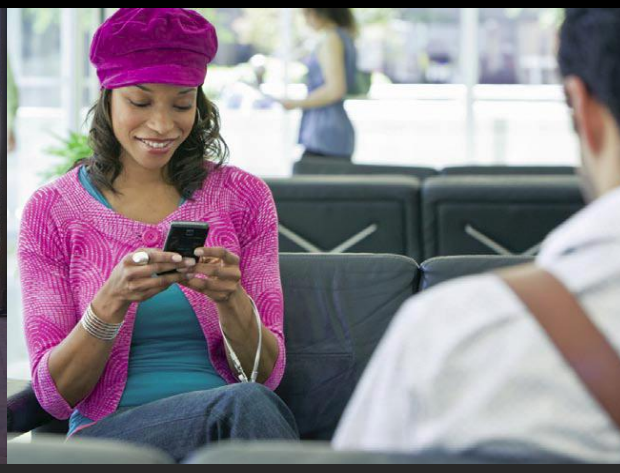
SYBASE®
An SAP® Company

MobiLink 12 Performance

WHITE PAPER





www.sybase.com

Contents:

Principal Author

Graham Hurst graham.hurst@sybase.com

Revision History

Version 1.0 - March 2012

Executive summary

MobiLink provides scalable, high-performance data synchronization between client databases and a central database (referred to as the consolidated database). Key factors in MobiLink synchronization performance are:

- The performance of concurrent synchronization script execution in the consolidated database.
- The network bandwidth.
- The client processing speed.
- The speed of the computer running the MobiLink server.
- A sufficient number of MobiLink database worker threads (each with a connection to the consolidated database).
- A large enough cache and sufficient computer memory to avoid excessive swapping to disk by the MobiLink server.

The performance of MobiLink synchronization can be summarized as follows:

1. The MobiLink server has modest hardware requirements, and makes good use of multiple processors. For example, in our tests with a SQL Anywhere consolidated database, MobiLink typically required less than half the processing that was required by the consolidated database.
2. The MobiLink server scales very well with additional clients.
3. You must test under high synchronization load to tune the concurrent performance of your synchronization scripts (e.g. by using indexed queries and by minimizing locking), and to help choose a range for automatic adjustment of the number of MobiLink database worker threads.
4. There is some per-synchronization overhead, so throughput is lower for smaller-sized synchronizations.
5. The choice of network protocol can affect performance, particularly for downloads. In these tests, HTTP was almost as fast as TCPIP, and the best performance with encryption occurred when compression was also used.
6. MobiLink state information is maintained either in the consolidated database or another database, not in the MobiLink server, so you can use multiple MobiLink servers. For high availability and scalability you can use multiple MobiLink servers with the Sybase Relay Server or a third-party load-balancing system to make them appear as a single server.

As an example of performance, here are the best total server times and synchronization rates for 10000 clients synchronizing 1000 92-byte rows each (for a total of 10 million rows being synchronized) with a SQL Anywhere consolidated database. The MobiLink server and consolidated database were each running on a dedicated Dell PowerEdge R510 server with two six-core Intel Xeon X5650 2.6 GHz processors and 48 GB of RAM:

Download from consolidated:	21.3 s or 469 syncs/s (41 MB/s)
Upload of inserts to consolidated:	33.2 s or 301 syncs/s (26 MB/s)
Upload of deletes to consolidated:	57.8 s or 173 syncs/s (15 MB/s)
Upload of updates (with conflict detection):	88.1 s or 114 syncs/s (10 MB/s)

The rest of this whitepaper provides more details in support of the above information.

Introduction

Sybase developed MobiLink synchronization technology to enable efficient and scalable data synchronization between many client databases and a single central database (referred to as the consolidated database) or application server. MobiLink is included with SQL Anywhere. Client databases can be UltraLite, or SQL Anywhere databases. Supported consolidated databases include SQL Anywhere, Sybase Adaptive Server Enterprise, Sybase IQ, Microsoft SQL Server, Oracle, MySQL, and IBM DB2 LUW.

The goal of this paper is to give you an understanding of MobiLink server performance and to help you assess its applicability to your data synchronization needs. First, this paper describes the time-consuming steps in MobiLink synchronization.

After a brief description of the testing methodology, this paper presents performance results and analysis for varying the following conditions:

- The number of MobiLink database worker threads and automatic adjustment of the number.
- The retry delay and connection limit when synchronizing with large numbers of client databases.
- The number of client databases synchronizing simultaneously.
- The amount of data transferred in each synchronization.
- The type of network protocol used to connect client databases to the MobiLink server.

We then give recommendations on how to get the best performance with MobiLink synchronization.

Differences from the MobiLink 11 Performance whitepaper

As well as using newer versions of MobiLink and SQL Anywhere, several other changes were made since the benchmark testing done for the MobiLink 11 Performance whitepaper. These include the following changes:

- Running the MobiLink server and the consolidated database on different computers instead of the same computer, to better reflect typical usage.
- Using more up-to-date server computers.
- Using Windows Server 2008 R2 for the server computers instead of Windows Server 2003. The registry settings previously used to allow more simultaneous network connections are not supported by the newer operating system.
- Using a stored procedure for uploaded updates with conflict detection and resolution, instead of using an upload_fetch script and conflict resolution scripts. This follows our current recommendation to detect and handle uploaded update conflicts in your upload_update script, and gives much better performance than using an upload_fetch script to detect conflicts.
- Performance testing of the new MobiLink server feature to automatically adjust the number of database worker threads, to compare with manual tuning.
- Performance testing of different network protocol options between the client databases and the MobiLink server.
- The new MobiLink server feature to automatically adjust its cache size was used, instead of explicitly setting a cache size. The default of using up to 70% percent of physical memory was used throughout.
- The MobiLink server now automatically caches remote schemas, reducing the per-synchronization overhead.

Time-consuming steps in MobiLink synchronization

To help you understand MobiLink server performance, we first briefly describe the MobiLink server architecture, and then look in detail at the steps in a single MobiLink synchronization, focusing on how they contribute to the time taken for a synchronization. After that, we look at scaling to more clients and identifying potential bottlenecks.

These are the main steps in a MobiLink synchronization:

1. The MobiLink client initiates the synchronization and transfers the upload stream to the MobiLink server.
2. The MobiLink server processes the upload stream, creating commands to run against the consolidated database.
3. The synchronization is assigned to a database worker thread and a database connection.
4. The upload data is applied to the consolidated database.
5. The rows to be downloaded are fetched from the consolidated database.
6. The database worker thread and database connection are released.
7. The MobiLink server prepares the download stream and transfers it to the MobiLink client, which applies it to the client database. This is the last step of a synchronization, using the default of no download acknowledgement.
8. If download acknowledgement is enabled, the MobiLink client sends an acknowledgement to the MobiLink server. The synchronization is again assigned to a database worker thread and a database connection, the acknowledgement is processed, and then the database worker thread and database connection are released.

MobiLink server architecture

The MobiLink server architecture is an extension of the staged event-driven architecture (SEDA). A synchronization passes through different stages, with at least one thread performing each stage. This architecture is designed to maximize synchronization throughput by isolating network activity from database activity, and to minimize the time each synchronization requires a connection to the consolidated database.

Initiation

A MobiLink synchronization is typically initiated by the client creating a network connection with the MobiLink server. Server-initiated synchronization is also an option, but it differs only in providing a mechanism to notify a client to start a synchronization. The order of events depends on the type of client. An UltraLite client establishes the network connection with MobiLink first and then builds the upload stream as it sends it, whereas a SQL Anywhere client builds the upload stream before establishing a network connection. (The difference in constructing the upload stream allows UltraLite clients to work with less memory.)

The client first requests a network connection with the MobiLink server. The MobiLink server uses a network stage thread to immediately accept the network connection, or refuse it if the connection limit is reached.

By default, the MobiLink server accepts up to 1024 network connections (-nc option), but operating system settings may cause network errors before that limit is reached. If the MobiLink server's limit is reached, additional synchronization requests are rejected, resulting in a network error for the client. Clients that experience such network errors must retry later. You can lower the network connection limit if you prefer to have synchronizations rejected instead of queued when the MobiLink server is busy, even though throughput might be reduced. Note

that a client must recreate either the first part (for UltraLite clients) or all (for SQL Anywhere clients) of the upload stream to retry a synchronization.

Upload processing

The network stage thread is used to read the bytes of the upload stream from the network into the MobiLink server's in-memory cache. The speed of the client-to-MobiLink network connection and the size of the upload stream are the main influences on the time taken to transfer the upload stream.

For an UltraLite client, the upload phase also includes some client processing time to determine what data it needs to upload (data that is new or has changed since the last synchronization). This normally requires a small amount of processing; it is much less than the client processing required in the download phase.

For a SQL Anywhere client, the upload stream is created before the client connects to MobiLink, so the time taken to create the upload stream occurs before the connection to MobiLink.

In the MobiLink server, the stream-stage thread is used to decode the uploaded bytes into MobiLink protocol commands. This stage includes decrypting the stream when encryption is enabled, and uncompressing the stream when compression is enabled. Encryption and compression add to the processing time required, though stream compression reduces the network transfer time by reducing the number of bytes transferred. The MobiLink protocol commands from this stage are added to a command queue.

The queued commands are carried out by a pool of threads in the command processor stage. Many commands are executed directly, but ones that require the consolidated database are collected into a list. Once the list is complete for a synchronization, the synchronization is assigned to a command processor thread. The command processor thread becomes a database worker thread by getting a database connection, and remains a database worker thread until that synchronization's list of database commands is completed.

At most, only half of the command processor threads can be database worker threads at the same time. The size of the command processor thread pool is twice the maximum number of database worker threads specified by the -wm or -w options for the MobiLink server. This allows the MobiLink server to perform non-database commands concurrently with database commands. Depending on the load on the MobiLink server, there may be fewer than half the command-process threads acting as database worker threads.

Database connection

A database connection is not created or assigned to a synchronization until the synchronization has reached the stage where database access is needed.

MobiLink maintains a pool of connections to the consolidated database for the use of database worker threads, as well as one or more shared connections for reading synchronization scripts from the MobiLink system tables and other administrative tasks. The rest of the database connections are created as needed by the database worker threads; one connection is dedicated to a database worker thread until the database command list is completed, and then it is returned to the database connection pool. You can specify a larger number of database connections for the pool than the number of database worker threads, but the most that will be used at once is one per database worker thread.

There are two cases where MobiLink will close a database connection from the pool and open a new one. The first case is if an error occurs. The second case is if the client requests a synchronization script version, and none of the available connections have last used that synchronization script version. If you have more than one synchronization script version, you may want to set the maximum number of pooled connections to be large enough that MobiLink will not need to close and open a new database connection each time a different synchronization version is requested.

To help minimize the time taken to create new database connections, you can do the following:

- Limit the number of database worker threads (-wm and -w options). (Choosing an optimal number of database worker threads for throughput is discussed in more detail later.)
- Set the maximum number of pooled MobiLink database connections (-cn option) to be your number of synchronization script versions multiplied by the number of MobiLink database worker threads. This avoids the need to create new database connections, since a connection for each version would be available for every database worker thread.

Applying upload

Once a synchronization has been assigned to a database worker thread, that thread carries out the database commands from the command queue. First, a simple query is used to check that the database connection is still active, and then authentication is performed.

The MobiLink database worker thread then performs the upload database commands in the command queue. It uses its database connection to apply the uploaded rows to the consolidated database by executing the upload scripts. By default, upload table scripts are only executed if rows have been uploaded. The MobiLink database worker threads have the same performance issues as any other clients transacting with the consolidated database, including simultaneous connections, size of transactions, concurrency issues, and, when the consolidated database is on a different computer than MobiLink, the network speed. In addition, if your upload scripts incorporate conflict detection via an upload_fetch script, the MobiLink server fetches each row to be updated from the consolidated database before applying an uploaded update, to check for a conflict. For better performance, we now recommend implementing conflict detection and resolution within your upload_update script instead of defining an upload_fetch script and conflict resolution scripts.

When the list of upload database commands in the command queue has been completed, the database worker thread starts on the download database commands.

