Abstract:

The emergence of mobile computing provides the ability to access information at any time and place. The rapidly expanding technology of mobile communication will give mobile users capability of accessing information from anywhere and any time. The wireless technology has made it possible to achieve continuous connectivity in mobile environment. When the query is specified as continuous, the requesting mobile user can obtain continuously changing result. In order to provide accurate and timely outcome to requesting mobile user, the locations of moving object has to be closely monitored. Within the last decade, the continuous development of mobile technology has motivated an intense research in mobile data services. Among these services, location-based services and location-depant queries have attracted a lot of attention from the software industry as well as from researchers. The objective of paper is to discuss the issues related to location aware query processing in mobile databases.

Keywords: Broadcast, Caching, Mobile, Query, Security,

I. INTRODUCTION

Mobile databases are gaining popularity and are likely to do so well into the future as portable devices become more and more popular and common. However, the growing need for monitoring applications (traffic control, ecology and environment monitoring) has forced an evolution on data processing paradigms, moving from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs).

A Mobile database System Architecture

For any mobile architecture, things to be considered are

- Users are not attached to a fixed geographical location
- Mobile computing devices: low-power, low-cost, portable
- Wireless networks
- Mobile computing constraints

B. Mobile Database System has three parties

(i) Fixed hosts: Perform the transaction and data management functions with the help of database servers

(ii) Mobile units: Portable computers, move around a geographical region that is a collection of mobile cells

- Mobile hosts retains network connection through the support of base stations
- Role of mobile hosts depend on the capacity

(iii) Base stations: It is a two-way radio installation in a fixed location, used to communicate with one or more mobile or portable radio transceivers. They are typically used by low-power two-way radios such as mobile phones, portable phones and wireless routers

C. MDS Data Management Issues

How to improve data availability to user queries using limited bandwidth?

Semantic caching

(i) Client maintains a semantic description of the data in its cache instead of maintaining a list of pages or tuples.

(ii) The server processes simple predicates on the database and the results are cached at the client.

Data Broadcast (Broadcast disk)
A set of most frequently accessed data is made available by continuously broadcasting it on some fixed radio frequency. Mobile Units can tune to this frequency and download the desired data from the broadcast to their local cache. A broadcast (file on the air) is similar to a disk file but located on the air.

The contents of the broadcast reflect the data demands of mobile units. This can be achieved through data access history, which can be fed to the data broadcasting system. For efficient access the broadcast file use index or some other method.

D. Security
One of the most important issues in mobile computing related to security in databases is the trust between system components. Traditionally, database systems based their operation and their security on that of the underlying operating system. It is clearly that some such degree of trust is needed for resource sharing between a mobile computer and a foreign environment for the mobile device to work properly in a foreign network.

II How MDS looks at the database data?

A. Location Dependent Data (LDD)

The class of data whose value is functionally dependent on location. Thus, the value of the location determines the correct value of the data.

Example: Hotel Taj has many branches in India. However, the room rent of this hotel will depend upon the place it is located. Any change in the room rate of one branch would not affect any other branch.

Schema: It remains the same only multiple correct values exists in the database LDD must be processed under the location constraints. Thus, the tax data of Pune can be processed correctly only under Pune’s finance rule.

Location binding or location mapping can be achieved through database schema or through a location mapping table. It needs location binding or location mapping function.

B. Location Dependent Data (LDD) Distribution

MDS could be a federated or a multidatabase system. The database distribution (replication, partition, etc.) must take into consideration LDD. One approach is to represent a city in terms of a number of mobile cells, which is referred to as “Data region”. Thus, Pune can be represented in terms of N cells and the LDD of Pune can be replicated at these individual cells.

C. Location Independent Data

The class of data whose value is functionally independent of location. Thus, the value of the location does not determine the value of the data.

Example: Person name, account number, etc. The person name remains the same irrespective of place the person is residing at the time of enquiry.

III MDS Query processing

Location dependent query

A query whose result depends on the geographical location of the origin of the query.

