Synchronization and replication in the context of mobile applications

Alexander Stage (stage@in.tum.de)

Abstract: This paper is concerned with introducing the problems that arise in the context of mobile applications with respect to data synchronization and replication. Existing solutions of data replication within databases are described as well as their correspondence to data synchronization. An advanced topic in contrast to traditional mobile computing, as there is peer to peer mobile computing, is also presented as well as its specific requirements and a way of replication within such environments. There is also a discussion of the standardization efforts in the field of data synchronization mechanisms and protocols for mobile devices.

1 Introduction

Data replication and synchronization have been topics of research for quite some time in the area of databases and distributed databases. Through the advent of mobile computing the results of this research have to be applied to a new area of application. Before going into details about synchronization and replication, two terms that are strongly connected to each other, a basic introduction into mobile computing has to be given.

1.1 Mobile Computing

Mobile computing gives users the chance to access data that is stored in a stationary database or some other data repository at any time and any place. This is also known as ubiquitous data access, which is realized through the use of modern communication links like wireless networks. Typical mobile devices are laptops, but also smaller devices like PDAs or smart cell phones. Mobile computing is regarded by many researchers as the worst case of distributed computing, since it is constrained by many factors, that have to be faced and overcome in order to provide reliable and trustworthy applications to end-users. The main constraints, that have to be taken into account in the context of this paper are:

- Resource-poorness of mobile devices compared to static computers
- Mobile connectivity is highly variable in performance and reliability. These factors are responsible for times of disconnected operations

This kind of users, also known as nomadic users, require local copies of their important data items, since they are often disconnected from their home network.[1][2][3]

When taking a look at the types of communication models that are being used in the mobile computing environment, the client-server model (in its different versions) and the peer-to-peer model can be distinguished. Within these models the participating units play different roles. In the client-server model, the mobile unit plays the role of the client, that requests a service from another computing system, which plays the role of the server, that is located at the fixed network. In the latter case, there is no distinction between clients and servers, all participating units are allowed to take in the role of a server or a client.[6]

1.2 Why Synchronization and Replication

The following scenario will show, how mobile computing, synchronization and replication are linked to each other.

Imagine a sales person, who is travelling from customer to customer and is collecting orders from them. The sales person is collecting all data on a laptop where he is also able to access data that indicates how long an order will take to be delivered and provides the possibility to calculate customer and order specific conditions. When the sales person is at a customer site, most of the time communication between the laptop and the central sales person's corporate database, where all required data should be entered or accessed, is not possible. This is where data replication comes in. As already stated, the needed data has to be copied onto the sales person's device in order to provide the required functionality. On a daily basis the sales person needs to send the new orders to the central system by any means of communication link or medium. When reconnecting to the corporate database or application the two data sets need to be synchronized.

That way, mobile devices become mobile databases as [8] understands them. Mobile databases appear and disappear over time in a network of connected and stationary databases as well as other mobile databases.

1.3 Topics covered and their dependencies

In the following, first data synchronization techniques will be covered, before data replication techniques within databases will be described. This order is applied, since the discussion of data replication techniques will refer to concepts of data synchronization. Therefore it is necessary to gain some insight into data synchronization, before being able to fully understand all aspects of data replication.

2 Synchronization Techniques

Data items that are located within a database, a file system or in memory as a variable within a program, can be modified by processes. If only one process at a time is trying to modify a data item, there is no need for synchronization. If more than one process at a time tries to modify a data item, it is necessary to ensure that no two processes access the data item at the same time. Accesses to the data item are put into a sequential order, which is called process synchronization.[5]

In distributed systems, identical data items may be managed at physically distributed places or machines. These data items can be manipulated independently from each other, resulting in different versions These logically connected data items need to be merged back into a consistent state. This process is called data synchronization. Data synchronization is at the centre of the following discussions and is meant by the term synchronization.[5]

2.1 Aspects

Synchronisation can take place between two or more systems, namely in a 1:1 or 1:n relation. That means that there is always one active system that starts and controls the synchronization process. The active system can be any of the participating systems.

Synchronization can be triggered manually or by certain events, which requires? an event notification mechanism. A system can explicitly inform another system that it wants to synchronize with it, which is called push synchronization, or pull synchronization where a system is being asked to engage in a synchronization process.[5]

Another aspect is which amount of data is being exchanged during the synchronization process. Incremental synchronization just exchanges the data items that have been modified on all systems since the last synchronization happened. In contrast to this method, full synchronization exchanges always all data items from one system to the other. Incremental synchronization can be achieved by storing timestamps of the last modification and comparing them between the participating systems. This introduces the need to synchronize the local time of a replica with the other replicas in the system. An alternative to that approach is to compare and exchange data structures within data items which is mostly done when synchronizing files. It might also be necessary to transform data items between not structural equal data structures that are equivalent.[5]

When data items are exchanged, it is possible that both data items have been changed within their respective location and cannot easily be merged. A conflict resolution mechanism has to be applied to put the data items back into a consistent state.[5]

One aspect that also has to be mentioned is the mapping between distributed data items and their dependent or referenced items. Within relational databases this can mostly be done by foreign and primary keys if the synchronization is done within homogeneous databases, which is also known as global unique identifier approach, that can be generalized to all kinds of data items.[5]

2.2 Layers of synchronization and solutions

As already described data items can be managed in different storage systems. The file system is one possible storage solution. Synchronization in this layer can be achieved in many ways. Tools and protocols that exist can be grouped into categories. Two categories are:

- file system tools like the UNIX tool rsync, that is especially designed for an efficient synchronization of files, or the coda file system, that is a distributed file system

- version control systems like CVS or the WEBDav protocol, that enables file transfer via http and incorporates locking and versioning

Synchronization mechanisms within databases are better known as replication mechanisms and will be described in the replication techniques section.