Download

The MobiLink database worker thread executes the download database commands for the synchronization. These commands execute the download scripts in the consolidated database, fetching the rows to be downloaded to the MobiLink client. The fetched rows are stored in the MobiLink server's cache. Any data conversion, such as character-set translation, is performed. As with uploads, MobiLink database worker threads have the same performance issues as any other database client applications when executing the download scripts. **Since they are run for every synchronization, it is critical for your download script queries to use indexes and avoid locking!**

After the fetching and data conversion commands are completed, the list of database commands has been completed. The download transaction is committed, any end synchronization scripts are executed and committed, the database connection is released back into the database connection pool, and the thread (which was the database worker thread) is returned to the pool of command processor threads.

Command processor threads then perform the remaining download commands for the synchronization, packing the rows to be downloaded into the download stream format. When the download stream has been created in memory, the network thread is used to transfer it to the MobiLink client.

Assuming the download queries have been tuned, the download phase usually involves less processing for the MobiLink server and the consolidated database, than the upload phase.

The time taken to transfer the download stream to the client depends on the size of the download, the network bandwidth, and possibly the client's processing speed (for UltraLite clients, and SQL Anywhere clients where the download does not fit in the download buffer).

Unlike the MobiLink server, the client does more processing in the download phase than in the upload phase. UltraLite clients process the download stream as it is received, whereas SQL Anywhere MobiLink clients buffer the download stream and then process it. When the download stream is processed, rows to be removed are deleted, new rows are inserted, changed rows are updated, indices are updated, and referential integrity is checked. This processing may take significant time with clients that have slow processors or slow storage media, such as flash memory.

If download acknowledgement has been enabled for the client, then once the client has received, processed, and committed the download stream, it sends a download acknowledgement to the MobiLink server. A database worker thread is acquired to run the non-blocking download acknowledgement script.

The best throughput and scalability is achieved when download acknowledgement is not enabled, which is the default.

Scaling up to more clients

Now that we have examined the time-consuming steps in a single synchronization, we consider the case of multiple clients simultaneously synchronizing.

The maximum number of simultaneous synchronizations for a MobiLink server is set by the network connection limit (-nc option), since each synchronization requires a network connection for the whole duration of a synchronization. When the server has the maximum number of network connections, any additional synchronization connection requests will be refused. With slow client computers or networks, the best throughput may require using a higher connection limit than the default of 1024, to allow more synchronizations simultaneously in the network transfer stage. However connection requests may be refused by the operating system even before the number of simultaneous synchronizations hits the MobiLink connection limit, particularly if the limit has been increased.

The maximum number of active synchronizations that can simultaneously be in the database stage is set by the maximum number of database worker threads (-wm or -w option). The optimal value depends on your consolidated database: it corresponds to the optimal number of database connections that can constantly run your synchronization scripts. In practice this number is lower than for other common database applications, because MobiLink synchronization is concentrating the work done over a long time on many client databases into condensed, simultaneous synchronizations.

While databases can support many simultaneous connections, the optimal number of busy connections is typically much smaller. For example, with a CPU-bound database load, the optimal number of connections supplying that load might be closely related to the number of CPU cores.

Consider the analogy of an old-fashioned bank, where each teller has to visit the bank's vault to deposit or withdraw cash for a customer. The customer is like a synchronization, the teller is like a MobiLink server's database worker thread, and the vault is like the consolidated database. With more tellers, more customers can be served at once, but too many tellers in the vault will get in each other's way. With too many tellers, the rate of transactions is less than the rate if a smaller number of tellers were in the vault.

The bank can get a bigger vault, but each vault will have an optimum number of tellers. Having significantly more or less tellers in the vault than the vault's optimum number would reduce the transaction rate, by underutilizing the vault or overcrowding it.

Extending the analogy, the number of customers allowed in the bank is like the MobiLink network connection limit. Since each customer needs to take time to fill out a deposit or withdrawal slip, and handle any transaction record or withdrawn cash, it makes sense to allow more customers than tellers in the bank.

Potential bottlenecks

The overall performance of any system, including throughput for MobiLink synchronization, is usually limited by a bottleneck at one point in the system. For MobiLink synchronization, the following might be the bottlenecks limiting synchronization throughput:

The performance of the consolidated database - Of particular importance for MobiLink is the speed at which the consolidated database can execute MobiLink scripts simultaneously, on as many connections as the number of MobiLink database worker threads, at the same time as any other database load. For best throughput, you will likely need a lot of testing and skilled tuning to reduce database contention for your synchronization scripts. This is by far the most common bottleneck for MobiLink performance, since few other applications can generate as concentrated a load on the consolidated database.

The bandwidth for client-to-MobiLink server communication - For slow connections, such as those over cellular or dial-up networks, the network may cause MobiLink clients and the MobiLink server to wait for data to be transferred.

The bandwidth for MobiLink to consolidated database communication - This is unlikely to be a bottleneck if both the MobiLink server and the consolidated database are running on the same computer, or if they are on separate computers connected by a high-speed network.

The speed of the computer running the MobiLink server - This is more important when you are using .NET or Java synchronization scripts, since they run on the MobiLink server computer whereas SQL synchronization scripts run on the consolidated database computer.

The number of MobiLink database worker threads - While a smaller number of database worker threads lessens the chance of contention in the consolidated database, too small a number may leave clients waiting for a free database worker thread, or underutilize the consolidated database. Too large a number can cause contention that lowers throughput. As of version 12, you can optionally have the MobiLink server automatically adjust the number of database worker threads, but performance testing may still be helpful to choose an optimal range for automatic adjustment.

The client speed - This is more likely to be a bottleneck in downloads than uploads, since downloads involve more client processing and writing to persistent storage. The MobiLink server does not need to wait for the client after sending the download.

The potential bottlenecks that are not specific to MobiLink are beyond the scope of this whitepaper. The effect of network bandwidth and general client-server performance of the consolidated database are not considered.

Performance testing

The goal of the tests for this whitepaper was to measure the performance characteristics of MobiLink synchronizations, so that you can gain insight into the scalability of the MobiLink server. In particular, we sought to determine the following:

- The performance of MobiLink with a large number of clients synchronizing simultaneously.
- The optimal number of MobiLink database worker threads to maximize throughput for a particular workload.
- The optimal retry delay when synchronization requests exceed the MobiLink server network connection limit and the effect of increasing the limit.
- The effect of varying the number of clients synchronizing simultaneously.
- The effect of varying the amount of data transferred in each synchronization.
- The hardware requirements for a MobiLink server, relative to those for the consolidated database.

To make the results easier to interpret, our testing methodology employed the following principles:

Vary only one parameter at a time – Since there are many variables that affect MobiLink synchronization throughput, we took care to vary only one variable at a time as much as possible. Often we would vary one parameter until optimal throughput was achieved, fix that parameter, and then vary another one.

Stress the MobiLink server – Since we are concentrating on MobiLink server performance, we wanted to make sure it was being stressed, and made choices to minimize the load on the consolidated database. We ran the MobiLink server and the consolidated database server on identical computers, with clients on separate computers. Typically we ran enough simultaneous synchronizations to keep either server computer's CPU utilization close to 100%. The MobiLink server and the consolidated database server were run on identical computers so that the CPU utilization of each could be used to infer their relative hardware requirements.

Keep it simple – In order to concentrate on the inherent performance of the MobiLink server, the synchronization scripts, the client application, and the database schema were all kept very simple. More complex schema and synchronization scripts would increase the load on the consolidated database, so by keeping them simple we maximized the relative workload of the MobiLink server.

The following sections describe specific aspects of the performance tests.

Database schema

All of the tests are based on synchronizing the data in a single table. The column types were chosen so that several popular data types would be used. The following SQL statement shows the table definition:

```
CREATE TABLE Purchase (
    emp_id      INT           NOT NULL,
    purch_id    INT           NOT NULL,
    cust_id     INT           NOT NULL,
    cost        NUMERIC(30,6) NOT NULL,
    order_date  TIMESTAMP     NOT NULL,
    notes       VARCHAR(64),
    PRIMARY KEY ( emp_id, purch_id )
);
```

A compound primary key was used to simplify partitioning the data among the clients; the first column indicates which client the row is associated with. Not sharing rows between clients avoided database contention, which helped to minimize the load on the database server and increase the relative workload of the MobiLink server.

Values

The data values that are used to fill the `Purchase` table are generated in either the client or consolidated database. The algorithms to generate the values ensure that large values are used for integer data, so that the packing scheme did not shrink the data when it was transferred to or from the client. The initial value for `cost` was 123456789.12 for all rows, and each update increments the `cost` value by one. All values used for the `notes` column are exactly sixty-four characters long. This was done to ensure that a constant number of bytes would be transferred for each row; in this case each row transferred used 92 bytes.

Timing framework

To conduct the performance tests, we developed a framework for timing MobiLink synchronizations. The components of the framework are the following:

Extra tables in the consolidated database – There is one table to hold information on the tests to be performed, such as the number of runs, number of clients per run, and what every client should do. There is another table to hold the timing information, which holds four timestamps for each synchronization: the start and end time as recorded on both the client and the consolidated database. Another table is used to keep track of the next data to download to each client. There are also some temporary tables and views related to the run information and timings.

The MobiLink synchronization scripts – Several synchronization versions, or sets of scripts, are employed. Each version has a corresponding publication defined in the client database. Two versions control the setup: one is for timed synchronizations, and one is for the clients to send their timings to the consolidated database at the end of the run. For the timed synchronizations, the following scripts are defined:

- begin_connection – defines connection-level variables used by other scripts
- begin_synchronization – records the server start time
- upload_delete – defines how uploaded deletes will be applied
- upload_insert – defines how uploaded inserts will be applied
- upload_update – defines how uploaded updates will be applied
- download_cursor – controls the number of rows to be downloaded
- end_synchronization – records the server end time

The client application – Since we want to stress the MobiLink server with a lot of simultaneously-synchronizing clients, we need a small, efficient client program that can have multiple instances run simultaneously on one or more computers. So that each instance of the client would have a small memory footprint, we chose to use UltraLite clients. For efficiency, we chose to implement the client in C, using the embedded SQL interface to UltraLite. To avoid slowing the clients with disk access, we disabled the usual UltraLite behavior of flushing to disk at the beginning of synchronization. For ease of multiprocessing, we created a Windows console program that can spawn multiple instances of itself. The first instance is used as a master process, and it spawns child processes that act as clients. This application also has an option to retry, after a specified delay, any synchronizations that fail from communication errors.

A supervisor application – This coordinates clients running on different computers.

All clients are kept in step through the use of *gates*. At a gate, each client waits for all the others to reach the same point before proceeding, like the starting gate at a race. For efficiency, the gate implementation uses operating system primitives (Windows event objects are used to wait for clients on the same computer, and Windows named pipes are used for waiting across computers). The gate implementation is very efficient; after a gate, all clients start within a few seconds.

The clients use gates before and after each timed synchronization. The gate before the synchronization ensures that all clients try to start synchronizing at the same time. The gate after synchronization ensures that no other client processing occurs until all client synchronizations are completed.

For synchronizations that are timed, the following steps occur:

		Client	MobiLink Server
	1	Prepare for synchronization	
⌛	2	Wait for all other clients (via "gate")	
🕓	3	Record client start time	
	4	Start synchronization, via ULSynchronize()	
🕓	5		Record server start time (in begin_synchronization script)
	6	Perform synchronization	Perform synchronization
🕓	7		Record server end time (in end_synchronization script)
🕓	8	Record client end time	
⌛	9	Wait for all other clients (via "gate")	

Table 1: Steps in a timed synchronization.

In the above table, the icons denote extra steps associated with the timing. These extra steps either involve recording a timestamp (denoted with 🕓) or waiting for all other clients at a gate (denoted with ⌛).

The client times are recorded in the memory of the clients, and then the times are sent to the consolidated database after all the timed synchronizations. The server times are recorded directly into the consolidated database.

In discussing the timing results, we refer to the following quantities derived from the recorded timestamp values:

client sync time = client end time – client start time

server sync time = server end time – server start time

total server time = last server end time – first server start time

If each client does multiple synchronizations of the same type, they are grouped into sets of simultaneous synchronizations. All synchronizations in a set are initiated at the same time, and the next set does not start until after they have all completed. In this case, we use the following formula to calculate the total server time:

$$\text{total server time} = \sum_{i=1}^{\#\text{ of sets}} \text{total server time for set } i$$

In other words, even if there are multiple sets of simultaneous synchronizations, we only count the times spent synchronizing (i.e. between the gates).

