IBM® DB2 Everyplace® Version 9.1

Performance Tuning Guide

DB2 Everyplace version 9.1 performance guidelines
and tuning recommendations

Version 1.1
July, 2008

DB2 Everyplace Development Team
IBM China Development Lab

1. Special Notices

This publication is intended to help DB2 Everyplace solution designers, system administrators, and sales representatives understand the factors that influence the performance of DB2 Everyplace® applications running on the officially supported platforms.

The information in this publication is not intended as a substitution of the IBM DB2 Everyplace product documentation provided by IBM. See the Library section of the IBM DB2 Everyplace Web Site for more information about what publications are considered to be product documentation. References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service. Information in this publication was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

The information contained in this publication was derived under specific operating and environmental conditions. While IBM has reviewed the information for accuracy under the given conditions, the results obtained in your operating environments may vary significantly. Accordingly, IBM does not provide any representations, assurances, guarantees, or warranties regarding performance. Any information about non-IBM ("vendor") products, in this document, has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness.

2. About this document

The purpose of this paper is to provide guidelines for application design and base application parameters tuning for DB2 Everyplace. Note that the tuning is affected by many factors, including the workload scenario, hardware performance, network conditions and the performance test environment. The objective of this paper is not to recommend that you use the values we used when testing our data scenarios but to make you aware of those parameters that made the most performance impact in the system. When tuning your individual systems, remember that it is important to begin with a baseline test and monitor the performance statistics to see if any parameters should be changed.

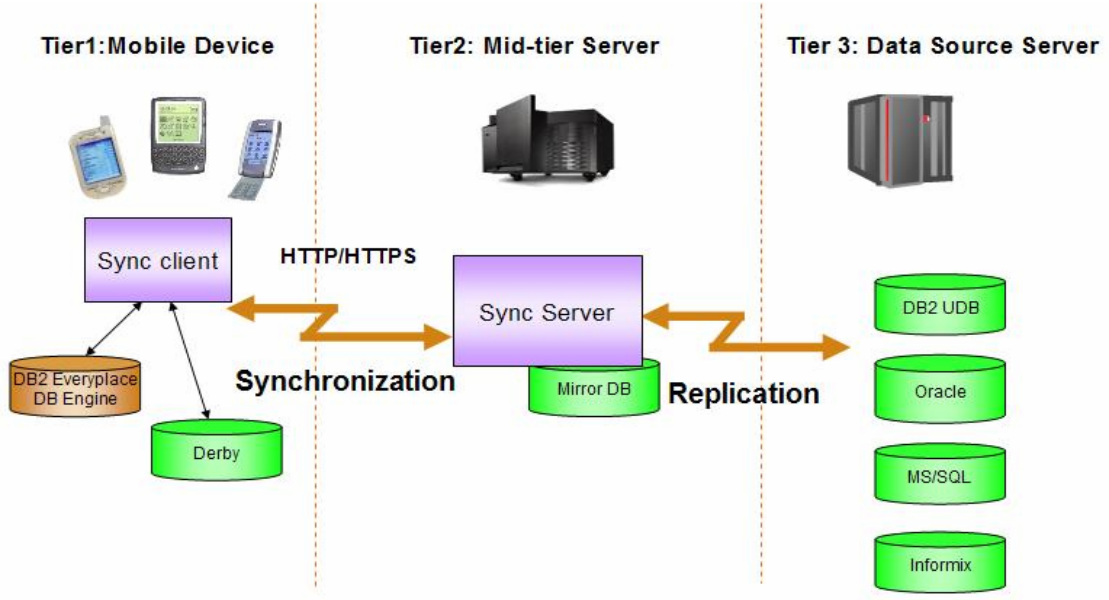
This paper provides:

- DB2 Everyplace architecture and performance problems you might meet in DB2 Everyplace-based solutions. This is mainly introduced in Chapter 3.
- Performance planning and design methodology we suggested. This is mainly described in Chapter 4.
- Performance design and tuning of configurations for DB2 Everyplace device side mobile applications and database engine. Chapter 5 mainly addresses this area.
- Performance design, tuning and scale configurations for DB2 Everyplace Sync Client, Sync Server and required back end to achieve optimal performance in production system. Chapter 6 covers this area.

In this paper when we state DB2 Everyplace, it implicitly means all DB2 Everyplace v 9.1.x builds. Some of the functionality described in this document may only work with the newer fix packs for DB2 Everyplace.

3. DB2 Everyplace Architecture and Performance Issues

3.1 DB2 Everyplace Architecture



As shown in the above diagram, the following are the main components of DB2 Everyplace product.

- Client database engine: high performance small footprint data store for managing and using data on mobile and embedded devices. In DB2 Everyplace solutions, either DB2 Everyplace database engine or Derby can be used as the client side database engine.
- Sync Client: synchronization client to synchronize data and files on mobile and embedded devices
- Sync Server: mobile user management and synchronization of enterprise data with mobile and embedded devices

In this diagram, you may notify the mirror database in the middle layer which is used to contain source database changes. The main purpose of mirror database is to avoid client side to access the source database directly. And there are two important different terms in DB2 Everyplace environments:

- Synchronization: the normal data synchronization between Sync Client and Sync Server. And the data will be exchanged between client side and the mirror database
- Replication: the data exchange process between mirror database and source database. Customer can either trigger replication manually or typically let it run periodically in a production system

  Client side change will be updated to mirror database after synchronization and

this change will be updated to source database after replication; and server side change will be updated to mirror database after replication and updated to client side after synchronization.

Besides the main components mentioned above, in a DB2 Everyplace application environment, there are other components that you need concern when you are considering performance issues:

- Client side mobile applications : mobile application design will heavily impact client side application performance
- Mirror database: In DB2 Everyplace environment, mirror database is very important for both synchronization and replication performance. Since both operations need write and read a lot data to and from mirror database.
- Source database: source database's performance will mainly impact replication performance, especially in first replication and there is a big data set in source database.

3.2 Main DB2 Everyplace performance issues

What is a performance problem? Basically we think that a performance problem means that one or more activities cost exceptional long time in the process of completing one operation. Performance issue may occur in normal operation such as daily data synchronization. In some conditions, such as big data set and a lot of concurrent users, you may meet performance problem easily. The following are main DB2 Everyplace performance problems you may meet in production system

- Slow database engine operation. Client application meet database performance problem when performance database engine operations. This problem probably needs analysis of database engine, mobile application design and SQL statements related.
- Slow data synchronization. The data synchronization between client side and server side is slow and may meet a lot of sync time out in the middle of the synchronization. Since data synchronization is an operation that includes a few parties, such as Sync Client, Sync Server and network, the sync performance problem may relate to each of this party.
- Slow data replication. As we described in above, data replication exchange the data between mirror database and source database. So slow data replication mainly relates to source database performance or mirror database performance.