Special constraints are imposed on mobile clients, as described in the introduction. These have to be considered in the case of synchronization. A couple of solutions exist, that are mostly concerned with synchronization in the application layer. Microsoft is offering a product called Active Sync, that can be used to synchronize between PC applications and Windows CE based mobile clients.[5]

2.3 Synchronization Protocol

In order to actually implement synchronization between two or more systems it is necessary to have a well-defined specification of the steps and the overall workflow of this process. The afore mentioned synchronization solutions have one big disadvantage. They are all using proprietary synchronization protocols that increase the efforts for server or PC application providers and mobile application providers to make their respective products interoperable.[5]

2.3.1 Standard SyncML

Based on the open extensible mark-up language (XML) standard, the SyncML specification forms the basis for specifying an open data synchronization protocol. Two parts of the specification can be distinguished, the representation protocol and the synchronization protocol. SyncML uses the client server communication model, where conceptually two components on both sides that are called SyncML Adapter are exchanging XML formatted messages, that are defined by the representation protocol. The Adapter also implements the protocol workflow via a variety of transport protocols. The SyncML Adapter provides a procedural interface for server and client side components, as there are the Sync Engine on the server side and client applications on the client side. The sync Engine is responsible for modifying the data repositories as well as providing conflict resolution services. The server side application is interacting with the Sync Engine. On the client side there is no concept like the Sync Engine. Instead, the client side application is directly interacting with the SyncML Adapter. The concrete synchronization mechanisms are not specified by the SyncML specification.

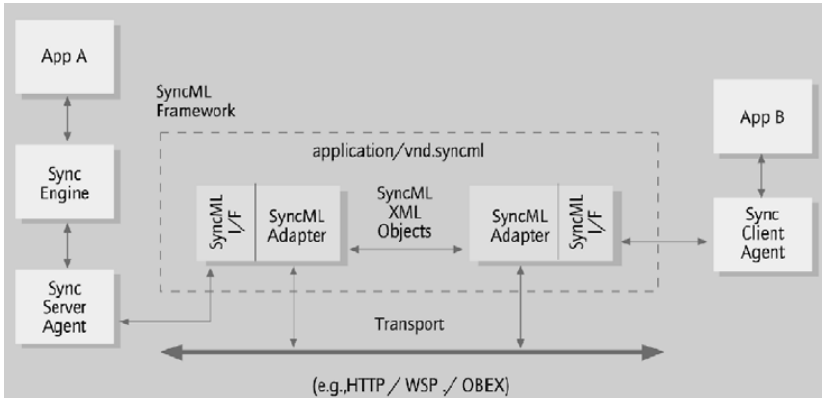

Figure 1: SyncML architecture [11]

For an in-depth description of the SyncML XML format please consider [11]. Only the main elements of this format will be described here. A SyncML Message element is composed of one SyncHdr and one SyncBody element. The former hold information about the source and the target of the synchronization process, therefore specifying the direction of the process, as well as a unique message identifier that is being used throughout the whole communication interaction. Within the SyncBody element the payload of the message, the actual data items are encoded, together with a SyncML Command element.

An alert Command is used to initialise a synchronization process. Commands can also have one of the values add, copy, delete, put or replace. These Commands are defining modification operations. Values like read and search are specifying query operations. In responses, the values status or results are used. Sequence and atomic values specify how subcommands should be executed, here the order and the transactional behaviour.

A possibility to map data items from the server to the client and vice versa is provided with the SyncML XML format.

What is more interesting in the context of this discussion, are the synchronization models that SyncML describes and that are defined by the synchronization protocol. The following scenarios are described within the specification.

- Two-way Sync: This model allows both sides, server and client to exchange modifications. The client sends its modifications first.

- Slow Sync: The server checks the complete data repository of the client for modifications.

- One-way Sync: Only one side sends its modifications, which results in two possible scenarios, from the client side only, or from server side only.

- Refresh Sync: One of the both sides sends its complete data repository to the respective other side for actualisation. Again, this gives two possible scenarios where the only the client sends its data repository or the server.

- Server-alerted Sync: the server informs the client, that it should issue synchronization.

3 Replication Techniques

Replication is used to achieve better availability and performance by using multiple copies of a server system, where availability is the main goal. If one of the copies, or replicas is not running, the service which is provided by a set of replicas can still be working. It is also of great use if the communication link between two systems is only intermittently available, which is most often the case in mobile applications. [10]

In the distributed systems community, software based replication, which is being described here, is seen as a cost effective way to increase availability. In the database community, however, replication is used for both performance and fault-tolerant purposes. [7]

Replication is both used in the database community as well as the distributed system community. In [13] an abstract functional model is introduced that is used to describe existing solutions in both fields. The abstract model is also used to compare the replication protocols. The classification that [13] introduces is semantically equivalent to the one that will be used in the following review of replication techniques.