In the test results, the total server time is reported instead of a direct throughput measure, such as the number of synchronizations divided by the total server time. Time is reported rather than rate, so that the longest values will be most prominent in the graphs, because minimizing the longest operations is most important for obtaining the best throughput. The discussions may refer to either maximizing throughput or minimizing total server time, since they are equivalent.

Testing environment

The timing tests were performed with the following software and hardware:

The tested software was SQL Anywhere 12.0.1.3554. The SQL Anywhere network database server was used as the consolidated database, the MobiLink server was used for synchronization, and the clients (described previously) used the UltraLite embedded SQL interface to UltraLite databases. In general, default options were used for MobiLink and SQL Anywhere. Unless indicated otherwise, the tests used the TCP/IP communication protocol, without encryption or compression. To improve the reproducibility of tests and maximize the load on the MobiLink server (by removing any disk bottleneck for the database), the SQL Anywhere database was run using its optional in-memory mode. The results were not significantly faster with the in-memory option, but showed less variability. Similarly, automatic adjustment of the SQL Anywhere multi-programming level was disabled to reduce variability; preliminary tests showed that the default minimum of 24 (the number of logical CPUs) gave the best results, so it was used.

The MobiLink server and SQL Anywhere ran on separate Dell PowerEdge R510 server computers. Each had two six-core Intel Xeon X5650 2.6 GHz processors and 48 GB of RAM, Intel Gigabit ET Quad Port network adaptors, and a single SAS 15K RPM hard drive. Hyperthreading was enabled, so both servers reported 24 cores. The operating system was 64-bit Windows Server 2008 R2. The 64-bit versions of the MobiLink server and SQL Anywhere database server were used.

The clients were run on a rack of up to twelve Dell PowerEdge 750 computers, running 32-bit Windows Server 2003. Each computer had a 3.2 GHz Pentium 4 processor with hyperthreading and 4 GB of RAM. For all tests, ten of the PowerEdge 750 computers were used to run the clients.

These client computers were networked to the MobiLink server computer using one Gigabit network switch and the MobiLink server computer was networked via a different port and Gigabit network switch to the consolidated database server computer. Both switches were isolated from other networks for timing runs.

Tests performed

In order to assess the performance and scalability characteristics of MobiLink, we used the timing framework to perform the following tests:

1. Vary the number of database worker threads to determine the optimal number to maximize throughput, and compare with letting the number vary automatically. This was done using 1000 clients synchronizing simultaneously.
2. Vary the delay between client retry attempts when the number of clients exceeds the MobiLink server network connection limit, and when the limit is increased to match the number of clients.
3. Vary the number of clients synchronizing simultaneously, to see if throughput drops off with more clients.
4. Vary the size of synchronizations, to see the effect on throughput.
5. Vary the network protocol used between the clients and MobiLink server, to see the effect of using different protocols, encryption, and compression.

In each case, four types of synchronization were measured:

- New data (insertions) downloaded from the consolidated database.
- Updates uploaded from the client databases, with conflict detection enabled (within the upload_update script).
- Deletions uploaded from the client database.
- Insertions uploaded from the client database.

All the timing runs follow the same steps. First, a new consolidated database is created and filled with enough data for the downloads. Then each client does one empty synchronization, to set up the client names in the MobiLink user table, to cache the remote schemas, and to make sure that MobiLink has established all database connections with the right synchronization script version. Then the timed synchronizations are performed, in the following order: downloads, updates, deletes, and then inserts. This exercises four types of synchronization. The order ensures that the consolidated database ends up with the same number of rows as it started with, and the UltraLite databases do not have more rows than those downloaded to it.

Downloads of updates and downloads of deletes were not tested. In downloads, there is no difference between downloading inserts and updates. The MobiLink server downloads the rows that are selected by your download_cursor script, and the client database uses an efficient internal “upsert” to apply the downloaded rows. Downloads of deletes would likely be faster, assuming your download_delete_cursor script only includes the primary key columns.

The details of these tests, including results and analysis, are described in the following sections.

Test 1: Optimal number of MobiLink database worker threads

In this test, we chose to use 1000 simultaneously synchronizing clients while varying the number of MobiLink database worker threads, first varying the number without automatic adjustment, then by varying the range for automatic adjustment. We chose to use 1000 clients to ensure that the MobiLink server was stressed and so that we might hit any point of diminishing throughput before running out of clients. Also, we wanted to be below the default MobiLink limit of 1024 network connections. The following were kept constant:

- 1000 clients on 10 computers with Pentium 4 3.2GHz processors
- 1000 rows per client synchronization (with 92 bytes per row)
- 10 sets of synchronizations for each client
- Total of 10 thousand synchronizations and 10 million rows

Using a fixed number of database worker threads

The results for different numbers of database worker threads, without using automatic adjustment, are shown in the following graphs (overview and detail) and table:

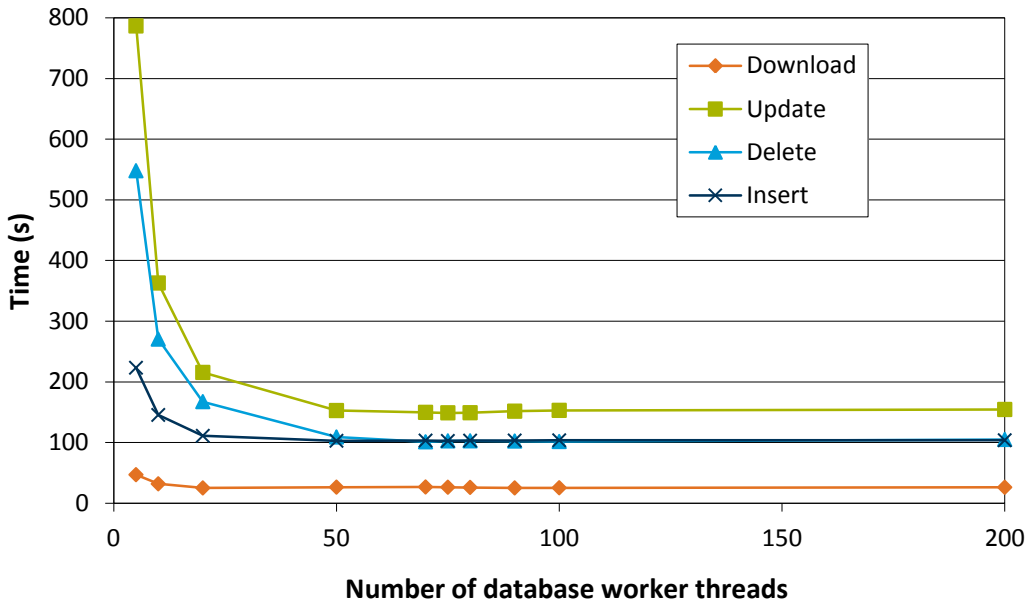


Figure 1: Total server time vs. fixed number of MobiLink database worker threads, with each point representing 1000 clients synchronizing 10 times.

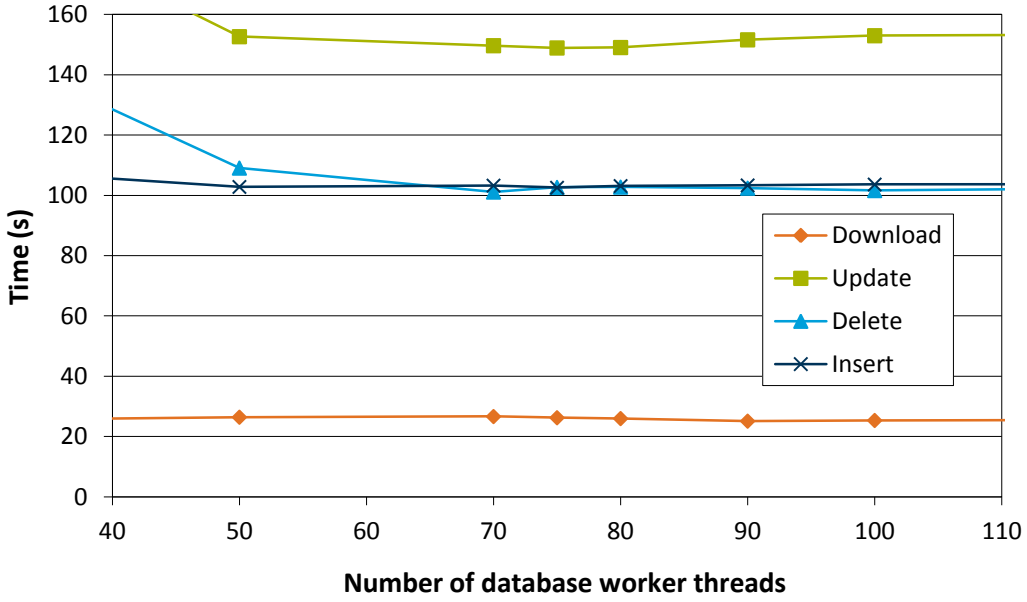


Figure 2: Zoomed-in view of Figure 1.

DB Workers	Download	Update	Delete	Insert	Sum
5	47.0	786.5	547.9	223.2	1604.6
10	31.9	362.9	270.4	145.4	810.5
20	**25.1**	215.5	167.3	111.0	518.9
50	26.5	152.7	109.1	102.8	391.0
70	26.7	149.7	**101.1**	103.3	380.8
75	26.3	**148.9**	102.7	**102.6**	**380.5**
80	26.0	149.1	102.8	103.1	381.0
90	25.1	151.6	102.4	103.4	382.5
100	25.4	153.0	101.6	103.6	383.6
200	26.2	154.4	105.1	103.8	389.4

Table 2: Total server time (in seconds) vs. fixed number of MobiLink database worker threads.

The best times in this test were obtained when using between 50 and 100 database worker threads. Of the tests performed, the one with 75 database worker threads has the smallest sum of the times for the four types of synchronization, so 75 database worker threads were used in subsequent tests. The curves are relatively flat above 50 database worker threads. Note that the timings have some variability—differences of a few seconds were common between identical runs—so often tests were repeated. For example, the test runs for 70 to 100 database worker threads were repeated five times and the run with the best total time is reported.

The optimum value for the number of database worker threads is most affected by the capabilities of the consolidated database with the load placed on it by concurrent execution of synchronization scripts. In these tests both the server computers had 24 logical CPUs and the SQL Anywhere multiprogramming level was fixed at that value, yet the best throughput was obtained with two to four times that many MobiLink database worker threads.

Using automatic adjustment of database worker threads

A new feature of the MobiLink server is the option to automatically adjust the number of database worker threads. If you specify a maximum number (via the –wm option) that is larger than the minimum number (via the –w option), the MobiLink server monitors performance and periodically adjusts the number. This works best when there is a sustained load, or at least load durations much longer than the 30 second period between adjustments. However these benchmark tests feature bursts of high load with no load in between, and the load durations are usually much less than 30 seconds, so these tests are not ideal for automatic adjustment.

