.

$$\int_{t_c}^{t_c+\Delta t} \sum_{k=1}^{n} |p_k(t) - c_k|dt = \sum_{k=1}^{n} \int_{t_c}^{t_c+\Delta t} |p_k(t) - c_k|dt \tag{2}$$

This indicates that we can compute the integral on each individual dimension and then sum up all those integrals to obtain the desired distance. For example, in Figure 4,

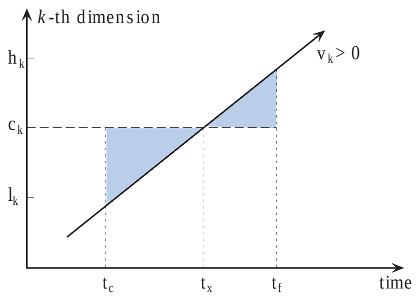


Fig. 4. Integral of Distance over Time

the mobile device has a positive linear velocity (it moves upwards) on the kth dimension, where the cell C_j's range is $[l_k, h_k]$. And t_f is assumed to be a future time point far enough from t_c, while t_x is the moment $p_k(t)$ passes c_k. Then the integral on that dimension can be expressed as a sum of two parts:

$$\int_{t_c}^{t_x} (c_k - p_k(t))dt + \int_{t_x}^{t_f} (p_k(t) - c_k)dt = \frac{1}{2} \cdot v_k \cdot (t_x - t_c)^2 + \frac{1}{2} \cdot v_k \cdot (t_f - t_x)^2 \quad (3)$$

Taking into account negative speeds, the appropriate integral value is:

$$\frac{1}{2} \cdot |v_k| \cdot ((t_x - t_c)^2 + (t_f - t_x)^2) \quad (4)$$

Depending on the concrete values of t_c and Δt, integrating on an individual dimension can involve one or two integral parts with corresponding ranges. Non-linear movements may involves additional integral parts because the mobile device may pass the middle point for more than once.

Intuitively, the closer a mobile device M_i gets to a grid cell C_j, the higher the probability that M_i will be interested in C_j is. This means *prob* is inversely proportional to $iDist_M$ for the period of consideration. Therefore, we give priority to those cells with large $iDist_M$ values when choosing victims. These are the least likely to be reused in the future because they are relatively far away from the mobile device M_i. When a victim is chosen, the storage slots it occupies is released and recorded in the *FreeSlots* list. The selection of a victim is repeated until *FreeSlots* has enough free slots for the new cell.

Routing Table Size Control Based on its own resource availability, a mobile device can determine how much local storage to use for its routing table. To avoid a possible overflow caused by routing table size growth, a mobile device can choose to ignore new route entries coming from peers, or remove existing ones from its own routing table. The decision can be made based on the proposed cost model, by computing the access possibilities of those grid cells in a routing table.

5 Query and Update Processing

Our scheme supports heterogenous queries. We consider the two arguably most popular query types: range queries and kNN queries. Overall, query processing proceeds as follows. After a location-based spatial query Q is issued on a mobile device M_{org}, termed *originator* of Q, the local storage contents are used to answer the query. If the query cannot be answered fully this way, the originator asks its neighbors for possible data with the help of its routing table. For those data portions that are unavailable in the routing table, requests are sent to the server. For a range query, a local refinement step is required to handle those grid cells that the query only covers partially. For a kNN query, the search bound is adjusted dynamically during the search to reduce the number of grid cells involved.

5.1 Range Queries

A range query Q_r issued on mobile device M_{org} is represented by $\langle pos, d \rangle$, where pos is M_{org}'s current position and the range is the circle centered at pos with radius of d.

When a range query Q_r is issued, the grid string is first used to identify the non-empty grid cells that intersect the range specified in the query. We use $C(Q_r)$ to represent the set of indexes of all these non-empty grid cells. Then, the local storage is checked to see if any of those involved cells are available locally. We use $C_l(Q_r)$ to represent the set of indexes of cells stored locally. If all cells are in local storage, i.e., $C_l(Q_r) = C(Q_r)$, the query is answered locally in full. Otherwise, the unavailable cells are retrieved from elsewhere. We let $C_p(Q_r)$ represent the set of indexes of grid cells that appear in M_{org}'s routing table, and $C_u(Q_r)$ represents the remaining grid cell indexes.

