Tuning for Replication and Event Publishing Performance
Note

Before using this information and the product that it supports, read the information in "Notices" on page 49.
### Contents

**Chapter 1. Introduction to tuning Q replication and event publishing**  
1

**Chapter 2. Tuning DB2 databases for replication and event publishing**  
3

- Considerations for database objects  
3
- Database parameter settings for Linux, UNIX, and Windows  
5
- Database manager parameter settings for Linux, UNIX, and Windows  
7
- Database environment variable settings for Linux, UNIX, and Windows  
8
- Database parameter settings for z/OS  
8

**Chapter 3. Tuning the replication and event publishing programs**  
11

- Multiple Q Capture programs  
11
- Multiple Q Apply programs  
12
- Q Capture and Q Apply tuning parameters  
12
- Q Capture parameters  
12
- Q Apply parameters  
17
- Search conditions and performance  
21
- Job or process priority  
21
- Improving performance for partitioned databases (Linux, UNIX, Windows)  
22
- Interpreting Q Capture and Q Apply monitoring statistics  
23

**Chapter 4. Tuning WebSphere MQ for replication performance**  
27

- Message limits for queues (MAXDEPTH)  
27
- Queue indexing (z/OS)  
27
- Queue buffer pools (z/OS)  
28
  - Defining buffer pools (z/OS)  
28
- WebSphere MQ logging  
29

- WebSphere MQ logging parameters for Linux, UNIX, and Windows  
29
- WebSphere MQ logging parameters for z/OS  
30
- WebSphere MQ logging configuration for z/OS  
30
- Channel parameters  
31
  - Channel batch size and interval  
31
  - z/OS channel parameters  
31
- Maximum uncommitted messages (MAXUMSGS)  
32

**Chapter 5. Performance tuning scenario: sleep_interval**  
33

- Checking performance statistics  
33
- Diagnosing the problem  
35
- Tuning the Q Capture program  
36
- Checking the results of tuning  
37

**Chapter 6. Performance tuning scenario: memory_limit**  
39

**Chapter 7. Viewing statistics in the monitor tables**  
41

**Product documentation**  
43

**Contacting IBM**  
43

**How to read syntax diagrams**  
45

**Product accessibility**  
47

**Notices**  
49

**Trademarks**  
51

**Index**  
53
Chapter 1. Introduction to tuning Q replication and event publishing

Q replication and event publishing are designed to replicate and publish a high volume of transactions and database events with low latency. The default settings provide high performance on a wide range of systems and configurations, but you might need to tune these settings for the best performance.

The following figure shows the major components of Q replication. Each of these components can be tuned to improve performance.

Figure 1. Major components of Q replication: DB2 database, Q Capture, WebSphere MQ, and Q Apply.
Tuning for Replication and Event Publishing Performance
Chapter 2. Tuning DB2 databases for replication and event publishing

Optimizing the performance of DB2® databases includes adjusting settings for database objects, parameters, and environment variables.

When you tune DB2 databases for replication and event publishing, your focus will be different at the source and target databases or subsystems:

**Source**
The Q Capture program has relatively little interaction with DB2 database or subsystem. The Q Capture program reads the logs, reads the Q Capture control tables at startup, and copies LOB data from the source table. Your DB2 database tuning should focus on your application workload rather than on the Q Capture program.

**Target**
The Q Apply program generates a large amount of database activity. Depending on your replication workload, the Q Apply program might perform many updates to target tables and to its control tables when it records each message that is applied plus conflicts or errors in both unidirectional and multidirectional replication. Treat Q Apply as a very large DB2 application and tune for its impact on memory, logging, and concurrency.

For bidirectional or peer-to-peer replication, you must tune for both a Q Capture program and a Q Apply program at each participating server.

**Considerations for database objects**
The disk drive and memory settings can significantly affect performance. You can use the RUNSTATS utility to provide the DB2 optimizer with statistics for determining the best data access strategy.

The following list describes the database objects that most affect replication performance:

**Buffer pools**
Define as much buffer pool space as you can for optimal performance. The primary consideration is your application workload. The Q Apply program performs many writes to the database, so ensure that the buffer pools that are dedicated to the target tables are large enough to avoid excessive I/O.

If possible, create a dedicated buffer pool for the table spaces that will contain your replication control tables, particularly the following tables that might have extensive data changes:

- IBMQREP_DONEMSG table on the Q Apply server.
- IBMQREP_SIGNAL table on the Q Capture server for bidirectional replication (this table receives numerous signal inserts from the Q Apply program).
- IBMQREP_DELTOMB table on the Q Apply server if you expect a large number of conflicting deletes in peer-to-peer replication.
- IBMQREP_EXCEPTIONS table on the Q Apply server if you expect a large number of conflicts.
• IBMQREP_SPILLEDROW table on the Q Apply server if you expect a large number of changes to be replicated while the Q Apply program is loading target tables.

**Disk system**
Whenever possible, use multiple disk drives to allow for parallel I/O. In addition, use disk controllers with a fast-write nonvolatile cache. Ideally, store DB2 data and indexes on different disks than the DB2 active logs.

**Logs**
For improved performance, the file system that holds your logs should span multiple disks and use disk striping. In addition, store the DB2 logs and the WebSphere® MQ logs on different sets of disk drives.

*z/OS*: Use VSAM striped log data sets for better read/write throughput.

**Spill files**
Allocate enough memory for the Q Capture program to avoid spilling, which can affect performance. When the Q Capture program reaches its memory limit, it spills to disk on distributed platforms and to Virtual I/O (VIO) by default on z/OS. If you cannot avoid spilling, store the spill file on a set of disk drives that is separate from the data and logs if possible. The Q Apply program does not use a spill file (it spills messages to a queue when the target table is being loaded).

*z/OS*: If you want to use a specific primary and secondary cylinder for VIO or use SYSDA instead of VIO, you can direct the spill file to a CAPSPILL DD card. For details, see [Directing spill files to storage](#). On z/OS, the largest spill files that you should use is two gigabytes (2 GB).

**Statistics**
Use the RUNSTATS utility on Q Capture and Q Apply control tables that receive extensive changes. Run the utility when these tables are at their production size to give the DB2 optimizer the necessary statistics to determine the best data access strategy for improved performance.

The following control tables are good candidates for using RUNSTATS:
• IBMQREP_SIGNAL (bidirectional replication)
• IBMQREP_DELTOMB (peer-to-peer replication)
• IBMQREP_EXCEPTIONS
• IBMQREP_SPILLEDROW
• IBMQREP_SRC_COLS (This table could grow large if many tables are being replicated. You can improve startup performance by using RUNSTATS after populating this table.)

The IBMQREP.DoneMsg table is created with the APPEND ON and VOLATILE keywords for improved performance, and does not require RUNSTATS:

**APPEND ON**
Allows rows to be inserted at the end of the table (appended). This makes inserts in the IBMQREP.DoneMsg table much faster. However, no free space information is kept, so you should occasionally run the REORG utility on this table.

**VOLATILE**
Tells the DB2 optimizer to always do an index scan when the Q Apply program accesses the IBMQREP.DoneMsg table. This parameter is only for Linux®, UNIX®, and Windows®.

You can also use RUNSTATS on the source and target tables:
Source tables
For source tables that contain large object (LOB) data that will be replicated, current statistics help the Q Capture program fetch data from the LOB columns.

Target tables
- Linux, UNIX, Windows: Use RUNSTATS on target tables after they are loaded with data.
- z/OS: If the Q Apply program loads the target tables by using the automatic load option, you do not need to use RUNSTATS on the target tables initially because statistics will be gathered automatically after the load.

Table spaces
Control tables that have extensive data changes (see the list under “Buffer pools”) should use DMS (database-managed space) raw-device table spaces for better performance. Consider placing these control tables in the same DMS table space as very active user tables, with a dedicated buffer pool. Whenever possible, define table-space containers that span multiple disk drives to allow for parallel I/O. In a high-volume environment, the extent size should be larger than the default of 32 pages (at the target database, an extent size of 64 pages is recommended).

z/OS: Control tables are created in segmented table spaces for better administration and space allocation. For a multi-extent data set, use a secondary extent size that is greater than 10 MB or 15 cylinders.

Database parameter settings for Linux, UNIX, and Windows
Your DB2 database parameter settings are likely to differ at the source and target. Monitor the system during peak load times and tune the source and target databases with different parameter settings as required.

You can use the autoconfigure command to obtain the initial values for database and database manager configuration parameters, then monitor the database and tune as necessary. The recommendations in the list below can help you tune to reflect the needs of the replication programs.

dbheap
Set a large size for the database heap to accommodate an increased log buffer size for the Q Capture program. Increase this value if you have a large number of tables, indexes, table spaces, and buffer pools. The default value of 1200 pages (UNIX), 600 pages (Windows 64 bit), and 300 pages (Windows 32 bit) might not be adequate for optimal performance in a high-volume environment.

logbufsz
To allow the Q Capture program to read log records from memory (log buffers) rather than from disk as often as possible, allocate a large amount of the database heap as a buffer for log records. At the source database, a value between 64 and 128 pages should be adequate for a large replication scenario. You might need to increase the log buffer size at the Q Apply server to accommodate heavy write activity at target tables.

Do not set this value to more than 35 percent of the database heap size. Unless your environment requires a higher value for this parameter, do not set the log buffer size to more than 512 pages because larger log buffers can degrade performance.
**logfilsiz**
To reduce the frequency of log archiving, allocate a large amount of space for the log file. The Q Apply program might have a significant impact on logging at the target database. However, large log files can increase database recovery time.