To test automatic adjustment and compare with using a fixed number, we used three ranges for automatic adjustment and two fixed values for the number of database worker threads, by specifying the following MobiLink server options:

- -wm 100: Automatically adjust between the default of 5 and 100.
- -w 25 –wm 100: Automatically adjust between 25 and 100.
- -w 50 –wm 100: Automatically adjust between 50 and 100.
- -w 75: Use a fixed value of 75 (the optimal value from runs using fixed numbers).
- -w 100: Use a fixed value of 100 (same as the maximum chosen for automatic adjustment)

Runs with each of the above settings were repeated 5 times, and the best results are shown in the following graph and table:

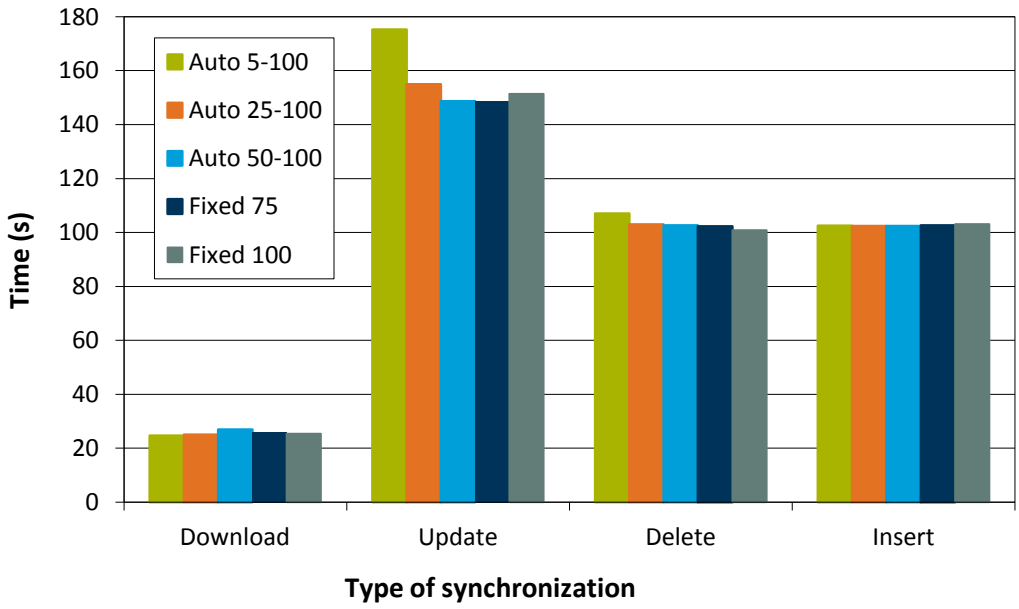


Figure 3: Best total server times from 5 runs with different database worker thread options.

DB Worker Option	Download	Update	Delete	Insert
Auto 5-100	**24.6**	175.3	107.0	102.5
Auto 25-100	25.0	154.9	103.0	**102.4**
Auto 50-100	26.9	**148.7**	102.7	102.4
Fixed 75	25.8	148.9	102.3	102.6
Fixed 100	25.4	151.4	**100.8**	103.1

Table 3: Best total server times from 5 runs with different database worker thread options.

The best results with automatic adjustment are similar or slightly better than the best results using either the near-optimal fixed number (from above) or the maximum used for the automatic range, even though the pulsed loads in these tests are not ideal for automatic adjustment. However, the variability between identical tests is generally larger, as shown in the following graph and table of the relative standard deviations:

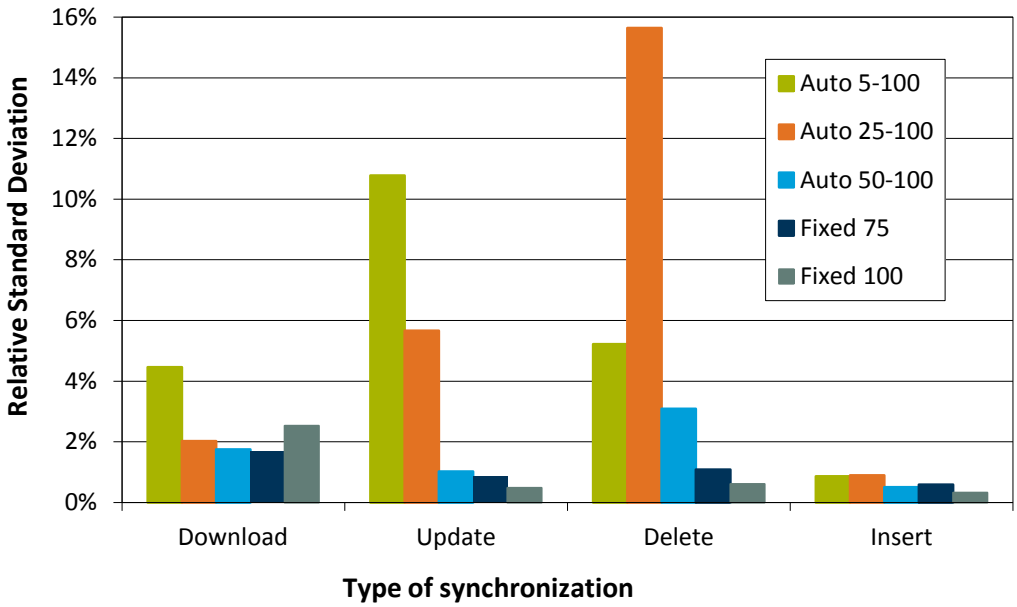


Figure 4: Relative standard deviations from 5 runs with different database worker thread options.

DB Worker Option	Download	Update	Delete	Insert
Auto 5-100	4.5%	10.8%	5.2%	0.9%
Auto 25-100	2.0%	5.7%	15.6%	0.9%
Auto 50-100	1.7%	1.0%	3.1%	0.5%
Fixed 75	**1.7%**	0.9%	1.1%	0.6%
Fixed 100	2.5%	**0.5%**	**0.6%**	**0.3%**

Table 4: Relative standard deviations from 5 runs with different database worker thread options.

The variability is lowest with the fixed number of database worker threads, and with automatic adjustment the variability is generally lower when the specified range is close to the optimal range determined from the tests without automatic variation.

The new MobiLink feature to automatically adjust the number of database worker threads works well even with these difficult tests, but it is worthwhile to determine an optimal range for automatic adjustment. Note that for

more realistic database schemas, where the consolidated database has to do more work performing the synchronizations, the adverse effect of contention with too many database worker threads may be much more apparent.

Even with this benchmark setup, there is no advantage to using 80 or more database worker threads for 1000 clients that try to synchronize simultaneously. More database worker threads results in decreased performance. The two most likely causes of decreased performance with increased database worker threads are the following:

Contention in the consolidated database - The chance of a connection being blocked by another connection increases with the number of database worker threads, and blocked connections reduce performance. In this test, we chose to have each client access separate data so that database contention would not limit performance. In most applications this is not possible, and contention in the consolidated database is usually the performance bottleneck.

Saturation of server processor or disk resources - If there is no further processing power available, adding database worker threads increases the operating system overhead for multitasking and increases hardware contention. In the tests with best performance, the MobiLink server CPU was saturated for download synchronizations and at the beginning of uploaded update synchronizations, whereas the consolidated database server CPU was saturated for the remainder of update synchronizations and for delete and insert synchronizations.

With 75 database worker threads, only 7.5% of the 1000 simultaneous synchronizations can use the consolidated database at once. The rest of the clients are either sending uploads, queuing for a database worker thread, receiving downloads, or have already finished synchronizing. From a client's perspective, the time to synchronize is longer when the client is closer to the back of the queue, because of the increased time waiting. This is a result of all synchronizations starting at almost the same time, which is the worst-case scenario. In cases where all synchronization requests are accepted, the average client time is minimized if the throughput is maximized (i.e. the total time is minimized). This keeps any queue moving most quickly.

You may also note the relative time taken by the different types of synchronizations. The downloads are usually faster than the uploads. For uploads, inserts are fastest and updates are usually slowest. When uploading updates, two versions of each updated row are uploaded (both the old and new to allow conflict detection), so more memory and network bandwidth is used.

When choosing a number of database worker threads to use, remember the following points:

- Perform tests that reflect the type of synchronizations that are expected for your real-world conditions, and choose a range for automatic adjustment that allows the best throughput.
- The best throughput requires a sufficient number of database worker threads, but too many can reduce throughput. If server CPU or disk is already saturated, or contention in the consolidated database is a bottleneck, then adding more database worker threads is counterproductive.
- With enough synchronization load to saturate either server's CPUs, using around 75 database worker threads gave the best throughput in our tests. This is about 3 times the number of CPU cores of the MobiLink server and the consolidated database server.

Test 2: Optimal client retry delay

As described in the Initiation section, the MobiLink server has a limit on the number of MobiLink client network connections, which is also a limit on the number of simultaneous synchronizations. If this limit is reached, additional synchronization requests are refused with a network error indicating the network connection was refused. Limitations of the network infrastructure, such as the operating system's network support, can also cause refused network connections.

When synchronization connection requests are refused, by either the MobiLink server limit or network limitations, the client application or user must retry the synchronization periodically until it succeeds. A MobiLink client does not automatically retry synchronizations. The delay between having a synchronization request refused and retrying can affect synchronization throughput. If the delay before retrying is too long, the MobiLink server may become idle. If the delay is too short, the MobiLink server may be kept busy refusing synchronizations instead of processing the active synchronizations.

This test attempts to determine an optimal retry delay, first when there are many more simultaneous synchronization requests than the default MobiLink server connection limit, then when the MobiLink server limit is set to the number of synchronizations so only the operating system will refuse connections. For this test, ten thousand clients attempted to synchronize one thousand rows each while varying the retry delay used by the client application. The following were kept constant:

- 10000 clients on 10 computers with Pentium 4 3.2GHz processors
- 1000 rows per client synchronization (with 92 bytes per row)
- Total of 10 thousand synchronizations and 10 million rows
- 75 MobiLink database worker threads

Exceeding the MobiLink server connection limit

The timing results when using the default MobiLink connection limit of 1024 are shown in the following graph and table:

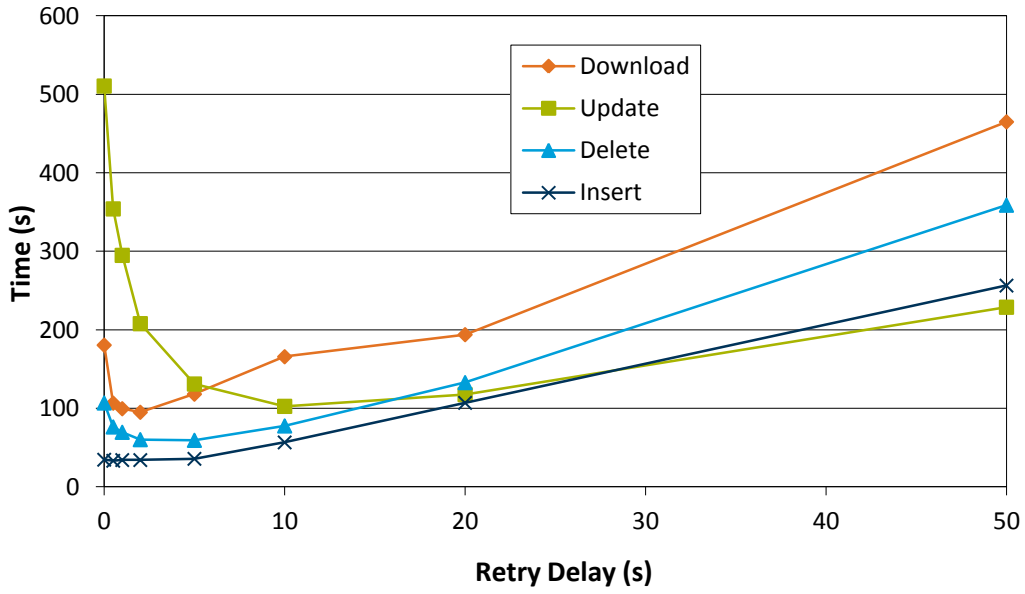


Figure 5: Total server time vs. retry delay for 10000 clients with the default connection limit of 1024.

Retry Delay	Download	Update	Delete	Insert	Sum
0	180.7	510.7	107.0	34.7	833.0
0.5	106.7	354.2	76.4	**33.3**	570.7
1	99.7	295.0	69.6	34.3	498.6
2	**95.1**	208.0	60.3	34.3	397.6
5	118.2	131.1	**59.4**	35.8	**344.6**
10	165.9	**102.7**	77.7	56.7	403.0
20	194.0	117.6	132.9	107.1	551.6
50	465.0	229.0	359.0	256.5	1309.4

Table 5: Total server time (in seconds) vs. retry delay for 10000 clients with default connection limit of 1024.

The retry delay has an effect on the throughput. The optimum retry delay depends on the type of synchronization. For example, the best time for uploading inserts occurs when the retry delay is 0.5 seconds; however for uploading updates the best retry delay is 10 seconds. In general, the best delay time corresponds to the speeds for the different types of synchronization; the quicker types of synchronization give the best throughput with shorter delays between retries. With too short of a delay, the MobiLink server spends too much time rejecting connection requests. With too long of a delay, the MobiLink server has unused capacity.