For the cells in $C_p(Q_r)$, requests are sent to peers according to the routing entries. Each such request is forwarded along a routing path until the peer holding the data is reached. This peer then sends the data to M_{org}. Due to the dynamic nature of wireless mobile ad hoc networking, it is possible that a mobile device along the routing path receives a request, but fails to find the relevant routing entry or grid cell data locally. Or, a mobile device may get a failure message from the lower protocol level when contacting a peer. When a device faces such situations, it returns a routing failure message to M_{org} along the reversed path. Each mobile device on the path back removes the relevant routing entry from its own routing table, till M_{org} places that cell in $C_u(Q_r)$. Finally, a request for the cells in $C_u(Q_r)$ is sent to the server, which in turn sends back the data to M_{org}.

5.2 *k*NN Queries

A kNN query Q_k issued on mobile device M_{org} is represented by $\langle pos, k \rangle$, where pos is M_i's current location and k is the number of nearest neighbors required.

Processing a kNN query Q_k is relatively complicated compared to a range query, as we cannot directly determine $C(Q_k)$, the set of grid cells intersected by Q_k. We conduct the kNN search by starting from the cell C_{org} where M_{org} is, then spiral through all surrounding cells from inner to outer. A search bound d_{bnd} is maintained during the procedure, which is the distance between the pos and the k-th nearest neighbor or max if less than k neighbors have been found thus far. At each step, we first decide the set of cells $cells_{srd}$ on a surrounding circle that need to be searched. Two kinds of cells are excluded from $cells_{srd}$: empty cells indicated by the grid string and those cells outside the search bound. For $cells_{srd}$, we first search the cells in local storage; then, for those ones not stored, we send requests to peers if they appear in the routing table, or

```
Algorithm kNNSearch($pos$)
Input:   $pos$ is the query originator's current position
Output:  $k$ nearest neighbors
1.  $d_{bnd}$ = $max$;
2.  decide the grid cell $C_{org}$ within which $pos$ lies;
3.  if ($C_{org}$ is not an empty cell)
4.      if ($C_{org}$ in storage)
5.          search $C_{org}$ and adjust $d_{bnd}$;
6.      else
7.          if ($C_{org}$ in routing table)
8.              send request for $C_{org}$ to peer;
9.          else
10.             send request for $C_{org}$ to server;
11.         search $C_{org}$ and adjust $d_{bnd}$ upon receiving;
12. while (TRUE)
13.     decide the next $cells_{srd}$ w.r.t $pos$ and $d_{bnd}$;
14.     if ($cells_{srd} == \emptyset$) break;
15.     for each cell $cell_i$ in $cells_{srd}$
16.         if ($cell_i$ in storage)
17.             search $cell_i$ and adjust $d_{bnd}$;
18.             remove $cell_i$ from $cells_{srd}$;
19.     if ($cells_{srd} \neq \emptyset$)
20.         $cells_{rt}$ = cells in $cells_{srd}$ and routing table;
21.         send requests for cells in $cells_{rt}$ to peers;
22.         send request for cells in $cells_{srd} \setminus cells_{rt}$ to server;
23.         search $cells_{srd}$ and adjust $d_{bnd}$ upon receipt;
```

Fig. 5. kNN Search Framework

otherwise to the server. The cells are searched as they are received. The loop terminates when we obtain nothing for the next cells$_{srd}$. The framework of kNN search on the query originator side is shown in Figure 5.

5.3 Updates

Updates to the data on the server can affect the grid cells in three different ways. First, an empty cell may become non-empty due to one or more data points being inserted. Second, a non-empty grid cell may become empty because all of its points are deleted. Third, the number of data points in a non-empty cell increases or decreases but remains non-zero, or point attributes change. This is the most likely scenario of the three in real life.