3.1 Replication in Databases

Server systems usually depend on a resource, mostly on a database system. Replicating the server without the database improves availability but not performance and leaves the single point of failure problem. Performance is not improved since all kinds of access, query and update operations are done on one single database resulting in query-update problems. Therefore in addition to the replication of the server system, the resource needs to be replicated, to gain substantial advantages over systems that rely on one resource.[10]

According to [13] databases are collections of data items that are controlled by a database management system. Replicated databases are therefore a collection or set of databases that store copies of identical data items. A data item can be referred to as the logical data item or by it’s physical location.

There are many technical implications that come with resource replication, or data replication as it will also be referred to. The main requirement in data replication is, that replicas should functionally behave like servers, that are not replicated. This strict requirement can not always be fulfilled since the mechanism to solve this problem - synchronous replication or eager replication, where every transaction updates all replicas - produces heavy distributed transaction load. Therefore synchronous replication is considered to be very hard to achieve in an efficient way [7]. Fortunately there is another way for propagating updates to all replicas. This mechanism is called asynchronous replication, since replicas are updated independently from each other and updates are sent from one server to all other replicas. This solution can lead to situations, where transactions are updating the same data items and may result in conflicts, which can be solved by ensuring the right time dependent order of the original transactions. That way all replicas take in the intermediate states the whole system got through and, at some point in time, get into a common, identical state. Keeping the order of updates requires synchronization, which imposes performance implications that require sophisticated synchronization techniques, resulting in a high degree of complexity [10]. Replication is therefore imposing a constant trade-off between consistency and efficiency [7].

Since we are dealing with a distributed system of replicas, situations have to be handled in which servers go down or network connections fail, introducing network partition problems. How these major problems are solved, introduces a classification schema that will be described in the next section. If not explicitly mentioned the subjects of the discussion are relational databases. Therefore basic knowledge in the field of relational database technology, the transaction concept and management are required in order to understand the following discussion.

It should be mentioned that replication, especially in databases with high transaction rates is a complex topic. In the simple case of only a few replica nodes, that apply the update anywhere, anytime, anyway replication scheme, combined with low transaction rates, this simple replication scheme works well with respect to deadlocks and reconciliation, that will be described later. When the system scales up, problem complexity grows drastically at cubic rates leading to inconsistent replicas and soon to system delusion, where there is no way to repair the inconsistencies [4].

3.1.1 Classification

The techniques that will be presented in the following, can be characterized by identifying two different dimensions. First where an update is allowed to take place, at a dedicated server or at any server in the cluster of replicas and second whether an update will be done synchronously at all replicas or asynchronously. This classification is done in [10] ,[4], [13]. Other authors have added the amount of interaction activities, which can be measured as network traffic, as a third dimension. Constant and linear growth interaction costs can be distinguished, which will not be presented in more detail here. Other requirements like ordering or uniformity constraints on the messages that are interchanged among the replica databases are therefore not considered here. [7] also introduces another distinction point for the different solutions. That is the way an update transaction terminates. Two possible ways are identified, voting and non-voting termination. Voting termination can be achieved by simple confirmation messages or by more complex means like a two phase commit protocol (2pc). Another aspect of database replication is the point in time at which a transaction behaves in a deterministic way, which ensures one copy serialisability.[7]

The abstract functional model that [13] describes, divides the process of replication into five different phases, that are used to compare different approaches with each other. The replication protocols are presented by assuming one operation transactions. The phases are:

1. Request Phase: A client sends a request either to one replica or to all replicas

2. Server Coordination Phase: The replicas coordinate their work with each other to order the steps within the execution

3. Execution Phase: The replicas execute the operation

4. Agreement Coordination Phase: The replicas ensure that they all have done the same thing. In databases this phase is mostly implemented by a 2pc.

5. Client Response Phase: The client receives a response. This can be done when all work has been finished, or before all necessary steps have been done.

The different replication solutions apply different techniques within the described phases or even skip a whole phase, or change the order of the phases, which makes them distinguishable from each other.

3.1.2 Single Master Replication

[10] regards this approach as the most straightforward and often pragmatic way for replication. When applying this mechanism, one replica is the designated server where update transactions are worked on. This replica is called the primary replica. The updates on this replica are then distributed to all other replicas, which are called secondaries. During the distribution process the order in which the transactions are executed at the primary is preserved. Thus, all replicas go through the same sequence of states. Queries can be executed by all replicas in between any two update transactions. [10] Since there is only one server that can be updated, conflicts that arise from simultaneous updates on different replicas cannot occur. The only requirement that has to be fulfilled at every point in time is that there is only one primary. The secondaries can also be called backups of the primary. It is also possible to have primaries for specific subsets of the data items that need to be replicated.[7] In the relational model the granularity of the subsets is typically a table.

3.1.2.1 Synchronous techniques

Synchronous[10], or eager[13] update propagation means the instantaneous propagation of updates to all replicas as part of one atomic transaction, which keeps all replicas synchronized in a consistent state and does not impose any concurrency anomalies. At the same time it reduces performance and increases response time by adding additional update operations. Synchronous replication typically uses locking techniques to cope with concurrent data modifications [4].

