Mobile Devices Synchronization

Vidya N. Kawtikwar
IT department
P.V.P.P College of Engineering
Sion(E) Mumbai, India
vidya.kawtikwar@rediffmail.com

Ramesh Sahabade
Computer Department
Terna Engineering College
Nerul, Navi Mumbai, India
rvs2002@rediff.com

Abstract- The synchronization process in mobile devices allows you to execute bidirectional updates on the required data. Any changes that have been made on the client device can be transmitted to the server database, and any changes on the server can be transmitted to the client device. In this way, you can keep the data on the client and the server synchronized. In this paper, different techniques to perform synchronization in mobile devices are discussed in detail.

Keywords*-Mobile Device, Mobile Database, Synchronization, server database*

I. INTRODUCTION

Recent advances in mobile technology and equipment have led to the emergence of a new computing environment and a variety of small sized mobile devices such as PDAs (personal digital assistants), smart mobile phones, HPCs (handheld PCs) and Pocket PCs have been popularized. As various network technologies are increasingly being associated with such mobile devices, the processing of business information can be available using mobile devices. As a result, business models that rely on mobile technologies are appeared. Mobile devices do not have much computing power and rely on batteries. Additionally, constant access to network is difficult due to narrow bandwidth. Therefore, it is not easy to process a large size of stored data and maintain a continuous connection with the server side database. For these reasons, mobile devices have mobile databases in order to achieve stable data processing. Mobile devices download replications of limited data from a connected server-side database using a synchronization device that has a stable wire communication function. Mobile devices process various tasks using the data downloaded in an offline state. The work on the network disconnected condition is a crucial point for mobility support.

In a disconnected environment, there are inevitable inconsistencies between the server-side database and the mobile database. Synchronization techniques can solve the data inconsistencies and guarantee the integrity of the data. Consequently, synchronization is an essential subject in mobile device computing environments. Commercial DBMS venders offer various solutions to data synchronization in a mobile environment. However, these solutions are not independent of the server-side database because they use database dependent information such as metadata or use specific functions of server-side database such as trigger and time stamp. In other words, the mobile database vender should be equivalent to the server-side database vender. The solution of operating a separate synchronization server in the middle tier is independent of the server-side database but dedicated to the mobile database. That is, the synchronization solution and the mobile database should be the identical vender product. Because of these restrictions, the extensibility, adaptability and flexibility of mobile business systems are markedly decrease. This problem must be solved in order to build efficient mobile business systems because upcoming mobile environments will have heterogeneous characteristics in which diverse mobile devices, mobile databases, and RDBMS exist.

II. SNNCHRONIZATION ARCHITECTURE

Synchronization is most often implemented in a distributed computing architecture with a client layer, a middle-tier layer, and an enterprise data layer. Each layer can be implemented using varying techniques, all aimed at accomplishing the same goal: providing a way to extend enterprise data to a variety of mobile devices.

Figure.1 shows the overall synchronization process.

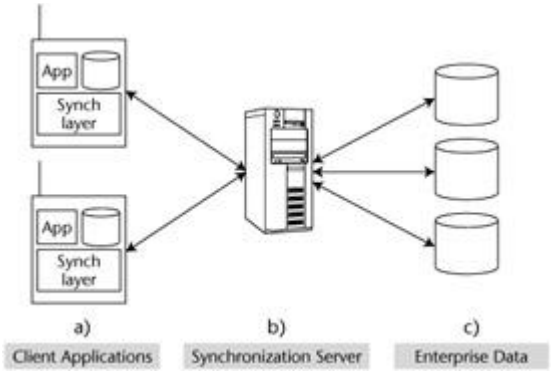

Figure. 1: Synchronization architecture

A. *Client*

When synchronizing data between an enterprise server and a persistent data store on the client device, a synchronization layer must be present to manage the two-way data communication .Ideally, this layer will have a minimal impact on your client application, while still

providing a simple, easy-to-use client API for controlling the synchronization process. By implementing a modular, self-contained synchronization layer, you can control the entire synchronization process with little interaction from the client application. In some cases, all that is required from the client is the invocation of the synchronization process; the synchronization layer does everything else from there.