We use a two-tuple $\langle idx, flag \rangle$ to represent an update, where idx refers to the grid cell of the object being updated, and $flag$ indicates which of the above three types of updates it is (numbered I, II, and III, respectively). The server (or its administrator) is responsible for modify the server-side data and index when an update happens. After that, the server notifies the clients of the update by simply broadcasting the two-tuples to them.

Each client M_i processes an incoming update as detailed in Figure 6. If M_i has an ongoing query q whose result so far is invalidated by the update, the query q is discontinued. If the update is of type I or II, M_i needs to invert the corresponding bit in the grid string. If the grid cell C_{idx} involved in the update resides in storage, M_i evicts it from the storage. Otherwise, if the cell C_{idx} has a routing entry in the routing table, it is removed from the table.

```
Algorithm update($idx, flag$)
Input:   $idx$ is the index of the grid cell updated
         $flag$ is the update type
1.  if (query $q$ is ongoing and $q$'s result so far covers $C_{idx}$)
2.      abandon query $q$;
3.  if ($flag$ is I or II)
4.      invert $C_{idx}$'s bit in the local grid string;
5.  if ($C_{idx}$ in storage)
6.      evict $C_{idx}$ from storage;
7.  else if ($C_{idx}$ in routing table)
8.      remove $C_{idx}$'s entry from routing table;
```

Fig. 6. Update Processing on A Device

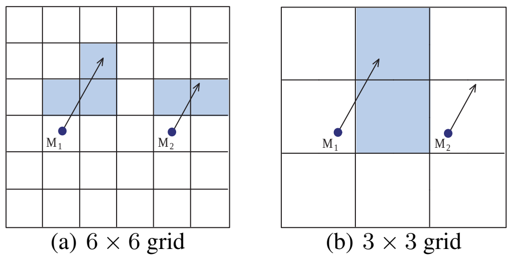
Our proposal is able to efficiently handle updates because the compact yet informative collaborative indexing scheme works successfully between the server and the clients.

In the experimental evaluation (see Section 6), we assume that updates happen at random across both space and time. We vary the *update ratio*, the ratio of the number of point updates during the experiment period to the total number of points used in the experiment, to see its impact on the performance of our proposal.

5.4 Effects of Grid Configuration and Δt

As a grid cell is the basic unit in our storage and sharing scheme, its size has important impact on system performance. If a cell is too large, which indicates it probably contains more data points, it will require too much storage space while actually most of data points within may not be used by the mobile device storing it. In contrast, if a cell is too small, new requests may become frequent, and query performance will deteriorate. The cell size also affects the probability estimation in Section 4.3 because the Manhattan distance in Formula 1 is relevant to the size of cell C_j.

See the example in Figure 7. The whole region of interest is partitioned using two different grids, a 6 × 6 grid and a 3 × 3 grid. Assume the existence of two devices M_1



(a) 6 × 6 grid (b) 3 × 3 grid

Fig. 7. Effects of Grid Cell Size and Δt

and M_2, whose current positions are represented as dots. The vector attached to each device indicates its movement, and its length indicates how far the device moves during Δt. Thus, M_1 moves faster than M_2 here. In the 6×6 grid, M_1 will need 3 cells within Δt, all of which are shaded in the figure. Though in the 3×3 grid, M_1 will only need two cells, the number of data points to be stored is considerably increased unless many empty cells are involved. This contrasts the situation of M_2 who does not need to store new cells in the 3×3 grid, at the cost of storing a large cell already.

Next, parameter Δt determines how far we will look into the future when estimating the probabilities for a cell to be reused. It also affects how many grid cells will be involved. Refer to Figure 7(a) and let M_1 and M_2 have the same velocity, but M_1 have a larger Δt than M_2 (note now a vector length indicates the Δt value). Consequently, M_1 needs to consider three cells while M_2 can do with two. Because different mobile devices can have different resources, computing capacities, and even movements, each mobile device should hold its own Δt when it computes the probabilities during storage replacement. Furthermore, a mobile device can use different Δt's to estimate probabilities at different times.