4. Application planning for performance and scalability

DB2 Everyplace is a middleware for you to build mobile applications. The process for ensuring that a production DB2 Everyplace application system will have acceptable performance and scalability includes more than tuning parameters after installation. This section describes our recommended performance methodology best practices, whose scope ranges from the early stages of planning a DB2 Everyplace system through the routine monitoring of the production system.

4.1 Performance Methodology

This section provides an overview of our recommended methodology for performance planning

1. Understand and document your projected workload, performance, and scalability objectives:
   - Number of PDA and laptop users.
   - Frequently performed operations, and the number of each performed per hour (synchronization, refresh and replication; client side database operations), both for typical users and peak hours.
   - Average data size of synchronization and number of synchronizations per hour.
   - Use of features with significant sync performance impact (for example, filters for sync, join filter feature, refresh frequency, static data, upload subscriptions, replication frequency).
   - Layout physical database architecture (for example buffer pools and table spaces based on the data model and disks available in your environment)
2. Your IBM representative has a capacity planning document or tool for DB2 Everyplace to help you make an initial rough sizing of the hardware configurations that should be able to support your workload. Please keep in mind that the capacity document or tool can only give a rough suggestion of the configuration and it's difficult to use a tool give an accurate recommendation. So output of the capacity planning tool or document can only be used for a reference and start point for the system
3. Plan and document your overall system topology and configuration.
4. Read and understand the performance design and tuning recommendations in this document. Be aware that performance tuning involves trade offs appropriate tuning techniques and parameter values depend on the unique circumstances of your configuration and workload.
5. Plan for an initial tuning period to maximize confidence and reduce risk before going into production. If possible, use automated test tools to drive a multi user test load based on your projected workload. Collect performance data in the

performance testing. During this tuning period iterate focusing on one area at a time, changing only a tuning parameters. Run the test workload to evaluate the effect of each set of changes before making additional tuning changes.

6. In production, perform routine performance maintenance, and monitor the performance of DB2 Everyplace Sync Server systems. The following tips can be used:
   - For synchronization, perform periodic database RUNSTATS and REORG for mirror database tables, as described in the tuning recommendations.
   - Maintain a periodic performance profile of key performance metrics (CPU, memory, network, and disk utilizations, for example, as well as overall throughput and the response times for key operations), using the available performance monitoring tools on your platform.
   - Validate your original workload projections against your production system.
   - Document the performance profiles over time to observe trends before they become a problem

4.2 Collecting performance data

The following list performance data that you may collect in performance testing.

Static data

• Physical configuration of all the machines used for the tests (processors speed, location of databases (remote /local), number of clients on each clustered node).
• Size of the initial data.
• Server and client build level.
• WAS configuration (specifically JVM settings like thread pool size, heap size, etc).
• Sync Server configuration (collect all property values stored in DSY.PROPERTIES table in DSYCTLDB database, timeout values, and trace levels).
• Database configuration settings for source, mirror, DSYCTLDB and DSYMSGDB (specifically, lock list, buffer pool size, log size, connections settings)
• Database engine side table schemas, data size and configurations etc

Runtime data

• Collect resource utilization during the execution of the scenarios: CPU, Disk.
• Network statistics (I/O).
• Memory usage and statistics.
• Average response time

5. Database Engine Performance Design and Tuning

DB2 Everyplace database engine performance tuning could fall into three main categories. In section 5.1, general design considerations and suggestions for DB2 Everyplace mobile applications and database are introduced. Section 5.2 mainly address DB2 Everyplace engine related attributes that could impact performance and section 5.3 give hints for database engine query performance tuning.

5.1. Applications and Database Design

5.1.1 Avoiding table scans of large tables

DB2 Everyplace is fast and efficient, but when tables are huge, scanning tables might take longer than a user would expect. It's even worse if you then ask to sort this data.

5.1.2 Always create indexes

Have you ever thought what it would be like to look up a phone number in the phone book of a major metropolitan city if the book were not indexed by name? For example, to look up the phone number for John Jones, you could not go straight to the *J* page. You would have to read the entire book. That is what a table scan is like. DB2 Everyplace has to read the entire table to retrieve what you are looking for unless you create useful indexes on your table.

5.1.3 Create useful indexes:

Indexes are useful when a query contains a WHERE clause. Without a WHERE clause, DB2 Everyplace is supposed to return all the data in the table, and so a table scan is the correct behavior.
DB2 Everyplace creates indexes on tables in the following situations:

- When you define a primary key, unique, or foreign key constraint on a table.
- When you explicitly create an index on a table with a CREATE INDEX statement.

For an index to be useful for a particular statement, one of the columns in the statement's WHERE clause must be the first column in the index's key.
Indexes provide some other benefits as well:

- If all the data requested are in the index, DB2 Everyplace does not have to go to the table at all.
- For operations that require a sort (ORDER BY), if DB2 Everyplace uses the index to retrieve the data, it does not have to perform a separate sorting step for some of these operations in some situations.

Create indexes for the most common queries you have on the application. Keep in mind that indexes create a big penalty on insert, update, delete queries and will affect synchronization time. Only use necessary indexes on the client. This technique will improve synchronization time. Remember that removing the indexes might degrade the performance of the client application, so you need to have balance between optimizing for the synchronization performance and SQL query performance. DB2 Everyplace database engine has built in automatic REORG, but applications can also call it to improve the queries performance when there is a large data change on the tables. Remember that REORG is performed after each synchronization.

Note: DB2 Everyplace does not support indexing on columns with data types like BLOB.

5.1.4 Bidirectional Index

DB2 Everyplace supports bidirectional index scanning. Since version 8.1 a single index can support several different output ordering queries of a table. Some other mobile databases require that at least 4 indexes are created to handle the queries listed below. For example, the index IDX1 on (X, Y) could assist queries with:

ORDER BY X ASC, Y ASC
ORDER BY X ASC, Y DESC
ORDER BY X DESC, Y ASC
ORDER BY X DESC, Y DESC
ORDER BY X ASC
ORDER BY X DESC

5.1.5 Join order:

For some queries, join order can make the difference between a table scan (expensive) and an index scan (cheap). Here's an example:

```
select ht.hotel_id, ha.stay_date, ht.depart_time
from hotels ht, Hotelavailability ha
where ht.hotel_id = ha.hotel_id and
ht.room_number = ha.room_number
and ht.bed_type = 'KING'
and ht.smoking_room = 'NO'
order by ha.stay_date
```

If DB2 Everyplace chooses *Hotels* as the outer table, it can use the index on *Hotels* to retrieve qualifying rows. Then it need only look up data in *HotelAvailability* three

times, once for each qualifying row. And to retrieve the appropriate rows from *HotelAvailability*, it can use an index for *HotelAvailability*'s *hotel_id* column instead of scanning the entire table.
If DB2 Everyplace chooses the other order, with *HotelAvailability* as the outer table, it will have to probe the *Hotels* table for *every row*, not just three rows, because there are no qualifications on the *HotelAvailability* table. DB2 Everyplace usually chooses a good join order. However, as with index use, you should make sure.