Because so many client devices are on the market, the synchronization client must have support for the leading mobile devices, including laptops, Windows CE devices, Palm OS devices, Symbian OS devices, as well as specialized devices with add-on features such as barcode scanners and other industrial components. Each of these devices can have a different mobile operating system with different network protocol support. The synchronization layer on the client takes care of the network communication from the device back to the synchronization middleware.

B. Middleware

The synchronization server is where the most of the synchronization logic is contained. Figure.1 illustrates the role of the synchronization server in relation to the other components of the synchronization architecture. This server is responsible for communicating with the client application to send and receive required data packets. In order to do this, it has to be able to communicate over a protocol that the client application understands. Most of the time, this protocol is IP-based, and often is HTTP. When the synchronization server receives the data from the client, it then has to execute the synchronization logic to determine how this data is transferred into the enterprise data source.

Many of the advanced synchronization features are implemented within the synchronization server. Some of these features include data, conflict detection and resolution, data transformation, data compression, and security. All of these features have to be implemented while still maintaining server performance and scalability.

Two common synchronization server implementations exist: as a standalone server application or as Java servlets running in a servlet engine. Both of these methods have benefits and drawbacks. The stand alone synchronization server is convenient because it does not require any additional software to execute. These servers are usually programmed using the C programming language, taking advantage of OS-level calls, leading to enhanced performance. This also means that the server has to be available for the operating system to which you are deploying or you are out of luck. In terms of scalability and availability, the server can either have its own built-in load balancing and failover mechanisms or use third-party load balancing solutions, such as the hardware-based systems provided by Cisco systems.

For Java servlet-based synchronization servers, you will need a servlet engine for deployment. Since J2EE application servers are now commonplace in most organizations, this requirement does not usually pose a problem. By using an outside servlet engine, the performance, scalability, and availability of the synchronization server now rely on the capabilities of the application server/servlet engine being used. The same goes for the server operating systems that are supported; that is, as long as the servlet engine works on a given platform, the synchronization servlet should work as well. That said, you should give extra consideration to the synchronization vendor's supported platforms and recommended application servers when deciding which application server and operating system to use.

C. Enterprise Integration

The final part of a complete synchronization solution is the enterprise integration layer. While this layer is often part of the synchronization server, we are discussing it separately because it provides different functionality. The enterprise integration layer enables you to communicate with various backend data sources. If you are using a commercial mobile relational database on the client, you will most likely have integration to enterprise relational databases on the server using ODBC, JDBC, or native drivers. In addition to providing integration to relational databases, you may also require access to other forms of enterprise data, such as ERP systems, CRM systems, or XML data. If this is the case, you will have to look for additional enterprise adapters for the solution you are implementing; or if you have the resources, you can create your own adapter.

III. DATA SYNCHRONIZATION PROCESS

The basic synchronization process is explained as shown blow in Figure.2

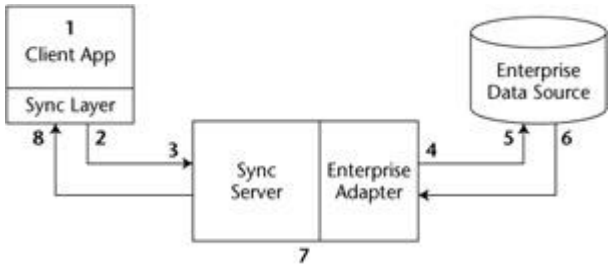

Figure.2: Basic synchronization process.