One possibility to establish this kind of replication within relational databases is to define a trigger on tables that should be replicated. The trigger updates all replicas remotely whenever an update or insert action takes place on a table. Since the data replication is done right within the originating transaction, the time at which the replication takes place cannot be controlled by external systems or persons. Figure 1 describes this technique with respect to the above mentioned abstract replication model with applying a 2pc before sending a response to the client in order to ensure consistency over all replicas. The primary is only committing the transaction if all replicas succeeded. The primary is also defining the order of the execution of the incoming transactions. After committing or aborting, a response message is send to the client. When using a trigger, a transaction has to be started before updating the primary and communicating with the secondaries. This transaction is executed as a distributed transaction which can be controlled by a 2pc protocol. Only database managements systems, that support distributed transactions can use the trigger approach. Eager replication is mostly used as a hot standby backup mechanism, where one primary is replaced manually by a secondary in case the primary fails [13].

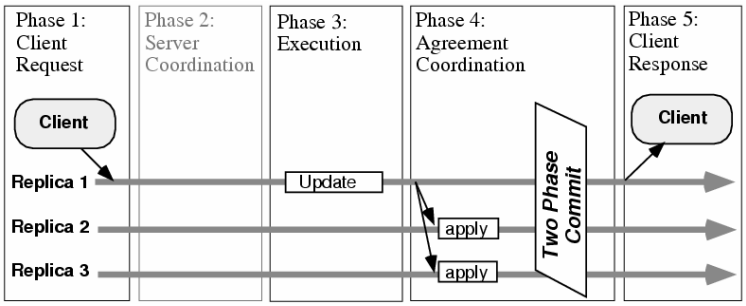

Figure 2: Eager primary copy [13]

Instead of propagating the updates, a technique can be used, that is replicating the requests of the original transactions and ensuring, that the same order of execution is used at all secondaries. This can be done in two ways. All requests can be run serially at the secondaries, which does not enable concurrency at the replicas. The other way is to use a process synchronization mechanism that ensures the same execution order at all replicas and let the transactions run concurrently. A request runs as a single distributed transaction at all replicas, which makes it possible to allow concurrency. The transaction synchronization mechanism, locking for example, at all replicas ensures, that the transactions run in the correct order, even if they run concurrently. Transaction termination is synchronized by a two phase commit protocol. If it is not desirable to abort a request if one replica fails, the global transaction doesn't need to be aborted as long as the transaction has run on the primary.

The following figure shows this technique with respect to the different phases of a replication process that was described above. Here Replica 1 is the primary, Replica 2 and 3 are the secondaries. This scheme can also be used in multi-master replication techniques or update everywhere [13], that will be introduced later, as [13] does. In this case all replicas have the right to update data items.

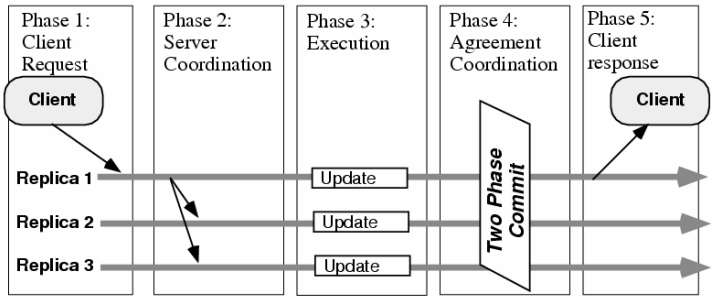

Figure 3: Eager update everywhere with distributed locking [13]

3.1.2.2 Asynchronous techniques

Asynchronous replication is the more popular approach according to [10], but has also drawbacks like stale data versions [4]. The term lazy replication [13] is also used for that kind of replication techniques. Lazy replication is used when performance is the main goal and response times have to be short [13].

In reference to the abstract functional replication model, an update is done on the primary within the execution phase and a response is immediately sent to the client. The agreement coordination phase takes place after the client response phase, where all replicas are brought into a consistent state. The primary has already coordinated the transactions and has brought them into the right execution order for the secondaries.

Updates can be extracted from the database system log, by only choosing the successful updates and filtering out the aborted transactions. This reduces the amount of data that has to be sent over the network, but imposes the restriction, that the primary has to wait until any running transaction at the time the replication should take place, terminates. Instead of sending complete records, only fields that have changed are transmitted to all other replicas, which is a further improvement.

An alternative to using the database log is to use a log table that is collocated to the table that should be replicated. Via a trigger all update operations are written to the log table. Periodically the log table entries are flushed to the replicas. Before sending the log table entries to the replicas a new log table is created and used to collect the update and insert operations. In parallel the replicas can be updated.

3.1.3 Handling failures

As long as both primary and secondary servers are reachable and alive the single master approach works well. In the case of failure special behaviour must be applied depending on the location of the failure.[10]

In the case of the failure of a secondary server, the whole system continues its work as described above. When the failed replica recovers, it has to detect which updates have occurred since it failed, which can be done by consulting the update log. It could be more efficient to get a whole new copy of the data from the primary server, depending on the downtime and on the transaction rate per second the whole system processes. During the catch up process, new transactions can be handled, that need to be processed by the secondary in order to really catch up.[10]

If the primary server fails, the first thing the whole system needs, is a new primary server. The decision which server takes over the role of the new primary must be agreed on by all servers. One replica therefore initialises an election algorithm, where each replica sends in its replica identifier. The replica with the highest identifier wins and becomes the new primary. The initialising replica is then informing all other replicas of the election result.[10]

After a new primary server is elected, all replicas need to have the same state, which means that all of them must have processed the same set of updates. To prevent a situation, in which replicas could be in a different state than the primary, a primary could stop processing new transactions before it received acknowledgements from all replicas, telling that the current transaction completed. This is actually a synchronous approach with the addition of a 2pc with all replicas, imposing all problems a 2pc implies.[10][7] To circumvent the usage of the 2pc, each replica can send the log address of the last log record it received from the failed primary. So all replicas can get consistent with the most up-to-date replica in the system. Still, some updates the failed primary sent could be lost.