6 Experimental Evaluation

6.1 Experimental Settings

We implement our proposals using JiST-SWANS [1], a Java-based MANET simulator. We use a dataset named NE [2] of 123,593 points in float that represent metropolitan area postal addresses. We transform the dataset into the data space of $[1000 \times 1000]$. For each point we generate at random four attribute values in integer. We consider four main performance aspects: (1) the overall response time; (2) the local/peer storage hit rate; (3) the local/peer storage use rate; (4) the number of messages used to forward routes/queries between mobile devices. We investigate how these aspects are affected by different storage sizes, grid configurations, mobile network scales, update ratios, and Δt settings. The simulation experiments are conducted on an IBM x255 server running Linux with four Intel Xeon MP 3.0GHz/400MHz processors and 18G DDR main memory.

Table 1 lists the parameters used in the simulation. The settings in bold are the de-

Parameter	**Setting**
Grid configuration	$\mathbf{50}\times\mathbf{50}$, $60\times60, \ldots, 100\times100$
Number of mobile devices	$\mathbf{50}, 60, \ldots, 100$
Storage slot size	32 (data points)
Storage slot count	$\mathbf{50}, 60, \ldots, 100$
Δt	$50s, \mathbf{100}\boldsymbol{s}, \ldots, 300s$
Data update ratio (%)	$\mathbf{0}, 10, \ldots, 60$
Speed range	0.1unit/s–1unit/s
Max hops to forward routes	3
Holding time	60s
Wireless routing protocol	AODV

Table 1. Parameters Used in Simulation

faults, used when their corresponding parameters are not varied. Initially, all data are stored in the simulated server, and no device stores any data in local storage. For a two-hour simulation period, every mobile device issues 10 to 100 queries whose type, range or kNN, are determined randomly. For a range query, the ratio of its radius to a grid cell's side length is randomly picked among $0.1, 0.2, \ldots, 1$. For a kNN query, k is chosen at random from 1 to 5. We vary the number of mobile devices from 50 to 100, which yields a moderate-scale MANET [4]. All devices move within the spatial domain according to the random waypoint mobility model [6]. We set the maximum hops to forward a routing message to 3. We decided this value through some preliminary experiments, in which the value 3 achieves good cache effects but does not incur considerably high additional costs.

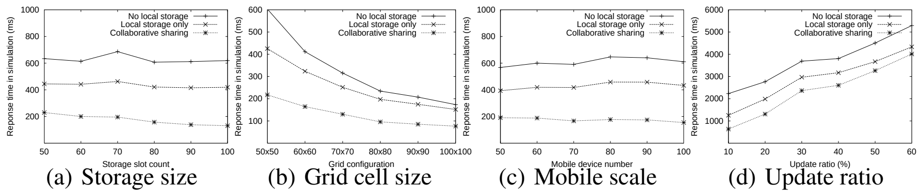
6.2 Response Time

The response time is defined as the elapsed simulation time from the moment that a query is issued at a mobile device M_{org} to the moment that M_{org} gets all answers. In the simulation, we set the mobile P2P channel bandwidth to 11Mbps (IEEE 802.11b), and the wireless C/S channel bandwidth to 384Kbps, which is what 3G wireless networks are expected to offer for mobility at pedestrian speed [4]. We compare three different strategies: no local storage, local storage only, and collaborative sharing. The average simulation results are shown in Figure 8. If no device storage is used at all, the response time is the longest. We also see that collaborative data sharing shortens the response significantly. This is because collaborative data sharing uses the faster wireless P2P channels rather than the slow wireless C/S channel.

Figure 8(a) shows that an increase in storage space favors collaborative sharing over the local storage strategy. This is because the extra space retains more data requested by peers via collaborative sharing. All strategies benefit from small grid cells, as shown in Figure 8(b), because in our cell-based scheme, a smaller cell contains less data and needs less time for transmission between devices or between devices and the server. As the number of devices increases, the response time of the collaborative sharing strategy decreases slightly, as shown in Figure 8(c). Increased mobility provides a higher collaborative sharing capacity, which, however, is countered by additional costs, including query and result forwarding via multiple hops. Compared to the aforementioned results, all strategies degrade in the presence of updates, as shown in Figure 8(d). This is so because updates may delay, if not invalidate, ongoing queries and evict data in local storage. Nevertheless our collaborative sharing strategy still performs the best, since inter-device sharing remains effective.