**logprimary**
Depending on your workload, you might need to increase the number of preallocated primary log files at the target because of increased write activity. Source logging is increased by a small amount depending on the number of updates against tables that are now defined with the data capture changes attribute.

Set this value in conjunction with the `logfilsiz` parameter and `logsecond` parameter.

**logsecond**
If replication activity is sporadic, allocate enough secondary log files at the target database to ensure sufficient time for log archiving to complete when the primary log files become full.

If your environment requires more (or fewer) preallocated secondary log files, you can set this value appropriately, along with `logfilsiz` and `logprimary`.

If you expect many long-running transactions that could exceed the number of active logs, set `logsecond` to -1 to enable infinite active logging.

**locklist**
The Q Capture program does very little locking of tables or rows, and this activity is confined to the Q Capture control tables. The locks that are held by the Q Apply program at any one time will depend on the nature of the workload that is being replicated. Consider increasing the `locklist` value if you are replicating many long-running source transactions.

**maxlocks**
To reduce the likelihood of lock escalation (the process of replacing row locks with table locks), increase the percentage of the lock list that must be filled before the database performs escalation. In most cases, set `maxlocks` to a value between 50 percent and 70 percent. Because lock escalation also occurs if the lock list runs out of space, ensure that the `locklist` value is high enough for both replication and your application workload.

**maxappls**
This parameter determines how many applications can be connected to the database at the same time. The default setting for this parameter is Automatic. With this setting, the database or subsystem dynamically allocates the resources to support new applications. If you do not want to set the value to automatic, use the following formulas based on the number of threads for each Q Capture and Q Apply program. The DB2 database or subsystem treats each thread as a separate application:

**Q Capture server**
\[
\text{maxappls} \geq \text{number of concurrent applications other than Q Capture} + \\
(5 \times \text{number of Q Capture programs on the database})
\]

**Q Apply server**
\[
\text{maxappls} \geq \text{number of concurrent applications other than Q Apply} + \\
(3 \times \text{number of Q apply programs on the database}) + \\
\text{number of receive queues} + \text{number of Q Apply agents} \\
\text{for all receive queues}
\]
If you use the Q Apply program to load multiple target tables in parallel, increase the result of this formula by 1 for each table that is being loaded in parallel. This additional number will account for the spill agents that apply messages from the spill queue.

**applheapsz**
This parameter controls the amount of memory from the DB2 application heap that each agent for an application can use to process SQL statements. The default of 256 4 KB pages might be too small for the Q Capture program or Q Apply program, depending on your workload. Increase the value if needed.

**stmtheap**
At the target database, consider increasing the default number of 4 KB pages that are used by the SQL compiler if the tables being replicated have a large number of columns or have long column names. In this case, the Q Apply program might build large SQL statements when applying the workload to the target table, and a large statement heap can help avoid runtime failures.

**num_iocleaners**
On the target database, the Q Apply program might dramatically increase the need for asynchronous page cleaners because of the volume of writing to target tables and to the IBMQREP_DONEMSG table. Because these transactions primarily change data in the buffer pools, increase the number of page cleaners to avoid synchronous I/O.

**num_ioservers**
Set this value to the number of CPUs for the database server to allow for prefetch I/O and asynchronous I/O, but do not set this value lower than the default of 3. If you use disk striping, set this value to match `num_iocleaners`.

**pckcachesz**
If you accept the default setting for the package cache size (-1), DB2 multiplies the value of the `maxappl` parameter by eight to determine the number of pages to allocate. If you set this parameter value manually, your main focus for tuning should be the target database, where many SQL statements are used repeatedly. Increase the package cache size on this database to help eliminate the need to compile dynamic SQL and access the system catalogs when reloading static packages.

---

**Database manager parameter settings for Linux, UNIX, and Windows**

The `intra_parallel` and `maxagents` parameters are instance-level configuration parameters that affect replication.

Tune the `intra_parallel` parameter and `maxagents` parameter with your overall application workload as the primary concern.

**intra_parallel**
Q replication and event publishing use relatively simple SQL statements, and intra-partition parallelism is not required. If your applications require intra-partition parallelism, you can set this parameter to YES.

**maxagents**
Set the value for maximum number of database manager agents based on the number of applications other than replication that will use the instance, plus the number of connections that the Q Capture or Q Apply programs
will require. To determine this number, use the calculation that is described for the \texttt{maxappl} parameter. (If you are replicating multiple databases within the instance, you must calculate the value for each).

\section*{Database environment variable settings for Linux, UNIX, and Windows}

The \texttt{db2_mmap_read}, \texttt{db2_parallel_io}, \texttt{db2codepage}, and \texttt{EXTSHM} environment variables affect replication performance.

The list below describes DB2 environment variable recommendations for replication.

\begin{itemize}
  \item \texttt{db2_mmap_read}
    \begin{itemize}
      \item AIX\textsuperscript{®}: Set this value to \texttt{OFF} to allow DB2 to read from JFS file system cache into memory (outside the buffer pool).
    \end{itemize}
  \item \texttt{db2_parallel_io}
    \begin{itemize}
      \item Set the parameter to * to enable parallel I/O for all (*) table spaces.
    \end{itemize}
  \item \texttt{db2codepage}
    \begin{itemize}
      \item You can expect a slight performance cost for code page conversion if the source and target databases use different code pages. To avoid this cost, use the same code page when it makes sense to do so.
    \end{itemize}
  \item \texttt{EXTSHM}
    \begin{itemize}
      \item AIX: Set this variable to 'ON' to allow for more than four apply agents. By default, replication uses sixteen agents. Setting this variable at the target database increases the number of shared memory segments to which a single thread can be attached.
    \end{itemize}
\end{itemize}

\section*{Database parameter settings for z/OS}

Your DB2 database parameter (DSNZPARM) settings might differ at the source and target. Monitor the system during peak load times and tune the source and target databases with different parameter settings as required.

The recommendations below can help you tune to reflect the needs of the replication programs.

\begin{itemize}
  \item \texttt{blksize}
    \begin{itemize}
      \item Specify the maximum block size of the archive log data set for the device.
    \end{itemize}
  \item \texttt{cachedyn}
    \begin{itemize}
      \item Many of the same SQL statements are used multiple times during replication, so allow DB2 to cache prepared, dynamic SQL statements for later use. These prepared statements are cached in the environmental descriptor manager (EDM) pool. If you specify \texttt{CACHEDYN=YES}, you should consider this usage when you calculate your EDM pool size. Also, to facilitate dynamic SQL you can bind the replication packages with the bind option \texttt{KEEPDYNAMIC(YES)}.
    \end{itemize}
  \item \texttt{dealclt}
    \begin{itemize}
      \item Specifying NOLIMIT for the tape unit deallocation period allows for the most optimization opportunities. In a data-sharing environment, if the Q Capture program is stopped for a period of time and all logs are archived to tape, this parameter allows the Q Capture program to merge the logs without needing to reallocate an archive read tape unit. These units might remain unused for a length of time that is greater than a specific
\end{itemize}
DEALLCT value. When all tape reading is complete, you can update this option with the SET ARCHIVE command.

**maxrtu**

Specify the number of data-sharing members plus one for the maximum number of dedicated tape units that can be allocated to read archive log tape volumes concurrently. In a data-sharing environment, this parameter and the `deallct` parameter allow the Q Capture program to optimize archive log reading from tape devices.

**outbuff**

To allow the Q Capture program to read log records from memory (log buffers) rather than from disk as often as possible, set aside the largest output log buffer that is possible depending on the memory space that is needed for other applications.

**unit**

Specify the device type or unit name for storing archive log data sets.

**Recommendation:** Recall archive logs from tape to DASD if the Q Capture program needs to read old logs. The Q Capture program is likely to read archive logs only if it was shut down for a significant period of time while DB2 instance continues to run.
Chapter 3. Tuning the replication and event publishing programs

Q replication and event publishing are designed to take advantage of parallel processing of transactions. You can achieve significant performance gains by creating multiple Q Capture programs to share your workload, or by configuring the number of threads that are used by the Q Apply program to apply transactions to target tables.

Within each of the replication programs, you can also improve performance by tuning memory allocation and other parameters.

Because Q replication and event publishing are designed for minimum latency and high throughput, most of the information about tuning focuses on achieving these two performance goals. However, the cost of faster performance in terms of CPU resources is also discussed where appropriate.

Multiple Q Capture programs

Using multiple instances of the Q Capture program can greatly increase overall throughput. But to correctly divide the workload, you must examine how transactions are built at the source database or subsystem.

Having more than one Q Capture program on a server provides multiple log readers, which allows more transactions to be written to queues and increases throughput. Depending on your operating system, you might see a 50-percent to 75-percent increase in throughput by using two Q Capture programs. With three to five Q Capture programs, you can expect a slightly higher increase. Throughput gains decline when you use more than five Q Capture programs for the same DB2 log.

If you decide to use multiple Q Capture programs, you must organize your data flow so that all tables involved in the same transactions are replicated by the same Q Capture program. The reasons for this prerequisite are:

• To maintain data integrity, all tables that are involved in the same transactions must be replicated by using the same send queue.
• Two or more Q Capture programs cannot write messages to the same send queue.

The trade-off for the higher throughput that you gain with multiple Q Capture programs is additional CPU overhead that is associated with multiple log readers. Using more than one Q Capture program might provide the best performance gains if the source system has multiple CPUs.

A single Q Capture program can write messages to more than one queue. This configuration does not increase the parallelism at the source, but might allow for increased parallelism at the target because multiple queues are processed in parallel by multiple Q Apply browsers (and the associated agent threads).

If multiple send queues go to the same remote queue manager, you can use multiple transmission queues and channels to improve performance. The Q Capture program puts messages on multiple queues in a serial fashion, so if the
number of queues grows too large you might achieve better performance by creating an additional Q Capture program.