5.1.6 Remote stored procedure adapter

This feature allows client device to execute stored procedures on a remote database for real time query or transaction. The result set is stored locally in a temporary DB2 Everyplace table. This feature requires the device to be in connected mode and the performance maybe not good if the network is not stable, and should only be used in extreme cases, where a local query to the local database can not be performed because of security or real time reason.

5.2. DB2 Everyplace Attributes

DB2 Everyplace lets you configure behavior or attributes of a specific database, a specific connection or a specific statement through the use of a series of attributes.

5.2.1 Data Flush Mode - Write Through vs. Buffered

DB2 Everyplace could flush the updated data either by handing to the operating system or pushing directly to the storage media after a COMMIT (in manual commit mode) or after completion of the SQL statement (in auto-commit mode).

- SQL_IO_BUFFERED -Changes are sent to the operating system. You can recover the data if an application stops unexpectedly, but you might not be able to recover the data if the operating system stops. Applications using this mode (the default) will perform considerably faster.
- SQL_IO_WRITETHROUGH -Changes are sent directly to the storage media. You can recover the data if either the application or the operating system stops unexpectedly. This mode increases reliability but decreases performance. Use this mode when data integrity is important or if hardware or operating system failure is a concern.

So for performance critical application, you could consider choose flushing data by operating system. However, for critical data application, considering in case the updated data might not be written to disk for the operating system, you might consider IO_WRITETHROUGH mode.
Note: this attribute is only enabled for Windows, Linux on X86 only.

5.2.2 Configure Bufferpool Size

There is an opportunity for you to choose proper bufferpool size according to your real environment. The size of bufferpool actually specifies the amount of memory that the DB2 Everyplace database should reserve for its bufferpools. You could set this connection attribute by SQL_ATTR_BUFFERPOOL_SIZE. If this value is not a multiple of 4K (4096 bytes), DB2 Everyplace rounds it down to the next smallest multiple of 4K.

NOTE: You cannot change the size of the bufferpool if a connection to the database already exists. New connections will use the bufferpool size of the existing connection. SQLConnect() will return a warning.

5.2.3 Autocommit Off

DB2 Everyplace supports transaction in which the application could commit each SQL statement either automatically or manually. You could set the transaction commit mode via connection attribute SQL_ATTR_AUTOCOMMIT. The supported values are:

- SQL_AUTOCOMMIT_ON - DB2 Everyplace automatically commits each statement, which is DB2 Everyplace's default behavior. In autocommit mode, all updates that are performed by a statement are made persistent automatically after the statement is executed. Statement-level atomicity is guaranteed.
- SQL_AUTOCOMMIT_OFF - The application must manually, explicitly commit or rollback a transaction.

If there are many non-query statements in your application, you could consider autocommit off to let all the updated data flushed to disk together to decrease the cost of frequent IO.

5.2.4 Main Memory vs. Storage Card

Memory cards have normally slower IO times than main memory on PDA devices. If memory cards are considered database storage, it will make database access slower. Different devices have different software and hardware implementation for the memory card readers. The IO time to access memory cards vary greatly on the device even if the memory card is the same. Test your solution with the actual device selected for production and measure its performance with production load before making the decision to use memory cards.

5.2.5 Checksum

Checksum is used for DB2 Everyplace database engine to verify that none of the files that DB2 Everyplace uses to store a database has been altered or corrupted. For any change in one page, calculation for checksum both the page header and the left content will be counted for checksum enabled mode. So for performance critical application, you could improve the database performance via CHECKSUM_OFF mode for no any change track calculation time. It is a connection attribute. The supported values are:

- SQL_TABLE_CHECKSUM_OFF - DB2 Everyplace stores files without checksum information. This is the default value.
- SQL_TABLE_CHECKSUM_ON - DB2 Everyplace stores files with checksum information.

Note: You can change this attribute only before establishing a database connection. Any changes you make to this attribute will only take effect when connecting to a directory that does not yet contain any catalog tables. After the first CREATE TABLE statement all the database files will be stored with checksums enabled.

5.2.6 Reorg

After a period of execution, there might be a lot of unused space, especially due to a great amount of deletion. Thus, for applications with numerous deletions from the DB2 Everyplace database, you could consider use REORG more often at the time that performance is not so critical, since REORG will take much time for moving space to disk. For performance critical application, you could disable the automatic REORG during the application but could leave the reorg between the application intervals.
For DB2 Everyplace, the attribute SQL_ATTR_REORG_MODE is used to specify whether automatic database reorganization is performed on user created tables and whether explicit REORG SQL statements are allowed. The supported values are:

- SQL_REORG_ENABLED - This is the system default. Database reorganization can be performed by DB2 Everyplace or explicitly by the user with a REORG.
- SQL statement. SQL_REORG_DISABLED - REORG SQL statements are restricted and automatic database reorganization of user-created tables is disabled. This option cannot be specified for an open cursor.

5.3. Query Performance

Query performance is not a one-time consideration. You should consider it throughout the design, development, and production phases of the application development life cycle.
SQL is a very flexible language, which means that there are many ways to get the same correct result. This flexibility also means that some queries are better than others

in taking advantage of the DB2 Everyplace optimizer's strengths.
During query execution, the DB2 Everyplace optimizer chooses a query access plan for each SQL statement. The optimizer models the execution cost of many alternative access plans and chooses the one with the minimal estimated cost. If a query contains many complex search conditions, the DB2 Everyplace optimizer can rewrite the predicate in some cases, but there are some cases where it cannot.
The time to prepare or compile an SQL statement can be long for complex queries. You can help minimize statement compilation time by correctly designing and configuring your database. You can use the DB2 Everyplace EXPLAIN facility to tune queries. The DB2 Everyplace compiler can capture information about the access plans of static or dynamic queries. Use this captured information to understand how individual statements are run so that you can tune them to improve performance.

5.3.1 Writing SQL Statements

SQL is a powerful language that enables you to specify relational expressions in syntactically different but semantically equivalent ways. However, some semantically equivalent variations are easier to optimize than others. Although the DB2 Everyplace optimizer has query rewrite capability, it might not always be able to rewrite an SQL statement into the most optimal form. There are also certain SQL constructs that can limit the access plans considered by the query optimizer. The following sections describe certain SQL constructs that should be avoided and provide suggestions for how to replace or avoid them.

5.3.1.1. Avoid complex expressions in search conditions

Avoid using complex expressions in search conditions where the expressions might limit the choices of access plans that can be used to apply the predicate. During the query rewrite phase of optimization, the optimizer can rewrite expressions to choose the best access plan; however, it cannot handle all possibilities.