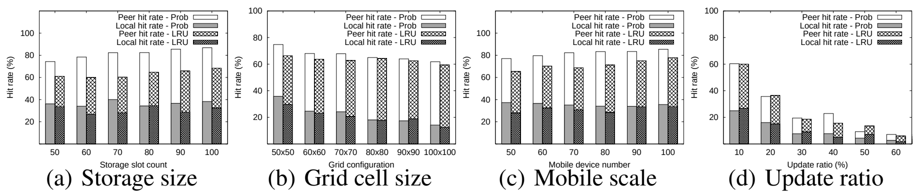
6.3 Storage Hit Ratios

A storage hit occurs when a desired grid cell is found without it having to be retrieved from the server. We distinguish between two types of storage hits: hits in the local storage of a device and hits in the storage of peers. For each device, we use the *local hit ratio* to represent the percentage of requested grid cells found in its local storage, and we use the *peer hit ratio* for the percentage found in peer storage. In the simulation, we compare our probability-based storage replacement policy with the traditional LRU policy. Our policy outperforms the LRU policy for almost all settings used in the experiments, as shown by the results reported in Figure 9. Our proposal predicts the



(a) Storage size (b) Grid cell size (c) Mobile scale (d) Update ratio

Fig. 8. Response Time in MANET Simulation



(a) Storage size (b) Grid cell size (c) Mobile scale (d) Update ratio

Fig. 9. Storage Hit Rate in MANET Simulation

near-future movement of a device and uses this for computing probabilities when it makes replacement decisions. In contrast, the LRU policy treats all grid cells from a static point of view.

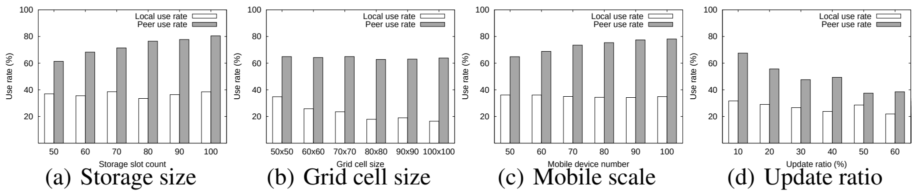
Referring to Figure 9(a), it is as expected that increased storage yields a higher hit ratio, as more data can be stored on the devices. Figure 9(b) shows that a coarser grid incurs a higher local hit ratio, but a lower peer hit ratio. Larger grid cells means that a bigger spatial region is stored on a device; hence, the device's future queries can find more data points locally, as queries on the same device exhibit locality. On the other hand, as adjacent devices are more likely to issue overlapping spatial queries than identical ones, they do not benefit from the larger grid cells and bigger spatial regions stored on the peers.

Nevertheless, larger grid cells imply the transfer of more data and may lead to longer response times, which is shown in Figure 8(b). According to the experiment covered by Figure 9(c), the peer hit ratio roughly grows proportionally with the number of device—with more devices, there is more storage for collaborative sharing, and hence more data can be obtained from peers instead of from the server. Referring to Figure 9(d), when updates are allowed both policies achieve lower hit ratios, but our proposal remains best for almost all cases. Updates tend to invalidate data in local storage, thus reducing the hit ratios. Our collaborative data sharing scheme is able to offset this effect to some degree.

We also consider the effect on the storage hit ratio by parameter Δt used in the probability-based replacement. The experimental results reported in Figure 12(a) indicate that Δt should be neither too short nor too long to ensure a high storage hit ratio.