Of the tests performed, the one with a retry delay of 5 seconds has the smallest sum of the times for the four types of synchronization, so that value was used in subsequent tests that use the default connection limit.

The number of refused synchronization requests from the same runs (using the default MobiLink connection limit) is shown in the following graph and table:

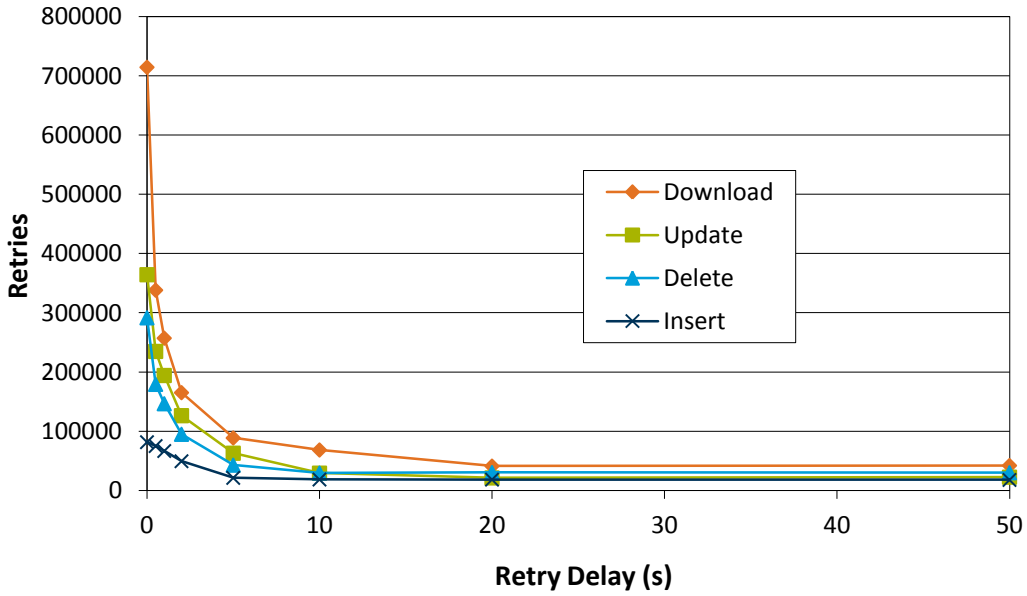


Figure 6: Number of client retries vs. the retry delay for 10000 clients with default connection limit of 1024.

Retry Delay	Download	Update	Delete	Insert
0	714616	364457	291695	81744
0.5	338340	234931	179385	75210
1	257282	194344	146663	66810
2	165317	126472	95042	49594
5	88830	63106	42950	21434
10	68530	29433	**29558**	18663
20	**41289**	**21418**	30475	18522
50	42236	22664	30134	**18007**

Table 6: Number of retries vs. the retry delay for 10000 clients with default connection limit of 1024.

From this data, you can see that with shorter retry delays, clients retry more often before successfully completing their synchronizations. Keep in mind that each entry in the table is the sum total of the number of retries for 10000 clients, and many clients have to retry more than once. In the worst case, for downloading with a delay of 0 seconds, each client retried over 71 times on average.

Increased MobiLink connection limit

The retry delay was also varied after raising the MobiLink connection limit to the number of simultaneous synchronizations. In this configuration the MobiLink server does not refuse any connections; but the operating system may still refuse connections, requiring clients to retry.

The timing results when using a MobiLink connection limit of 10000 are shown in the following graph and table:

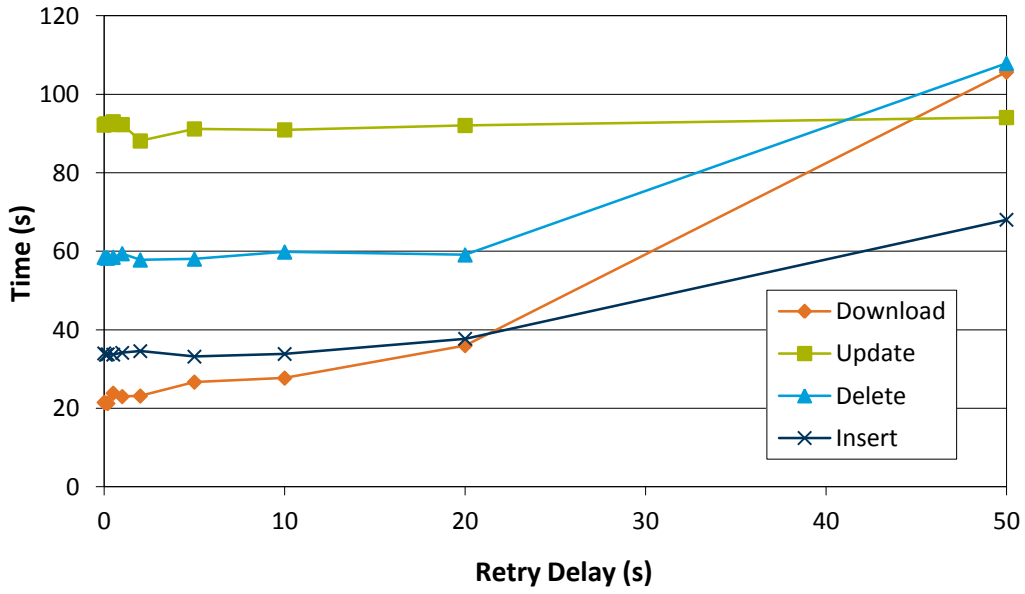


Figure 7: Total server time vs. retry delay for 10000 clients with connection limit of 10000.

Retry Delay	Download	Update	Delete	Insert	Sum
0	21.5	92.2	58.5	34.0	206.2
0.1	21.4	92.5	58.4	33.7	205.9
0.2	**21.3**	92.3	58.3	33.8	205.7
0.5	23.9	93.0	58.5	33.7	209.1
1	23.0	92.3	59.4	34.2	208.9
2	23.2	**88.1**	**57.8**	34.6	**203.7**
5	26.7	91.2	58.0	**33.2**	209.2
10	27.7	91.0	59.8	33.9	212.4
20	36.0	92.1	59.1	37.7	224.9
50	105.7	94.1	107.9	68.1	375.8

Table 7: Total server time (in seconds) vs. retry delay for 10000 clients with connection limit of 10000.

Of the tests performed, the one with a retry delay of 2 seconds has the smallest sum of the times for the four types of synchronization, so that valuewas used in subsequent tests with a connection limit of 10000.

Compared to the results with the default connection limit, the timings depend much less on the retry delay. In particular, without having the MobiLink server refuse connections there is little penalty for using retry delays less than 2 seconds. The timings are also better, especially for downloads, and the numbers of retries (shown below) are much smaller.

The number of refused synchronization requests from the same runs (using a MobiLink connection limit of 10000) is shown in the following graph and table:

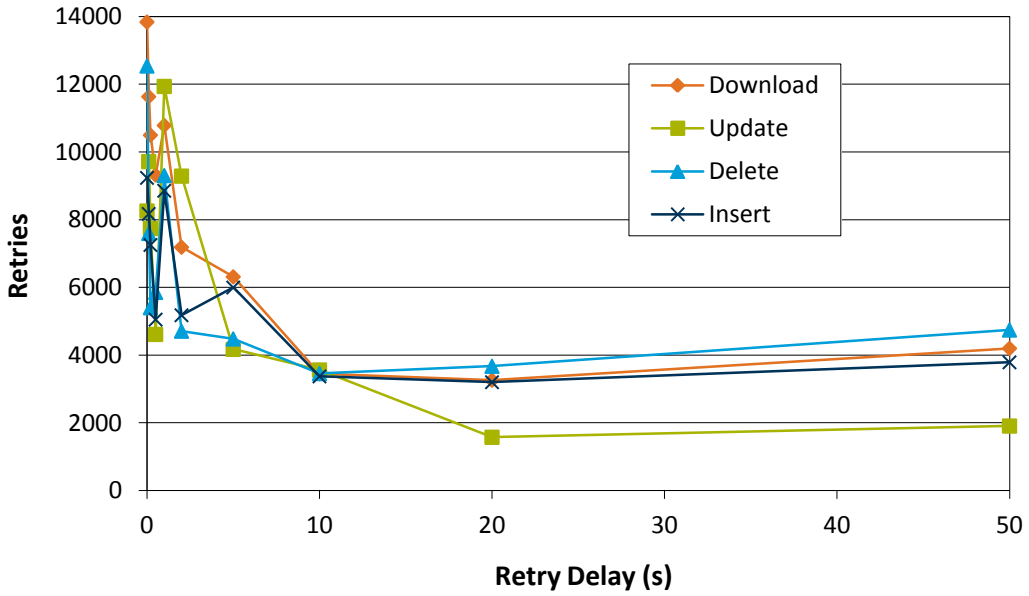


Figure 8: Number of client retries vs. the retry delay for 10000 clients with connection limit of 10000.

Retry Delay	Download	Update	Delete	Insert
0	13842	8260	12543	9238
0.1	11637	9719	7597	8173
0.2	10500	7744	5403	7256
0.5	9322	4614	5857	5052
1	10787	11940	9306	8858
2	7190	9290	4709	5180
5	6315	4173	4481	5999
10	3451	3564	**3457**	3373
20	**3262**	**1575**	3673	**3202**
50	4200	1907	4744	3789

Table 8: Number of retries vs. the retry delay for 10000 clients with connection limit of 10000.

The effect of the MobiLink connection limit is addressed in more detail in the following test.

Test 3: Scalability with respect to number of clients

This test varied the number of clients, while keeping the number of database worker threads at the best value from Test 1, to see how MobiLink scales with the number of clients. The number of synchronizations per client was adjusted to ensure that the same amount of data would be transferred overall, and that the size of each synchronization was constant. For example, with 10000 clients we used 1 synchronization (of each type) per client, with 500 clients we used 20 synchronizations per client, and with 200 clients we used 50 synchronizations each. The following were kept constant:

- Clients on 10 computers with Pentium 4 3.2GHz processors
- 1000 rows per client synchronization (with 92 bytes per row)
- Total of 10 thousand synchronizations and 10 million rows
- 75 MobiLink database worker threads

The tests for more than 1000 clients were performed twice: once with the default network connection limit of 1024 and retry delay of 5 seconds, and once with the network connection limit raised to 10000 and a retry delay of 2 seconds.

The results with the default connection limit (and a retry delay of 5 seconds) are shown in the following graph and table:

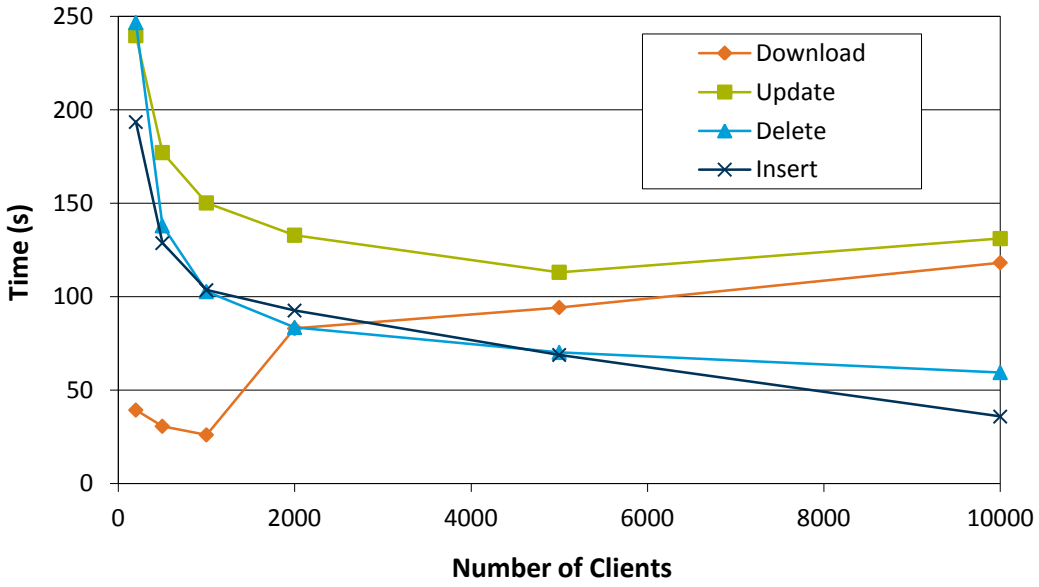


Figure 9: Total server time vs. number of clients, with each point representing 10000 synchronizations. The connection limit was the default of 1024 and the retry delay was 5 seconds.