As a result, finding a new primary, reconfiguring the replicas and so forth can be quite costly and if it happens too often, can dramatically degrade system performance.

Another problem that must be handled is, if the communication network partitions into two or more subnets of working replicas. If one can ensure that there is only one partition that handles requests, which holds the primary, the system could still work. If the primary is down, no partition can process requests anymore. It is therefore more flexible for each partition to determine whether it is allowed to hold the primary by applying a majority consensus. A set of communication replicas is allowed to elect a new primary or keep the existing primary if it forms a majority of the overall number of replicas. A partition that holds more then the half of the overall number of replicas can form the new working system. The other partitions have to check constantly for recovering replicas, since it could acquire the majority in the case of one replica joining the partition. If no partition has a majority, the system cannot process further requests. This situation could arise if there is an even number of total replicas, that are split into two partitions with an equal number of replicas in it. To get around this problem, a quorum consensus algorithm can be applied. This requires that all replicas get weights, that are considered during the process of figuring out the majority. For example if the system of replicas consists of two replica servers and one of them is given a weight of two and the other a weight of one, the server with the weight of two is always the new working partition.

3.1.4 Multi Master Replication

Like the case that was shown in the introduction, it is often a planned event, that a network splits into partitions and that each partition, in this case a single device and the central stationary database server and the rest of the corporate network, keeps processing updates. So the afore mentioned quorum and majority consensus mechanisms do not apply in this situation.

What is known as multi master replication can be understood as a system of replicas with multiple primary servers as defined in the single master replication schema. The multiple primary servers handle updates independently from each other while being disconnected. When a disconnected primary reconnects to the network, that is organized in a primary master topology, an exchange of recorded updates has to be done. During the time span in which the two systems are disconnected, equivalent data items can be updated on both systems, which results in conflicts when the both systems try to merge their data items. Conflicts can be avoided by designing applications not to produces conflicting updates.[10] If this cannot be achieved there are other ways to get around this problem.

3.1.4.1 Synchronous Techniques

As already stated, the single master synchronous replication can be implemented as a multi master synchronous replication scheme. Please consider the discussion above.

3.1.4.2 Asynchronous Techniques

As already stated, asynchronous replication has one major drawback, namely concurrency anomalies, that result in concurrent data access operations. Most asynchronous schemes use multi-version concurrency control mechanisms to detect non-serialisable behaviour. Isolation schemes provide the transaction with the most recent committed values, which could be a very old value [4]. Serialization problems can only be detected when transactions have been committed at the different replicas, making a reconciliation necessary. Normally, this cannot be done automatically, without possibly losing information [4].

One solution is to use unique timestamps. Such timestamps can be constructed by concatenating the local clock time to a unique replica identifier. Each data item that is being updated at a replica is tagged with such a timestamp. An update originated at a replica is only applied to the main database if it adheres to Thomas write rule [10]. That is, only an update with a greater numerical value than the one that is actually stored in the primary is applied. If the timestamp is lower, the update is discarded, or is fed into another conflict solution mechanism. This can be a person or some other program, that applies a synchronization rule. When the update gets propagated to the other replicas, all replicas share the same state. This model is also known as optimistic replication, since all replicas are allowed to change any data. Metadata is kept, that is used for reconciliation of concurrent changes, or conflicts. [14] describes an approach for optimistically hierarchical replicated data using vector time pairs, in contrast to the current discussion, which focuses on relational data.

Figure 3 depicts an asynchronous multi master approach that uses reconciliation between the different replicas. Multi master replication is also referred to as update everywhere in [13].

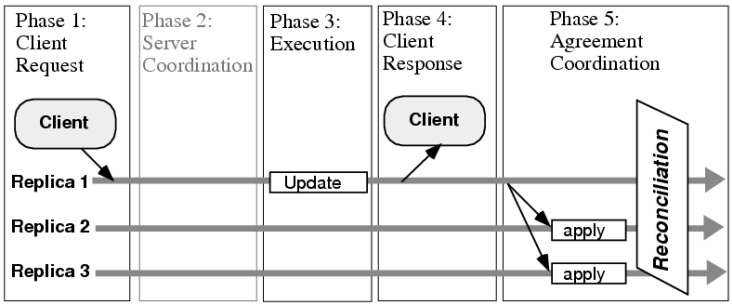

Figure 4:Lazy update everywhere [13]

3.1.5 Other Techniques

In [15] an overview of many other replication techniques is given, that are mostly modifications of the introduced techniques, but all keep as correctness criterion one-copy-serialisability. Just two other techniques should be mentioned here.

