IBM® DB2 Everyplace® Version 8.1 Performance Tuning Guide

DB2 Everyplace version 8.1 performance tuning recommendations for Sync Server, Sync Client and Database.

Version 1.0

November, 2003

DB2 Everyplace Performance Team
IBM Silicon Valley Laboratory
555 Bailey Avenue
San Jose, California 95141

Table of Contents

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.

Trademarks IBM

2. About this document

The DB2 Everyplace Sync Server enables synchronization between relational data and back end databases. The Sync Server provides a single synchronization point. Clients synchronize to the central Synchronization Server, which then synchronizes with back end databases. This enables clients to share data through the Synchronization Server and get concurrent updates to the same data. The server can also synchronize files to the device.

The purpose of this paper is to provide guidelines for base parameter and application tuning for DB2 Everyplace. Note that the tuning is affected by many factors, including the workload scenario, client performance, 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 our testing. 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:

- Performance tuning hints for configuring resources to scale DB2 Everyplace and required back ends to achieve optimal performance.

- Guidelines on performance and isolation testing prior to deploying DB2 Everyplace into production.

This paper details the results of performance testing on DB2 Everyplace. In this paper when we state DB2 Everyplace, it implicitly means all DB2 Everyplace v 8.1.X builds. Some of the functionality described in this document may only work with the newer fix packs for DB2 Everyplace.

The term concurrency used in this document refers to the number of DB2 Everyplace synchronizations per hour.

2.1. Organization

This document presents performance tunning tips, techniques, and best practices to help you get the most from DB2 Everyplace servers on AIX, Linux, Solaris and Windows platforms, DB2 Everyplace clients on Palm, WinCE, Pocket PC, Linux, Symbian and more. The topics covered are listed in the following table.

Chapter	*Description*
1	**Special notices** - Special Notices and Trademarks.
2	**About this document** - Overview of the full document.
3	**Overview of DB2 Everyplace components** – Describes the DB2 Everyplace components, the basic architecture, and the new features for the DB2 Everyplace version 8 releases.
4	**Important considerations for the design of applications** - Brief description of the most important considerations when designing a DB2 Everyplace solution.
5	**Planning for performance and scalability** – How to plan for performance using DB2 Everyplace.
6	**Database engine performance guidelines** – Client database engine guidelines for performance.
7	**Sync Client performance guidelines** - Sync Client engine guidelines for performance.
8	**Sync Server performance guidelines** – Sync Server engine guidelines for performance.
9	**Performance tuning a Sync Server environment** - Tuning considerations for database servers, HTTP server, Application Server and DB2 Everyplace server.
10	**Other performance tuning resources** - References to other document with more information about DB2 and Websphere performance tuning.
11	**The author** – Thanks to everyone who contributed to this document.
Appendix A	**Duplicate a Sync Server system for testing** – This chapter describe 2 ways of backing up a complete DB2 Everyplace system. This allows the environment to be duplicated for testing or performance analysis.

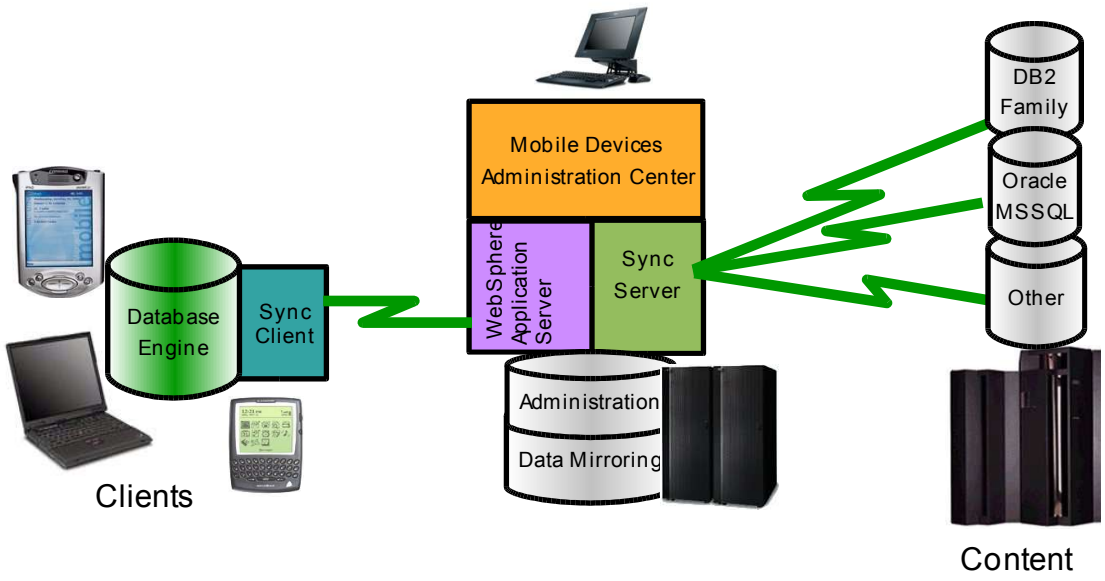
3. Overview of DB2 Everyplace components