Number of Clients	Download	Update	Delete	Insert
200	39.3	239.7	246.7	193.5
500	30.6	177.1	137.9	128.7
1000	**26.0**	150.2	102.7	103.5
2000	83.0	132.9	83.5	92.6
5000	94.2	**113.1**	70.2	68.8
10000	118.2	131.1	**59.4**	**35.8**

Table 9: Total server time (in seconds) vs. number of clients. The connection limit was the default of 1024 and the retry delay was 5 seconds.

The results with the raised connection limit are shown in the following graph and table. The graph includes the same data as above for 200, 500 and 1000 clients, since those runs did not exceed the MobiLink connection limit and did not have any retries.

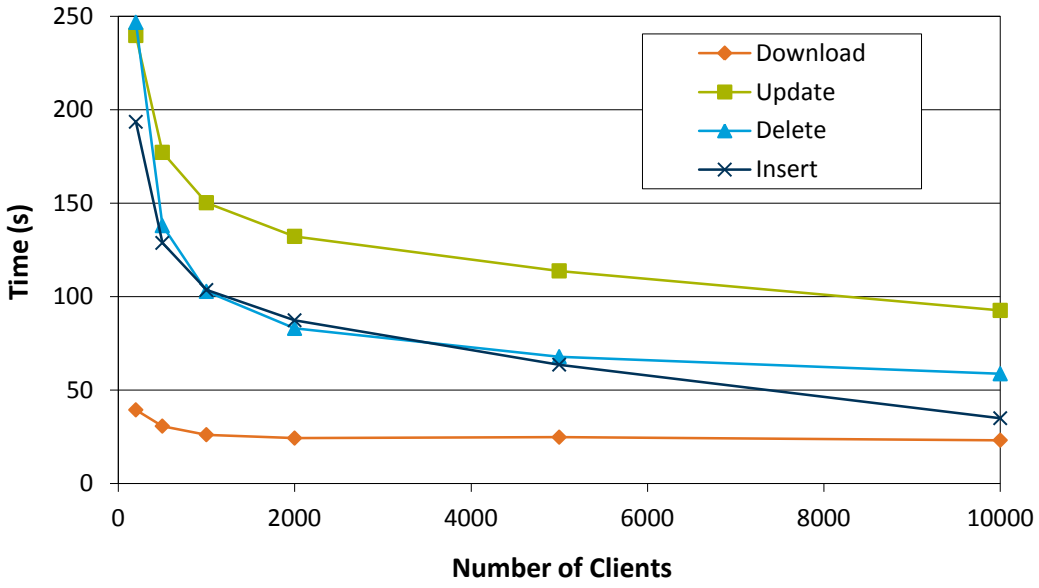


Figure 10: Total server time vs. number of clients, with each point representing 10000 synchronizations. The connection limit was 10000 and the retry delay was 2 seconds.

Number of Clients	Download	Update	Delete	Insert
2000	24.2	132.2	82.9	87.3
5000	24.7	113.7	67.8	63.5
10000	**23.1**	**92.6**	**58.7**	**34.9**

Table 10: Total server time (in seconds) vs. number of clients. The connection limit was 10000 and the retry delay was 2 seconds.

As the graphs show, the MobiLink server scales ideally with an increasing number of clients synchronizing, for all types of synchronization with the higher connection limit, and with the default limit for all types except downloads and updates. When the MobiLink server is refusing connections, the throughput drops significantly for downloads once the MobiLink connection limit is exceeded.

The observed scalability suggests that tests with a smaller number of clients could be used to estimate results with a larger number of clients.

To illustrate the effect of the connection limit, the following graph combines the previous two graphs. Solid lines are used for results with the default connection limit, and dashed lines are used for the results with the raised connection limit. The deviations between solid and dashed lines are due to exceeding the MobiLink connection limit.

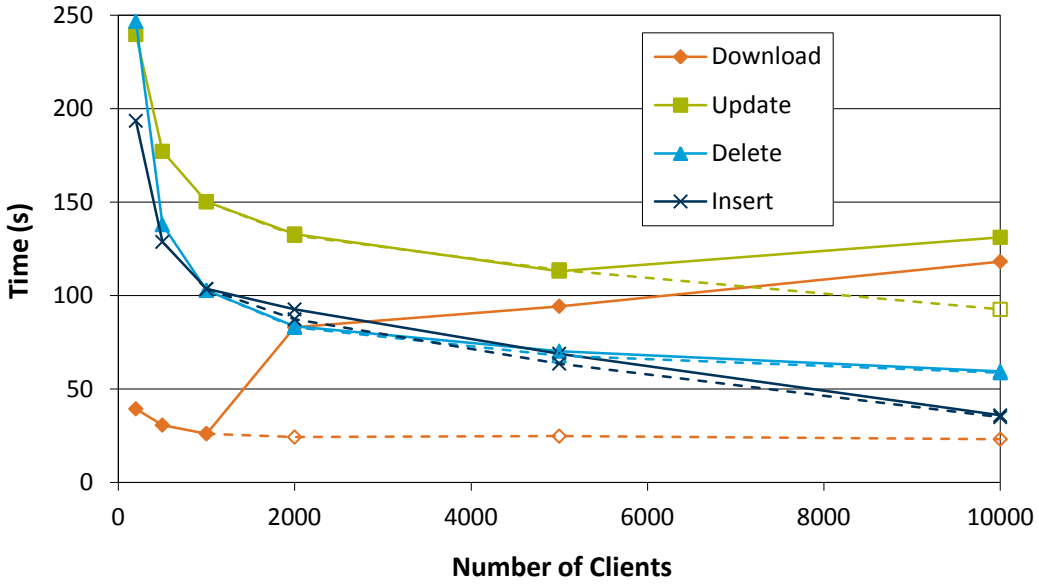


Figure 11: Total server time vs. number of clients with different connection limits. The connection limit was the default of 1024 for the solid lines and 10000 for the dashed lines.

The cost of exceeding the MobiLink connection limit is greater than we observed with the benchmarks for the MobiLink 11 Performance whitepaper, most likely due to a change in the Windows operating system. The earlier benchmarks used Windows Server 2003 and we had changed registry settings to enable a large dynamic backlog for accepting network connections. The operating system would buffer many more connection requests, even when the MobiLink connection limit was exceeded, resulting in fewer retries. In Windows Server 2000 and 2003 the dynamic backlog feature provided optional protection against "TCP/IP SYN flooding" attacks. Alternate protection is provided in Windows Server 2008, and the dynamic backlog feature is not available, so there are more retries in these tests.

When you consider using MobiLink for a large number of clients, remember the following points:

- It is important to run tests that accurately simulate real-world use to minimize total server times.
- Exceeding the MobiLink connection limit can adversely affect throughput, especially if the operating system cannot be configured to use a large backlog for connection requests.

Test 4: Size of each synchronization

This test varied the number of rows transferred in each synchronization, to see the effect of synchronization size on performance. We used 1000 clients so that the number of rows per synchronization could be widely varied without changing the total number of rows synchronized.

In this test, the number of rows per synchronization was changed without changing the total amount of data transferred by changing the number of synchronizations per client. For example, each client could synchronize 10000 rows 1 time, 5000 rows 2 times, 2000 rows 5 times, and so on. The following were kept constant:

- 1000 clients on 10 computers with Pentium 4 3.2GHz processors
- Total of 10 thousand rows synchronized per client, 10 million rows overall
- 75 MobiLink database worker threads

The results are shown in the following graph and table:

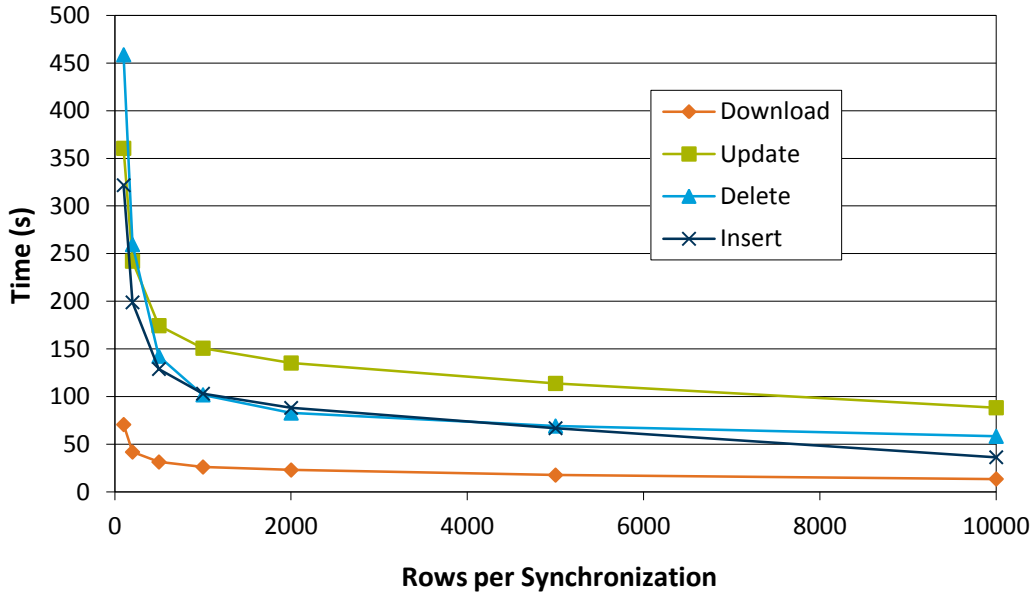


Figure 12: Total server time vs. number of rows per synchronization, with each point representing 10 million rows synchronized.

Rows	Download	Update	Delete	Insert
100	70.7	360.5	458.7	321.8
200	41.9	242.2	259.5	198.9
500	31.4	174.5	142.4	128.9
1000	26.1	150.8	101.9	103.1
2000	23.0	135.3	82.8	88.2
5000	17.7	113.9	69.2	66.9
10000	13.6	88.4	58.3	36.4

Table 11: Total server time (in seconds) vs. number of rows per synchronization.

The size of synchronizations has an effect on the throughput. A large number of small synchronizations are the slowest, and the total server times get faster as we move towards a smaller number of larger synchronizations. This is consistent with a fixed overhead for each synchronization, and its significance is diminished with larger synchronization sizes.

This becomes clearer when the total times from above are divided by how many synchronizations each client does, to give an average total server time per set of 1000 simultaneous synchronizations. The following graph shows that time versus the number of rows synchronized:

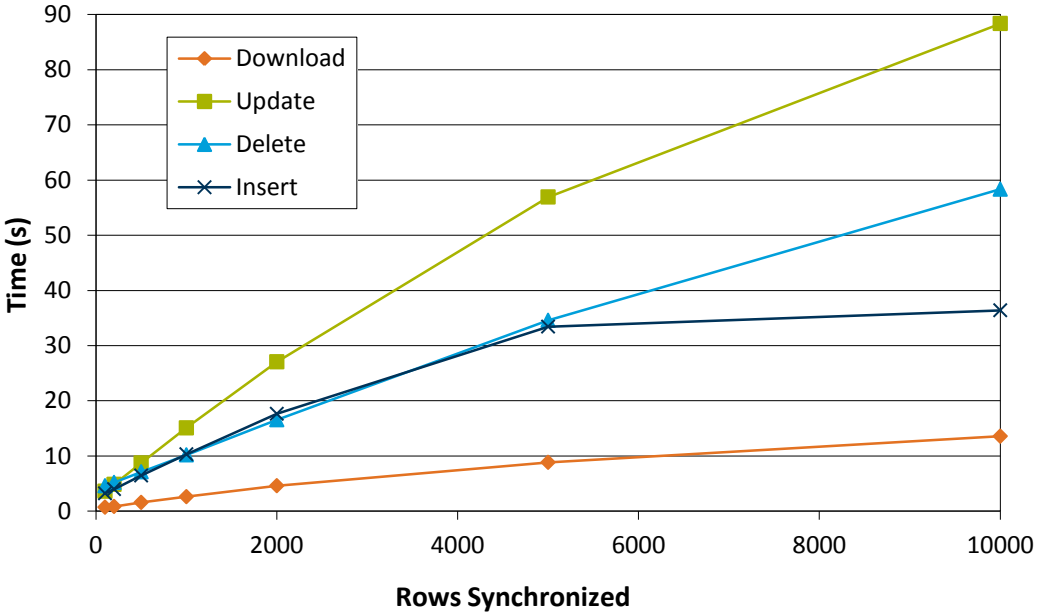


Figure 13: Average total server time for a set of 1000 synchronizations (in seconds) vs. the number of rows synchronized. For this graph, the total number of rows synchronized is not the same for each for each point.

