Optimizing RDM Server Performance

Abstract

This article is a first in a series that will discuss ways that users can boost the performance of their RDM-based application. For this article, the topic covers suggested ways to modify the initialization parameters, achieving higher performance without having to modify your application code.

Contents

1 PART ONE: RUNTIME INI SETTINGS

One of the most frequent questions I am asked when talking to developers creating RDM-based applications is - How can I maximize performance? Unfortunately, over that past ten years, I have learned there is no single answer that magically works for everybody every time. Each application has unique characteristics that can make the optimum settings for one system completely inappropriate for another. This is the first in a series of articles that will shed light on ways users can boost the performance of their RDM-based application. The topic for this article is modifying the settings in the rdmserver initialization file (rdmserver.ini) to increase the performance of RDM Server. It is possible to achieve considerable performance gains without having to modify a single line of code or changing the physical structure of the database.

1.1 Background

First things first, before getting started it's necessary to have an understanding of what is being optimized. Optimization techniques that are appropriate for reading data may not affect writing and vice-versa. So, before beginning, have an idea what the performance goals are and how they are going to be measured.

Second, test under conditions that are similar to how the application is expected to be deployed. If a typical database is to have 500 million rows do not perform benchmarking with only 50,000 rows. If the application will deploy on systems that are running a significant number of processes beyond the database, make sure that the database is tested in that environment. A lot of development effort can be wasted tuning a database server for conditions that are not realistic.

Finally remember that the database runtime settings are only one item in a series of things that need to be considered when optimizing your application. Other things that need to be considered, and will be touched on in future articles, include:

1.1.1 Application Design

How efficiently your database performs is very dependent on how accurately you can tell it what you are looking for. Creating efficient access plans or giving the optimizer the opportunity to create and efficient plan can make all the difference.

1.1.2 Application Architecture

Choosing how your application accesses the database can influence the performance characteristics of the database. If the database server is located on the same box as the application perhaps you want to use the Direct-link architecture and avoid the RPC overhead. If you are going to have 50 threads that all need access to data stored in the database perhaps you need to have a server connection queue that will limit the amount of resources necessary across all of the threads. These application architecture design questions are an important part in developing an efficient database application.

1.1.3 Database Design

The physical structure of the database can affect the database performance. Choosing the proper amount of normalization, the proper allocation of indexed columns, and the physical characteristics and location are important.

1.1.4 Hardware

Database systems are heavily bound to the system's CPU, I/O, and memory use. As a result selection of disks, controllers, RAID, RAM, and operating system will severely affect how your database performs.

2 RDMSERVER.INI SETTINGS

The following addresses the rdmserver.ini settings that have impact on the application performance. They grouped into the following logical units:

- Checkpoints
- Transactions
- Cache
- Threads
- Encryption
- Locking
- Miscellaneous

2.1 Checkpoints

RDM Server uses Checkpoint processing to migrate data pages that have been modified in the server cache to the database files where they belong. During a checkpoint operation normal processing of database calls continues, but at a degraded pace. By tuning the Checkpoint options a developer/administrator can optimize applications that perform a significant number of data modification operations.

The checkpoint process first copies all modified data into the checkpoint buffer. The buffer is then written to the checkpoint image file. Finally, the modified data in the checkpoint buffer is written to the database files (or to the hot file if backup mode is active).

A checkpoint operation is triggered automatically by the engine whenever one of the following two conditions is reached:

1. The percentage of dirty cache pages is greater than the CheckPointThreshold setting.
2. The size of all the dirty cache pages is greater than the MaxCheckPointSize setting.

In addition a developer can manually request a checkpoint operation by calling the 'd_checkpoint' API.

2.2 rdmserver.ini Settings

2.2.1 CheckPointDevice

Default Value: sysdev

The **CheckPointDevice** setting identifies the name of the RDM Server device that will contain the checkpoint file. If not specified the checkpoint file will be located in the catalog directory. It is best for both reliability and performance to locate it on a device that is associated with a different disk drive (different controller ideally) than the one(s) on which your database files are stored. It is okay, however, for CheckPointDevice and ChangeLogDevice to be the same.