The diagram below illustrates the main components of DB2 Everyplace system.

Component Name	*Description*
Client database engine	High performance small footprint data store for managing and using data on mobile and embedded devices.
Sync Client	High performance Sync 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

3.1. Architecture

The diagram below illustrates the main components of a DB2 Everyplace system.



Single Server architecture

At the core of the system is the Sync Server, which is the central source for describing, locating, organizing, and managing the data. It replies to client requests providing them with the necessary data through a secured access. End-users use the Sync Client and client database engine. The Sync Client

interacts directly with the Sync Server through HTTP requests. End-users applications can access the client database engine through standard API's like CLI, JDBC, ADO .net or ODBC, and can control the synchronization process through API's like iSync for C, iSync for Java and iSync for .Net.

3.2. New performance features in DB2 Everyplace

This section summarizes what's new in DB2 Everyplace for improved performance and scalability, for readers familiar with earlier releases. For more information on these features, see the product documentation.

Component	*New Features*
Client database engine	1. Improve the performance of queries made by applications like IBM Sync. 2. Index exploitation for LIKE predicate (For examples, C1 LIKE 'match%'). 3. Path length reduction: Tables with many columns, ORDER BY. Delete/Update statement with indexes.
Sync Client	1. Smart Sync, reduces the network handshaking on the MDSP protocol. 2. Resume mode is more efficient in version 8.1.X compared to version 7.2.X 3. HTTP 1.1 support. This improves synchronization time by reusing the same connection to route all the HTTP requests. 4. Time-out feature was redesigned to support networks with lower QOS. Clients are more reliable in networks with a low QOS. 5. Non-blocking sockets support on all platforms. 6. New Java Sync, C Sync and .Net Sync API with support to enable, disable and reset subscriptions sets on the client side. 7. Incremental time-out.
Sync Server	1. Increased concurrency support compared with previous release. 2. Faster replication for large volumes of data. 3. Improved synchronization performance.

4. Important considerations for the design of applications

Below are the descriptions of how DB2 Everyplace addresses each issue:

Data integrity

Resume mode, conflicts, rejected records.

Automatic conflict resolution

Predefine set of rules that allow DB2 Everyplace to always route the records to a point where they can be handled.

Portability

Target directory, synchronization to memory cards, and the ability to move the card to another compatible device, synchronize to laptops, PDAs, cellphones, etc.

Synchronization response time

Automatic resume mode, user can select what subscription sets to synchronize (enable, disable and reset), asynchronous message building, message store, delta configuration, horizontal and vertical filtering, complex filtering (join filtering), smart sync.

Concurrency

Mirror design, asynchronous message building, message store, using different mirrors for different sets of users.

Data flexibility

Synchronize several source databases into one database on the device, synchronize several source databases into several databases on the device in a single synchronization, horizontal and vertical filters, group/user parameters, several devices per user.

Support other database vendors

JDBC subscription supports DB2 family, Oracle, Informix, Cloudscape, Sybase, MS SQL Server, Domino, and possibly any JDBC driver with trigger support.

