Raima®

RDM Embedded 10 Comparison Benchmark

Date: February 3rd, 2011
Author: Jeff Parsons, Senior Engineer
Wayne Warren, Chief Architect

Abstract

RDM Embedded 10 was re-architected to optimize multi-user performance in multi-core and multiple networked computer environments. This report shows benchmark results demonstrating up to an 18-fold improvement in transaction performance compared to an earlier version of RDM Embedded.

Contents

Background

Raima customers of RDM Embedded 9.1 or earlier are keenly interested in how the new architecture of RDM Embedded 10 will impact the performance of their applications, which are already running just fine.

The answer to this question is situational. When the RDM Embedded database is used by a single program as a structured data store, the newest version has little to offer. Version 10 has a new architecture that specifically targets multi-user applications running on computers that have multiple core processors or multiple computers that are connected through a high speed LAN. It has features that allow databases to be accessed and updated in parallel, which means that performance can be scaled up with additional hardware. This is the future of computer hardware, and RDM Embedded version 10 is prepared to shine in this world.

This benchmark compares RDM Embedded version 8.1 against RDM Embedded 10.0 as they Insert, Update and Delete records in a database while simultaneously a load of reader processes are running. These measurements may not be applicable for some applications, but for many, they show that version 10 scalability will improve their performance under higher loads.

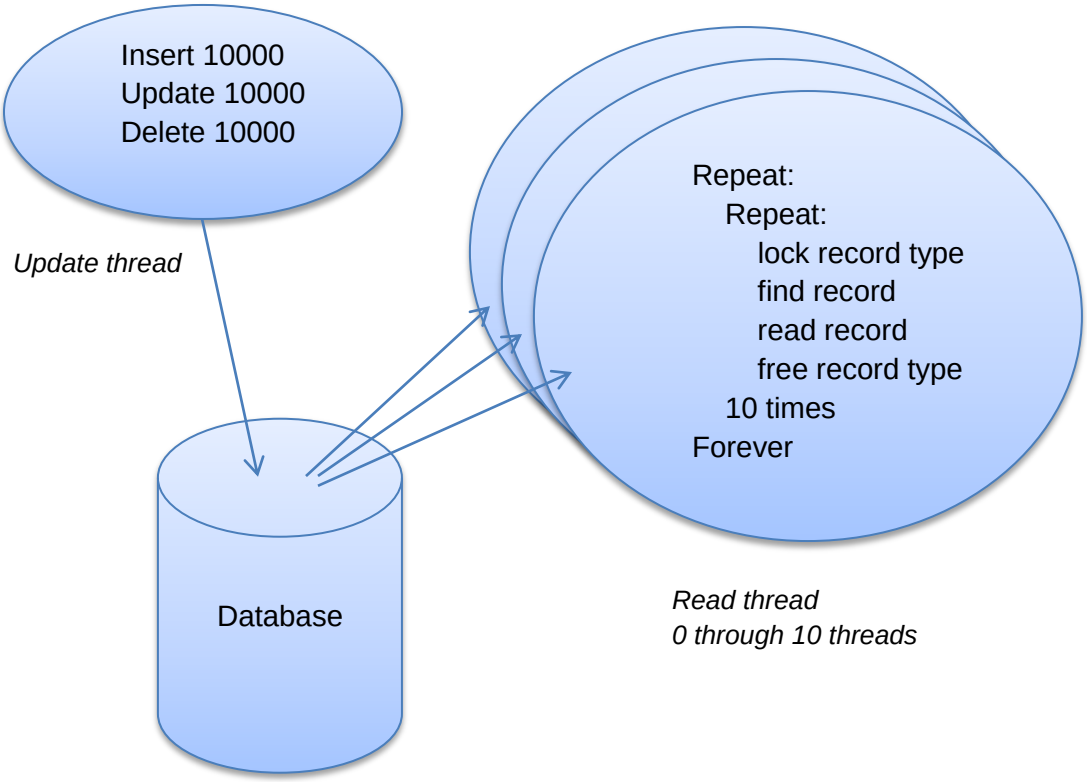
To achieve high performance with earlier versions of RDM Embedded, an option to skip the 'sync' operation (flushing all file buffers to physical disk), which is very costly but necessary for durable transactions, is turned on by setting SYNCFILES=0 in the rdm.ini file. By skipping the 'sync' operations, transactions are much faster, but the application programmer is accepting the risk that an unexpected interruption of computer processing might cause a database corruption. RDM Embedded 10 uses the 'sync' operation sparingly in order to maintain transaction safety without paying a high performance price.