2.2.2 MaxCheckPointSize

Default Value: 2,048,000 (bytes)

The **MaxCheckPointSize** determines the size of the Checkpoint buffer. When the buffer is full a checkpoint operation will occur. If you have a 4K byte page size and 5000 cache pages then a Checkpoint will occur when 500 pages are dirty (4096 * 500 = 2,048,000). This effectively gives you a CheckPointThreshold of 10%. The larger the size of the buffer the less frequent the checkpoint operation will occur.

2.2.3 CheckPointThreshold

Default Value: 40%

The **CheckPointThreshold** specifies the percentage of the RDM Server database cache that, when dirty, will trigger a checkpoint operation. If you have the default threshold (40), a 1K pagessize, 5,000 cache pages, and have upped the MaxCheckPointSize to 20M (20,971,520 bytes). A checkpoint will still occur when 2,000 pages are dirty even though your buffer will only be 10% full.

2.2.4 FileCommit

Default Value: 1 (true)

This parameter will cause RDM Server to perform a file sync operation on each database file (on some platforms the operation uses write-through or unbuffered writes) when it completes a checkpoint operation. In a file sync operation, RDM Server makes sure that data has been written to disk before proceeding. While a file sync operation is expensive, it is only recommended that you set this value to 0, if your data is not persistent. Setting this value to 0 can cause corruption if there is an abnormal server abort.

2.2.5 LogCheckpoints

Default Value: 0 (false)

This option will place a notification in the RDM Server log files each time a checkpoint operation occurs. This can be used during development efforts to detect how often checkpoint operations are occurring. This should not be set in a production environment.

2.3 Transactions

Transaction processing is typically the biggest bottleneck of update intensive applications. By recognizing the needs of your application it is possible to tune the engine to provide optimal performance. It is first important to have a general understanding of how RDM Server handles transactions. RDM Server uses change logging to record each change made to a database during a transaction. When your application commits a transaction, the server appends a transaction end marker to the change log and optionally flushes the change log to disk. It is the presence of the transaction end marker on disk makes the changes for the transaction permanent. The actual writes to the database files are all done by the check point processing described earlier.

2.4 rdmserver.ini Settings

2.4.1 ChangeLogDevice

Default Value: sysdev

The **ChangeLogDevice** setting specifies the name of the RDM Server device that will contain the change log files. If not specified the change logs will be located in the catalog directory. It is best for both reliability and performance to locate the change log on a device that is associated with a different disk drive than the one(s) on which your database files are stored. It is okay, however, for CheckPointDevice and ChangeLogDevice to be the same.

2.4.2 ChangeLogBufSize

Default Value: 262,144 (bytes)

The **ChangeLogBufSize** setting determines the size of the change log buffer in bytes. When the change log buffer is full, it is flushed to the change log file. A larger value will result it somewhat fewer flushes and slightly better performance.

2.4.3 ChangeLogSize

Default Value: 2,000,000 (bytes)

This sets the maximum size of the change log file which contains the history of the database changes for recovery purposes. When the number of changes consumes this amount of space in the change log, RDM Server will perform a change log cycle operation in which a new change log file is created. A change log cycle operation is expensive for applications that perform numerous transactions because it requires a delay is starting any new transactions until the cycling has been completed (see **CycleTimeout**). Increasing this setting will decrease the number of cycles that are required and can increase performance.