5. Planning for performance and scalability

The process for ensuring that a production DB2 Everyplace 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. This section also includes an overview of the configuration and application design choices a DB2 Everyplace administrator faces, focusing on the performance implications of those choices.

5.1. Performance methodology

This section provides an overview of our recommended performance best practices . The primary goal is to avoid surprises!

1. Read and understand the configuration choices and trade offs in this document.

2. Plan and document your overall system topology and configuration.

3. 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), both for typical users and peak hours.

    - Average data size synchronization and number of synchronizations per hour.

    - Use of features with significant performance impact (for example, join filters, 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).

4. Your IBM representative has a *capacity planning tool* for DB2 Everyplace tool to help you make an initial rough sizing of the hardware configurations that should be able to support your workload.

5. Read and understand the performance 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.

6. 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 multiuser test load based on your projected workload. During this tuning period iterate focusing on one area at a time, changing only a small number of

tuning parameters. Run the test workload to evaluate the effect of each set of changes before making additional tuning changes.

7. In production, perform routine performance maintenance, and monitor the performance of DB2 Everyplace Sync Server systems. The following tips can be used:

    - Perform periodic database RUNSTATS and REBIND, 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.

5.2. Collecting performance data

Static data

- Physical configuration of all the machines used on the tests (processors speed, location of databases (remote /local), number of clients on each cluster node).
- Size of the initial data.
- Server and client build level.
- WAS detailed 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 and DSYCTLDB (specifically, lock list, buffer pool size, log size, connections settings).