- An eager replication technique that is called read-one-write-all (ROWA), where a read operation can be executed at any replica, write operations have to be done on all. This is a modification of the single master replication model.

- The read-one-write-all-available (ROWAA) approach is a modification of the ROWA technique, but instead of writing to all replicas, only the available replicas are written, which introduces the problem that replicas might not be up-to-date.

3.2 Replication requirements in the mobile context

In mobile applications, a replica or copy is not always connected to the rest of the system. In this case it doesn't make sense to wait until an update takes place and is propagated to all replicas[13]. Therefore lazy replication algorithms are required, that asynchronously propagate replica updates to other nodes after the updating transaction commits. Statically connected systems use lazy replication to improve response time. According to [4], disconnected operations and message delays in conjunction with lazy replication are responsible for more frequent reconciliation.

4 Oracle Lite 10g – a mobile database solution

As an example for a mobile database solution the Oracle Lite 10g is considered in this chapter. Oracle Database Lite 10g is especially designed to facilitate the development, deployment, and management of disconnected mobile database applications. A disconnected mobile application is an application that can run on mobile devices without requiring constant connectivity to the server. An disconnected database application requires a local database on the mobile device, whose content is a subset of data that is stored on the enterprise data server. Modifications made to the local database by the application have to be synchronized with the server data. [12]

At the heart of Oracle Lite 10g is the Mobile Server, that forms the tier between the mobile device, that is equipped with an Oracle Lite database and the enterprise Oracle database server. The Mobile Server manages the disconnected applications and is responsible for delivering and updating these applications, as well as providing tools for the definition of databases for clients and keeping them synchronized with the enterprise Oracle database.[12]

4.1 Application Model and Architecture

The Oracle Lite 10g application model defines a container called “publication” for each application that is installed on the Mobile Server. A publication is comparable to a database that holds many publication items, which are comparable to database tables. These publication items are defined by Structured Query Language (SQL) Queries that contain variables. These can hold user specific values in order to customize or filter the data items that will be within the database of the user. A user can have a subscription to a publication, which means that he or she can use the application and the data that the publication provides for the user. The Mobile Server takes care about creating and populating the users database, therefore providing a snapshot of the data that is stored in the enterprise Oracle database.[12]

Whenever the user connects to the Mobile Server, a synchronization of the data snapshot the user initially retrieved can happen, since the data on the user’s device could have been modified. Synchronization may be initiated by the user using the Oracle Database Lite 10g Mobile Synchronization application (msync). The following figure illustrates the described architecture schematically.[12]

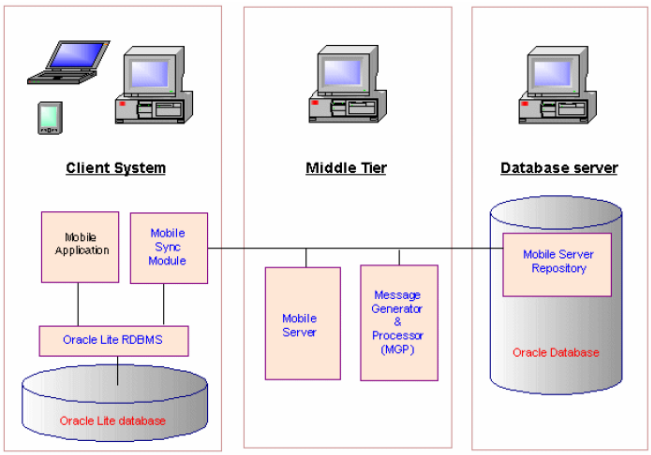

Figure 5: Oracle 10i Lite Architecture [12]

4.2 Synchronization

MSync is a lightweight application that resides on the mobile device. Mobile Sync makes it possible to synchronize data between handheld devices, desktop and laptop computers and Oracle databases.[12]

Mobile Sync synchronizes the snapshots in Oracle Database Lite with the data in the corresponding Oracle data server. The Mobile Server also coordinates the synchronization process.

Triggered by a mobile user, the Mobile Sync application first authenticates the users with the Mobile Server. The changes made to the Oracle Database Lite are then uploaded to the Mobile Server. It then downloads the changes that are relevant for the user from the Mobile Server and applies them to the Oracle Database Lite. The Mobile Sync application can also encrypt, decrypt, and compress transmitted data.[12]

A snapshot keeps track of changes made to it in a change log, which is a difference to a base table. Users can update the data in the snapshot in Oracle Database Lite while the device is disconnected, and can synchronize them with the Oracle database. Three types of publication items can be used to define synchronization, which are fast refresh, complete refresh and queue-based.[12]

The most common method of synchronization is a fast refresh “publication item” where changes are exchanged in two directions, from client to server and from server to client. A background process called the Message Generator and Processor (MGP) periodically collects the changes sent by all clients and applies them to the server’s database tables. It then composes new data, that can be downloaded by each client during the next synchronization, based on predefined subscriptions. [12]

The complete refresh publication item is defined as follows. All data for a publication is downloaded to the client, which is especially done during the first synchronization session. This form of synchronization takes longer because typically much more data rows are downloaded to the client device. Existing data rows on the client device are not considered and deleted. [12]

The most basic form of publication item is queue based, because there is no synchronization logic created with it. A queue-based publication item is used whenever no actual synchronization is required. Data collection on the client, for example, does not create conflicts and doesn’t need server side updates.[12]

Oracle Database Lite uses an asynchronous method for synchronization between Oracle Database Lite clients and the Oracle database server through the Mobile Server. The Mobile Sync module works independently of the MGP and vice versa. The default synchronization method is the fast refresh mode as displayed in the following figure.