If you use multiple Q Capture programs, you can also set different Q Capture program parameters based on the tables that are being replicated. For example, you might set a larger memory limit or shorter commit interval for a Q Capture program that is replicating tables with a higher volume workload.

### Multiple Q Apply programs

Having multiple Q Apply programs, each with its own set of control tables, might not improve throughput, but this configuration could lower latency.

On the target server, you can choose one of two methods for dividing the flow of data that is being applied:

- Maintain a single Q Apply program and let it create a browser thread for each receive queue.
- Create multiple Q Apply programs.

Even a single Q Apply program with a single browser thread can deliver excellent performance throughput. Each browser thread can create many agent threads, which increases the potential for parallel processing.

### Q Capture and Q Apply tuning parameters

You can tune a subset of the Q Capture and Q Apply program parameters to achieve performance goals such as lower latency, higher throughput, or reduced CPU usage.

The default values for these parameters should provide a high level of performance in most environments. However, in some high-volume replication or publishing environments, or in situations where large or long-running transactions are typical, you might want to improve performance with one of the following actions:

- Allocate more memory to the Q Capture or Q Apply programs
- Commit WebSphere MQ messages more frequently
- Dedicate more program threads to replication

### Q Capture parameters

The Q Capture program parameters that you can tune for performance are the `commit_interval`, `memory_limit`, and `sleep_interval` parameters. You can also set the `max_message_size`, `monitor_interval`, and `pruning_interval` parameters.

The following figure illustrates how these Q Capture parameters help control the program’s interaction with DB2 and WebSphere MQ.
The commit_interval parameter specifies how often, in milliseconds, a Q Capture program commits transactions to WebSphere MQ. At each interval, the Q Capture program issues an MQCMIT call. This call signals the queue manager to transmit messages that were placed on send queues to the target queues for the Q Apply program or user applications.

For most workloads, the default commit interval of 500 milliseconds (a half second) is satisfactory. Lowering this value is not likely to improve throughput, but will potentially improve latency. Committing messages more frequently requires more CPU usage and I/O overhead.
If you cannot increase the amount of memory that is allocated for the Q Capture program, you can reduce the commit interval. Reducing the commit interval flushes messages from memory more often and prevents the Q Capture program from spilling transactions to disk.

**Note:** The commit interval will not affect a long-running batch transaction with no interim commits. Transactions of this nature must be completely captured and written to queues before any of the transactional data can be committed.

You can raise the commit interval if you want to conserve CPU resources in a low-volume environment. However, raising the commit interval will increase latency.

**memory_limit parameter**

The `memory_limit` parameter specifies the amount of memory that a Q Capture program can use to build transactions in memory.

On z/OS, the default value of 0 causes the Q Capture program to calculate a memory allocation that is based on the Q Capture region size in the JCL or started task. The default value of 500 MB on Linux, UNIX, and Windows is sufficient for most workloads.

You might need to increase the memory limit if the Q Capture program is spilling transactions to disk or virtual I/O. Ideally, the Q Capture program should never need to spill transactions. If the `TRANS_SPILLED` column in the IBMQREP_CAPMON table has a value greater than 0 (you can also view this value in the Q Capture Throughput window in the Replication Center), try increasing the memory limit.

For large transactions or batch processing, you might need to increase the memory limit or lower the commit interval to avoid spilling. You can dynamically increase these parameters while the Q Capture program is running. You can then return these values to their previous settings to avoid excessive memory allocation or CPU consumption after the batch is processed.

**Determining minimum memory requirement**

In some cases you might want to reduce the amount of memory that a Q Capture program uses but still avoid spilling to disk (for example, in environments with low volume or small transactions). To determine the minimum amount of memory that you need, you can use the following formula. The formula is based on the concept that the memory required at any given time is equal to the number of transactions that are being processed multiplied by the size of each transaction. The formula uses statistics from the IBMQREP_CAPMON table or Q Capture Throughput window in the Replication Center:

\[
\text{memory_limit} \geq 1.25 \times \text{MAX(MAX_TRANS_SIZE)} \times \left(\frac{\text{MAX(TRANS_PROCESSED)}}{\text{monitor_interval}} \times \frac{\text{commit_interval}}{1000}\right)
\]

**Note:** The units for the `monitor_interval` and `commit_interval` parameters in the formula are seconds.

Where:

**MAX(MAX_TRANS_SIZE)**

The largest value that is recorded in the MAX_TRANS_SIZE column (the largest transaction that the Q Capture program processed).
MAX(TRANS_PROCESSED)
The largest value that is recorded in the TRANS_PROCESSED column (the most transactions that the Q Capture program processed).

Dividing by the monitor interval shows transactions processed per second. Because the Q Capture program keeps transactions in memory until the commit interval is reached, you need to multiply by this interval, which is divided by 1000 to give the value in seconds. You can find the runtime values of the commit_interval and monitor_interval parameters by using the qryparms parameter with the MODIFY or asnqccmd commands, or the Change Parameters – Running Q Capture Program window in the Replication Center.

You can modify the safety factor of 1.25 if your past workload is not a good predictor of your future workload.

sleep_interval parameter
The sleep_interval parameter specifies the number of milliseconds that a Q Capture program waits after reaching the end of the active DB2 log.

In a DB2 for z/OS data sharing environment, the Q Capture program sleeps for the interval that is specified by the sleep_interval parameter when the log buffer is less than half full. This allows you to reduce the cost of the merge log read in a configuration where fewer replicated data changes occur.

You might need to tune the default value of 5000 milliseconds (5 seconds) depending on your workload:

High volume
When the source database or subsystem has heavy traffic, the Q Capture program will be reading the log continually, so there is no need to shorten the sleep interval.

Medium volume
If the Q Capture program is keeping up well with the volume of transactions that are written to the log, you can experiment with lowering the sleep interval to reduce idle time and latency. You can determine the minimum effective sleep interval for your system by checking latency statistics to find the point where further reductions in the sleep interval do not reduce latency. Values that are too low (for example, below 400 milliseconds) might result in higher-than-necessary CPU usage. The capture program wakes up and reads the log, and if there are no new transactions the program goes back to sleep. The wake-up and sleep cycle increase the CPU usage.

Low volume
Consider raising the sleep interval to save CPU usage in an environment where the source database has low traffic, or where targets do not need frequent updates. If you see high CPU usage when there are no transactions to process, check the value of sleep_interval.

max_message_size parameter
The max_message_size parameter determines the size of the memory buffer that a Q Capture program uses for each send queue. You set this parameter for the publishing queue map or replication queue map that contains the send queue.

After a transaction is built in memory, the Q Capture program uses the max_message_size parameter to determine the size of the message to publish. If
the transaction size in bytes is larger than the maximum message size, then the
transaction will be split into smaller messages.

Use the following guidelines for setting \texttt{max\_message\_size}:

- One typical large transaction should fit into one message, so set the value of
  \texttt{max\_message\_size} to be slightly higher than the maximum size of a typical
  transaction.
- For very large transactions that exceed the value of \texttt{max\_message\_size} that you
  derived above, ensure that you set \texttt{max\_message\_size} such that at least one row
  of the transaction fits into one message.
- The value of \texttt{max\_message\_size} must be less than or equal to the WebSphere
  MQ parameter \texttt{MAXMSGL}, which sets the maximum message size for a queue.

If you are replicating or publishing large object (LOB) data, the value of
\texttt{max\_message\_size} determines how often the Q Capture program accesses the
source table to fetch the LOB data in multiple chunks (one chunk per message). A
low maximum message size can impede Q Capture performance in replicating or
publishing LOB data.

The \texttt{max\_message\_size} parameter also determines the largest message size that the
Q Apply program will see. If LOB data is sent in a separate LOB messages
(\texttt{LOB\_SEND\_OPTION = 'S')} and the LOB size is larger than the value for the
\texttt{max\_message\_size} parameter, the LOB data is sent as multiple messages. These
multiple messages can slow Q Apply performance by requiring more frequent
updates to the target table to replicate an entire LOB value.

A value for \texttt{max\_message\_size} that is too high can create spikes in overall memory
usage by the Q Capture and Q Apply programs when processing LOB values. Such
spikes are even more likely if code page conversion is required between the source
and target.

\textbf{monitor\_interval parameter}

The \texttt{monitor\_interval} parameter, which sets the frequency of Q Capture inserts
into the IBMQREP\_CAPMON and IBMQREP\_CAPQMON tables, does not affect
performance. However, if you want more current statistics to use for tuning,
consider lowering the value for this parameter to make the monitor data more
accurate.

By default, rows are inserted into these tables every 300,000 milliseconds (5
minutes). Typically, a Q Capture program commits WebSphere MQ transactions at
a much shorter interval (the default commit interval is a half second). If you use
defaults for the monitor interval and commit interval, each insert into the monitor
tables contains statistics for 600 commits.

If you want to monitor the Q Capture program at a more granular level, use a
monitor interval that is closer to the commit interval. Setting the monitor interval
lower than 50 milliseconds is not recommended because many rows will be
inserted into the monitor tables and increase CPU usage.

\textbf{prune\_interval parameter}

The \texttt{prune\_interval} parameter determines how often the Q Capture program
prunes the IBMQREP\_CAPMON, IBMQREP\_CAPQMON, IBMQREP\_SIGNAL, and
IBMQREP\_CAPTRACE tables.
If you shorten the prune interval from its default of 300 seconds (5 minutes), you increase the CPU overhead for processing pruning queries. However, shortening this interval is unlikely to affect overall throughput or latency.