5.3.1.2. Avoid join predicates on expressions

Using join predicates on expressions limits the join method to nested loop. Some examples of joins with expressions are as follows

*WHERE SALES.PRICE * SALES.DISCOUNT = TRANS.FINAL_PRICE*
WHERE UPPER(CUST.LASTNAME) = TRANS.NAME

5.3.1.3. Avoid expressions over columns in local predicates

Instead of applying an expression over columns in a local predicate, use the inverse of the expression. Consider the following examples:

XPRESSN(C) = 'constant'

INTEGER(TRANS_DATE)/100 = 200802

You can rewrite these statements as follows:

C = INVERSEXPRESSN('constant')

TRANS_DATE BETWEEN 20080201 AND 20080229

Applying expressions over columns prevents the use of index start and stop keys, requires extra processing at query execution time. These expressions also prevent query rewrite optimizations such as recognizing when columns are equivalent, replacing columns with constants, and recognizing when at most one row will be returned. Further optimizations are possible after it can be proven that at most one row will be returned, so the lost optimization opportunities are further compounded. Consider the following query:

SELECT LASTNAME, CUST_ID, CUST_CODE FROM CUST

*WHERE (CUST_ID * 100) + INT(CUST_CODE) = 123456 ORDER BY 1,2,3*

You can rewrite it as follows:

SELECT LASTNAME, CUST_ID, CUST_CODE FROM CUST

WHERE CUST_ID = 1234 AND CUST_CODE = '56' ORDER BY 1,2,3

If there is a unique index defined on CUST_ID, the rewritten version of the query enables the query optimizer to recognize that at most one row will be returned. This avoids introducing an unnecessary SORT operation. It also enables the CUST_ID and CUST_CODE columns to be replaced by 1234 and ‘56’, avoiding copying values from the data or index pages. Finally, it enables the predicate on CUST_ID to be applied as an index start or stop key.

5.3.1.4. Avoid non-equality join predicates

Join predicates that use comparison operators other than equality should be avoided because the join method is limited to nested loop. However, nonequality join predicates cannot always be avoided. When they are necessary, ensure that an appropriate index exists on either table because the join predicates will be applied on the nested loop join inner.

5.3.1.5. Avoid multiple aggregations with the DISTINCT keyword

Avoid using queries that perform multiple DISTINCT aggregations in the same subselect, which are expensive to run. Consider the following example:

SELECT SUM(DISTINCT REBATE), AVG(DISTINCT DISCOUNT) FROM DAILY_SALES GROUP BY PROD_KEY;

To determine the set of distinct REBATE values and distinct DISCOUNT values, the input stream from the PROD_KEY table might need to be sorted twice, of which the cost for execution is too expensive.

5.3.1.6. Avoid redundant predicates

Avoid redundant predicates, especially when they occur across different tables. In some cases, the optimizer cannot detect that the predicates are redundant, which might result in longer query runtime.
The following section provides an example with two redundant predicates that are defined in the WHERE condition. Identical predicates are defined on the time dimension (DT) and fact (F) table:

AND ("DT"."SID_0CALMONTH" = 199605
AND "F". "SID_0CALMONTH" = 199605
OR "DT"."SID_0CALMONTH" = 199705
AND "F". "SID_0CALMONTH" = 199705)
AND NOT ("DT"."SID_0CALMONTH" = 199803
AND "F". "SID_0CALMONTH" = 199803)

The DB2 Everyplace optimizer does not recognize the predicates as identical, and treats them as independent. This leads to underestimation of cardinalities, suboptimal query access plans and longer query run times.
The above predicates are transferred to the ones shown below. Only the predicates on the fact table column "SID_0CALMONTH" remain:

AND ("F". "SID_0CALMONTH" = 199605
OR "F". "SID_0CALMONTH" = 199705)
AND NOT ("F". "SID_0CALMONTH" = 199803)

5.3.1.7. Use parameter markers to reduce compilation time for dynamic queries

The DB2 Everyplace database engine can avoid recompiling a dynamic SQL statement that has been previously run by storing the access section and statement text in the dynamic statement cache. A subsequent PREPARE request for this statement will attempt to find the access section in the dynamic statement cache, avoiding compilation. However, statements that only differ because of literals used in predicates will not match. For example, the following 2 statements are considered different by the dynamic statement cache:

SELECT AGE FROM EMPLOYEE WHERE EMP_ID = 26790
SELECT AGE FROM EMPLOYEE WHERE EMP_ID = 77543

Even relatively simple SQL statements can result in excessive system CPU usage due to statement compilation, if they are run very frequently. If your system experiences this type of performance problem, consider changing the application to use parameter markers to pass the predicate value to the DB2 Everyplace compiler, rather than explicitly including it in the SQL statement. However, the access plan might not be optimal for complex queries that use parameter markers in predicates.

5.3.2 Tuning SQL statements using the EXPLAIN facility

The EXPLAIN facility is used to display the query access plan chosen by the query optimizer to run an SQL statement. It contains extensive details about the relational operations used to run the SQL statement such as the plan operators, their arguments, order of execution, and costs. Since the query access plan is one of the most critical factors in query performance, it is important to be able to understand the explain facility output in order to diagnose query performance problems. Explain information is typically used to:

- understand why application performance has changed
- evaluate performance tuning efforts

5.3.2.1 Analyzing performance changes

To understand the reasons for changes in query performance, you need the before and after explain information which you can obtain by performing the following steps:
1. Capture “explain” information for the query before you make any changes and save the resulting explain tables. Alternatively, you might save the output from the db2exfmt explain tool. However, having the explain information in the explain tables allows easy querying with SQL, in order to perform more sophisticated analysis.
2. Save or print the current catalog statistics. You might also use the db2look productivity tool to help perform this task. The EXPLAIN contains all the relevant access plans at the time the statement is explained. The db2exfmt tool will automatically format the statistics contained in the snapshot.
3. Save or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE INDEX. The db2look tool will also perform this task.
The information that you collect in this way provides a reference point for future analysis. For dynamic SQL statements, you can collect this information when you run your application for the first time. For static SQL statements, you can also collect this information at bind time. To analyze a performance change, you compare the information that you collected with the information that you collected about the query and environment when you start your analysis.
As a simple example, your analysis might show that an index is no longer being used as part of the access plan. You might then choose to perform one of these actions:
. Reorganize the index
. Gather explain information when rebinding your query.
After you perform one of the actions, examine the access plan again. If the index is used again, performance of the query might no longer be a problem. If the index is still not used or if performance is still a problem, perform a second action and examine the results. Repeat these steps until the problem is resolved.

5.3.3 Conclusion for query performance

To avoid potential performance issues, use these best practices when you write your SQL statements and configure your settings. You can minimize this impact in several ways:

- By writing SQL statements that DB2 Everyplace optimizer can more easily optimize. The DB2 Everyplace optimizer might not be able to efficiently run SQL statements that contain non-equality join predicates and other complex search conditions.