Because of the common use of SYNCFILES=0, this benchmark compares both old and new versions in their non 'sync' modes. Note that if both versions were compared in 'sync' mode, the performance advantages of version 10 would be even greater than those that are reported here.

Application Overview

The application tests performance in single writer, multiple reader scenarios. The main thread performs a series of 10,000 inserts, followed by a series of 10,000 random updates and then finishes with deleting all records in the database. When running the test there can be 0 or more background threads that are actively reading data from the database. The test provides the total run time (in milliseconds) for each of the three different write operations (insert, update, delete). It does not report any information on the threads that are reading from the database.

Operations, whether insert, update or delete, are performed within transactions. Because transactions are safe, double-writes to disk, they always carry significant overhead. A single operation, by contrast, is very quick. This means that if operations can be grouped into the same transaction, the overall throughput is increased, because fewer transaction commits are required to perform the same number of operations. The test results below show the total time required for the same number of operations, but with varying levels of blocking them into transactions. The first results show 1 operation per transaction, the second shows 10 per transaction, and the last one shows 100 per transaction.



Insert Operations

The insert operations serially insert data into the database using a sequential primary key value.

Update Operations

The update operations do a lookup on a random key (d_keyfind) and then update a field in the record (non-indexed).

Delete Operations

The delete operations loop through the table (d_recfrst/next) and delete all of the records in the table.

Read Operations

The read threads perform key lookups (d_keyfind) followed by a field read based on a randomly generated primary key value. In this test each thread did 100 lookup/reads per lock unit.

Direct-Link Library

The references to TFST in this report indicate the direct-link usage of the RDM Embedded 10 Transactional File Server. This makes the use of versions 8.1 and 10 the most similar and fair for comparison.

Variables

The benchmark always uses 10,000 records, but varies the executions on two axes:

1. *The number of reader threads*. The first table uses 10 reader threads, but subsequent tables show differences from 0 reader to 10 reader threads.
2. *The number of operations per transaction*. By nature, a transaction has a certain fixed overhead. By grouping more update operations in a single transaction, the total throughput increases. The ability to group operations within a transaction is application dependent.

Schema

The test was run using a very simple database schema. The same schema source file was used for both RDMe 8.1 and RDMe 10.0.

```
database testdb {
    data file "testdb.d01" contains rec_200;
    key  file "testdb.k01" contains rec_200.id;

    record rec_200 {
        unique key int32_t id;
        int16_t        hexkey[3];
        char           payload[200];
    }
}
```

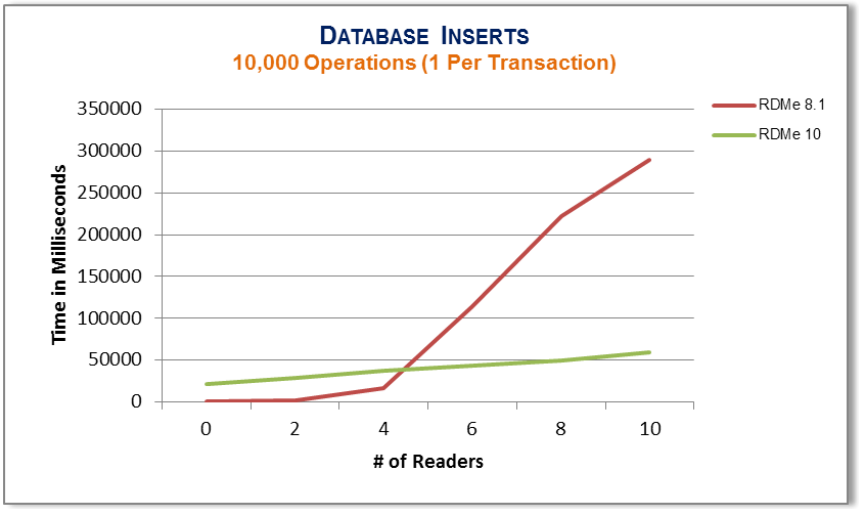
Hardware Specifications

- ***Processor:*** Intel ® Core™2 Duo CPU E8400 @ 3.00 GHz
- ***Memory:*** 8.00 GB
- ***Operating System:*** Windows 7 – 64bit