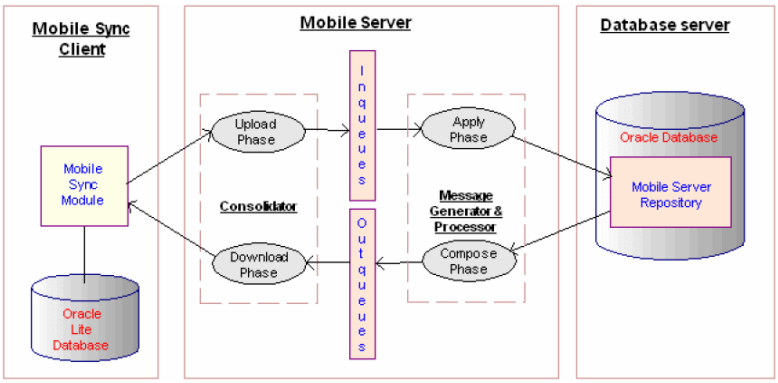

Figure 6: Oracle 10i Lite Fast Refresh [12]

Fast refresh is an incremental refresh. Uploaded changes are stored in so-called in-queues during the upload phase. Changes that are stored in out-queues are downloaded and applied to the client. The MGP periodically checks the in-queues and takes anything found in an in-queue and applies it to the database during the apply phase. Changes generated by all clients and server-side applications to the Oracle database are composed and stored in an out queue until the next time a client is synchronized. The upload and download phases are performed independently of any apply or compose phase. During any synchronization session, download occurs after upload, and compose occurs after apply. For each user, the MGP takes any content of the in queue and applies it to the base tables on the Oracle database. Any conflicts are detected and resolved at this time. The apply phase is completed when the changes uploaded by all users are processed.[12]

4.3 Evaluation

The Oracle 10g Lite approach is very well suited for developing disconnected applications from scratch, where there is no existing database that has to be accessed, that is probably no Oracle product. If the database cannot be accessed directly from the Mobile Server, the solution lacks some support. Since organisations mostly have a data access layer around their databases, that uses technologies like CORBA, Enterprise Java Beans (EJB), or some other technology, that encapsulate logic that needs to be executed before accessing the database (it mustn't even be a relational database). The only way to use Oracle Lite 10g in such an environment, is to duplicate the data access code into the mobile application that might be written in a different programming language than the existing code, resulting in the need to port that mostly mission critical code. The results are higher maintenance costs for the data access code. The distribution of this code could dramatically reduce the benefits the Oracle Lite 10g provides and lead to the decision not to use the solution at all or only partially. This would put much effort back onto the shoulders of the application development team.

5 Peer to Peer Mobile Computing

Most of the discussion above was tailored towards the client server communication model. Now the focus gets to the peer to peer communication model in the context of mobile computing.

In the client server model it was assumed, that the mobile clients periodically reconnect to a static infrastructure of database servers and networks, which hinders users to become truly mobile. Communication between mobile devices has not been allowed or even possible. Any to any communication, as introduced by the peer to peer computing model, suffers from scalability problems when it comes to synchronizing more and more units with each other. [3]

5.1 Requirements

In peer to peer mobile computing, users are not bound to particular geographic places, which would impose restrictions on their mobility. Still, they need to synchronize the data that they use on their mobile devices with other users. So any to any communication needs to be possible. A mobile device like a laptop should be able to communicate with a stationary desktop as well as with a palmtop. All three devices should be able to communicate without the restrictions of a predefined communication model like the client server model. In a system like this, all participants should be equal with respect to the ability to propagate updates. Security issues that arise in that context can be overcome without even touching the replication system.[3]

Another requirement is to support large replication factors. While the techniques that are used within database systems are mostly designed to support a limited amount of replicas of the same data item, the replication factor is naturally much higher in the peer to peer communication model. If one supposes that mobile devices will spread more and more and that these devices, even the smallest get sophisticated communication abilities, even higher replication factors will be seen in the future.[3]

The constraints on mobile device resources, compared to stationary computers, make it impossible to just replicate data in a careless way onto mobile devices. Instead, finely grained control mechanisms for the selection of data items that should be replicated have to be provided.[3]

5.2 The Ward Model

The ward model, as described in [3] tries to satisfy these requirements as well as provides a dynamically reconfigurable synchronization topology. In this model many geographically collocated computers are grouped together into a ward, where all computers are equal peers. A ward has always one ward master, which is taking in the role of a server. A ward master is also just one of the equal peers and can be re-elected in case it gets unreachable or fails. The ward master is not required to store data on behalf of the ward members, but has to know them all. Additionally the master is the only interface to other wards, which limits the knowledge at every replica within a ward to the other ward members. Other replication approaches require, that all replicas know each other and synchronize with each other, which makes the ward model more scaleable compared to the presented approaches. The ward masters of the wards form another, top-level ward. The ward masters are communicating with each other and import and export updates from or into their own wards. Wards can be constructed dynamically, members can leave and join wards as their geographical location changes. Wards are destroyed when the last member of a ward disconnects.[3]

