mFinity
Enterprise Mobility Management Platform

Data Integrity in Mobility

mFrontiers Inc.
Subsidiary of DBValley Corp.

The Need for Offline Data Access

The blending of personal and business use of IT devices is driving an increasing diversity of smartphones and tablets into the business enterprise workspace. This entices business to move corporate business applications and enterprise data accessibility beyond the desktop and on to mobile devices.

There are situations in which a mobile application (mobile app) may require access to data where wireless communication is not available, such as in a hospital, an electrically noisy industrial facility, a remote field location, or a shielded building. Wireless networks are not at 100% coverage and probably will never get there. Even if wireless communication is available, it can be unstable. During the first half of 2013, overall problems associated with placing calls, messaging, and data-related activities averaged 11 problems per 100 calls (PP100) for full-service carriers.[i]

For an enterprise mobile app to support offline data access and maintain data integrity, the developer must consider many challenges to synchronizing the data between the server and multiple mobile devices.

The Data Synchronization Problem

Maintaining synchronization of multiple instances of data in a distributed computing environment has long been a challenge to Information Technology. For many organizations, these challenges are further complicated by the need to synchronize data between numerous types of mobile devices. Currently, the mature solutions to this problem, developed for distributed computing, are simply not available or are not practical on mobile devices.

Here are just some of the issues that a mobile app developer must consider:

- Transport Related
    - Messaging convention or communication protocol
    - Serialization format
    - Security during transport
- Local Persistency Related
    - Storage solution – SQL, NoSQL, Filesystem
    - Security - encryption
- Synchronization – Internal to mobile application
    - Between display and local database
- Synchronization - External to application

- Relationship Model (ex: Master-Slave or Multi-Master)
- Conflict resolution when update collides
- Network interruptions and availability
- Concurrency and threading

Within the space of mobile devices, there are two main approaches to addressing all the above issues. One approach is at the application level, by embedding the remote synchronization code as part of the mobile app and having corresponding code on the server side application. The second approach is at the database level, with the mobile app writing to a local database and having middleware handle synchronization of the local database to the server database.

Application Level Data Synchronization

At the application level, the aforementioned synchronization issues must be addressed as part of the application or application framework. Not all Enterprise Mobility Management (EMM) platforms offer offline application development support, but among the ones that do, this is the most popular method of providing offline support.

Figure 1 illustrates a typical implementation of this strategy.

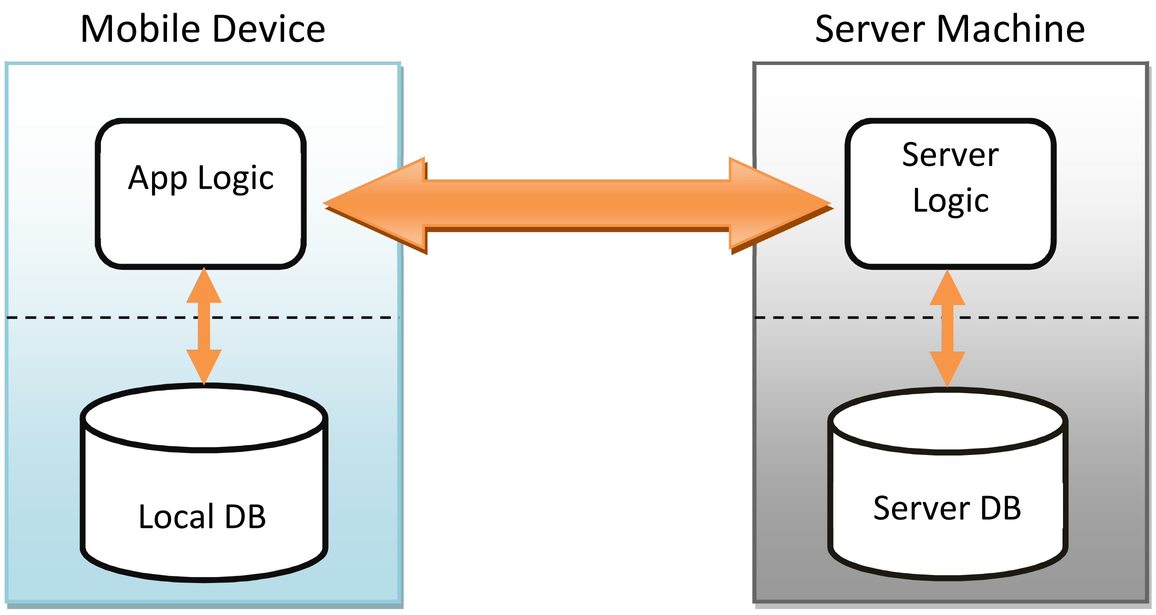

Figure 1. Application Level Data Synchronization

More advanced EMM platforms offer a set of APIs to address the mobile app's data synchronization needs. They provide a different API for each type of application being developed; Android (Java), iOS (Objective C), or hybrid applications using JavaScript. These platforms also usually provide services such as encryption, multi-user support, and search functionality.