Example What is the distance of Pune railway station from here? The result of this query is correct only for “here”.

Situation: Person traveling in the car desires to know his progress and continuously asks the same question. However, every time the answer is different but correct.

Requirements: Continuous monitoring of the longitude and latitude of the origin of the query. GPS can do this.

IV Security Issue in mobile database

Secure network connection

The mobile database and the central database have to be synchronized at specific times. The synchronization is implemented in the system software of the mobile database and is performed over the http protocol. Using http has the significant advantage of using a widely available protocol and possibly the dis-advantage that its performance may be lower than a proprietary protocol for the database synchronization operation.

Encrypted local database

The local database on the mobile device is encrypted and each time the user opens the mobile database, he has to enter his password. In case the mobile device is stolen or violated by an intruder, the data that is stored on the local database is not readable. The encryption algorithm is part of SQL Server Mobile Edition and unfortunately we were not able to find documentation for the specific algorithm. We assume that the vendor does not simply rely on obscurity and that the encryption is based on one of the established symmetric key encryption algorithms. If the build-in encryption algorithm of the mobile database is considered insufficient, it is of course possible to implement this feature within the client application.

User authentication at the database server

The synchronization of the small-footprint database that is installed on the mobile device with the central database is performed with database replication technology. For this purpose, there is an appropriate publication at the database server. A publication is the meta-data package of information about which data is replicated. The mobile database uses the publication of the database server for the synchronization operation. In order to connect to the publication an appropriate user account on the database server has to be used. This means that the application user has to be authenticated at the database server.

Authentication at the web server
As already noted, the communication between the mobile database and the central database is performed over https. At the server side the communication link is handled by a web server. Hence, it is possible to take advantage of standard web server authentication and require the user to authenticate at the web interface level. This requirement is very important since it provides protection for the mobile database agent that is executed at the server side within the web server. Without web server authentication every network user would be able to contact the server-side agent by simply using the appropriate URL.

Server-side mobile agent account

Both endpoints of the communication link are handled by mobile database agents. During a synchronization process, the agent operations on the server-side can either be executed by the default agent account of the server’s operating system or in the context of a dedicated account of the server’s operating system. We use a dedicated operating system account for the execution of the agent service. The account has been granted the minimum permissions that are necessary for its role. This decision satisfies the common security rule of granting minimum sufficient permissions.

Separate user accounts for the authoring and the read-only application In case a user has to use the application both as an author of announcements and as a reader of all announcements we can either assign two accounts to the user, an authoring account and a read-only account, or grant both functionalities to a unique user account. Even though the security of the application would not be lowered by using a unique account, we preferred to use two separate, dedicated accounts. This approach reflects in a more natural way the structure of the application.

Application provided security

For authoring operations, each user has access only to his own data. A set of database triggers implemented in the database server, check that the data manipulation operations of the user are valid. This check prevents all users from accidental or malicious modifications of data for which they have no authorization. More precisely, an author can create new announcements that are signed with his name, can delete or update announcements that are signed with his name, and has no access to announcements created/signed by other users. The above functionality resembles in a loose sense the virtual private database technology (VPD) of Oracle [3].

The read-only client

The read-only part of the MDA is implemented as a separate client application. The read-only client provides access for viewing all announcements. We apply certain techniques to assure the security of the central database:

- The publisher of the database server that is used for the synchronization of the read-only application is defined to be read-only. Consequently it is not possible to apply any modification to the central database from the read-only application.

- Read-only clients have no access to the main table of the central database. Instead the read-only clients read the announcements from a replicated instance of the main table. A set of database triggers implemented in the database server keeps the replicated table always updated. In case an accidental or malicious modification of the data in the replicated table would occur, it would have no effect on the main table of the application.

Communication between the servers

The announcements are also available over http as a web page. A dynamic web page with aspx code gives a list of the announcements. The web server must have access to the database in order to read the data. For this reason we have to deal with a common security issue in database-driven websites: Choosing the appropriate database account that the web server is using to access the database. We created a specific account in the database that has only one permission: To perform a select on the replicated announcements table. This decision too, applies the principle of granting the minimum sufficient permissions.