The data items, that are stored within a ward, can change over time, as computers are joining and leaving the ward. The data items that a ward holds are called the ward set, which is replicated optimistically. By applying selective replication, each ward member can only store data that it actually needs, therefore improving efficiency and resource utilization. This can be achieved manually or with automated tools. This also applies to inter-ward communication, where different wards can store different ward sets. [3]

Mobility within the ward model is supported in two ways. Intra-ward and inter-ward mobility. Intra ward mobility occurs within a limited geographical area. All computers a moving ward member encounters are ward members and therefore communication and synchronization can take place. Inter-ward mobility is more challenging. Again two scenarios are supported, ward changing and ward overlapping. Basically, ward overlapping is a very lightweight solution, which allows multi-ward membership, without changing any ward sets, whereas ward changing is more heavyweight since it requires, that the new ward adds the data of the new member to its ward set, and the old ward possibly reduces its ward set.[3]

5.3 Peer to Peer and Oracle Lite 10g

The Oracle Lite 10i architecture is tailored towards the client-server communication model and requires as stated in [12] an installation of a full fledged oracle database on the server site. If this paradigm could be modified in a way that an Oracle Lite installation could take over the role of a full Oracle installation, one step towards a peer to peer model would be done. Another step would be to put the middle tier of the Oracle Lite 10g solution onto the mobile device, making the installation a quite heavyweight solution. Other issues like scalability, security, and an open and standardized communication protocol would still have to be solved, in order to make this Oracle specific solution truly peer to peer enabled.

The ward approach could be used to build up an architecture, that relies on static Oracle database servers, that play the role of ward masters, building also the static backbone of a replicated database system. Mobile users could then connect to all possible ward masters, which are then responsible for managing the required data items as part of their respective ward sets. This model would at least provide flexibility, that most users need, providing a compromise between true mobility and reliable, consistent and efficient data replication. [4] describes this model as two-tier replication, with base nodes, that are statically connected and mobile nodes, that are usually disconnected. It is shown, that it is very well suited for mobile computing. Weather the benefits of both approaches, the ward and the two tier replication scheme can be really combined in order to provide a very flexible and efficient replication solution for mobile computing needs to be investigated in more detail as it is possible within the context of this paper.

6 Future Requirements, Developments and Applications

The predominant mobile unit, at least in industrialized countries, is the car. It is being developed towards an intelligent mobile unit, equipped with mobile computer devices. An initiative formed by European car manufacturers has joined to develop what is called car to car communication [16]. The main idea is to create a mobile network of interconnected cars, that are communicating with each other in order to improve security and provide other services to the actual inmates of the car. This peer to peer network of ad-hoc joining and leaving cars will need to exchange data in real time as well as replicate data from car to car. The presented ward model could build the basis for an effective communication model, that is highly flexible towards its configuration of participating members of the network. The notion of a mobile ward has to be investigated, which could apply in this scenario.

References

[1] M. Satyanarayanan: Fundamental Challenges in Mobile Computing, Annual ACM Symposium on Principles of Distributed Computing, Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, Pages 1 – 7, 1996

[2] Vietanh Nguyen: Mobile Computing & Disconnected Operation: A Survey of Recent Advances, http://www.cse.ohio-state.edu/~jain/cis788-95/ftp/mobile_comp/, 1995

[3] Ratner, Reiher, Popek, Kuenning: Replication requirements in mobile applications, Mobile Networks and Applications, Volume 6 , Issue 6 (November 2001), Pages: 525 - 533

[4] Gray, Helland, O'Neil, Shasha: The Dangers of Replication and a Solution, International Conference on Management of Data, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, Pages: 173 - 182

[5] Vanda Lehel: Synchronisation von Informationsobjekten zwischen Portalen, Diplomarbeit Hamburg 2002

[6] Alfredo Goñi, Arantza Illarramendi: Mobile Computing: Data Management Issues

[7] Matthias Wiesmann, Fernando Pedoney, Andr´e Schiper ,Bettina Kemmez, Gustavo Alonsoz: Database Replication Techniques: a Three Parameter Classification, Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000), 2000

[8] Fausto Giunchiglia and Ilya Zaihrayeu: Coordinating Mobile Databases, http://citeseer.ist.psu.edu/715356.html

[9] Hagen Höpfner: Replikationstechniken für mobile Datenbanken, Diplomarbeit, Magdeburg 2001

[10] Bernstein, Philip A.; Newcomer Eric: Principles of Transaction Processing, Morgan Kaufmann Publishers, Inc., 1997

[11] SyncML Representation Protocol, version 1.0.1, http://www.syncml.org/docs/syncml_represent_v101_20010615.pdf

[12] Oracle® Database Lite Developer's Guide 10g (10.0.0) Part No. B13788-0

[13] Wiesmann, Pedone, Schiper, Kemme, Alonso: Understanding Replication in Databases and Distributed Systems, Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000)

[14] Russ Cox, William Josephson: Optimistic Replication Using Vector Time Pairs, http://citeseer.ist.psu.edu/691408.html

[15] Ceri, Houtsma, Keller, Samrati: A Classification of Update Methods for replicated Databases, 1994, http://citeseer.ist.psu.edu/ceri94classification.html

[16] Car to Car Communication Consortium. http://www.car-to-car.org/