- By using the DB2 Everyplace EXPLAIN functionality to review potential query access plans, especially which index on the table is used, and then determine how to tune queries for best performance.

5.4. More Tips

5.4.1 Disconnect DB2 Everyplace Database properly

DB2 Everyplace features crash recovery that restores the state of committed transactions in the event that the database exits unexpectedly, for example during a power failure. The recovery processing happens the next time the database is started after the unexpected exit. Your application can reduce the amount of work that the database has to do to start up the next time by shutting it down in an orderly fashion. Furthermore, for autocommit off applications, if there are numerous SQL statements that don’t commit after a long period, the log file will increase dramatically

5.4.2 Put DB2 Everyplace first in your classpath

The structure of your classpath can affect DB2 Everyplace startup time and the time required to load a particular class. The classpath is searched linearly, so locate DB2 Everyplace's libraries *at the beginning of the classpath* so that they are found first. If the *classpath* first points to a directory that contains multiple files, booting DB2 Everyplace can be very slow.

6. Synchronization Performance Design and Tuning

Section 6.1 describes the considerations for performance when design synchronization solutions based on DB2 Everyplace Sync Server and Sync client. This design might mainly impact design and organization of the subscriptions, and the table structure that need to be synchronized, etc..
Section 6.2 describes the performance tuning hints for the synchronization process between mobile devices and Sync Server's mirror database. It gives tuning advice for different parameters and design for different component. And Section 6.3 describes the performance tuning hints for replication process between mirror database and source database.

6.1 Synchronization Design for Performance

6.1.1 Data size

Synchronize only data that needs to be synced. Tables are organized into subscriptions sets. Each subscription set can be enabled, disabled, or reset separately to avoid syncing unnecessary tables. This technique can be used to set up different synchronization needs. For example: Daily data vs Weekly data.
Synchronize often when you make client changes. Sync Client executes the REORG function on database at the end of each synchronization to reclaim the space of the deleted records. This will reduce table size and improve client database performance.
Remove columns that are not needed from client tables using vertical filtering. Minimize the number of records needed on the client side using horizontal filters on the server side. This will reduce data size and sync time for the synchronizations.
Try to separate static data from volatile data in your table design for the source. DB2 Everyplace uses row-based synchronization so that when any column in a row changes the server sends the full row to the client and vice versa. If a source table contains a column that does not change, consider moving it to another table.
Example:
Initial source table design:

TPROD (PRODUCT_ID, NAME, PRICE, STOCK, GROUP)

This table should be broken into the following two tables:

TPROD(PRODUCT_ID, PRICE, STOCK)
TPRODINF(PRODUCT_ID, NAME, GROUP)

If the table TPROD changes very frequently, the two table design will minimize the data size in the synchronization, because the NAME and GROUP fields are static, and PRICE and STOCK are volatile.

6.1.2 Memory cards

Memory cards have normally slower IO times than main memory on PDA devices. If this option is being considered database storage it will make synchronization and database access slower. Different devices have different software and hardware implementation for the memory card readers. The IO time to access memory cards vary greatly on the device even if the memory card is the same. Test your solution with the actual device selected for production and measure its performance with production load before making the decision to use memory cards

6.1.3 Data partitioning

Data partitioning means using different mirror for different subscriptions. Subscriptions with different replication needs should go to different mirrors. Using this technique allows the Sync Server to be able to replicate faster and provide the replicated data to the clients in faster cycles.
Example:
• data only changed on the weekend, for example, use replication on demand on a Saturday for product catalog
• data changed during the day, use short replication cycles, for examples prices, stocks, orders, to allow the clients to have access to the new data during the day.

6.1.4 Table aggregation to reduce subscription number

Include maximum number of tables in one subscription to minimize number of subscriptions going to a specific client. This provides faster synchronization, because Sync Client will exchanges a smaller number of messages with the server.

6.1.5 Upload subscription considerations

Upload subscriptions insert data directly on the source, skipping the mirror. This type of subscription should be used when there is a need of reflecting the changes on the source at the time of synchronization. Remember that upload subscription is insert only and does not scale as well as JDBC in concurrency environments because it needs a direct connection to the source database.

6.1.6 Filters

• Use simple filters per group or user with parameters to minimize the data being sent to the device.
Example:

USERID = $USER AND DATA > :activemonth

$USER and :activemonth will be replaced by Sync Server at synchronization time. $USER will get the value of the user that is performing the synchronization and :activemonth is a parameter defined in the group level where this user belongs.
• Minimize the use of complex filters (join filter) to improve concurrency and sync time. Complex filters have a huge penalty in performance and scalability. It creates a complexity overhead for the DB2 queries executed by Sync Server on the mirror database, which will affect concurrency and sync time.
• Use filters between source and mirror to minimize the size of the mirror, this will reduce replication time and improve synchronization response because mirror tables will contain less rows.
• Create indexes on the filtered columns in the mirror database to improve the query performance of Sync Server. [Editing a subscription using Mobile Device Administration Console or XML Scripting might drop the indexes for the tables in that subscription depending on the changes that were performed, remember to recreate the indexes again].

6.1.7 Replication cycle

Use long replication cycles if possible. Replication competes for the same resources that synchronization uses. If your replication cycle is too frequent this will reduce concurrency and server availability. Keep in mind that you should maximize the replication time to achieve better concurrency. A drawback to this design is that replication could take longer because of the amount of data that will be accumulated in the mirror for those large replication cycles.

6.1.8 Replication

• Use different database mirrors for different types of users that use a different data set (partition data). This allows DB2 to minimize transaction logs and perform better in query response time. It also allows replication to happen on a mirror without affecting the other. This way the users of a mirror do not get affected by the load and replications on the other mirror.
• Keep mirror database close to Sync Server. This will allow DB2 queries to be executed faster. If possible have a dedicated TCP/IP backbone between mirror and

Sync Server. Use the same design between source and Sync Server if you use Upload subscriptions.

- Do not share TCP/IP connection between Sync Server and mirror database with other machines. Minimize other network traffic not related with Sync Server to achieve better query performance.
- Maximize replication time cycle. When Sync Server is replicating, it does not allow synchronization to occur on the mirror being replicated. Use replication on demand, or large replication cycles if possible, when concurrency is high.

6.1.9 Conflicts

Design solutions with no/minimal conflicts if possible. Conflicts between the device and the source are very expensive for several reasons:

- record goes back to the client and is rejected.
- application has to have special code to handle rejected records if the normal rules of Sync Server does not apply.
- under the normal scenarios Sync Server will send 2 records to the device for each rejected record. One is the record with the data accepted by the server that will replace the client record, the other is the rejected record.