Client-side data encryption

We also tested a common but very important feature, that of encrypting the user data in the database. Even though this feature is not directly relevant to the announcements application, we consider it very important for secure mobile database applications and more generally for secure database applications. The user gives a password to the client application and all his critical data is encrypted at the client-side before it is permanently stored in the database. This encryption guarantees the confidentiality of the data against any database user including the local database administrators. The approach is very simple: The client application applies a symmetric key encryption algorithm, for example AES, and stores the encrypted data into the database. When the user reads the data, he provides his password and the data is decrypted. We verified this approach and it works transparently as soon as the user has given his password. A shortage of the current mobile platform was that some library functions, like for example the function "PasswordDeriveBytes", were not provided by .NET Compact Framework v2.0. We overcame this problem by providing a hand-coded implementation of the required function that was absent.

V Broadcast Algorithms

To partition data among on-demand and broadcast channel various algorithms have been developed. We study a wide range of broadcast algorithms for query dissemination in two major categories: pull and push algorithms.

A. Pull Algorithms

In a broadcast pull environment, responses to requests are transmitted on a dedicated shared broadcast channel. Following are the specific algorithms that we study:
First Come First Served (FCFS) [13] - It broadcasts the data items in the order they are requested. To avoid redundant broadcasts, a request for a data item that is already in the queue is ignored.

Longest Wait First (LWF) [12] - This schedules the data item that has the largest total waiting time, that is, the sum of the time each pending request for the item has been waiting for. We implement LWF using three values for each unique item requested: (i) the number of outstanding request for the data item, (ii) the last recording of the total waiting time and (iii) the time of this recording. For instance, at time 0, when a data item is requested for the first time, the three fields of its service queue entry are set to (1, 0, 0). At every subsequent request at time,”t,” this entry gets updated. We repeat the update process for every new incoming request and also at the time of scheduling decision to avoid using stale total waiting time values.

PULL RxW [11] - This algorithm selects the data item with the maximal RxW value where R is the number of outstanding requests for that item and W is the amount of time that the oldest outstanding request for that item has spent in the service queue. Using two threaded lists that keep the service queue entries sorted in R and W order, the scheduling decision time is significantly reduced.

PULL RxW.a [11] - It is a modified version of RxW algorithm and reduces the scheduling decision time further. The algorithm broadcasts the first item it encounters whose RxW value is greater than or equal to α times the current threshold value. The threshold is computed as the running average of the RxW value of the last item broadcast and the previous threshold. After each broadcast decision, the threshold is updated accordingly. The scheduling overhead decreases as alpha decrease. The most commonly used settings for the approximation algorithm are PULL RxW.90, (α = 0.9), and PULL RxW.0, (α = 0). RxW.0 compares only two items, the item with the most outstanding requests and the one that has been waited for the longest time. While RxW.90 compare every item to be broadcasted.

B. Push Algorithms

The fundamental push algorithms are Broadcast Disks [8] and Teletext System [10]. In these schedules all items being accessed are partitioned into a number of disks. Items on the smaller numbered disks are broadcast more frequently than those on the larger numbered disks. We use the optimized parameters for Broadcast Disks assuming perfect knowledge of access probabilities.

PUSH MAD [8, 10] - It selects the data item whose access probability is highest since last broadcast. In the extreme case, MAD requires examining all items in the database before making a scheduling decision. In the experiments MAD is run twice; the first run records the schedule generated and the second run use this recorded schedule for taking measurements. This results in an optimistic off-line algorithm with the highest throughput.

VI Semantic Caching

The driver defaults to a simple per-statement caching with a FIFO queue. However, the frame-work provides two extension points for application programmers to change this default behavior.