2.4.4 CycleTimeout

Default Value: 4 (seconds)

A change log cycle cannot proceed when there are active transactions. It will wait the amount of time (in seconds) specified by the parameter **CycleTimeout**. If unspecified, the default timeout value is 4 seconds. If all active transactions do not all complete before the timeout, the change log file is allowed to grow an additional 10% the system will retry cycling the log file when that 10% is exhausted. New transactions are not allowed to start (they wait) while the attempt is made to cycle the change log file. Long running transactions can result in the timeout. You can force a change log cycle to occur by calling s_backupBegin (you don't have to do the backup, you could just end). So, if you know when there are few, if any, transactions executing you could force a periodic change log cycle and minimize the chances that a timeout will occur. The cycle event does slow down system performance because it is preventing new transactions from starting for the duration of the CycleTimeout period.

2.4.5 LogBlobs

Default Value: 1

Logging BLOB data can be quite expensive. It is possible to tell the engine to not log the contents of blob fields. In this case the blob data would not be recoverable in case of abnormal server termination.

3. 0 Blob data logging is off. A call d_bloblog can override this setting .
4. 1 Blob data logging is on. A call to d_bloblog can override this setting.
5. 2 Never log BLOB data. Cannot be overridden.
6. 3 Always log BLOB data. Cannot be overridden.

2.4.6 AsyncTransactions

Default Value: 0 (false)

This flag improves system performance at the cost of reliability. RDM Server normally (**AsyncTransactions**=0) flushes the change log buffer to the change log file at each transaction end. The change log buffer is also flushed on every checkpoint operation. Enabling **AsyncTransactions** causes RDM Server to only flush the change log buffer during checkpoints. This means, however, that those transactions that are committed between checkpoints will be lost in the event that the system abnormally terminates. Except for those lost transactions, the database will remain in a transaction consistent state.

2.4.7 MaxTrends

Default Value: 0 (false)

2.4.8 MaxTrendWaits

Default Value: 0 (false)

By default (with these parameters set to 0), RDM Server flushes the change log buffer every time a transaction is ended (via either a commit or rollback). In a high transaction throughput application (where several transactions are being committed per second), some performance increases can be achieved by allowing multiple transaction commits to be grouped into a single change log flush operation.

The **MaxTrends** parameter specifies the number of transaction ends (commits) to occur before flushing the change log. Each transaction end waits until the specified number has been received. Then the change log is flushed and control is returned to each of the sessions that initiated the transaction end operation. The **MaxTrends** should be set to a value that is no greater than the number of active sessions that are issuing transactions.

The **MaxTrendsWait** parameter specifies the number of milliseconds to wait for the **MaxTrends** number of transaction ends to arrive. If **MaxTrends** transaction ends do not arrive within the **MaxTrendsWait** timeframe, the change log buffer will be flushed and control will return to the sessions that issued the transaction ends.

Note that transaction ends do not return until **MaxTrends** have arrived or **MaxTrendWait** has elasped. Hence the need to have high transaction rates otherwise use of these parameters will actually lower performance.

DO NOT USE THESE PARAMETERS IF AsyncTransactions IS ENABLED!

2.5 Cache

The server engine cache manager maintains a database cache that contains frequently accessed pages from the database files. The cache is used by the server to temporarily store data being actively used so that the disk will not be accessed so often. The Checkpoint Processing described earlier is how RDM Server transfers modified data from the database cache to disk.

2.6 rdmserver.ini Settings

2.6.1 MaxCachePages

Default Value: 5,000 (pages)

This specifies the number of database pages that will be kept in the internal engine cache. The pages size can be different for each database file. By increasing the number of cache pages a read-intensive application can reduce the number of disk accesses. However, many operating systems (Windows, Linux) have aggressive file cache strategies that can effectively perform this optimization as well. If you have a database page size of 4K then the amount of memory allocated for cache will be 20,480,000 bytes (5,000 * 4096).

2.6.2 GrowCachePages

Default Value: 0 (off)

When a large cache page is evicted for a smaller page the system will not reallocate the page. This means that if you have one file with a larger page size than all of the other files that eventually your entire cache will be sized based on the largest page size. The **GrowCachePages** setting (when turned on) will tell the engine to reallocate the cache pages whenever a new page is not the same size as the pages being evicted.