With DB2 Everyplace, it is easy to design solutions without conflicts. A conflict-free design can be achieved by using several techniques such as:

- upload subscriptions
- horizontal and vertical filters
- parameters per user and group and $USERNAME.
- complex filters (lookup filters)

Conflict records handling on the client side can be done using Sync API. This API will return the rejected records to the client application. The application can then handle/apply the record as desired. For example, it can insert the record into another table or save it into memory.

6.1.10 Partition of user or data properly

Partition users into groups that have some filter component in common. Then make a subscription for each group so that each group of users uses a different mirror. Create the filters between the source and the mirrors, instead of between the mirror and the client. Do not forget to create appropriate indexes on the source. If needed, apply additional filters from mirror to client to reduce even further the data that goes to the client.

A simple example:

Imagine 1000 users that can be divided into 10 groups. Suppose the source database

has 10 MB of data, but each group is interested in only 20 % of the data and each client is interested in only 10 % of the data. The mirror for each subscription will contain only 2 MB of data instead of 10 MB. A smaller mirror will make the inserts, updates, deletes and selects required for synchronization and replication faster.
A drawback to this design is that replication could take longer because of the filtering needed to partition the data. There is also the issue that if you have more than one subscription against the same source table you will find that data can take 2 replication cycles to get to all the clients.

6.2 Synchronization Performance Tuning

In the synchronization process, a few components are used in server side, including the databases, application server, web server, and DB2 Everyplace Sync Servers. From the tuning side, every component needs to be well tuned for a better synchronization performance.

6.2.1 Sync Client Tuning Considerations

Timeout

The timeout value on the client sets the maximum time the client will wait for a server reply. The default is 30 seconds. For synchronizations of 5 to 15 Mb, it is more common to use a timeout value of 3 to 5 minutes.
Increase the timeout value of the client until the synchronization success rate is acceptable, otherwise the client will timeout before it receives the reply message from Sync Server and synchronization will fail. Next time the client synchronizes, it will resume from where it stopped. However, it is good practice to increase this value to the acceptable waiting time for a synchronization to be performed.
If the server is busy, especially when concurrency is high, increase the timeout value of the client until the synchronization success rate is acceptable. However, it does not mean that time out value should be set to too big. Too long wait time or continuous timeout on client side might imply server side performance issue.

Message size (Network setting)

You can tune the message size parameter to improve synchronization performance. The most important factors to consider when selecting a message size is the quality, speed, latency and reliability of the network.
If you have a high quality network with a high latency you might reduce the sync time if the number of messages is reduced. This can be achieved by increasing the message size which will reduce the number of handshakes between server and client and the number of messages.

On the other hand if your network is not reliable, it is better to use a smaller message size to allow a faster recovery. In this case message, delivery failure will occur more frequently, and the client has requested or sent less data each time a failure occurs. It is very important especially to set message size smaller when the network is very slow and not stable.
The setting of message size is completed by setting the network type in client side. Faster network means bigger message size. Please refer to DB2 Everyplace information center for the mapping of network setting and exact message size.

6.2.2 Tuning Remote Back End Database Servers

In DB2 Everyplace product, several back end database servers are required for core functionality. These servers include database server for the control database (DSYCTLDB), a database server for the messages exchanged between client and server (DSYMSGDB) and a database server for the mirror database. DB2 Everyplace can also be used with other user repository (e.g. Lightweight Directory Access Protocol (LDAP)) in Lotus Expeditor installation, but this scenario is not covered in this paper.
For maximum performance, different back end servers should reside on separate systems from DB2 Everyplace system. The primary benefit of having such a configuration is to avoid resource contention from multiple servers residing on a single server. Back end servers sharing DB2 Everyplace resources and databases resources would impact the amount of throughput we could achieve. Back end servers for the DB2 Everyplace configuration include:
• Source database: This is the database containing data that need to be synchronized. In most of the cases, this database is an existing database and it should stay in an independent machine; source database's performance mainly impact performance of replication. This database needs to be well tuned if the database size to be replicated is big.
• Mirror database: Mirror database contains changes of source data. Its performance is very important for data synchronization. In a typical environment, it is recommended that you put this database in an independent machine. Don't put it on the same computer as source database machine or Sync Server machine. And make sure there is big tablespace and bufferpool for the tables created by the subscriptions
• DSYCTLDB database: This database mainly contains configuration information of the system. Its data content would not be very big. It is recommended that you perform periodical maintenance for this database. In the basic configuration mode, this database is in the same machine with Sync Server and in the remote configuration mode, this database is in a different machine with Sync Server.
• DSYMSGDB: This database mainly contains the data messages exchanged between Sync Clients and Sync Server. This database's performance will also impact synchronization performance. It is recommended that you put this database on a

different machine with the mirror database. And make sure there is big tablespace and bufferpool for the main table dsy.msgstore.

The following sections give some basic configuration suggestions for different databases when there are more than 50 sync users. These parameters are used in our test environment and you may use them as a base for your production system

DSYCTLDB database:
db2 update db config for dsyctldb using applheapsz 2048
db2 update db config for dsyctldb using app_ctl_heap_sz 1024
db2 update db config for dsyctldb using maxappls 256
db2 update db config for dsyctldb using locklist 2048
db2 update db config for dsyctldb using buffpage 10000
db2 update db config for dsyctldb using dbheap 2048
db2 update db config for dsyctldb using sortheap 4096

Mirror/Source database:
db2 update db config for mirrordb using applheapsz 2048
db2 update db config for mirrordb using app_ctl_heap_sz 1024
db2 update db config for mirrordb using maxappls 512
db2 update db config for mirrordb using locklist 2048
db2 update db config for mirrordb using maxlocks 500
db2 update db config for mirrordb using buffpage 20000
db2 update db config for mirrordb using dbheap 4096
db2 update db config for mirrordb using sortheap 4096

dsymsgdb database
db2 update db config for dsymsgdb using applheapsz 2048
db2 update db config for dsymsgdb using app_ctl_heap_sz 1024
db2 update db config for dsymsgdb using maxappls 512
db2 update db config for dsymsgdb using locklist 2048
db2 update db config for dsymsgdb using maxlocks 500
db2 update db config for dsymsgdb using buffpage 20000
db2 update db config for dsymsgdb using dbheap 4096
db2 update db config for dsymsgdb using sortheap 4096

Along with the above database parameter changes, there are several other important updates needed for the databases:

- Ensure that the database server has an adequate number of disks especially for the MIRROR database, DSYMSGDB and SOURCE database. In our test, we used 4 to 6 drives. You can create tablespaces on these disks and specify the tablespace used by subscription tables in the mirror database when creating subscription. Also, if possible, due to constant logging from the databases, dedicate the database logs to a separate

disk or even more than one disks from where the physical databases reside.

• Use SMS table space type for DB2 temporary table spaces for DB2 Everyplace MIRROR and SOURCE databases. Failure to follow this recommendation could result in excessive time spent in buffer writes on UPDATES and other SQL requests which require nested queries, as well as excessive disk utilization on the DB2 server.