1. The application user can initiate the synchronization process manually, or it can be programmed into the application. At the point where the synchronization is initiated, the data that has changed since the last synchronization is prepared for sending to the synchronization server. This data preparation often involves compressing the data and, optionally, encrypting the data.
2. Once the data is prepared, a connection is established with the synchronization server. At this point, the user is usually authenticated with the server; the data packets are then sent over the communications network (wireless or wireline) to the synchronization server.
3. The synchronization server receives the data to be synchronized. It then uses the synchronization logic to determine whether the data needs to be transformed before it is sent to the enterprise data source. If it does, transformation can occur at this time.
4. The enterprise adapter provides integration to the enterprise data source. This adapter may simply be an ODBC or JDBC driver for enterprise relational databases, or custom code for other more complex data sources.
5. Using the appropriate enterprise adapter the synchronization server can authenticate the user against the enterprise data source (optional) before it starts the data transfer. Once the user is authenticated, the server can update the enterprise data source with the changes from the client application. At this time, the synchronization server can also detect if there are any conflicts in the data being updated and, if there are, take appropriate action.
6. After the update has been committed, the relevant changes that took place on the server since the last synchronization are prepared for sending back to the client application.
7. The synchronization server takes the enterprise data source's changes and performs any transformation that may be required before sending the updates to the client application. Again, this data is usually compressed and possibly encrypted for additional security.
8. The data is sent to the client application, where it is updated in the mobile data store.

.

IV. SYNCHRONIZATION MODES

Synchronization modes are as follows:

A. Snapshot

Snapshot synchronization makes it possible to move large amounts of data from one system to another. It involves deleting a table on one system and copying a table from another system in its place. The result is two tables with identical data sets.

This form of synchronization is used when there are moderate to large amounts of data that need to be synchronized or when data is not changed at the remote location. Additionally, due to the large amount of data being transferred, it is best suited for reliable networks with high bandwidth.

Snapshot synchronization is a suitable candidate for instances where a complete, or nearly complete, set of data has to be transmitted. An example might be a salesperson who requires updated product catalogues or price lists. The main office can send these updates via a snapshot so that all the data is updated to the remote worker at once. Since the remote worker will not be making changes to the remote product list, no data will be lost.

B. Netchanges

For most mobile applications, a net change mode of synchronization is more efficient than using snapshot synchronization. In these cases, only the changed data is sent between the remote and enterprise databases, saving network bandwidth and reducing connection times. In order to accomplish this, the remote database keeps track of the original data and the most recent change that has occurred since the last synchronization. In this way, even if the data were to change several times on the client, only the original data and most recent data have to be transmitted to the server. The changes to the enterprise data source can then happen in a single transaction since there is only one update to be made. The same approach is taken when the server database is updated and synchronized to the remote database.

This form of synchronization is well suited for applications in which the user may synchronize several times in a day, sending only small amounts of data each time. It is ideal for coping with the bandwidth limitations on most of today's wireless networks. It is also a good technique for situations in which larger amounts of data are synchronized less frequently.

The net change mode does have one drawback: If you require knowledge of each individual transaction that occurred to the data, then keeping track of only the most recent change may not be suitable for you. In this case, you might find a transaction log-based synchronization approach to be more appropriate.

V. SYNCHRONIZATION TECHNIQUES

The synchronization solutions available today use many techniques to move data from client applications to enterprise servers. Different synchronization techniques are as follows

A. Clock synchronization

Clock synchronization is a hindrance from computer science and engineering which deals with the indication that internal clocks of several computers may vary. Even when primarily set precisely, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. There are several hitches that occur as a consequence of rate variances and several solutions, some being more suitable than others in certain situations. In serial communication, some people use the term "clock synchronization" just to deliberate getting one metronome like clock signal to pulsation at the same frequency as another one frequency synchronization and phase synchronization. Such "clock synchronization" is used in synchronization in telecommunications and instinctive baud rate detection.

B Event based synchronization

A synchronization method and apparatus describes event objects to permit synchronization of execution units (e.g., threads). In one procedure, the synchronization method and apparatus is used in aggregation with a UNIX operating system. By describing event objects on which threads or other execution objects can wait upon, multiple threads can wait on one event, or otherwise, one thread can wait on multiple events. Besides, using the event-based synchronization method and device, it is conceivable to specify behavior, mainly when one thread or other execution object waits on multiple events. For example, the performance indicated can be that a condition is gratified if any of the events occur, if all of the events occur, or some other logical combination of events occurs.