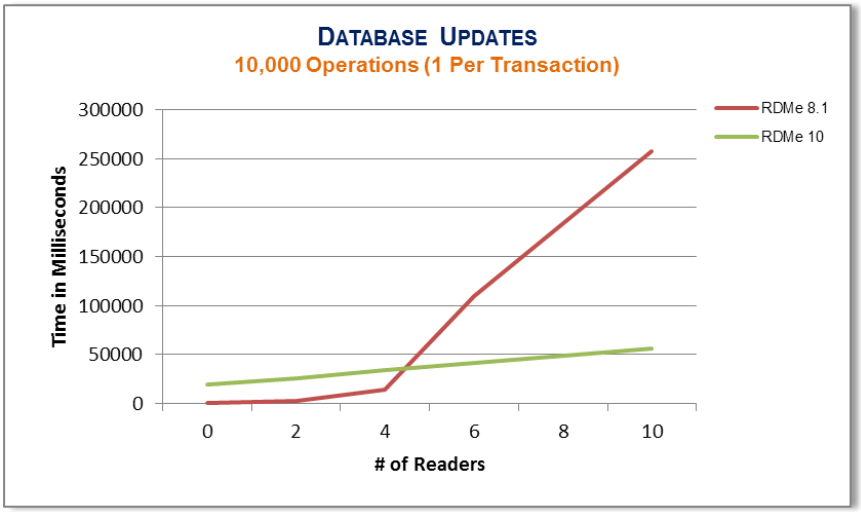
The Results

One Operation/Transaction, Variable Number of Readers

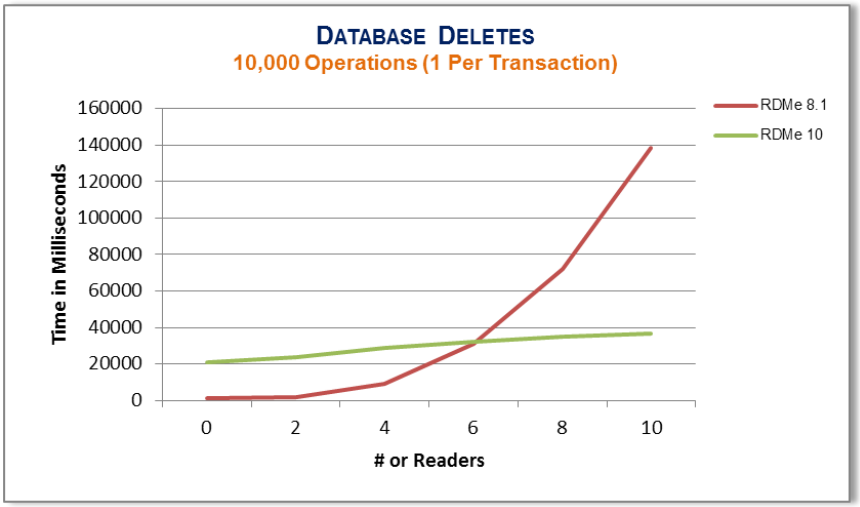
This test demonstrates the scalability of RDM Embedded 10 using test runs with different numbers of reader threads. As can be seen the performance of RDM Embedded 10 remains more consistent as additional threads are added, while RDM Embedded 8.1 times start increasing exponentially when there are more than 2 read threads active.



10,000 operations (1 per transaction) - Inserts		
# Readers	**RDMe 8.1**	**RDMe 10**
0	1299	21283
2	2122	28961
4	17035	36978
6	113982	43919
8	222539	49547
10	289439	59142