For bidirectional replication, the IBMQREP_SIGNAL table can grow large. The Q Apply program inserts a signal into the IBMQREP_SIGNAL table for every transaction that it receives and applies to make sure that the Q Capture program does not recapture the transaction. You can shorten the pruning interval to manage the size of this table.

**Q Apply parameters**

The two types of Q Apply operating parameters are replication queue map parameters and Q Apply instance-level parameters.

**Replication queue map parameters**

These parameters affect the way a Q Apply program processes messages on a single receive queue, and are most important for performance tuning. These parameters are:

- **num_apply_agents** (number of agent threads)
- **maxagents_correlid** (number of agent threads for batch jobs with the same job name)
- **memory_limit** (memory buffer for the receive queue)

You set these parameters when you create or modify a replication queue map.

The following figure illustrates how these Q Apply parameters help control the program's interaction with WebSphere MQ and DB2.

![Diagram showing Q Apply parameters](image)

Figure 3. Important tuning parameters for the Q Apply program: 1. **num_apply_agents** and 2. **memory_limit**. This figure shows how some of the Q Apply program's activities are controlled by operating parameters that you can tune: A browser thread gets messages from the receive queue and rebuilds transactions from them. The **memory_limit** parameter controls the size of the memory buffer that is used to hold these reconstructed transactions. Next, the browser schedules which transactions can be applied in parallel by a number of agents that you set with the **num_apply_agents** parameter. Then the agents take available transactions, rebuild SQL statements, and execute them.

**Q Apply instance-level parameters**

These parameters affect overall program operation, but have little impact on performance. These parameters are:

- **monitor_interval** (frequency of inserts into the monitor table)
- **prune_interval** (frequency of deletes from the monitor and trace tables)
You set these parameters at the Q Apply instance level.

**num_apply_agents parameter**

The `num_apply_agents` parameter determines how many agent threads will be used by the Q Apply program to take reconstructed transactions from the browser and apply them to target tables. You want to optimize this parameter to increase throughput and also to keep CPU use low.

For the `num_apply_agents` parameter, a value higher than 1 allows the Q Apply program to process transactions in parallel. You set this parameter for the replication queue map that specifies the receive queue.

Your setting for `num_apply_agents` might depend on your operating system:

- **Linux UNIX Windows**
  - The default value of 16 agents should provide good performance for most medium- to high-volume replication workloads.

- **z/OS**
  - The optimal number of agents might be slightly lower than the default of 16. If you are seeing lock contention at target tables that increases latency, consider one or more of the following actions:
    - Reduce the number of agents.
    - Use row-level locking instead of the typical default of page-level locking at the target tables.
    - Reduce the deadlock timeout interval.
    - If contention results from batch jobs on a z/OS Q Capture server, set the `maxagents_correlid` parameter to limit parallelism for transactions from the same job.

Here is an example of how the number of agents scaled for some performance workloads tested on a p690 Regatta system that used AIX 5.2, with separate source and target LPARs with eight CPUs and 8 GB of RAM, and a typical transaction size of 10 rows (200-byte rows):

- With a workload of 600 transactions per second or less, two agents were sufficient.
- Four agents provided good performance with a workload between 600 and 900 transactions per second.
- For more than 1,200 transactions per second, eight or more agents were needed.

You can experiment with increasing the number of agents to see if throughput increases. You can use the Latency window in the Replication Center to check how quickly transactions were applied for intervals before and after you adjust the value of `num_apply_agents`.

The tuning goal for this parameter is to use the smallest number of agents that you need to handle the workload. If you use too many agents, each agent will sleep when it has no transactions to process. The sleep and wake-up cycles can increase CPU usage.

If the percentage of sleep time for each agent is greater than 40 percent, then you probably have too many agents for your workload. You can use the following formula to calculate the percentage of sleep time for each agent:

\[
\text{Percent of time agents sleep} = \left[ \frac{\text{APPLY_SLEEP_TIME}}{\text{num_apply_agents}} \right] \times \frac{\text{monitor_interval}}{100}
\]
Where APPLY_SLEEP_TIME is the value of the column in the IBMQREP_APPLYMON table, num_apply_agents is the value of this replication queue map parameter, and monitor_interval is the value of this Q Apply parameter.

If the percent sleep time for agents is zero, you can check the memory usage for the receive queue. If the value of the MEM_FULL_TIME column in the IBMQREP_APPLYMON table is greater than 0, you might need to increase the number of agents to clean transactions from memory, or you might need to set a higher memory limit. If the percent sleep time is too high, you might need to reduce the value of the num_apply_agents parameter.

If the values for both MEM_FULL_TIME and APPLY_SLEEP_TIME are greater than 0, then you either have a long transaction or batch job, or the target database needs to be tuned.

**maxagents_correlid parameter**

The maxagents_correlid parameter specifies the maximum number of agent threads that can process transactions with the same z/OS correlation ID or job name. This parameter allows you to reduce lock contention by limiting parallelism for transactions from the same batch job on a z/OS Q Capture server. Reducing lock contention can lower latency.

The maxagents_correlid parameter works only with Q Capture servers on z/OS. The Q Apply server can be z/OS, Linux, UNIX, or Windows.

Batch workloads on z/OS can lead to lock timeouts and deadlocks when the Q Apply program tries to apply transactions in parallel. When the target server uses a page locking scheme lock time outs are more likely to occur. These timeouts and deadlocks can substantially increase latency.

The num_apply_agents parameter specifies the total number of agent threads that are available. The maxagents_correlid parameter limits the number of agent threads that the Q Apply program uses to process transactions from the same job.

For the maxagents_correlid parameter, a value of 1 means that transactions with the same correlation ID are applied serially. A value higher than 1 allows the Q Apply program to process transactions in parallel. You set this parameter for the replication queue map that specifies the receive queue. The value of the maxagents_correlid parameter cannot be set higher than the value of the num_apply_agents parameter.

You can view the values in the DEADLOCK_RETRIES column and JOB_DEPENDENCIES column in the IBMQREP_APPLYMON table to decide how you should adjust the value for the maxagents_correlid parameter. The DEADLOCK_RETRIES column identifies the number of retries from deadlocks during the monitor interval. The JOB_DEPENDENCIES column identifies the number of transactions that were delayed during the monitoring interval.

If you experience persistent lock contention, try setting the maxagents_correlid parameter to half the number of total apply agents specified by the num_apply_agents parameter. If the value of the DEADLOCK_RETRIES column becomes 0, you can increase the value of the maxagents_correlid parameter and continue monitoring. If the value of the DEADLOCK_RETRIES column is still high, try lowering the value of the maxagents_correlid parameter. The value of the JOB_DEPENDENCIES column should increase as you lower the value of the
maxagents_correlid parameter because more dependent transactions are waiting for processing. The tuning goal is to determine a value for the maxagents_correlid parameter that reduces or eliminates deadlocks to achieve your latency requirements.

**memory_limit parameter**

The memory_limit parameter determines the amount of memory that a Q Apply program can use as a buffer to process transactions from one receive queue. You set this parameter for the replication queue map that contains the receive queue.

When the memory limit is reached, the browser thread stops getting messages from the receive queue and waits for agent threads to apply more transactions to the target table to free memory. If a single transaction is large enough to exceed the memory limit, the browser thread will apply partial rows of the transaction before processing more rows of the same transaction. Both of these actions slow performance.

A useful approach for determining the memory that is required for each receive queue is to start with a high value for the memory_limit parameter, such as 32 MB, and then check the value of the following columns in the IBMQREP_APPLYMON table to determine whether this value is adequate for your workload:

- The CURRENT_MEMORY column measures the memory usage by a Q Apply program for a receive queue. If the value is close to your setting for the memory_limit parameter, you might need to increase the amount of memory that is allocated for the receive queue.
- The MEM_FULL_TIME column measures the number of milliseconds that a Q Apply program could not process transactions from a receive queue because agents were using all available memory. If this value is greater than 0, increase the memory allocation for the receive queue. If the value of MEM_FULL_TIME is 0, increasing the amount of memory available for a receive queue will not speed processing time or improve throughput.
- The MONSTER_TRAN column measures the number of transactions that exceeded the memory limit.
- The APPLY_SLEEP_TIME column measures the number of milliseconds that all Q Apply agents went to sleep mode because they could not get a transaction from the memory. If this value is 0 and MEM_FULL_TIME is greater than 0, you might need to increase the value of the memory limit. If MEM_FULL_TIME is equal to 0 and the percent sleep time for agents is greater than 30%, you might need to reduce the value of the num_apply_agents parameter. If both MEM_FULL_TIME and APPLY_SLEEP_TIME are greater than 0, then you either have a long transaction or batch job or your target database needs to be tuned.

In Version 9.5, the default setting for the Q Capture program is to replicate LOB values inline with the transaction messages. You should adjust the memory limit so that transaction messages with LOB values do not exceed the memory limit and result in slower performance.

**Requirement:** The memory limit for a receive queue must be at least three times greater than the max_message_size parameter for the Q Capture program.

**monitor_interval parameter**

The monitor_interval sets the frequency of Q Apply inserts into the IBMQREP_APPLYMON table. If you want more current statistics to use for tuning, consider lowering the default value for this parameter.
Changing the interval (inserting more or less frequently) does not affect the rate at which transactions are applied to target tables.

The more often the monitor data is recorded, the more accurate the monitor data will be when queried. However, under a very heavy workload a shorter monitor interval (more frequent inserts into the IBMQREP_APPLYMON table) can reduce the rate at which messages are pruned from the receive queue because the same thread handles both tasks. A shorter monitor interval will also increase CPU usage.

Setting the interval lower than 50 milliseconds is not recommended.

**prune_interval parameter**

The `prune_interval` parameter determines how often the Q Apply program prunes the IBMQREP_APPLYMON and IBMQREP_APPLYTRACE tables.