Runtime data

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

6. Database engine performance guidelines

Advanced indexing

The database engine supports bi-directional index scanning. Since version 8.1 a single index can support several different output ordering queries of a table. 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
```

Remote stored procedure adapter

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 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.

Design guidelines

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.

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.

7. Sync Client performance guidelines

7.1. Planning considerations for synchronization time optimization

- Data size
- Number of subscriptions
- Network bandwidth between the device and the server
- Latency of the network
- Upload bandwidth vs. download bandwidth
- Device type (for large synchronizations, CPU, Unicode)
- Network type (WiFi, Bluetooth, cable, infrared, etc.)
- Network quality (wireless vs. cable)
- Network complexity
- Network distance between mirror database server and Sync Server, for JDBC and DataPropagator subscriptions
- Network distance between source database server and Sync Server for upload subscriptions
- Client message size

7.2. Synchronization

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 setup different synchronization needs. For example: Daily data vs Weekly data.

Synchronize often when you make client changes. Sync Client execute 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 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.

Indexes

Minimize the number of indexes on the client to improve the synchronization time. Index creation on the device will affect the performance of the synchronization because INSERT, UPDATE and DELETE operations will become more expensive and REORG will also take more 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.

Timeout

The time-out value on the client sets the maximum time the client will wait for a server reply. The default is 30 seconds. For synchronizations of 5-15 Mb, it is more common to use a time-out value of 3 to 5 minutes.

Increase the time-out value of the client until the synchronization success rate is acceptable, otherwise the client will time-out 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 time-out value of the client until the synchronization success rate is acceptable.

Message size

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 request/send less data each time a failure occurs.

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.

Miscellaneous

If using SSL the synchronization time will increase. The device has to perform extra computation to encrypt the data and in some devices this feature is also very CPU, battery intensive.

Target directory allows you to have multiple databases on the same device and synchronize them with the same server or different servers. This feature can be used to allow synchronization to be performed at the same time as transactions are being performed on a different client database. It can also be used to perform two synchronizations on the same device at the same time. In the case of the network bandwidth and CPU is not totally in use, this technique will reduce the synchronization time.

By using mirror partition technique you can also improve the synchronization time. See the details about mirror partition on the "*replication*" and "*trade-off*" sections of the "*Sync Server Performance Guidelines*" chapter.

7.3. Network

The network connection between the server and the client plays the most important role in synchronization time. Here some examples of normal network speeds, latency and QOS:

The normal network bandwidth between client and server are:

- 56 K bits download / 19 K bits upload with low latency and high QOS, for modem
- 56 K bits download/upload with low latency and high QOS, for serial
- 56 K bits download/upload with high latency and low QOS, for infrared
- 1.5 M bits download/ 128 bits upload with low latency and high QOS, for DSL

- 100 M bits download/upload with low latency and high QOS, for Fast Ethernet
- 64 K bits download / 9.6 K bits upload with high latency and low QOS, for GPRS
- 11 M bits download/upload with high latency an low QOS, for WIFI 11b
- 56 K bits download/upload with high latency an low QOS, for WIFI 11b

The network bandwidth described above is never the actual network speed due to overhead from TCP/IP stack, encryption, HTTP, MDSP. Normally only 80 % of this network speed is usable for data transfer.

Mobile phone networks and wireless networks in general have high latency. This influences the synchronization time because the client is required to acknowledge each message received to the server. The number of messages that the client needs to exchange with the server play a more important role in these type of networks.

Most of the wireless networks in the market share bandwidth with other users, so if you are predicting that the usage of the network is going to be very high in the production area, don't use the total bandwidth to estimate synchronization time.

The size of message plays an important role in networks or devices with bad QOS of the network layer. If the quality of the network is poor and the client is not able to receive or send a full message, it will start from the beginning of that message. In this cases the choice of a good message size is important to improve synchronization time and success rate.