• The table space type (SMS, DMS) for SOURCE and MIRROR has impact on Sync Server performance. For SQL queries performed by Sync Server that are insert operations (for example, transferring rows from source to mirror, incoming rows from the clients), preallocated space like in a DMS improves the SQL queries performance. The administrative overhead of DMS is higher and requires administrator to ensure that DMS space does not get exhausted. If you use DMS, make sure that you have enough spaces and that mirror pruning is opened. If you don't have a good feeling what is the best type for your scenario, use SMS. Refer to DB2 administration guide for further details on using SMS/DMS.

• For MIRROR/SOURCE database and DSYMSGDB, create big buffer pools (e.g. 200M or more for big data set) and assign it to be used with the tablespace you created.

• The SOURCE/MIRROR, DSYCTLDB and DSYMSGDB, need to provide Sync Server with sufficient connections. If there are not enough connections available, clients cannot be synchronized. This is very important in concurrency scenarios. Make sure the instance parameter, MaxAgents, and the database parameters, MaxAppls and Avg_Appls, are set properly. Also, adjust the DB2 Everyplace parameters (refer to section 6.2.5), “Jdbc.MaxConnections”, “DB2ClientSession.Connections” and “messagestore.db.conpoolsize” for the connection configuration. If the concurrent client number is n, set those three parameters at least 2*n.

• For SOURCE/MIRROR and DSYCTLDB, adjust the PCKCACHESZ to cache more prepared SQL statements to increase performance. Make sure that 150 or more SQL statements can be cached in each database.

• Reorganize the tables in the MIRROR and DSYCTLDB databases periodically. When a row in a table is deleted, the space occupied by the row is not necessarily reclaimed until the table is reorganized. Perform the following command on each databases, "db2 reorgchk update statistics on table all". This issues the utility RUNSTATS command that scans the tables and index space to gather information of space and efficiency of indexes. This information is stored in the DB2 catalog and used by the SQL optimizer to select access paths to data. The reorgchk utility then produces a report, recommending which tables should be reorganized. To reorganize any of the listed tables, issue the "db2 reorg table <table-name>" command. Since DB2 Everyplace 9.1.2, there is an internal task to enable automatic reorg and runstats for mirror database tables, you can enable it to make the maintenance work automatically(refer to http://publib.boulder.ibm.com/infocenter/db2e/v9r1f1/index.jsp?topic=/com.ibm.db2e.doc/common/dbewhatsnew.htm).

• For MIRROR/SOURCE and DSYMSGDB, ensure that the usual necessary transaction log size needed for production is available through primary transaction log

files (see DB2 LOGPRIMARY parameter for details). Primary transaction log is preallocated at database creation time. Secondary transaction log is allocated as needed.

• Use the database parameter LOGBUFSZ to change the amount of log buffer. This allows specifying the amount of database shared memory to be used as buffer for the log records before writing them to the disks. Buffering the log records allows more efficient log I/O, as the records would be written to the disk less frequently and more log records will be written during each write.

• Application heap size (APPHEAPSZ) is a database configuration parameter that defines the number of private memory pages available to be used by the database manager on behalf of a specific agent or sub agent. The heap is allocated when an agent or sub agent is initialized for an application. The amount allocated is the minimum amount needed to process the request given to the agent or sub agent. As the agent or sub agent requires more heap space to process larger SQL statements, the database manager will allocate memory as needed, up to the maximum specified by this parameter. You should change the default value to the optimal value for your production environment.

6.2.3 Tuning web server parameters

In DB2 Everyplace production system, you may use an independent web server to receive requests and distributes the request to application server. IBM HTTP Server is the main web server supported by DB2 Everyplace now and the following are some tuning tips for IBM HTTP server:

• Keep Alive Timeout: This value indicates how long the HTTP Server should allow a persistent connection to remain open and inactive before closing it. We set this value to be 10 seconds; this is the max time a message can take to be applied on the Sync Client side. This is because we want to be conservative in our test, and therefore we assume each user will be opening new TCP connections every time a message takes more than 10 seconds to be applied on client side. However, in a live environment, it can be helpful to increase the KeepAlive timeout. Keep in mind that a higher KeepAlive timeout may increase contention for HTTP server processes. If you are running out of HTTP processes, decrease this value.

• MaxClients (for UNIX systems) & ThreadsPerChild (for Windows): 300 or higher depending on workload

• KeepAlive: On

• MaxKeepAliveRequests: 0 (to allow an unlimited number of requests)

• MaxRequestsPerChild: 250000

• MinSpareServers: 5, which is the default; but monitor the activity of the servers to determine if this needs to be increased

• MaxSpareServers: 10, which is the default; but monitor the activity of the servers to determine if this needs to be increased

• StartServers: 100 (varies depending on value of MaxClients/ThreadsPerChild)

• Turn off additional logging for each requests: In the IBM HTTP Server config file, access logging can be turn off by commenting out the line “#CustomLog /usr/HTTPServer/logs/access_log common.”

6.2.4 Application Server Tuning

Parameter tunings are based on WebSphere Application Server 6.0. This section will again focus on recommendations stemming from performance impacts experienced by our team. It is not our goal to define recommendations for every tuning parameter offered by the WebSphere Application Server product. For more details on tuning WebSphere Application Server, see the Tuning Section of the InfoCenter located at: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp

Parameter tuning

Java Virtual Machine (JVM): Set the JVM heap size larger than 256MB. For best and most consistent throughput, set the starting minimum (ms) and maximum(mx) to be the same size. Also, remember that the value for the JVM heap size is directly related to the amount of physical memory for the system. In our tests, with a physical allocation of 4GB RAM, we ranged between a fixed JVM heap size of 512MB to 768MB. Never set the JVM heap size larger than the physical memory on the system.

Garbage collection: Using the “Xnoclassgc” parameter will allow for more class reuse, thus causing less garbage collections to occur.

Servlet engine pool size: Increase this parameter higher than the default. You need set it according to concurrent sync users and other users that are using the same application server. In our testing, we used 100 for both the minimum and maximum settings. Increase this value if all the servlet threads are busy most of the time.

Vertical scalability

In our testing, we implemented vertical cloning across our WebSphere Application Server systems. Vertical cloning refers to the practice of defining multiple clones of an application server on the same physical machine. Vertical cloning provides a mechanism to create multiple JVM processes that fully utilize the processing power to near 100 %. During testing, a single application server did not drive the load to an optimal near 100 %, thus we added a second clone of the application server. One best practice in regards to cloning is to not blindly add clones to the environment. The best way is to begin by performance testing with one instance of the application server and then measuring the throughput and system’s resource utilization. For configuring a vertical clone, if the CPU is utilized 85 % (or more), it is better not to add an additional clone. Thus, it is hard to pinpoint the number of horizontal or vertical clones any one environment will need without doing extensive tests. Overall, workload management does provide a huge impact in trying to meet your overall performance objectives.