Shortening the prune interval from its default of 300 seconds (5 minutes) is unlikely to affect overall throughput or latency. In general, the control tables whose pruning is set by this parameter are not very large.

The pruning of two Q Apply control tables, IBMQREP_DONEMSG and IBMQREP_DELTOMB, is not governed by the `prune_interval` parameter. These tables are for the Q Apply program’s internal processing and are managed completely by Q Apply (the tables do not provide useful tuning information). The IBMQREP_DONEMSG table is pruned every 2 seconds, and the IBMQREP_DELTOMB table is pruned every hour. You cannot change these settings.

### Search conditions and performance

You can specify a search condition for each Q subscription or publication to filter rows for replication or publication. Search conditions affect performance because the Q Capture program must evaluate the search condition for every row.

The search condition, a restricted SQL WHERE clause, can be used to filter rows that you do not want to publish or replicate. The Q Capture program evaluates the search condition after rebuilding transactions in memory from log records.

Throughput rates are affected for Q subscriptions or publications that include a search condition. The impact depends on your operating system and the number and speed of processors. The CPU costs of a search condition will also vary, but you should expect an increase.

Because fewer messages are sent with a search condition, reduced I/O for logging persistent messages and reduced network traffic might make up for some of the performance costs of a search condition.

### Job or process priority

Depending on your priority for replication or publishing throughput, you might be able to raise the job or process priority for the Q Capture and Q Apply programs. Raising the process priority can increase replication throughput, but can also decrease overall transaction processing for a source database or subsystem.

If you want higher replication or publishing throughput but do not want to lower the priority for the database or subsystem, you might want to add additional instances of the Q Capture program to help share the workload.
The job or process priority for WebSphere MQ should be the same as the Q Capture and Q Apply programs. Raising the priority of WebSphere MQ will not improve performance because the queue manager, queues, and channels can convey data only as quickly as the data is provided and retrieved by the Q Capture and Q Apply programs.

On z/OS, you can define service classes in Workload Manager (WLM) to assign the job priorities.

In a C program on UNIX systems, you can use the `setpri(process_id, priority)` command to set the process priority.

To determine the process ID of a Q Capture program or Q Apply program that was started for a user ID, you can run the `ps` command. The command `ps -ef` generates a full listing of all processes, and the command `whoami` returns the current user ID. The following examples return the process ID for a Q Capture program (asnqcap) or Q Apply program (asnqapp) that were started by the current user ID:

```
ps -ef | grep 'whoami' | grep asnqcap
```

Or:

```
ps -ef | grep 'whoami' | grep asnqapp
```

### Improving performance for partitioned databases (Linux, UNIX, Windows)

The Q Apply program provides two tuning parameters that you can use to improve replication performance with partitioned databases – `buffered Inserts` and `commit count`.

Like any application that carries out a large number of INSERT, UPDATE, and DELETE operations at a database with multiple partitions, Q Apply’s performance can be hindered by greater CPU resources that are required to handle communication between partitions. These resources are especially taxed at the coordinator partition, which by default handles all communication and coordinates transactions that involve other partitions. Performance issues are most frequent for workloads that involve multi-row transactions.

The `buffered Inserts` and `commit count` parameters enable Q Apply and the target database to group row operations in batches, reducing the amount of inter-partition communication that is required. Used alone or in tandem, these parameters can reduce CPU consumption at the target database and improve Q Apply performance, reducing latency and increasing throughput.

The following sections provide more detail about each parameter.

**buffered Inserts**

If you specify `buffered Inserts=y` when you start the Q Apply program, the program internally binds appropriate files with the INSERT_BUF option. This bind option enables the coordinator node in a partitioned database to accumulate inserted rows in buffers rather than forwarding them immediately to their destination partitions. One 4K buffer is used for each target partition. When a buffer is filled, or when another SQL statement such as an UPDATE, DELETE, or
INSERT to a different table or COMMIT/ROLLBACK are encountered, all of the rows in the buffer are sent together to the destination partition.

Using this option offers several advantages:

- Only one message is sent from the target partition to the coordinator partition for each buffer that is received by the target partition.
- A buffer can contain a large number of rows, especially if the rows are small.
- Parallel processing occurs as row operations are performed across partitions while the coordinator partition is receiving new rows.

In a replication workload that has a large number of INSERT operations, the `buffered_inserts` parameter can significantly improve performance at the target database.

**Important:** If you start Q Apply with `buffered_inserts=y`, all row conflicts or SQL errors cause the relevant browser thread to stop reading from the receive queue. This behavior replaces any error or conflict handling options that were set for the Q subscription, including the option to ignore certain SQL errors. Errors and conflicts are not logged in the `IBMQREP_EXCEPTIONS` control table, but a warning message is issued. When errors or conflicts occur, you must stop Q Apply, restart message processing on the receive queue, and then start Q Apply with `buffered_inserts=n` to get past the error. For this reason, the `buffered_inserts=y` option is only recommended in environments where errors and conflicts are unlikely to occur.

**commit_count**

The `commit_count` parameter specifies the number of transactions that each Q Apply agent thread applies to the target table within a commit scope. By default, the agent threads commit after each transaction that they apply. If you specified `commit_count=5`, for example, each agent thread issues a COMMIT after it has applied a maximum of five transactions.

Using this parameter can reduce CPU consumption at the coordinator partition by reducing the number of global commits, which can incur a significant cost.

**Note:** Use a higher value for `commit_count` only with row-level locking. This parameter requires careful tuning when used with a large number of agent threads because it could cause lock escalation resulting in lock timeouts and deadlock retries.

**Interpreting Q Capture and Q Apply monitoring statistics**

Monitoring statistics were added for Version 9.7 that you can use to help assess and tune performance for the replication programs. These statistics are recorded in the `IBMQREP_CAPMON` table, `IBMQREP_CAPQMON` table, and `IBMQREP_APPLYMON` table.

**IBMQREP_CAPMON table**

Table 1 on page 24 provides interpretive detail about monitoring statistics that were added for the Q Capture program in Version 9.7.
Table 1. Q Capture monitoring statistics added for Version 9.7

<table>
<thead>
<tr>
<th>Column</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGREAD_API_TIME</td>
<td>The number of milliseconds that the Q Capture program spent using the DB2 log read application program interface (API) to retrieve log records. This counter can help you estimate how much time Q Capture spends interacting with the database or subsystem. You might want to monitor this number if end-to-end latency is higher than desired and you want to isolate whether the issue is with Q Capture or the database. If the value is high, the database manager might be spending time retrieving archived log records and you could increase the amount of space designated for active logs.</td>
</tr>
<tr>
<td>NUM_LOGREAD_CALLS</td>
<td>The number of log read API calls that Q Capture made. You can use this statistic along with LOGREAD_API_TIME to calculate the average amount of time that Q Capture spends reading each log record.</td>
</tr>
</tbody>
</table>
Table 1. Q Capture monitoring statistics added for Version 9.7 (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_END_OF_LOGS</td>
<td>You can use these two statistics together to tune the amount of time that the Q Capture program is idle after it reaches the end of the log and has no more records to retrieve, and also to tune the <code>sleep_interval</code> parameter with other performance metrics.</td>
</tr>
<tr>
<td>LOGRDR_SLEEPTIME</td>
<td>The number of seconds that the Q Capture log reader thread slept because there were no changes to capture or because Q Capture is operating at its memory limit. For example, in a workload with many small transactions, if latency is higher than desired but sleep time is also high, work might be coming to DB2 while the log-reader thread is sleeping, and if <code>sleep_interval</code> is too high, Q Capture does not replicate these transactions until the thread wakes up, increasing latency. You need to tune this carefully based on an awareness of your typical workload, because the potential trade-off is higher CPU consumption when Q Capture makes log read API calls and finds no new transactions. Other observations about using these two metrics:</td>
</tr>
<tr>
<td></td>
<td>• The <code>sleep_interval</code> value multiplied by the <code>NUM_END_OF_LOGS</code> value gives an indication of the total time that Q Capture slept because it reached the end of the log.</td>
</tr>
<tr>
<td></td>
<td>• If LOGRDR_SLEEPTIME is greater than the total time that Q Capture slept, Q Capture is running near its memory limit. Consider increasing the value of the <code>memory_limit</code> parameter. Note: The value in the CURRENT_MEMORY column is taken at each monitor interval, and is only a weak indication of how memory was used during the monitor interval. The log-reader thread could have slept many times during the interval because Q Capture reached its memory limit, but at the time the snapshot was taken memory usage might be low. Conversely, memory usage might be high when the snapshot is taken, but the thread might never have slept because of the memory limit.</td>
</tr>
<tr>
<td></td>
<td>• On z/OS, these counters can be used to tune the <code>sleep_interval</code> to reduce the number of F calls that Q Capture makes to request that the database manager flush its log buffer.</td>
</tr>
</tbody>
</table>

**IBMREP_CAPQMON table**

Table 2 on page 26 provides interpretive detail about monitoring statistics that were added for send queues in Version 9.7.
Table 2. Send-queue-level monitoring statistics added for Version 9.7

<table>
<thead>
<tr>
<th>Column</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQBYTES</td>
<td>The number of bytes put on the send queue during the monitor interval, including data from the source table and the message header. You can use this number to monitor the raw throughput for a queue and calculate WebSphere MQ message overhead.</td>
</tr>
<tr>
<td>MQMESSAGES</td>
<td>The number of messages put on the send queue during the monitor interval. You can use this information to determine whether you need to adjust the maximum message size value for the send queue (MAX_MESSAGE_SIZE in the IBMQREP_SENDQUEUES table), or the maximum message length (MAXMSGL) attribute that is defined for the actual WebSphere MQ queue. If the number of messages that are put on the send queue is much greater than the number of transactions published (TRANS_PUBLISHED column in IBMQREP_CAPQMON), transactions are not fitting into a single message, which causes additional CPU usage for WebSphere MQ.</td>
</tr>
</tbody>
</table>

IBMQREP_APPLYMON table

Table 3 provides interpretive detail about monitoring statistics that were added for the Q Apply program in Version 9.7.