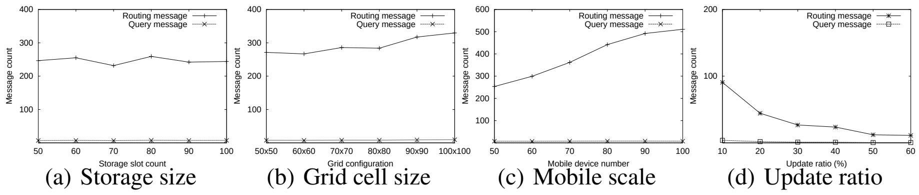
6.4 Storage Use Ratios

It is also of interest to know how much of a device's data is actually used in query processing. As for the hit ratios, we distinguish between the *local use ratio*, the percentage of the stored grid cells used by local queries, and the *peer use ratio*, the percentage used by peer queries. Simulation results are reported in Figure 10. In addition, we study the effect of parameter Δt used in our probability based replacement policy on the storage



(a) Storage size (b) Grid cell size (c) Mobile scale (d) Update ratio

Fig. 10. Storage Use Rate in MANET Simulation



(a) Storage size (b) Grid cell size (c) Mobile scale (d) Update ratio

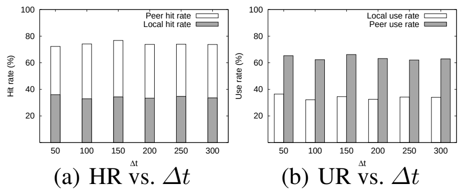
Fig. 11. Message Count in MANET Simulation

use ratio, as reported in Figure 12(b). Most results here are in line with their counterparts as reported in Section 6.3 on the storage hit ratios. Similar reasons as for the previous batch apply to these findings.

6.5 Message Counts

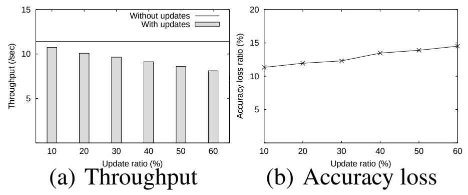
We also explore the wireless P2P message consumption of our method, distinguishing between two kinds of messages: *routing messages* and *query messages*. The former are used to disseminate routing table entries between peers, while the latter are used when forwarding queries and relevant grid cell data in the MANET. Average simulation results are reported in Figure 11. For the four sets of experiments covered, the query message cost is consistently very small compared to the routing message cost. However, the two are related: it is the extra routing messages that bring about the small number of query messages and fast retrieval of data for any device issuing queries. The extra routing messages pay off as they are utilized and amortized across the queries issued by multiple mobile devices. This study indicates that our query processing based on collaborative data sharing is efficient and robust.

From Figure 11(a), we see that the routing message cost of our proposal is insensitive to storage size variations. In contrast, the routing message cost exhibits an increasing trend as the grid becomes finer, as shown in Figure 11(b). Our storage and sharing mechanism uses grid cells as the basic units, the number of which increases as the grid granularity becomes finer. As a result, the increased numbers of grid cells involved in the storage and sharing produce more routing messages between the mobile devices. Figure 11(c) shows that the routing message cost increases almost linearly with the number of mobile devices, which demonstrates the scalability of our method. As shown in Figure 11(d), the message cost decreases as the update ratio increases. As updates tend to cause less data to be shared among peers, and as evictions due to updates do not invoke routing messages (all peers remove the relevant routing entries), the numbers of routing messages decrease. As less data are retrieved from peers, inter-device query messages decrease, too.



(a) HR vs. Δt (b) UR vs. Δt

Fig. 12. Effect of Δt



(a) Throughput (b) Accuracy loss

Fig. 13. Query Throughput and Accuracy

6.6 Throughput and Accuracy Under Updates

In this batch, we do not limit the number of queries each mobile device can issue and stick to range queries only because their search ranges can be exactly determined for accuracy concerns. We then examine the impact of updates on the system wide query *throughput*, the number of queries successfully answered per second in the simulation, and the *accuracy loss ratio*, the ratio of cells invalidated by co-occurring updates for an abandoned query. Figure 13(a) shows that the presence of updates reduces the throughput compared to the cases without updates. This is so, as updates not only invalidate queries, but also consume resources that otherwise could be used by queries. As seen in Figure 13(b), the accuracy loss ratio increases as more updates occur, but stays below 15%. These results indicate that our proposal is robust and reliable under updates.