It is important to understand the different upload vs download speed of the client network connection to the server. On a normal DSL line the client might take 10 time to synchronize the same amount on the upload direction than it would have taken in the download. When estimating for synchronization time always take into account this important difference.

8. Sync Server performance guidelines

8.1. Planning considerations for replication time optimization

- Number of records per table.
- Data size from mirror to source.
- Data size from source to mirror.
- Source concurrent changes while replication is active (version 8.1.4 and above).
- Network bandwidth and latency between the source database and Sync Server.
- Network bandwidth and latency between the mirror database and Sync Server.
- Index on the filtered columns from the source.
- Replication cycle frequency.

8.2. Subscriptions and replication

Data partitioning

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 replications cycles, (for examples prices, stocks, orders), to allow the clients to have access to the new data during the day.

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 the Sync Client will exchanges a smaller number of messages with the server.

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.

Table modes

When creating a subscription, take advantage of the modes (insert, update and delete) which can be specified per table. This allows the Sync Client to automatically reject the records locally that do not follow this rule and will reduce data size sent to the server.
On version 8 and above when a table is read only mode, all the rejects records are rejected locally on the client side. If the customer application does not contain the logic to refresh a table in the case of a reject cause by UPDATE, or DELETE operations, the table will be out of sync until the next refresh or until that record is updated on the server side. The same behavior is true for upload subscriptions, on records that are delete or updated, but upload refresh will result on a empty table by design.

Filters

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

```
USERID = $PRICE AND DATA > :activemonth
```

  $PRICE and :activemonth will be replaced by Sync Server at synchronization time. $PRICE 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. [When editing a subscription Mobile Device Administration Console/Scripting might drop the indexes for the tables in that subscription depending on the changes that were performed, remember to recreate the indexes again]. In version 8.1.4 Sync Server will try to recreate the indexes that where created for the internal tables.

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 replications cycles.

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.

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.

Data size

Try to separate static data from volatile data in your table design for the source. DB2 Everyplace uses a 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.

Trade off

Partition users into groups that have some filter component in common. Then make a subscription for each group so that each group of users use 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 db 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 to be 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.

9. Performance tuning a Sync Server environment

9.1. 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 application databases, a database server for the control database and a database server for the mirror database. DB2 Everyplace can also be used with a Lightweight Directory Access Protocol (LDAP) user repository in WEA 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 base DB2 Everyplace configuration include

- a remote database server for WebSphere Application Server database (WAS) or WebSphere Portal database (WPS),
- a remote database server for data on source or mirror databases,
- a remote database server for data on DSYCTLDB database.

This section will once 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 DB2 product. For more details on tuning DB2, see the tuning references at:

http://www-3.ibm.com/software/data/education/bookstore/ref.html

Preloading databases for performance

DB2 Everyplace uses several databases to maintain information about its internal structures. When the server is initially installed, these databases are empty. The first time a client synchronizes, entries are added to tables within those databases. This is done automatically by the server, and is part of the normal server processing. However, this can have an impact on performance. The performance of relational databases will typically decline as tables grow unless the tables are periodically reorganized.

So that the data shown in this paper is indicative of the server's performance, and not limited by database performance, the databases are preloaded before the performance scenarios are run. This is done by automation scripts that load all data needed for the test cases in our test environment. Then the server is

stopped and all of the database tables are reorganized.

Any performance testing of DB2 Everyplace which uses more than 50 unique users will need to follow a similar procedure after the databases are populated, otherwise sub-optimal performance will be seen.

Database parameter tuning

For our testing we used IBM DB2 as our database server. Assuming that the database aliases are respectively WAS, DSYCTLDB, SOURCE and MIRROR, here are the specified parameters and values used in our environment:

```
db2 update db config for <database alias> using <db parameter> <value>
```

WAS database:

```
db2 update db config for was using applheapsz 256
db2 update db config for was using app_ctl_heap_sz 256
```

DSYCTLDB database:

```
db2 update db config for wps using applheapsz 1024
db2 update db config for wps using app_ctl_heap_sz 1024
db2 update db config for wps using maxappls 40
db2 update db config for wps using locklist 100
db2 update db config for wps using buffpage 1000
db2 update db config for wps using dbheap 1200
db2 update db config for wps using sortheap 256
```

SOURCE/MIRROR database:

```
db2 update db config for wms using applheapsz 1024
db2 update db config for wms using app_ctl_heap_sz 1024
db2 update db config for wms using maxappls 256
db2 update db config for wms using locklist 512
db2 update db config for wms using maxlocks 50
db2 update db config for wms using buffpage 1000
db2 update db config for wms using dbheap 1200
db2 update db config for wms using sortheap 256
```

Along with the above database parameter changes, there are several other updates needed for MIRROR 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 a 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.

- The SOURCE/MIRROR and DSYCTLDB, need to provide Sync Server with sufficient connections. If there are not enough connections available, clients can not 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, Jdbc.MaxConnections and DB2ClientSession.Connections, for the connection configuration.

- 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.

- Ensure that the database server has an adequate number of disks especially for the MIRROR/SOURCE, and Session databases. In our testing, we used 4 to 6 drives. Also, if possible, due to constant logging from the databases, dedicate the database logs to a separate disk (s) 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 don't have a good feeling what is the best type for your scenario, use SMS. Please refer to DB2 administration guide for further details on using SMS/DMS.

- Separate transaction logs and data table table spaces in different physical hard drives (see DB2 NEWLOGPATH parameter).

- Ensure that the usual necessary transaction log size need 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 to specify 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. The default value for LOGBUFSZ is 8 (4KB pages), an optimal value would be 128/256 (4 KB pages), depending on the amount of memory on the system.

- If the data size on the production system is high, tune the buffer pools for source and mirror database. This is very important for performance of SQL queries.

- 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. When your applications receive an error indicating that there is not enough storage in the application heap, increase the value of APPHEAPSZ. You should change the default value (128 4KB pages for DB2 EE or 64 4KB pages for DB2 EEE) to the optimal value for your production environment.

9.2. Web server tuning tips

IBM HTTP server parameter tuning tips

- 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 second, 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 testing, and therefore we assume each user will be opening new TCP connections every time a message takes more than 10 second to be applied on client side. However, in a live environment, it can be helpful to increase the Keep-Alive timeout. Keep in mind that a higher Keep-Alive 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 apache servers to determine if this needs to be increased.

- MaxSpareServers: 10, which is the default; but monitor the activity of the apache 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."

Another option to relieve contention on the Websphere Aplication Server system is to install the web server remotely from the portal application server when using Websphere Portal Server. In this environment, where both the web server and the database servers are remote, the portal server does not have to contend with these processes for system resources.

9.3. Application server tuning

Parameter tunings are based on WebSphere Application Server V4.0, Advanced Edition. This section will once 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://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html

Parameter tuning

Java Virtual Machine (JVM): Set the JVM heap size larger than 256MB. For the best and most consistent throughput, set the (ms) starting minimum and (mx) maximum 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.

From the WebSphere Portal application server in the WebSphere Administrative Console,

- Select the JVM Settings tab
- Select the Advanced JVM Settings button
- Modify the command line arguments to include the new parameter above.

Servlet engine pool size: Increase this parameter higher than the default. In our testing, we used 60 for both the minimum and maximum settings. Ideally, set this value and monitor the results using the WebSphere Resource Analyzer V4.0.2. Increase this value if all the servlet threads are busy most of the time.

Process Priority: On the "Advanced" tab of the WebSphere Portal application server, you can change the process priority from the default value of 20. Depending on what this Java process is competing with on the system, this value can make a big difference in performance. When running the web server on the same box as the WebSphere application server, setting this value less than 20 helps to give the application server more CPU cycles relative to many web server processes.

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 portal application server did not drive the load to an optimal near 100 %, thus we added a second clone of the portal application server.

One best practice in regards to cloning is to not blindly add clones onto 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.

9.4. DB2 Everyplace Tuning

Join filter (out of scope filters)

By turning JoinFilter filtering off you can gain some extra performance boost. 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, if the integrity feature is turned off.

Filters

Another performance improvement can be achieved in the case where you don't use filters at all, between the mirror and the client.

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 transfered between mirror and source in one way.

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

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.

By default, all message loggers and traces are disabled, except for error. Use the DSYGdflt.properties file located in “db2everyplace_root/server/properties/com/ibm/mobileservices/” where “db2everyplace_root” is the DB2 Everyplace root directory. Each server administrator must determine the amount of logging needed during production and the trade-offs of lower performance due to the overhead of logging. We keep the default setting of the product in our test enviroment, which is only to log errors and prune the log table after 7 days.

DB2 Everyplace Trace settings:

- Trace.Path=db2everyplace_root/Server/logs
- Trace.MaxFiles=10
- Trace.MaxSize=10000000
- Trace.Level=ERROR|CONSOLE

DB2 Everyplace DPropagator Trace

- Capture.Trace=0

Pruning of DSY.LOG table on DSYCTLDB database

- Log.KeepDays=7
- Log.PruneToSize=10000

Properties

There are several changes between version 8.1, 8.1.1 and 8.1.2 and 8.1.4, where these properties defaults values are being stored. Check DB2 Everyplace documentation to understand where this properties are stored for your version. For version 8.1.0, 8.1.1 and 8.1.2 some of the properties are stored in DSYGdflt.properties and in 8.1.4 and after in the table DSY.PROPERTIES of DSYCTLDB database. The important parameters in DSY.PROPERTIES table for performance are:

- **DB2ClientSession.Connections** and **Jdbc.MaxConnections** - The first 2 parameters are 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 number of concurrent synchronizations does not always directly relate with 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 (default values DB2ClientSession.Connections = 12, Jdbc.MaxConnections = 25).

- **ThreadPoolCount** – This parameter specifies the number of threads Sync Server can allocate to be able to perform internal tasks for synchronization and replication (default value ThreadPoolCount = 20). It is a good practice to increase this value in higher concurrency environments, 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** – 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 (see

DB2 Everyplace documentation for more information on this parameter).

- **Server.ResponseTimeout** - 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** - 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. Setting this period shorter will result in a more aggressive pruning of the control tables which will improve the performance of synchronizations and replications as well as reducing the storage consumption of Sync Server. A value of -1 (default) will disable this pruning function.

Below is an example sample how these values can be updated. Refer to DB2 Everyplace documentation for the defaults values. Use the " dsysetproperty" command to change a property value.

Example:

```
dsysetproperty DSYGdflt Jdbc.MaxConnections 50
```

9.5. Example of a DSYGdflt.properties configuration file

This example is from a DB2 Everyplace version 8.1.2 installation:

```
# The size of thread pool used by the Sync Server
ThreadPoolCount=20

# The directory/path where the trace files will be written. You must have
# write authority in order to messages to be logged. Also, all \
# characters must be doubled. For example:
#      Trace.Path=d:\temp\logs   <-- invalid
#      Trace.Path=d:\\temp\\logs <-- valid
Trace.Path=/home/db2inst1/db2everyplace81/Server/logs

# The maximum number of trace files in the trace cycle
Trace.MaxFiles=10

# The maximum trace file size (in bytes) of the trace files (minimum:
# 1000000)
Trace.MaxSize=10000000

# The level(s) for which messages to log. Using the list below,
# specify the keyword(s) of the levels that you want to be logged,
# separating each keyword # with a | (vertical bar) character.
# For example, to log all error messages (keyword: ERROR, you would type:
# Trace.Level=ERROR
#
#      Keyword    Description
#      -------    ------------------------------------------------------------
#      NONE       Turn trace off
#      *          Trace all messages
#      ALL        Trace all DSY SyncServer message types
#      CONSOLE    Additionally display trace messages to console
#      ENTRY      Type: Trace entry/exit messages
#      ERROR      Type: Trace error messages
#      GENERAL    Type: Trace general messages
#      MSGDUMP    Type: Trace incoming and outgoing messages from the server
#      REQUEST    Type: Trace HTTP requests
Trace.Level=ERROR|CONSOLE

# The flag which controls the tracing of the DPropR Capture process.
# A value of 1 will cause tracing to take place.
Capture.Trace=0

# Set the server name
Server.Name=default

# Set the server IP address
Server.IP=127.0.0.1
#
Server.Port=8080
```

Apendix A - Duplicate a Sync Server system for testing

Using Mobile Device Administration Console XML scripting

This approach is compatible between different releases. The prerequisites for this approach are 2 machines with the same level or newer of DB2 and DB2 Everyplace installed. This section describes how this is done with the VNURSE sample environment. In this scenario we have the VNURSE source database, the M_VN2 mirror database and the Sync Server control database DSYCTLDB. The steps outlined here are needed for the duplication.

Step 1

For VNURSE, M_VN2 and DSYCTLDB you have to backup the database configuration. If these databases are in a different instance, you have to backup the different instance settings as well. If you have multiple source and mirror databases you have to do it for all of them.

This section assumes all VNURSE, M_VN2 and DSYCTLDB are in the same instance. The commands to execute would be:

```
db2 GET DB CFG FOR DSYCTLDB > DSYCTLDB.output
db2 GET DB CFG FOR VNURSE > VNURSE.output
db2 GET DB CFG FOR M_VN2 > M_VN2.output
db2 GET DBM CFG > instance1_dbm.output
```

Step 2

The next step is to extract the source table schema's.

- Open DB2 Control Center, Right click on the VNURSE database. Select option "Generate DDL" and extract the full source table definitions, indexes, etc. into a DDL file.

- Open this file and remove all entries which are not related to table spaces, buffer pools, data tables, etc. of your data source, especially make sure that all entries for internal Sync Server System tables are removed. Remove also all the triggers on your source tables that were not created by you.

Step 3

Extract the source table data.

- Open DB2 Control Center, right click on the VNURSE database, click on TABLES, select one table and right click and select the EXPORT option and save table to a file. Repeat this for each source data table in your database.

Step 4

Now the Mobile Device Administration Console configuration for all subscriptions, Mobile Device Administration Console users, etc. needs to be extracted.

- Goto %DSYINSTDIR%\Server\bin ($DSYINSTDIR\Server\bin on Unix/Linux platforms).
- Execute: dsyadminxml -x MDAC_config.xml

  (./dsyadminxml.sh -x MDAC_config.xml on Unix/Linux platforms).

Step 5

Now another instance and the databases needs to be created.

- Create another instance.
- Create the databases VNURSE, M_VN2 and DSYCTLDB (using dsyctldb.bat)
- Update the instance configuration using update dbm cfg using PARAMETER VALUE for each parameter in the instance1_dbm.output file.
- Update the database configuration for VNURSE, M_VN2 and DSYCTLDB for each parameter in the above files with