Table 3. Q Apply monitoring statistics added for Version 9.7

<table>
<thead>
<tr>
<th>Column</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROWS_PROCESSED</td>
<td>The number of rows that were read from receive queues and applied but not yet committed to the target. This value might be higher than ROWS_APPLIED because it might include rows that were rolled back. You can use ROWS_PROCESSED to determine whether the Q Apply program is still processing when a large transaction (large enough to consume most of the memory, but not large enough to be treated as a monster transaction), might make it appear that Q Apply is &quot;stuck.&quot; Before Version 9.7, counters such as TRANS_APPLIED, ROWS_APPLIED, QDEPTH, and CURRENT_MEMORY were used to make this determination. However, these counters are updated only after the transaction is committed to the target.</td>
</tr>
</tbody>
</table>
Chapter 4. Tuning WebSphere MQ for replication performance

Because WebSphere MQ is the transport mechanism for replication and publishing, you must tune your queue managers and queues to maintain optimal performance.

Message limits for queues (MAXDEPTH)

In a high-volume replication environment, you must pay particular attention to the number of messages that each queue is set up to hold. This number is set by the MAXDEPTH parameter.

The default value of the MAXDEPTH parameter for your operating system might be too small for a high-rate workload, particularly for the transmission queue (xmitq) at the source, and the receive queue at the target.

Increase the MAXDEPTH for the transmission queue if you plan to replicate or publish a large number of transactions, or if you expect many large transactions that will be divided into multiple messages. To determine the size of the transmission queue, consider these factors:

- The number of source tables that you are replicating or publishing data from
- The number and size of rows in the tables
- The value of the Q Capture max_message_size parameter, which determines the way that large transactions are divided into multiple messages
- How long you want messages to collect on the transmission queue in the event of a network outage
- Whether the Q Apply program will run continuously
- Whether other applications are sharing the transmission queue

Also, consider increasing the MAXDEPTH value for the transmission queue if the value of the commit_interval parameter for the Q Capture program is longer than the default of 500 milliseconds (a half second). A longer commit interval increases the number of messages that are put on the transmission queue at the same time. This effect is heightened if messages from multiple send queues are funneled into one transmission queue.

Set the MAXDEPTH value for the destination receive queue to be at least as large as the MAXDEPTH value for the transmission queue so that the receive queue can accommodate a large volume of transactions.

If the Q Apply program loads the target tables, set the MAXDEPTH value for spill queues based on the number of changes that you expect at the source while target tables are being loaded.

Queue indexing (z/OS)

Receive queues on the target system should be created with an index to speed pruning of the queue after transactions are applied. You set the INDXTYPE parameter when you define a local queue.

The Q Apply program gets messages from the receive queue twice: once in first-in, first-out order for applying to the target table, and a second time to delete the
message after it is applied. Because messages can be applied in a different order than they arrived on the queue, for the second MQGET operation at the receive queue the Q Apply program must look for the message by using its message identifier (MSGID).

A receive queue with INDXTYPE(MSGID) allows the queue manager to maintain an index of message identifiers that dramatically improves the speed at which the Q Apply program can prune receive queues. Otherwise, the program must search the queue sequentially.

Note: Queue indexing is available only on the z/OS platform.

**Queue buffer pools (z/OS)**

You can improve performance on z/OS by creating a larger buffer pool for queues that are expected to carry a high message load. The transmission queue at the source and the receive queue at the target are likely to have the heaviest and most consistent replication or publishing message load.

Creating a larger queue buffer might be particularly useful at the target system because of the number of WebSphere MQ operations that are performed on the receive queue. The message is put from the channel to the queue. The browser thread gets the message from the queue. The pruning thread gets the message a second time, deletes it from the queue and commits it.


**Defining buffer pools (z/OS)**

On z/OS, you can associate one or more local queues with a dedicated buffer pool to increase the amount of memory that is used to cache messages on the queue.

**Procedure**

To define the buffer pools:

1. Use the DEFINE BUFFPOOL command to define a buffer pool for holding messages in main storage, including the number of 4 KB pages that the buffer pool will contain. For high-volume queues such as the source transmission queue, set the number of pages significantly above the default of 1000, for example to 50,000.

2. Use the DEFINE PSID command to define a page set, and set the BUFFPOOL parameter to point to the buffer pool that you created. This buffer pool will be used for caching the contents of the page set.

3. Define a storage class, which maps one or more queues to a page set, by using the DEFINE STGCLASS command and specifying a page set identifier.

4. When you define the queue, specify the storage class in the STGCLASS attribute.

You can define up to 16 buffer pools for each queue manager. The maximum number of buffers is determined by the amount of storage that is available in the
queue manager address space. Usually, the more buffers that you have, the more efficient the buffering and the better the performance of WebSphere MQ.

### WebSphere MQ logging

The queue manager logs all persistent messages in active and archive logs in a manner similar to DB2. Extra overhead is required for message logging, which can affect performance.

By default, all WebSphere MQ messages that are used in replication and publishing are persistent. In the event of a forced shutdown or system failure, persistent messages can be recovered and data at the source and target remains in synch.

Ideally, you should store the log files required by WebSphere MQ on a separate physical disk from the data files that are used for message queues. This separation of files can dramatically reduce I/O contention and improve performance.

If possible, use disk striping for the queue manager logs to allow for parallel I/O and to speed recovery time.

### WebSphere MQ logging parameters for Linux, UNIX, and Windows

You can update WebSphere MQ logging parameters to tune the performance of WebSphere MQ.

You specify queue manager logging parameters in the mqs.ini file on Linux and UNIX operating systems. On Windows systems, you set these parameters by using the Log page of the queue manager properties notebook that can be accessed from the WebSphere MQ Services snap-in, or within the log stanza in the registry.

On distributed platforms, the following parameters affect the performance of WebSphere MQ:

**LogType**

Specify linear logging rather than the default of circular logging to ensure that no log data is lost, and that in the event of a system failure all messages can be recovered and replicated. You will need to monitor the disks where archive logs are stored to ensure that they do not become full, however.

When linear logging is configured, a message is entered in the queue manager error log to show which log files can be deleted. The MS0L SupportPac also provides a Java-based utility for maintaining linear log files on UNIX or Windows operating systems.

**LogBufferPages**

Specify the largest possible value for the amount of memory that is allocated to buffer records for log writes. The maximum number of 4 KB buffer pages is 512. Larger buffers lead to higher throughput, especially for larger messages. If you specify 0 (the default), the queue manager selects the size.

**LogFilePages**

This parameter specifies the size of each primary and secondary log file in units of 4K pages. This parameter controls how much data can be written to a log file before another file is used. Increasing this value can improve message throughput by reducing the amount of switching between log
files, and can also speed the task of log archiving. A value between 4096 pages and the maximum of 16,384 pages (16 MB and 64 MB) is recommended. You specify a value when you create the queue manager and cannot change the value.

LogPrimaryFiles
If you expect large transactions or long-running batch jobs without interim commits, increase the number of primary log files from the default of 3 to accommodate your workload.

LogSecondaryFiles
Typically, you should allocate two secondary log files for every three primary log files.

LogWriteIntegrity
Set the method that is used to write log records (SingleWrite, DoubleWrite, TripleWrite) based on your business needs. The default of TripleWrite might have a small extra performance cost but ensures complete integrity of the log records in case of a disaster recovery scenario. If the disk where your logs are stored has a battery-backed cache, then you can set this parameter to SingleWrite.

WebSphere MQ logging parameters for z/OS
The OFFLOAD, OUTBUFF, and WRTHRSH parameters can affect replication performance.

The CSQ6LOGP macro within the WebSphere MQ system parameter module controls logging options. For a typical replication environment, you can accept the default values for most logging parameters. You can tune the following parameters for replication:

OFFLOAD
Archive logging must be enabled for replication, so maintain the default value of YES for this parameter. Otherwise, when the active log fills, it will be overwritten with new records, and if there is a need for message recovery, some replicated data could be lost.

The MS15 SupportPac provides a utility for monitoring the use of archive logs and deleting any logs that are no longer required for any kind of backout, restart, or recovery.

OUTBUFF
Specifies the total size, in kilobytes, of the output buffers for writing to the active and archive log data sets. Each output buffer is 4 KB. In a high-volume replication environment, set this value to the maximum if possible to reduce the disk I/O.

WRTHRSH
Specifies the number of 4 KB output buffers that are filled before being written to the active log data sets. The optimum log buffer threshold for production systems is 15, which corresponds to the maximum number of buffers that are written in a single log I/O.

WebSphere MQ logging configuration for z/OS
Store the queue manager log data sets and page data sets across multiple volumes. For optimal performance, store the logs on DASD volumes with low usage.
Also, use a DASD volume that is fast-write cache enabled, and ensure that the volume has an adequate cache size to reduce log I/O and improve performance.

Allocate enough primary space for the log data sets, with no secondary allocation. When the first log data set fills, WebSphere MQ will switch to the next available one. Always preformat the log data sets.

When you create the log data sets, dual logging improves the speed of recovery in a disaster scenario without adding significant performance costs versus single logging. In WebSphere MQ version 5.3, all writes to dual logs are done in parallel if the primary log data set is on a write-cached DASD.