As you can see, there is a near linear relationship between the average time for a set of 1000 synchronizations and the number of rows synchronized. With no rows synchronized (and thus no upload scripts invoked), all lines will converge on the time for 1000 empty synchronizations.

- When considering the sizes of your synchronizations, remember that there is some per-synchronization overhead, so throughput rates are slower for smaller-sized synchronizations.

Test 5: Network protocol

There are several network protocol options that can be used for the connection between the MobiLink client and the MobiLink server. For the tests so far, TCP/IP connections without encryption or compression were used. One can choose to use either TCP/IP or HTTP. With encryption those become TLS or HTTPS, and there are three options for encryption: RSA, FIPS-approved RSA (referred to as FIPS) and ECC. Note that FIPS and ECC are separately licensed options. With any of those options, one can also enable zlib compression, to reduce the size of data transferred between the MobiLink client and server.

In this test, different network protocol options were used for connections between the MobiLink client and server, to see the effect on performance. We used 1000 clients to stay below the default MobiLink limit of 1024 network connections. The following were kept constant:

- 1000 clients on 10 computers with Pentium 4 3.2GHz processors
- 1000 rows per client synchronization (with 92 bytes per row)
- 10 sets of synchronizations for each client
- Total of 10 thousand rows synchronized per client, 10 million rows overall
- 75 MobiLink database worker threads

As with the other tests, a dedicated Gigabit network was used for connections between the clients and the MobiLink server.

TCP/IP or HTTP

The results for runs without encryption are shown in the following graph and table:

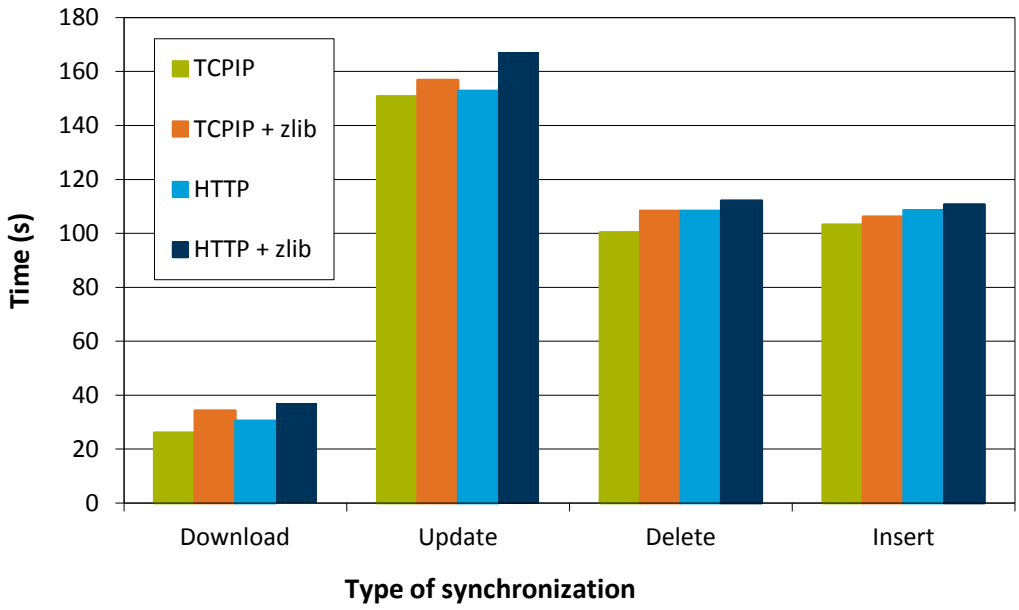


Figure 14: Total server times with TCP/IP and HTTP, with and without zlib compression.

Protocol	Download	Update	Delete	Insert
TCPIP	**26.1**	**150.8**	**100.4**	**103.2**
TCPIP + zlib	34.3	156.9	108.3	106.2
HTTP	30.5	152.9	108.3	108.6
HTTP + zlib	36.9	167.2	112.1	110.7

Table 12: Total server times with TCP/IP and HTTP, with and without zlib compression.

For these tests, times with TCP/IP are best, and those with HTTP are within 10% except for the 17% longer download time. Compression increases the time for both TCP/IP and HTTP, less than 10% for uploads, and more for download synchronizations: 31% for TCP/IP and 21% for HTTP. With slower networks, compression can improve throughput, but it does not seem to help for these tests without encryption.

Encrypted TCP/IP (TLS)

The results for runs with unencrypted TCP/IP and encrypted TCP/IP (called TLS) without compression are shown in the following graph and table:

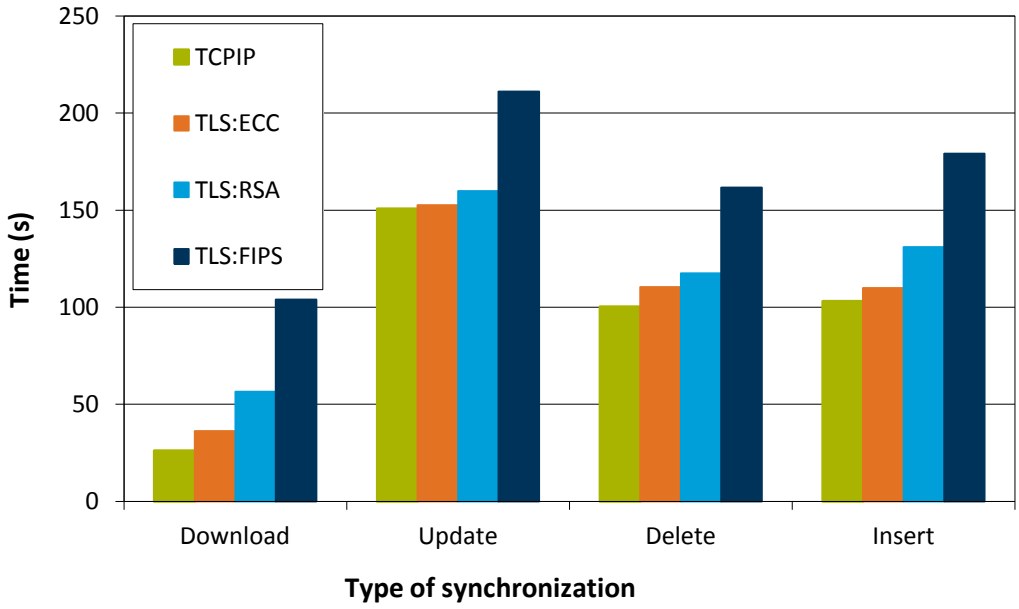


Figure 15: Total server times with TCP/IP with different encryption options and no compression.

Protocol	Download	Update	Delete	Insert
TCPIP	**26.1**	**150.8**	**100.4**	**103.2**
TLS:ECC	36.2	152.6	110.4	109.9
TLS:RSA	56.4	159.9	117.4	131.0
TLS:FIPS	103.9	211.1	161.7	179.1

Table 13: Total server times with TCP/IP with different encryption options and no compression.

The greatest difference between encryption options occurs for the download times, but for all types of synchronization the relative order is the following:

1. No encryption
2. Encryption with ECC
3. Encryption with RSA
4. Encryption with FIPS approved RSA

The results for using both encryption and compression for TCP/IP are shown in the following graphs and table:

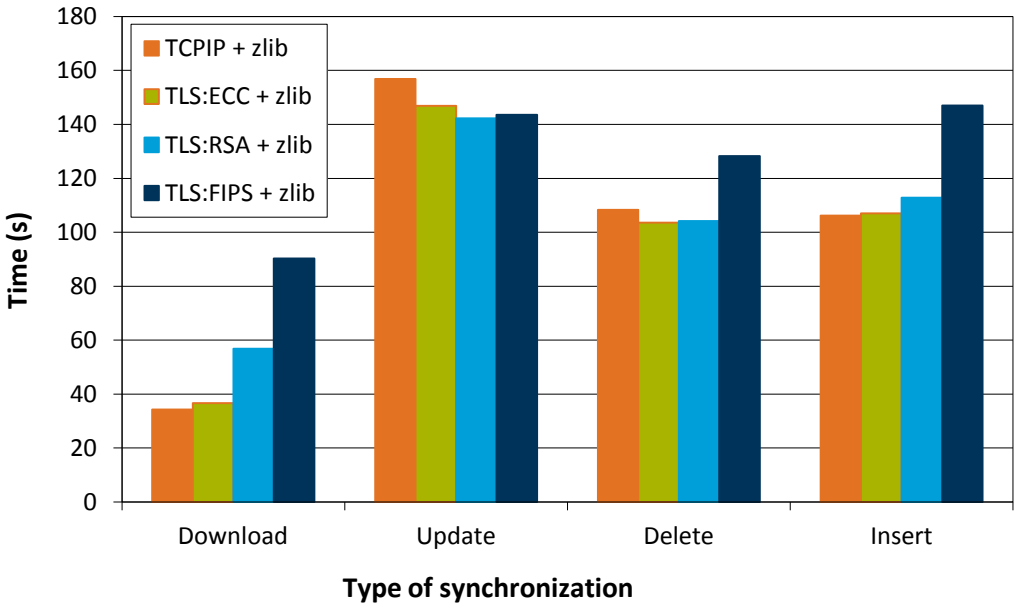


Figure 16: Total server times with TCP/IP with different encryption options and zlib compression.

Protocol	Download	Update	Delete	Insert
TCPIP + zlib	**34.3**	156.9	108.3	**106.2**
TLS:ECC + zlib	36.7	146.9	**103.6**	107.0
TLS:RSA + zlib	56.8	**142.2**	104.2	112.8
TLS:FIPS + zlib	90.3	143.6	128.3	147.0

Table 14: Total server times with TCP/IP with different encryption options and zlib compression.

With encryption, compression either adds no time, as with download using ECC and RSA, or significantly decreases the time, especially for FIPS encryption. For updates and deletes (except FIPS), the results with encryption are faster than without, and with compression there is less difference between encryption types. Even with a fast network, it seems worthwhile to always use compression if using encryption.

Encrypted HTTP (HTTPS)

The results for runs with unencrypted HTTP and encrypted HTTPS without compression are shown in the following graph and table:

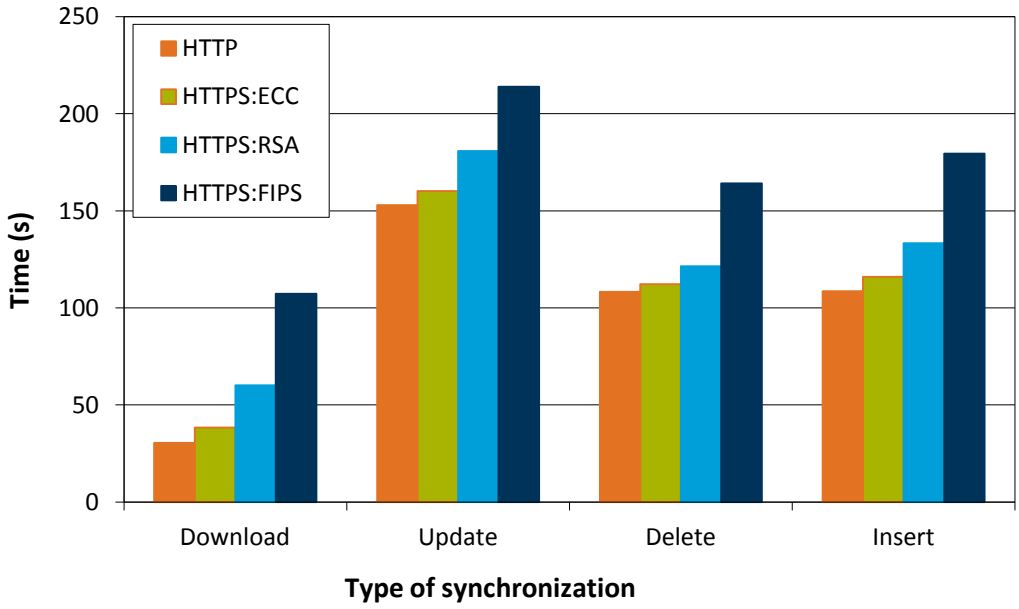


Figure 17: Total server times with HTTP with different encryption options and no compression.