6.2.5 DB2 Everyplace Tuning

Join filter (out of scope filters)
By turning JoinFilter filtering off you can gain some extra performance boost, especially when you defined a lot of filters on subscription. Change the value of property RowFilter.OutOfScope.Delete to 1. Read the JoinFilter documentation to understand what are the problems that may create in your environment. It may incur filter out of scope problem if you turned off join filter feature and filters are defined. Refer to Info Center for details of join filter feature and out of scope problem

Filters and application design
Another performance improvement can be achieved in the case where you don't use filters at all, between the mirror and the client. However, in many production systems, filters are a necessary function that must used. When designing tables and filters on subscription, avoid using too many cross table reference in filters. It is a best practice to refer one or two tables only in filter design. It will need a big performance overhead if complex filter is defined and join filter is enabled

Table DSY.log
The table DSY.log increases very quickly in size in a production environment. Make sure you can predict the size and clean this table by using the DB2 Everyplace default property Log.KeepDays (default 7) and Log.PruneToSize (default 10000), or by cleaning it manually.

Logging and Tracing
It is important that the amount of logging performed during a high workload or production environment is monitored closely. DB2 Everyplace has incorporated several runtime logging capabilities. Besides the logs in dsy.log table, you can change trace level for the information that will be in trace file. You can change trace setting in "DSYGdflt.properties" property file (which is in "DB2 Everyplaceveryplace_root/server/properties/com/ibm/mobileservices/" where "DB2 Everyplaceveryplace_root" is the DB2 Everyplace root directory). You can also change trace setting using "dsytrace" command. These two ways both require that you restart Sync Server. Starting from DB2 Everyplace 9.1.2, there is a new way to change trace level without restarting Sync Server. You can access a web page to change trace level and you don't need restart Sync Server. Please refer to info center (http://publib.boulder.ibm.com/infocenter/db2e/v9r1f1/index.jsp?topic=/com.ibm.db2e.doc/common/dbewhatsnew.htm).

Properties
There are some important properties in either property file or dsy.properties table that will impact performance

• Jdbc.MaxConnections , messagestore.db.connpoolsize and DB2ClientSession.Connections (in dsy.properties table). Those parameters are the size of database connection pools related to concurrency. Increase them if the number of concurrent synchronizations is increasing; otherwise the synchronizations will be serialized and will take longer. Remember that the number of concurrent synchronizations does not always directly relate with the number of total users in the system. The important factor to estimate is the number of concurrent synchronizations, not the total number of concurrent users on the system. Set these two values at least double the number of the concurrent sync users.

• ThreadPoolCount(in DSYGdflt.properties file) – This parameter specifies the number of threads Sync Server can allocate to be able to perform internal tasks for synchronization and replication . It is a good practice to increase this value in higher concurrency environments. Set this value at least double the number of the concurrent sync users. Consider also increasing the JVM maximum Java heap size (Xmx) when you increase this parameter. This will reduce Garbage Collection and avoid out of memory exceptions.

• RowFilter.OutOfScope.Delete (in dsy.properties table) – Join filter default value, this parameter has 2 values: 0 and 1. If set to 1, it means that Join Filter will be activated by default every time you create a subscription. This setting only changes default value, you can set value for every subscription table separately when you are defining subscriptions (see DB2 Everyplace documentation for more information on this parameter).

• Server.ResponseTimeout(in dsy.properties table) – The Server.ResponseTimeout value will allow the Sync Server more time in heavy load scenarios before it informs the client to retry again. Adjust this one as needed, for example increase it to 60 seconds for larger synchronizations.

•MaxSyncPeriod.Days(in dsy.properties table) – The period in days which a user may go without synchronizing a Subscription before it will be required to perform a refresh on the next synchronization request. This setting is important for controlling storage of mirror database. It will control the history of days to be kept in mirror database tables. Setting this period shorter will result in a more aggressive pruning of the control tables. improve the performance of synchronizations and replications. and reducing the storage consumption of Sync Server. A value of -1 (default) will disable this pruning function.

For those parameters in property file, you can change them in the file directly. And for those parameters in dsy.properties table, you can change them in the table directly or use “dsysetproperty” command. Below is an example sample how these values can be updated. Refer to DB2 Everyplace documentation for the defaults values.

Example:

dsysetproperty DSYGdflt Jdbc.MaxConnections 50

6.3 Replication Performance Tuning

Replication is the process of transferring data between source and mirror database. You may meet replication performance problem when there is large data set.

Database log size needed for replication
Typically the mirror and the source need the same amount of log size for replication. It is very important to predict the replication log size for DB2 Everyplace product. If a replication cycle is not performed because there is not enough log size, the replication will not be performed until the log size is changed to the required log size. Log size in DB2 Everyplace is normally several times the max data size being transferred between mirror and source in one way.

Enable multipage file allocation on mirror database
If you are running first time replication and the data size is very big, you can consider run db2empfa command on mirror database to enable multipage file allocation. This is one tuning option for DB2 when there is big data insert operation

Enable more active db2 page cleaning on mirror database
If you are running first time replication and the data size is very big, you can consider enable more active db2 page cleaning on mirror database by decreasing CHNGPGS_THRESH. Here is an example :

“db2 connect to mirror db”
“db2 update db cfg using CHNGPGS_THRESH 5”

Increase number of rows committed every time when performing replication
Where there is a lot of rows need to be replicated from source to mirror, you can increase the number of rows committed every time when performing replication. In dsy.properties table, modify the parameter "Jdbc.Replication.OperationsPerTransaction" from default value (50) to a bigger one e.g. 2000.

7. Other performance tuning resources

DB2 Everyplace Performance Tuning Guide 8.1 Luis Alves IBM Silicon Valley Lab

DB2 Everyplace: http://www3.ibm.com/software/data/db2/everyplace/

DB2 UDB V8 and WebSphere V5 Performance Tuning and Operations Guide, Redbook SG247068
http://www.redbooks.ibm.com

Websphere Application Server 6.0 InfoCenter
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp

DB2 Database for Linux, Unix and Windows 9.1 Info Center
http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

Netfinity Performance Tuning with Windows NT 4.0, Redbook SG24528700.
Even though it is a little dated, the sections on use of the Windows Performance Monitor are very useful.

Harvey W. Gunther, WebSphere Application Server Development Best Practices for Performance and Scalability, IBM WebSphere White Paper, Version 1.1.0 September 7, 2000.

8. About the document and the author

This document is an updated version of DB2 Everyplace v8.1 Performance Tuning guide written by Luis Alves. This version updates out of date information and adds more tuning advices from latest production information.

Jing Jing Xiao (xiaojj@cn.ibm.com)
Fei Peng Wang(wangfp@cn.ibm.com)
Liang Qi (qiliang@cn.ibm.com)
DB2 Everyplace Development Team
IBM China Development Lab