Use VSAM striped data sets for better performance. Switching to active logs that use VSAM striping can lead to improved throughput in situations where performance is being constrained by the log data rate.

Channel parameters

You can tune the channel batch size and interval to improve replication performance. For z/OS, you can also adjust the channel parameters to improve performance.

Channel batch size and interval

When you define a channel, you can tune the number of messages that are committed by the queue manager between two synch points. This grouping is called a batch. Messages are always sent individually over the channel, but they are not committed and removed from the transmission queue until the synch point is reached.

By increasing the size of a batch (which is set by the BATCHSZ parameter) from its default of 50, you can boost performance in a high-rate workload. Throughput is improved because the queue manager requires fewer commits. In general, the heavier your workload, the higher you should set the batch size. However, if the batch size is too large, you pay a penalty of higher I/O and CPU usage caused by memory paging of uncommitted messages.

For a high-volume replication environment, set BATCHSZ between 50 and 640. Higher values are not likely to improve throughput and can consume excessive resources.

You can also set a time interval for committing batches when you define a channel. If the time interval specified by the BATCHINT parameter expires, the queue manager commits a batch of messages even if the BATCHSZ value was not reached. A batch interval can conserve CPU in a low-volume environment, but might impact response times. For replication, start by accepting the default of 0 for BATCHINT, and tune this parameter in tandem with BATCHSZ.

z/OS channel parameters

You can tune the channel batch size and interval to improve replication performance. For z/OS, you can also adjust the channel parameters to improve performance.

The CSQ6CHIP macro within the WebSphere MQ system parameter module controls channel initiator options. For a typical replication environment, you can
accept the default values for most channel parameters. You might want to change the values of the following parameters to improve performance:

**DISPS**
Specifies the number of dispatchers to use for the channel initiator. If you have fewer than 50 channels, keep the default setting of 5. Otherwise, allow one dispatcher for each 50 current channels.

**ADAPS**
Specifies the number of adapter subtasks to use for processing channel requests. For a high-volume replication environment, a value of 30 gives optimal performance.

**CURRCHL**
Specifies the number of channels that are actually being used. The default value is 200, so if you are using fewer channels, tune this parameter to match your configuration. If you set the value too high, channels will not be evenly distributed across dispatchers.

---

**Maximum uncommitted messages (MAXUMSGS)**

The replication and event publishing programs put and get messages from queues within a synch point, so the MAXUMSGS parameter might be a factor in your WebSphere MQ tuning. MAXUMSGS is a queue manager parameter that limits the number of messages that can be put on queues or retrieved from queues within a synch point.

The Q Capture program puts messages on send queues and then commits them at an interval that you specify in the commit_interval parameter. The Q Apply program commits the messages that it deletes from the receive queue within a WebSphere MQ unit of work.

The number of allowed uncommitted messages should be greater than or equal to the sum of the message limit (MAXDEPTH) of all queues that are defined within the queue manager. If you reduce the default MAXUMSGS value, ensure that the new value is large enough to handle your peak replication workload.
Chapter 5. Performance tuning scenario: sleep_interval

This basic scenario that illustrates how you can use the Replication Center to check performance statistics, dynamically tune your replication environment, and then check for performance improvement.

In the scenario, IBM® WebSphere Replication Server is installed in a test environment. You created a unidirectional Q subscription between identical source and target tables, and you configured the Q Capture and Q Apply programs using the default parameter values.

You want to test the capabilities for very rapid (low-latency) replication by inserting rows into the source table and checking to see how quickly they are replicated to the target.

Checking performance statistics

Detailed performance data is a key to effective tuning. You can use the Replication Center to view performance statistics.

The Q Capture and Q Apply programs save statistics in three control tables: IBMQREP_CAPMON and IBMQREP_CAPQMON at the Q Capture server, and IBMQREP_APPLYMON at the Q Apply server. With the Replication Center reporting windows, you can view these statistics and tailor them to your needs.

After inserting a row at the source table and confirming that the row was replicated to the target table, you want to check how much time elapsed between the row being committed at the source and committed at the target.

To do so, you use the Latency window in the Replication Center as follows:

1. You open the window by using the following path in the Replication Center object tree: Q Replication → Operations → Q Apply servers. In the contents pane, you right-click the TARGET server and select Reports → Latency.
   The Latency window opens. The Q Apply schema is set to ASN, which identifies the Q Apply program whose performance you want to check.

2. You select End-to-end latency from the Information to display field.

3. You accept the default settings for Range of time, which collects statistics for a 24-hour period, and for Time intervals.
   The following figure shows the setup for the Latency window.
4. You click **Show Report** to view the statistics, and the **End-to-End Latency** column shows a value of 3,849 milliseconds (3.8 seconds). You insert another row and click **Show Report** again. This time, the value is 5.3 seconds. You try one more row, and the latency is 3.7 seconds.

The following figure shows the Latency report window after your three inserts:
The latency is higher than you expected. The next step is to try to locate the problem.

**Note:** By default, a Q Apply program inserts statistics into its monitor table every 5 minutes. If you insert a row at the source, the resulting statistic will not immediately appear unless you lower the monitor interval.

### Diagnosing the problem

You can use the Latency window to view statistics for the components that make up the end-to-end latency. One possible cause for high latency is the `sleep_interval` parameter.

To get a complete picture of latency, you need to look at three components:

- The time required to capture a transaction from the DB2 recovery log and put the corresponding message on a queue (Q Capture transaction latency).
- The time required for messages to travel between source and target queues (queue latency).
- The time required to get the message from a queue and apply the transaction to the target table (Q Apply latency).

You can use the Latency window to view statistics for all of these components. To do so, select **Composite latency** from the **Information to display** field and click **Show Report**. The following figure shows the report window that opens:
You notice that the queue latency and Q Apply latency are very low, less than one-tenth of a second. The Q Capture transaction latency accounts for almost all of the total end-to-end latency.

Why is it taking so long for the transactions to be captured and put on the send queue? The most likely reason is the Q Capture sleep interval. By default, the Q Capture program waits (or "sleeps") for 5000 milliseconds (5 seconds) when it reaches the end of the active DB2 log and finds no new transactions. This default conserves CPU resources in an environment where the program can easily stay ahead of the volume of updates at the source.

Because this is a test system with no activity except for a small number of updates to test replication, it is likely that the Q Capture program was in a 5-second sleep cycle when your insert was committed. Three to five seconds might have elapsed between the commit and the time that the Q Capture program woke up and read the log again.

To test this theory, you can reduce the Q Capture sleep_interval parameter, and then recheck the latency statistics to look for improvement.

### Tuning the Q Capture program

You can change the value of parameters such as the sleep interval while the Q Capture program continues to run.

You use the Change Parameters – Running Q Capture Program window as follows:

1. You use the following path in the Replication Center object tree to open the window: **Q replication → Operations → Q Capture servers**. In the contents pane, you right-click the SOURCE server and select **Change Parameters → Running Q Capture Program**.

2. You verify that the correct Q Capture schema of ASN is displayed, and then select the SLEEP_INTERVAL keyword in the table. The CURRENT_VALUE column shows that the Q Capture program is running with the default value of 5000 milliseconds.

3. In the **Value** area, you use the arrows to reduce the sleep_interval parameter to 1000 milliseconds (1 second), as shown in the following figure.
4. You click OK. The Q Capture program begins running with the shorter sleep interval.

You also could dynamically reduce the Q Capture sleep interval using the chgparms parameter with the MODIFY or asnqccmd commands instead of the Replication Center:

```
f myqcap,chgparms sleep_interval=1000
asnqccmd capture_server=source capture_schema=ASN chgparms sleep_interval=1000
```

### Checking the results of tuning

If the longer sleep interval caused the higher-than-expected delay, the values for Q Capture transaction latency and end-to-end latency should both be lower after you reduced the sleep interval.

You insert two more rows, and then check the statistics again by using the Latency window. The following figure shows the result.
The Q Capture transaction latency and end-to-end latency have both dropped sharply, while the queue latency and Q Apply latency have remained about the same. This data confirms that the sleep interval was causing the extra delay.

Another factor that affects the latency of replicated transactions is the Q Capture commit interval. By default, the program waits 500 milliseconds (a half second) before committing messages that it has created from log records to the WebSphere MQ queue manager. You could reduce the overall latency of transactions by cutting this interval, although the default value is considered optimal for a typical replication workload.
Chapter 6. Performance tuning scenario: memory_limit

Tracking down Q replication performance problems can be an iterative process that involves checking the source database, the target database, and WebSphere MQ. In the sleep interval tuning scenario, you were able to pinpoint the problem in the first area that you checked, at the source.

In a high-volume production environment, the sleep interval is less of a factor. When a database is being continually updated, the Q Capture program rarely sleeps.

For more helpful tuning information, you can use the Q Capture Throughput and Q Apply Throughput windows to view statistics about such important performance indicators as number of rows and transactions processed, memory usage, transaction size, and conflicts.

The following example briefly summarizes a scenario in which you use the Replication Center to check and tune performance:

1. Throughput seems lower than expected, so you use the Q Capture Throughput and Q Apply Throughput windows to check the number of rows and transactions processed per second.
2. The throughput numbers at the source are less than what you require, so you use the Q Capture Throughput window to check whether any transactions were spilled because of inadequate memory for Q Capture. Spilling might affect performance throughput.
3. You find that spilling occurred, and some transactions exceeded the memory limit, which was set at 32 MB.
4. You increase the memory limit to 64 MB while the Q Capture program is running, and then check the number of rows processed by the Q Capture program. The numbers improve.
Chapter 7. Viewing statistics in the monitor tables

In addition to using the Replication Center to view performance statistics, you can use a command prompt or one of the DB2 command line tools to issue SQL SELECT statements and view performance statistics.