```
update db cfg using PARAMETER VALUE
```

Step 6

Create the data tables in the source.

- Open DB2 Control Center, right click on the VNURSE database, click TABLES, select one table and right click and select the IMPORT option and import the data from the corresponding file to the table. Repeat this for each source data table in your database.

Step 7

Recreate the Mobile Device Administration Console configuration.

- Goto %DSYINSTDIR%\Server\bin ($DSYINSTDIR\Server\bin on Unix/Linux platforms).
- Execute: dsyadminxml.bat -d MDAC_config.xml.

  (./dsyadminxml.sh -x MDAC_config.xml on Unix/Linux platforms).

Using DB2 backups

In this section we describe how DB2 backups can be used to duplicate a Sync Server configuration. The prerequisites for this approach are 2 machines with the same level of DB2 and DB2 Everyplace installed. This approach is not compatible between different releases, since internal structures in the control database DSYCTLDB might have changed.

As above, we assume VNURSE, M_VN2 and DSYCTLDB for our example.

Step 1

In this step the databases need to be backed up. Issue the following commands on a CLP:

```
backup db VNURSE
backup db M_VN2
backup db DSYCTLDB
```

Step 2

In this section you restore the databases in another instance. Make sure the instance you restore has the same UDB version level and the same FP level as the one where the backups where taken. Then issue the following commands on a CLP (make sure the path logged on is where the directory of the backups is).

```
restore db VNURSE
restore db M_VN2
restore db DSYCTLDB
```

Step 3

In this section we outline what needs to be done before you can start:

- Change the user/password for Sync Server if needed to access the databases in the new instance. Goto %DSYINSTDIR%\Server\bin. Call dsyreset.bat -group group_name for all you Mobile Device Administration Console users to make sure all clients get a clean refresh and Sync Server is not getting inconsistent with aborted/failed syncs in the old environment.

10. Other performance tuning resources

IBM web site - http://www.ibm.com

DB2 Everyplace - http://www-3.ibm.com/software/data/db2/everyplace/

DB2 UDB / WebSphere Performance Tuning Guide, Redbook SG24-6417-00.

IBM WebSphere V4.0 Advanced Edition: Scalability and Availability, Redbook SG24-6192-00.

Websphere InfoCenter - http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html

WebSphere RedBooks - http://publib-b.boulder.ibm.com/redbooks.nsf/portals/WebSphereRedbooks

Netfinity Performance Tuning with Windows NT 4.0, Redbook SG24-5287-00. 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.

Sam Boreman, ST Component Document, Java Tech. Centre, IBM Hursley Park, June 18, 2002. This document is about the IBM JVM s Storage component, responsible for object allocation and garbage collection. It explains the algorithms used, has a tutorial/reference for reading verbosegc output, and also has a reference to the command line parameters and messages.

Tuning Garbage Collection with the 1.3.3 Java Virtual Machine, on Sun s Java website. It explains the Sun JVM s garbage collection algorithm and tuning considerations.

11. The author

Thanks to the following for contributing to this paper:

Thuan Bui, DB2 Everyplace Performance Team, Silicon Valley Lab

Martin Oberhofer, DB2 Everyplace Performance Team, Silicon Valley Lab

Tenni Gan, DB2 Everyplace Performance Team, Silicon Valley Lab

Bruno Oliveira, DB2 Everyplace Performance Team, Silicon Valley Lab

Rui Barbosa, DB2 Everyplace Performance Team, Silicon Valley Lab

Rajesh Kartha, DB2 Everyplace Performance Team, Silicon Valley Lab

Tim Alper, DB2 Everyplace Team, Silicon Valley Lab

Dilip Biswal, DB2 Everyplace Team, Silicon Valley Lab

The Author,
Luis Alves
DB2 Everyplace Performance Team
Silicon Valley Lab