7 Conclusion

Assuming a hybrid mobile environment within which mobile devices can communicate wirelessly with not only an application server via a slow channel, but also with peer devices via fast P2P channels, this paper proposes a collaborative and predictive data sharing scheme that exploits the P2P capabilities of mobile devices.

Based on a uniform grid, we maintain a simple yet efficient collaborative indexing structure on the server and each mobile device within the server's coverage. Each device is able to issue spatial queries, and the devices request, store, and share data in units of grid cells. In contrast to the traditional C/S mobile computing, this collaborative sharing scheme exploits the high P2P bandwidth, thus shortening query response time significantly. Special routing tables are used to direct request forwarding among peers. A predictive cost model is proposed for storage replacement and routing table maintenance on resource-limited devices. This model takes into account the predicted movement of each device when assigning to its grid cells probabilities that they are to be reused by future queries. Extensive experiments conducted on a MANET simulator, elicit design properties of our proposals, indicating that they are efficient in answering queries and robust to data updates.

Acknowledgments

The work of Zhiyong Huang, Hua Lu and Beng Chin Ooi was in part funded by A*STAR under grant no. 032 101 0026. Christian S. Jensen is also an adjunct professor at Agder University College, Norway.

References

1. JiST/SWANS. http://jist.ece.cornell.edu.

2. The R-tree Portal. http://www.rtreeportal.org.
3. D. Barbará and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. In *Proc. SIGMOD*, pp. 1–12, 1994.
4. S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, editors. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, New Jersey, 2004.
5. A. Brilingaitė and C. S. Jensen. Enabling routes of road network constrained movements as mobile service context. *GeoInformatica*, 2006 (to appear).
6. J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. MOBICOM*, pp. 85–97, 1998.
7. Budiarto, S. Nishio, and M. Tsukamoto. Data management issues in mobile and peer-to-peer environments. *Data Knowl. Eng.*, 41(2-3): 183–204, 2002.
8. C.-Y. Chow, and H. V. Leong, and A. T. S. Chan. GroCoca: Group-based peer-to-peer cooperative caching in mobile environment. *IEEE Journal on Selected Areas in Communications*, 25(1): 179–191, 2007.
9. S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. VLDB*, pp. 330–341, 1996.
10. T. Hara. Cooperative caching by mobile clients in push-based information systems. In *Proc. CIKM*, pp. 186–193, 2002.
11. H. Hu, W. S. Wong, D. L. Lee, B. Zheng, and J. Xu. Proactive caching for spatial queries in mobile environments. In *Proc. ICDE*, pp. 403–414, 2005.
12. G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proc. P2P Computing*, pp. 75–91, 2001.
13. K. Lee, W.-C. Lee, B. Zheng, and J. Xu. Caching Complementary Space for Location-Based Services. In *Proc. EDBT*, pp. 1020–1038, 2006.
14. B. Liu, W.-C. Lee, and D. L. Lee. Distributed caching of multi-dimensional data in mobile environments. In *Proc. MDM*, pp. 229–233, 2005.
15. H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu. UCAN: A unified cellular and ad-hoc network architecture. In *Proc. MOBICOM*, pp. 353–367, 2003.
16. C. Lindemann and O. P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *Proc. P2P Computing*, pp. 73–80, 2002.
17. J. Nievergelt and H. Hinterberger. The grid file: an adaptable, symmetric multikey file structure. *ACM TODS*, 9(1): 38–71, 1984.
18. Q. Ren and M. H. Dunham. Using clustering for effective management of a semantic cache in mobile computing. In *Proc. MobiDE*, pp. 94–101, 1999.
19. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. SIGMOD*, pp. 331–342, 2000.
20. Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. SIGMOD*, pp. 611–622, 2004.
21. B. Xu and O. Wolfson. Data management in mobile peer-to-peer networks. In *Proc. DBISP2P*, pp. 1–15, 2004.
22. L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. In *Proc. INFOCOM*, 2004.
23. B. Zheng and D. L. Lee. Semantic caching in location-dependent query processing. In *Proc. SSTD*, pp. 97–116, 2001.