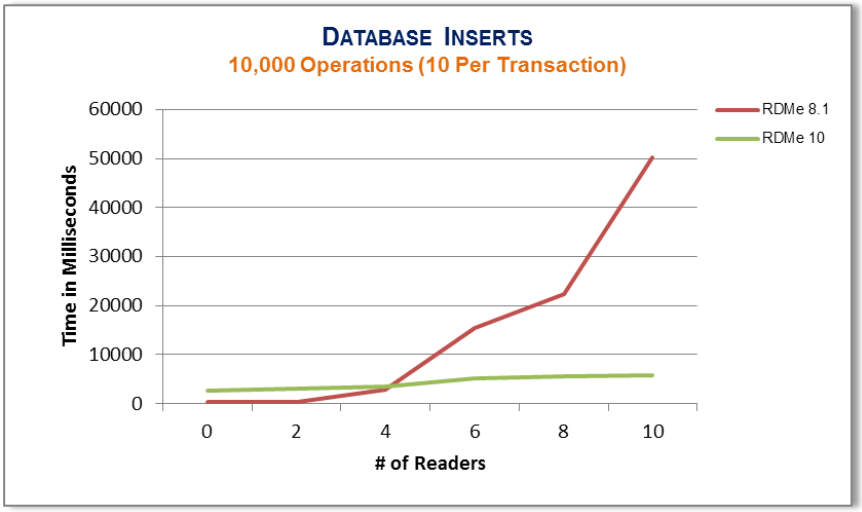
10,000 operations (1 per transaction) -Updates		
# Readers	**RDMe 8.1**	**RDMe 10**
0	1058	19447
2	2298	25293
4	14449	33803
6	109264	41827
8	183882	49246
10	256963	56159



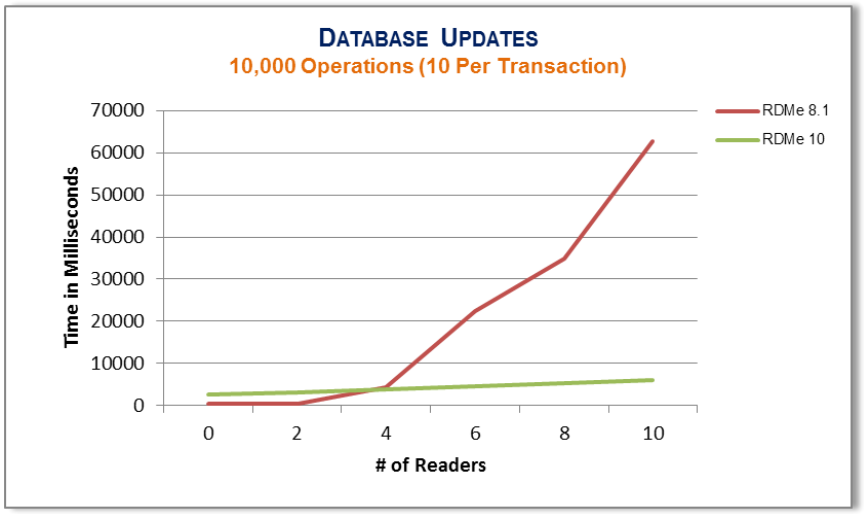
10,000 operations (1 per transaction) - Deletes		
# Readers	**RDMe 8.1**	**RDMe 10**
0	1268	20925
2	1538	23827
4	9062	28525
6	30775	32193
8	72116	34764
10	138646	36698

Ten Operations/Transaction, Variable Number of Readers

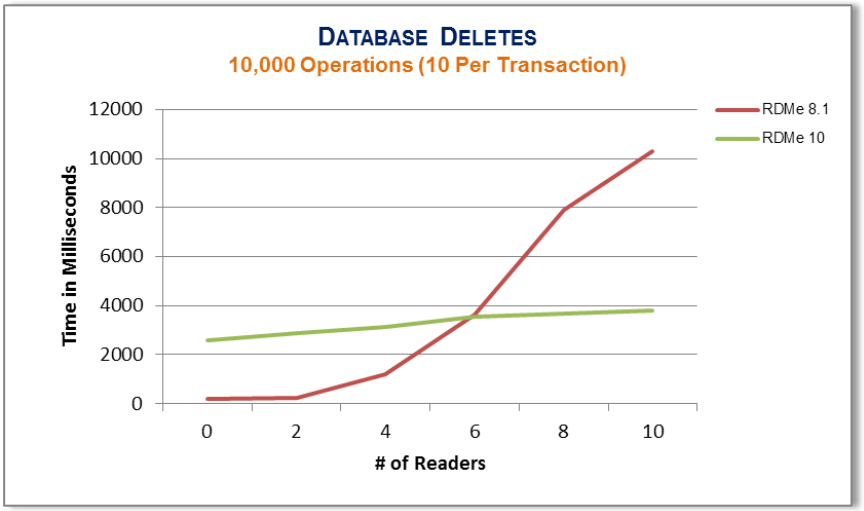
The difference between these results and the ones above is the blocking of ten operations per transaction instead of just one. The basic shape of the charts remains the same.