Usually, these EMM platforms offer tools like “wizards” to create the interface classes for the developer, but the developer is responsible for creating the implementation classes manually. For example, the platform will generate interface classes/methods to handle data update collisions, but the actual logic and algorithm must be coded by the developer. The developer must also implement fallback measures for when the network connection fails during a transaction.

Examples of implementation class methods the developer must code are:

- get
- add
- replace
- remove

Sample code:

```
function getUser() {
  return { userList : [{name: ‘carlos’, age: 100}, {name: ‘arnonld’, age:49}]
};
 }

function addUser(data) {
  return Logger.debug(‘Got data from Store to ADD: ‘ + data);
}

function updateUser(data) {
  return Logger.debug(‘Got data from Store to REPLACE: ‘ + data);
}

function removeUser(data) {
  return Logger.debug(‘Got data from Store to REMOVE: ‘ + data ‘‘);
}
```

SOAP is one commonly used approach to share data between applications. Its transport format is XML, which tends to be verbose and quite large. For a thick client or a desktop PC, this is acceptable. However, for more resource constrained mobile devices, pairing JSON with a REST API is a more logical option. JSON’s serialized data will be less verbose than XML, making it lightweight and less process intensive for the mobile device. For this reason, the most widely implemented transport method for mobile devices is REST + JSON.

Drawbacks of Application Layer Data Synchronization

Although some support and guidance is provided by EMM platforms that implement data synchronization at the application layer, there is still a lot of manual code development required, and the developer must possess theoretical knowledge and actual implementation know-how to do data synchronization in a distributed environment. Reading and learning the API documentation for a particular product also takes away from the time that the developer could spend coding the actual business logic. Although using a platform is better than developing a fully custom synchronization solution, it requires considerable education and manual coding to provide offline data access and synchronization. The custom code will require more testing and increases the chance of errors that could lead to data integrity issues.

Database Level Data Synchronization

Much like at the application level, the synchronization issues identified above still need to be addressed at the database level. However, the application developer does not have to develop the classes or code logic themselves, because they are provided by the synchronization software. This opens the door for all kinds of low level optimizations that can benefit the mobile app.

Figure 2 illustrates a typical implementation of this strategy.

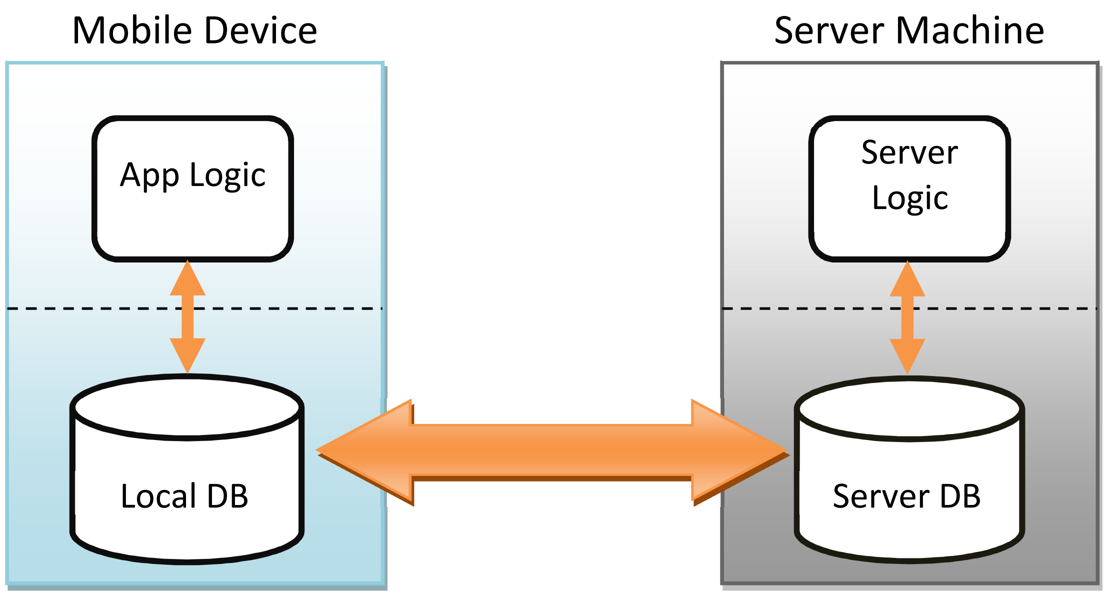

Figure 2. Database Level Synchronization

One such optimization is taking advantage of low level data storage interfaces. Most enterprise quality databases have a variety of database access APIs for the application developer. These range from query dialects to database abstraction interfaces (e.g.. ODBC) to high level language interfaces. All of these incur a performance cost for using them. In vendor provided database level data synchronization, the sync can happen below these higher level interfaces, thus improving performance. In addition to this, the application developer does not need to learn or understand those low level data interfaces. Often times, they are not even permitted access to them.