Protocol	Download	Update	Delete	Insert
HTTP	**30.5**	**152.9**	**108.3**	**108.6**
HTTPS:ECC	38.4	160.2	112.2	116.0
HTTPS:RSA	60.1	180.8	121.4	133.4
HTTPS:FIPS	107.3	213.9	164.1	179.4

Table 15: Total server times with HTTP with different encryption options and no compression.

As with TCP/IP, the greatest difference between encryption options occurs for the download times, and again for all types of synchronization the relative order is the following:

1. No encryption
2. Encryption with ECC
3. Encryption with RSA
4. Encryption with FIPS approved RSA

The results for using both encryption and compression for HTTP are shown in the following graph and table:

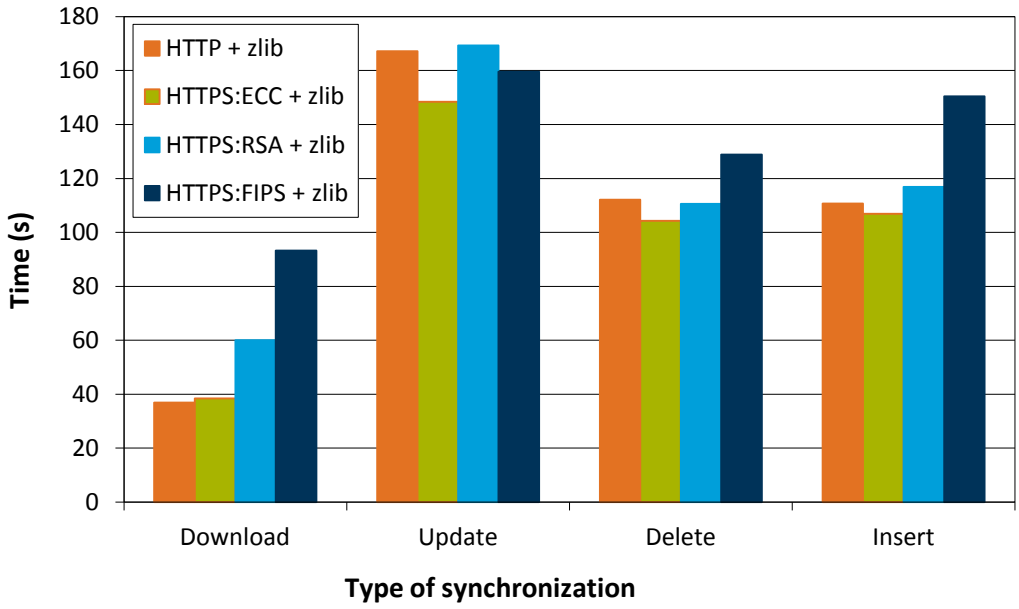


Figure 18: Total server times with HTTP with different encryption options and zlib compression.

Protocol	Download	Update	Delete	Insert
HTTP + zlib	**36.9**	167.2	112.1	110.7
HTTPS:ECC + zlib	38.5	**148.4**	**104.2**	**106.9**
HTTPS:RSA + zlib	60.0	169.3	110.5	116.9
HTTPS:FIPS + zlib	93.2	159.6	128.8	150.4

Table 16: Total server times with HTTP with different encryption options and zlib compression.

With encryption, compression either adds no time, as with download using ECC and RSA, or significantly decreases the time, especially for FIPS encryption, just as with TLS. For uploads, the results with ECC encryption are faster than with no encryption, and with compression there is less difference between encryption types. As with TLS, with HTTPS it is best to also use compression.

Hardware requirements

To assess the relative hardware requirements of the MobiLink server, we ran it on an identical computer to that for the consolidated database, and watched the CPU utilization on the two computers. We used Microsoft's Process Explorer to see the CPU utilization.

The overall CPU utilization was usually 90% to 100% on one of the server computers during timed synchronizations. The exceptions were for downloads with a small number of clients, and for the smallest number of database worker threads. When the CPU utilization approached 100% for either server computer, we were confident that the bottleneck was processing speed on that server computer. The MobiLink server computer CPU utilization was the bottleneck for downloads, and just at the very beginning of synchronizing uploads, particularly updates, when the MobiLink server was receiving the uploaded data and processing it before applying it to the consolidated database. Once uploads were being applied to the consolidated database, its CPU utilization would be 90% to 100%, whereas the MobiLink server CPU utilization would be less than 50%, typically around 35%.

For download synchronizations, the MobiLink server CPU utilization was close to 100% and the consolidated database server CPU utilization was typically less than 50%. Recall that these tests were designed to minimize the load on the consolidated database, by using simple schema and synchronization scripts and also by running the database entirely in memory, yet the MobiLink server only seems to have been the bottleneck for downloads.

As long as you have sufficient RAM, the MobiLink server uses very little disk access so, unlike most database servers, it does not require fast disk access or large storage capacity. During these tests, the consolidated database was run entirely in memory, so was not disk bound, but typical consolidated databases are disk bound, which could result in different performance characteristics.

To help ensure that the network was not a bottleneck, the network traffic between MobiLink clients and the MobiLink server used a different network and MobiLink server Ethernet port than for the traffic between the MobiLink server and the consolidated database. This gave a little better performance than when all traffic used the same network and MobiLink server Ethernet port.

When you consider the hardware requirements for MobiLink, remember the following point:

- MobiLink should need less processing power, and much less disk capacity or performance, than the consolidated database.

Recommendations

This section summarizes the MobiLink performance tips mentioned previously, and gives advice on how you might perform small-scale tests to predict both how MobiLink will perform in your large-scale deployment, and also the hardware resources you should dedicate to MobiLink.

Performance tips

Following these tips will help you to get the best performance out of MobiLink:

- Test and tune the concurrent performance of your synchronization scripts under high load. In our experience, contention in synchronization scripts is the most common performance problem in MobiLink systems.
- The MobiLink server and consolidated database operate as a system. Tuning one without tuning the other is less effective than tuning them together in an iterative fashion.
- Use a range for automatic adjustment of the number of MobiLink database worker threads that gives you optimum throughput. For example, around 75 database worker threads or automatic adjustment between 50 to 100, gave optimum throughput in our tests. A larger number did not significantly hurt performance in these tests, but with a real-world database schema contention would be much more likely with higher-than-optimum numbers, so it is better to use a range for automatic adjustment than a high fixed value.
- Choose a network connection limit (via the MobiLink -nc option) either to maximize throughput or to enforce the maximum number of simultaneous synchronizations that you want to allow. Increasing it beyond the default can improve throughput, but the operating system limitations may still refuse connections.
- Ensure that the computer running the MobiLink server is running at least a 64-bit operating system and has enough physical memory to accommodate the MobiLink server memory requirements in addition to its other memory requirements.
- Ensure that there is enough available memory to avoid swapping the MobiLink cache to disk. Check the MobiLink server message log or console for warnings like the following:

  ```
  MobiLink server has swapped memory pages out:9000 in:8000
  concurrent pages:10000
  ```

  If you see such messages, check that you have not specified a –cm or –cmax option that is too small (the default is 70% of physical memory) and that you have sufficient available physical memory.
- If you are using more than one script version, consider setting (via the MobiLink –cn option) the maximum number of MobiLink database connections to be your typical number of synchronization versions times the number of MobiLink database worker threads to reduce the need for MobiLink to close and create database connections.
- Use the minimum logging verbosity that is compatible with your business needs. By default verbose logging is off, and MobiLink does not write its log to disk. Test with different logging options to see if your logging verbosity is lowering the performance of your system.

Large deployments

A single MobiLink server can handle tens of thousands or hundreds of thousands of client databases. In these tests, we looked at up to ten thousand clients synchronizing *simultaneously*. Depending on the type of synchronization, this took from 21 to 88 seconds, which is equivalent to 408 thousand to 1.6 million client databases synchronizing over an hour, or almost 10 million to 41 million synchronizations in a day. The hardware requirements for this were quite modest; a single instance of MobiLink can handle a very large number of client databases, usually with less hardware than that required by the consolidated database to which it is connected.

If you determine that a single instance of MobiLink running on a dedicated server computer would not meet your performance or availability requirements, you can use multiple MobiLink servers in a server farm. Note that this requires purchasing the separately licensed high availability option. For example, you can use the Relay Server (included with SQL Anywhere) and multiple MobiLink servers. Another example is to use a hardware load balancer, or application delivery controller, with multiple MobiLink servers.

A synchronization hierarchy is another architecture that employs multiple MobiLink servers. In a synchronization hierarchy, clients synchronize with secondary consolidated databases that are periodically synchronized with a primary consolidated database. You could use MobiLink for both synchronization layers, if the secondary consolidated databases are SQL Anywhere databases. A synchronization hierarchy is much more complicated than using the Relay Server or a load balancer. Also, it does not address scalability directly since you still have the problem of synchronizing the total amount of data with the primary consolidated database. However a synchronization hierarchy may fit your infrastructure or business needs. For example, if your MobiLink servers need to be geographically distributed or if you want to avoid having your primary consolidated database tied up with lengthy synchronizations from clients with slow processors or slow network connections.

Applicability

While the tests reported here can give you some idea of quantitative MobiLink performance, if you want to assess performance for your MobiLink setup, you should do tests using your schema, data, consolidated database, synchronization scripts, clients, and networks.

To do this type of testing, we suggest the following procedure:

1. Determine your synchronization needs. You should know or estimate how many users will be synchronizing data over the time period of interest, to estimate how many users would be likely to synchronize simultaneously. You should also determine the characteristics of a typical synchronization, including the type and size of data that is uploaded and downloaded, and whether the data was inserted, updated, or deleted.
2. Set up a pilot implementation. As much as possible, use your actual synchronization scripts, consolidated database, and client and server hardware. Create clients that perform the typical synchronizations with typical data, using or simulating actual clients on your intended client hardware and network. For the number of clients, if it is impractical to use the total number you expect, then use the new MobiLink Replay utility to record typical synchronizations and replay them via multiple simulated clients.
3. Run the MobiLink server with the MobiLink Monitor. The MobiLink Monitor collects timing information for the phases of each synchronization, and records any warnings or errors that occur. In our tests, we found that using the MobiLink Monitor (on a separate computer than the MobiLink server) does not affect throughput. If you need to locate SQL bottlenecks, you may need to enable logging to file with some level of verbosity (such as -vcemns). The SQL Anywhere Monitor can also be used to monitor MobiLink servers. It is a web-based tool that monitors the health and availability of MobiLink servers. It collects and displays the same type of periodic information that is logged with the -ppv option for the MobiLink server.

If you are interested in using the timing framework (called MLBench) that we developed for the tests reported in this whitepaper, or a more flexible timing framework called Floodgate, contact Sybase Consulting Services (http://www.sybase.com/support/consulting). Sybase Consulting Services provides Performance and Tuning services, as well as Architectural Definition and Review, Prototyping, Solutions Development, and many other services.

.

SYBASE, INC.
WORLDWIDE HEADQUARTERS
ONE SYBASE DRIVE
DUBLIN, CA 94568-7902 USA
Tel: 1 800 8 SYBASE

www.sybase.com

SYBASE®
An SAP Company