C Non-block synchronization algorithm

Java provides supports for additional atomic operations. This allows to develop algorithm which are non-blocking algorithm, e.g. which do not require synchronization, but are based on low-level atomic hardware primitives such as compare-and-swap (CAS). A compare-and-swap operation checks if the variable has a certain value and if it has this value it will perform this operation. Non-blocking algorithm are usually much faster than blocking algorithms as the synchronization of threads appears on a much finer level (hardware).

D. Default synchronization algorithm

The default synchronization algorithm starts when an attention identifier (AID) key is pressed. An attention identifier (AID) key is any key that generates a presentation space update. Primarily, the state of the terminal is UNINITIALIZED. The procedure waits for a period of time for updates to the presentation space. User can modify the wait time in the Timeout field in the preferences window. The nonappearance wait time is 1200 milliseconds. If Timeout is set to 1200 milliseconds, and an update arises during the last 600 milliseconds, the process waits for an additional 600 milliseconds for extra updates. During this extra wait period, added update occurs during the last 300 milliseconds, the algorithm waits again for another 600 milliseconds for further updates. This continues until no updates are received during the last half of the last additional time period. At this point, the state of the terminal is either LOADED (keyboard locked) or READY (keyboard unlocked), reliant upon the OIA status.

E. SAMD algorithm

SMAD synchronization algorithm based on message digests for synchronizing between server-side databases and mobile databases. The SAMD algorithm is performed with only SQL functions of relational databases, so that it is not dedicated to particular venders and is available for use in combination with any server-side databases and mobile databases. Therefore, extensibility, adaptability and flexibility are guaranteed when a mobile business system is authorized. This feature is important in order to build efficient mobile business systems because the upcoming mobile business environment has heterogeneous characteristics in which diverse mobile devices, mobile databases and RDBMS exist.

The SAMD synchronization algorithm must keep the following restrictions.

1) Every database table must have a primary key.

2) The primary keys of the data table and the message digest table have an identical value for a given row.

3) A new row is inserted into the mobile database and another one into the server-side database; the primary key values of the two rows cannot be identical.

VI. CONCLUSION

There are different techniques to perform synchronization in mobile devices. Each of the techniques provides value for certain situations, but not every technique will be right for your particular application .When implementing your synchronization layer, keep these techniques in mind to ensure that your solution is as efficient as possible for your application.

REFERENCES

[1] Gye-jeong kim, seung-cheon baek, hyun-sook lee, han-deok lee,moon jeun, Joe (2006), "LGeDBMS: A small DBMS for embedded system with flash memory", 32nd international conference on very large data bases, pp.1255-1258,.

[2]. Joshua savil, (2008), "Moblink Synchronization Profiles', A white paper from Sybase iAnywhere, October 17.

[3]. Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik, (2009), Life Science Journal 2012;9(3) "A synchronization algorithm for mobile devices for ubiquitous computing"

[4]. Mi-seon choi,young-kuk kim, juno chang (2008),

"Transaction-centric split synchronization mechanism for mobile E-business applications", April 20th.
[5] Mi-Young Choi, Eun-Ae Cho, Dae-Ha Park, Chang-Joo Moon, Doo-Kwon Baik (2010), "A Database Synchronization Algorithm for Mobile Devices", Vol. 56, No. 2, May 2010.
[6] Santashilpal Chaudhuri, Amit kumar saha, David B.Johnson (2007), "Adaptive clock synchronization in sensor networks", March 31st.
[7] Xianzhong tian;younggang miao; Tongsen HU;
Bojie fan; Jian pian; wei xu (2009), "Maximum likelihood estimate based on time synchronization algorithm for wireless sensor networks".
[8] Ziad itani, Hassan diab, Hassan Artail (2005), "optimistic pull based replication for mobile devices".
[09] Dr. Venkatesh. J, Aarthy. C " An efficient database synchronization for mobile devices using SAMD algorithm", Life Science Journal 2012.