Cache Strategy

The cache strategy controls the way the driver decides what to store in the cache. The second one deals with cache replacement – the way the cache is kept-up-to-date. Both extension points are implemented as java interfaces (cf. Listings 1 and 2) and are set during the driver’s initialization phase.

The first extension point provides a way to customize the caching strategy – the way the driver decides what to store in the cache. The second one deals with cache replacement – the way the cache is kept-up-to-date. Both extension points are implemented as java interfaces (cf. Listings 1 and 2) and are set during the driver’s initialization phase.
A preemptive cache example: To show the capabilities of this interface, let’s have a look at how the included preemptive caching strategy is implemented. It uses the idea of expanding the column selections of the SQL statement to include all columns, leaving the other clauses untouched (i.e. a SELECT a, b, c FROM xyz becomes a SELECT * FROM xyz).

First, the processQuery method uses the FROM (and following WHERE, GROUP BY . . . ) clause as cache index key. It simply cuts the front of SQL statement off and returns the part beginning at the FROM identifier (i.e. a SELECT a, b, c FROM xyz becomes xyz). When the entry is not found, the cacheMiss method simply takes the result of the processQuery method and adds a SELECT * to its beginning, thus creating a valid SQL query which is then sent to the database server. Finally, the postProcessQuery method creates a new result set based on the original query. For each column definition in the original SQL statement the corresponding column from the database result set is selected and added to the new one. Afterwards, eventual renaming is applied where necessary (as defined by SQL AS clauses). Finally, the new result set is returned to the client.

Of course, this kind of caching can fail quite easily, for example with the use of SQL functions or joins. However, the application developer knows the properties of the strategy used in his application and can thus write the SQL statements in a more robust way. Also, there is always the option of not using caching for some statements. Furthermore, most problems can be avoided with a more intelligent SQL analysis. However, please note that the current version of MIDP does not implement any regular expression engine, a developer must hence implement his own, a comprehensive undertaking in it’s own.

Cache Replacement Strategy

The cache replacement strategy controls the actual content of the cache. This is done via the three methods listed in Listing 2. The sessionStart and sessionEnd methods are both only called once during the drivers lifecycle, namely during the initialization and termination phases, respectively. The newEntry method on the other hand is called every time the cache content is to be modified.

During the initialization phase of the driver, the cache content index is read from the RMS by the cache manager and is then passed to the sessionStart method of the cache replacement strategy. This method identifies the obsolete cache entries and returns them in order to be deleted. Similarly, the sessionEnd method is called during driver shutdown, indicating whether or not the cache should be cleared or not. Overall, the job of these two methods is to move the cache into a valid state. For instance, because the default FIFO strategy indicates a cache flush during driver shutdown, it also marks the entire cache content for deletion during initialization.

The task of the newEntry method, on the other hand, is to manage the cache content. It is called whenever a new entry is to be added to the cache and identifies at most one cache entry which is to be removed from the cache. The default FIFO strategy, for instance, does return the oldest cache entry once the cache reached a predefined size of ten entries.

VII Conclusion

Developing a secure mobile database application is an important task. Our experience with developing and testing the application is satisfactory in several aspects. The efforts to implement the mobile database application were reasonable, it works reliably and it is efficient and user-friendly. For the security of a mobile database application, our case study showed that there are sufficient tools and techniques available to provide a security level comparable to the security level of conventional platforms. The few shortages that we faced are most likely technical issues that should be overcome in the forthcoming versions of the system software of the mobile platforms. Finally, an important issue is the lack of appropriate documentation for certain encryption algorithms that are used within the system software of mobile platforms.

This paper presented our caching framework for mobile devices. It is an extension for the MyMIDP database driver for MIDP 2.0 compatible devices. As mentioned in Section 1 we had to consider device and API limitations during the design and development process. The given goals were reached. Our current development version of the MyMIDP driver (incl. the described cache implementations) has a footprint of 27kB, only. The caching framework provides flexible and transparent data caching for database result set.

References

4. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In 28th Int’l Conf. on Very Large Databases (VLDB), Hong Kong, 2002.