Another example of an optimization is a custom communication protocol can be used to move the data from the local database to the server database. This allows for efficient compression of data, efficient storage of data in packet payloads, and encryption at the lowest levels. The communication protocol can be exposed to the application developer so they can add in their own customizations, such as custom encryption, or a custom file based transport.

Oracle Database Mobile Server (DMS)

Database Mobile Server provides the infrastructure to develop, deploy, and manage mobile apps for the Oracle RDBMS. It is highly specialized software that takes advantage of low level optimizations. It is the most efficient software tool to move your data between the local database and the server database. It uses a publish/subscribe model that is layered on top of an Oracle RDBMS schema. In the next several paragraphs, 3 key areas of DMS will be covered, namely, data subsetting, conflict detection, and synchronization.

Using the publish/subscribe model, permits data subsetting down to the row and column level. Data subsetting addresses the problem of what to do when different mobile users require different data. The two key reasons for doing this are security and application specific needs. Data subsetting uses a parameterized view as part of the subscribe model to determine what data a given user has access to.

DMS also provides advanced conflict detection and resolution. This addresses the problem of what happens when 2 mobile users change the same data rows and try and sync their data at the same time. It also handles the case where synchronization is disconnected and both the server and client made different changes to the same data. . DMS provides two built in rules for handling conflict and those are “Server side wins” and “Client side wins.” It uses internal versioning to detect conflicts. In the “Client side wins” approach, the client side changes will get applied to the server. The opposite is true in “Server side wins”. DMS will change record inserts to updates where appropriate without the need for an administrator to intervene. If neither of these prebuilt rules will do the job for an application, then DMS allows for custom rules. The custom rules can used to peek at the actual data in the records to determine how best to resolve a conflict. For example, a record may have a status column and once that is set to a certain value no more changes are allowed regardless of where those changes originate.

DMS provides bi-directional synchronization capability that can be initiated either manually or automatically. If done automatically, the mobile app can setup conditions or events that trigger the sync. Since it is tightly coupled with the Oracle database, it can have either client or server initiated syncs. This is a significant advantage over application level sync because a device level monitoring agent does not need to be developed.

An advantage over application level sync is DMS offers a wide range of synchronization models. Because the Oracle RDBMS provides change level tracking, DMS can offer a fast sync model where only changed records are transmitted to and from the local database to the server database. In addition to fast sync, it offers a complete refresh model, and a queue based model. For each of these sync models, the application developer can define priority levels on the application data and the highest priority data is synchronized first. This capability is not typically available in application level data sync.

Database level synchronization provides significant advantages over application level synchronization. Tight coupling with a feature rich RDBMS creates an opportunity to take advantage of its features and provide an extensive option set for the mobile application to work with much lower latencies and improved performance.

mFinity – Database Level Synchronization

The mFinity™ Enterprise Mobility Management Platform (EMMP) supports development of mobile apps that require offline data access and automatic server and mobile device synchronization once communication is available. mFinity delivers this functionality through tight integration with Oracle Database Mobile Server at both the server side and on the mobile client side. mFinity provides database level synchronization to ensure data integrity. In addition to facilitating synchronization, mFinity simplifies the mapping of data between the host and mobile databases.

Conclusion – Data Integrity in Mobility

Effective data governance in the world of enterprise mobility requires policies and procedures that ensure data integrity. Mobile apps that require offline access to enterprise data will have the highest level of data integrity when data synchronization is performed at the database level rather than the application level. The mFinity Enterprise Mobility Management Platform, tightly integrated with Oracle Database Mobile Server, uses database-level synchronization to provide the highest level of data integrity for mobile applications. mFinity is the platform of choice for effective data governance in enterprise mobility.

Authors

Michael Brey is Director, Embedded Database Technologies, for Oracle Corporation, michael.brey@oracle.com

Kyoungin Park is CTO, for mFrontiers Inc, kipark@mfrontiers.com

[i] PRNewswire Press Release, "J..D. Power Reports: Overall Wireless Network Problem Rates Differ Considerably Based on Type of Service," August 29, 2013.