2.7 Threads

The RDM Server engine architected a highly threaded manner. There are a number of settings that can control the properties of these threads.

2.8 rdmserver.ini Settings

2.8.1 NumRPCThreads

Default Value: 16 (threads)

Specifies the number of worker threads the RPC system should start up when it is initialized. The default is 16. This is the maximum number of remote client requests that can be handled simultaneously. Other requests will be queued until one of the threads finishes its work and is available to work for another client. Care should be taken to not increase this so high that thread scheduling fragmentation results in lower throughput than having some requests queued. For multi-processor machines it is advisable to increase the number of threads, but it is never necessary to have this value higher than the number of simultaneous server connections.

2.8.2 NCPThreadPriority

Default Value: high

2.8.3 IOThreadPriority

Default Value: high

On platforms (e.g., Windows) that allow thread priorities to be specified, these parameters are used to set the priorities for the threads used in performing network communications (NCP) and input/output operations (I/O). We recommend that you leave these set to high unless the RDM Server engine is running on a real-time system in which there are application threads that must run at a higher priority. Higher priority means that the operating system will assign a greater amount of CPU time to those threads.

2.9 Encryption

RDM Server supports encryption of both communications and data. The following settings can control how encryption is handled. Allowing non-encrypted clients can improve performance, but may not be secure in some environments.

2.10 rdmserver.ini Settings

2.10.1 EncryptAllCommunications

DefaultValue: no

This parameter specifies whether all communication is required to be encrypted. The default is no which indicates that communication is accepted from a client that is not sending requests in an encrypted format.

2.11 Locking

When using instance locking we use a hash algorithm for checking if a record is locked or not. This setting allows you to modify the hash algorithm to optimize performance for situations where there are a number of un-freed locks.

2.12 rdmserver.ini Settings

2.12.1 NumTableLockBuckets

Default Value: 20

Setting to a large number (e.g. 100) will significantly improve performance when a large number of un-freed locks (e.g. within a single transaction) are being placed by individual sessions. Example from SQL would be insert from file, update/delete where many rows are being changed. The cost for the performance, as always, is that more memory will be required. You can compute the cost as follows. Each bucket uses 4 bytes per record type (table) per login session plus 24 bytes. Hence, with NumTableLockBuckets=50 on a database with 25 record types/tables and 75 active sessions the memory requirements will be = 4*50*25*75 + 24 = 375,024 bytes.

2.13 Miscellaneous rdmserver.ini Settings

2.14 rdmserver.ini Settings

2.14.1 MaxUsersOn

Default Value: 10

The **MaxUsersOn** parameter specifies the maximum number of login sessions that will be allowed by the RDM Server engine. This allows the system to statically allocate login session tables at engine start-up reducing the amount of needed multithread synchronization. This value can be no higher than the value of **MaxUserLicensed**.

2.14.2 SkipLoginLogout

Default Value: 0 (false)

By default, the RDM Server engine will display a message on the console log (RDS.LOG) showing each session that logs in and logs out. Setting **SkipLoginLogout** to 1 will disable the output of these session login/logout messages.

2.14.3 FileCreateSize

Default Value: 32768 (bytes)

2.14.4 FileExtendSize

Default Value: 32768 (bytes)

These values specify how large a database file will be by default when it is created and by how much the file is expanded when the current allocation is filled. A large file creation/extend size results in some performance improvements by reducing the number of times the file system has to expand the file.

3 CONCLUSION

Through manipulation of the RDM Server initialization file settings it is possible to achieve measurable performance improvement in an application without modifying a single line of code. Not every application will see equal performance variance through ini modifications, so the only sure fire way to determine what works best for your application/system is to have a benchmark application and run that application using different combinations of settings.

Want to know more?

Please call us to discuss your database needs or email us at info@raima.com. You may also visit our website for the latest news, product downloads and documentation: www.raima.com.