The performance statistics are stored in the following control tables:

**IBMQREP_CAPMON**
Contains statistics for a single Q Capture program. The Q Capture program inserts a row into this table at each monitor interval. If you use the MONITOR_TIME column in an ORDER BY clause, you can view the rows chronologically.

**IBMQREP_CAPQMON**
Contains Q Capture statistics that are grouped by send queue. The Q Capture program inserts a row into this table for each send queue at each monitor interval. If you use the SENDQ and MONITOR_TIME columns in an ORDER BY clause, you can view the rows chronologically for each send queue.

**IBMQREP_APPLYMON**
Contains Q Apply statistics that are grouped by receive queue. The Q Apply program inserts a row into this table for each receive queue at each monitor interval. If you use the RECVQ and MONITOR_TIME columns in an ORDER BY clause, you can view the rows chronologically for each receive queue. If you use an ORDER BY clause for the QSTART_TIME column, you can view statistics before and after the receive queues or the Q Apply program restarts.
Product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product interface, in a suite-wide information center, and in PDF file books.

You can also order IBM publications in hardcopy format online or through your local IBM representative.

To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order

You can send your comments about documentation in the following ways:
• Online reader comment form: www.ibm.com/software/data/data/rcf/
• E-mail: comments@us.ibm.com

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You can also provide feedback on products and documentation.

Customer support

For customer support for IBM products and for product download information, go to the support and downloads site at www.ibm.com/support/us/

You can open a support request by going to the software support service request site at www.ibm.com/software/support/probsub.html

My IBM

You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/us/

Software services

For information about software, IT, and business consulting services, go to the solutions site at www.ibm.com/businesssolutions/us/en

Information Management product support

For Information Management product support, news, and other product information, go to the Information Management support site at www.ibm.com/software/data/support/

Federation, replication, and event publishing products support

For support, go to:
• IBM InfoSphere Federation Server www.ibm.com/software/data/integration/support/federation_server/
- IBM InfoSphere Replication Server
  www.ibm.com/software/data/integration/support/replication_server/
- IBM InfoSphere Data Event Publisher
  www.ibm.com/software/data/integration/support/data_event_publisher/

**Classic products support**

For support, go to:
- IBM InfoSphere Classic Federation Server for z/OS
  www.ibm.com/software/data/integration/support/classic_federation_server_z/
- IBM InfoSphere Classic Replication Server for z/OS
  www.ibm.com/software/data/infosphere/support/replication-server-z/
- IBM InfoSphere Classic Data Event Publisher for z/OS
  www.ibm.com/software/data/integration/support/data_event_publisher_z/
- IBM InfoSphere Data Integration Classic Connector for z/OS
  www.ibm.com/software/data/integration/support/data_integration_classic_connector_z/

**General information**

To find general information about IBM, go to [www.ibm.com](http://www.ibm.com).

**Product feedback**

You can provide general product feedback through the Consumability Survey at

**Documentation feedback**

You can click the feedback link in any topic in the information center to comment on the information center.

You can also send your comments about PDF file books, the information center, or any other documentation in the following ways:
- E-mail: comments@us.ibm.com
How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

```
>>>required_item
```

- Optional items appear below the main path.

```
>>>required_item
   <optional_item>
```

If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

```
>>>required_item
   <optional_item>
```

- If you can choose from two or more items, they appear vertically, in a stack.
  If you must choose one of the items, one item of the stack appears on the main path.

```
>>>required_item
   <required_choice1>
   <required_choice2>
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
>>>required_item
   <optional_choice1>
   <optional_choice2>
```

If one of the items is the default, it appears above the main path, and the remaining choices are shown below.

```
>>>required_item
   <default_choice>
   <optional_choice1>
   <optional_choice2>
```

- An arrow returning to the left, above the main line, indicates an item that can be repeated.
If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.

**Fragment-name:**

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, column-name). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).
Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere™ Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- IBM InfoSphere Business Glossary
- IBM InfoSphere Business Glossary Anywhere
- InfoSphere DataStage® and QualityStage Administrator
- IBM InfoSphere FastTrack
- IBM InfoSphere Information Analyzer
- IBM InfoSphere Information Services Director
- IBM InfoSphere Metadata Workbench

For more information about IBM product accessibility status, go to [http://www.ibm.com/able/product_accessibility/index.html](http://www.ibm.com/able/product_accessibility/index.html)

Accessible documentation

Accessible documentation for IBM Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not
been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**

IBM trademarks and certain non-IBM trademarks are marked on their first occurrence in this information with the appropriate symbol.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
Index

A
accessibility 43
ADAPS parameter 31
agent threads, Q Apply program 18, 19
applheapsz parameter 5
APPLY_SLEEP_TIME column 18, 19
autoconfigure command 5

B
BATCHINT parameter 31
BATCHSZ parameter 31
blksize parameter 8
buffer pools
  DB2  3
    for queues  28

C
cachedyn parameter 8
channel batch size and interval 31
channel parameters for z/OS 31
commit_interval parameter 13
control tables
  table spaces for  3
  using RUNSTATS  3
  viewing statistics  41
  with extensive data changes  3
CSQ6CHIP macro 31
CURRCHL parameter 31
customer support 43

D
database manager parameters (Linux,
  UNIX, Windows) 7
database parameters (Linux, UNIX,
  Windows) 5
database parameters (z/OS) 8
DB2 instance parameters 7
DB2 tuning
  buffer pools  3
  database manager parameters (Linux,
  UNIX, Windows) 7
database manager parameters (Linux,
  UNIX, Windows) 5
database parameters (z/OS) 8
evironment variable settings (Linux,
  UNIX, Windows) 8
statistics 3
table spaces 3
db2_mmap_read environment variable 8
db2_parallel_io environment variable 8
db2codepage environment variable 8
dbheap parameter 5
deallct parameter 8
disk system
  DB2 storage  3
  WebSphere MQ storage  29
DISPS parameter 31
documentation
  accessible 43
DSNZPARM settings 8

E
evironment variable settings (Linux,
  UNIX, Windows) 8
EXTSHM environment variable (AIX) 8

I
IBM support 43
IBMREP_APPLYMON table 23, 41
IBMREP_CAPMON table 23, 41
IBMREP_CAPQMON table 23, 41
IBMREP_DELTOMB table 3
IBMREP_DONEMSG table 3
IBMREP_EXCEPTIONS table 3
IBMREP_SIGNAL table 3
IBMREP_SPILLEDROW table 3
indexing queues 27
INDXTYPE parameter 27
intra_parallel parameter 7

J
job priority 21

K
KEEPDYNAMIC(YES) bind option 8

L
large object (LOB) data, message size 15
legal notices 49
locklist parameter 5
log stanza in Windows registry 29
logbufsz parameter 5
logfilisz parameter 5
logprimary parameter 5
LOGDR_SLEEPTIME 23
LOGREAD_API_TIME 23
logs
  DB2  3
  WebSphere MQ  29
logsecond parameter 5

M
MAX_MESSAGE_SIZE, tuning 23
max_msg_size parameter 15
maxagents parameter 7
maxappls parameter 5
MAXDEPTH parameter 27
maximum uncommitted messages 32
maxlocks parameter 5
MAXMSGL parameter, WebSphere MQ
  tuning 23
maxrtu parameter 8
MAXUMSGS parameter 32
MEM_FULL_TIME column 18, 19
memory_limit parameter
  Q Capture  14
  memory_limit parameter, Q Capture
    tuning 23
message limits for queues 27
monitor_interval parameter
  Q Apply  21
  Q Capture  16
MQBYTES  23
MQMESSAGES  23
mqs.ini file (Linux, UNIX) 29
num_apply_agents parameter 18, 19
NUM_END_OF_LOGS  23
num_lcleanser parameter 5
num_ioservers parameter 5
NUM_LOGREAD_CALLS  23

O
outbuff parameter (DB2 for z/OS) 8

P
parallelism of Q Apply program 18, 19
pckcachesz parameter 5
performance tuning 23
process priority 21
product accessibility
  accessibility 47
prune_interval parameter
  Q Apply  20, 21
  Q Capture  17

Q
Q Apply parameters 17
Q Apply program
  diagram of tuning parameters  17
  monitoring statistics  23
  multiple instances  12
  parallelism  18, 19
Q Capture parameters 12
Q Capture program
  diagram of tuning parameters  12
  memory requirements  14
  monitoring statistics  23
  multiple instances  11
spill files 3
queue manager logs 29

© Copyright IBM Corp. 2004, 2009
queues
  buffer pools for  28
  indexing    27
  message limits  27

R
Replication Center
  Change Parameters window  36
  Latency window   33
ROWS_PROCESSED   23
RUNSTATS      3

S
scenario for tuning
  memory_limit  39
  sleep_interval  33
schemas
  multiple Q Apply  12
  multiple Q Capture  11
screen readers  43
search conditions  21
setpri command  21
sleep_interval parameter    15, 35
sleep_interval parameter, Q Capture
  tuning  23
software services  43
spill files  3
stmheap parameter  5
support, customer  43

T
table spaces for control tables   3
trademarks  51
TRANS_SPILLED column  14
tuning scenario
  memory_limit  39
  sleep_interval  33

U
uncommitted messages, maximum  32
unit parameter   8

W
WebSphere MQ logging configuration for
  z/OS  31
WebSphere MQ logging parameters
  Linux, UNIX, Windows  29
  OFFLOAD parameter  30
  OUTBUFF parameter (WebSphere MQ)  30
  WRTHRSH parameter  30
  z/OS  30
WebSphere MQ Services snap-in   29