10,000 operations (10 per transaction) - Inserts		
# Readers	**RDMe 8.1**	**RDMe 10**
0	245	2653
2	338	3057
4	2881	3538
6	15434	5238
8	22258	5525
10	50276	5761



10,000 operations (10 per transaction) -Updates		
# Readers	**RDMe 8.1**	**RDMe 10**
0	319	2558
2	513	3067
4	4401	3732
6	22349	4641
8	34959	5193
10	62638	6089

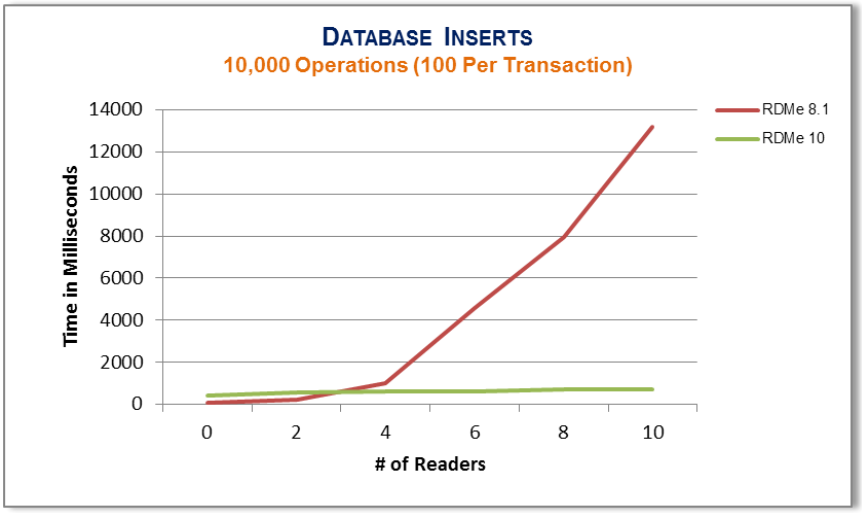


10,000 operations (10 per transaction) - Deletes		
# Readers	**RDMe 8.1**	**RDMe 10**
0	179	2586
2	235	2891
4	1205	3116
6	3648	3526
8	7918	3656
10	10304	3788

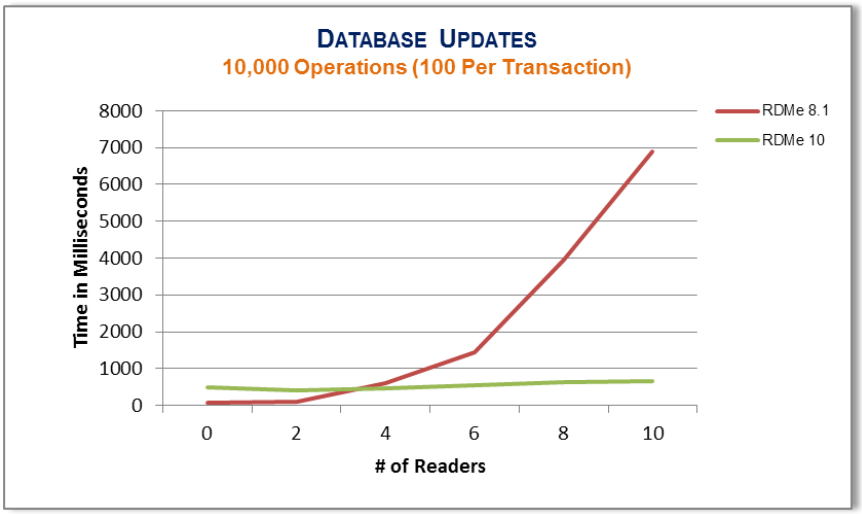
One Hundred Operations/Transaction, Variable Number of Readers

The following results have one hundred inserts, updates and deletes per transaction. While the shape of the chart remains the same, the time per operation decreases dramatically, as is expected with larger blocks.

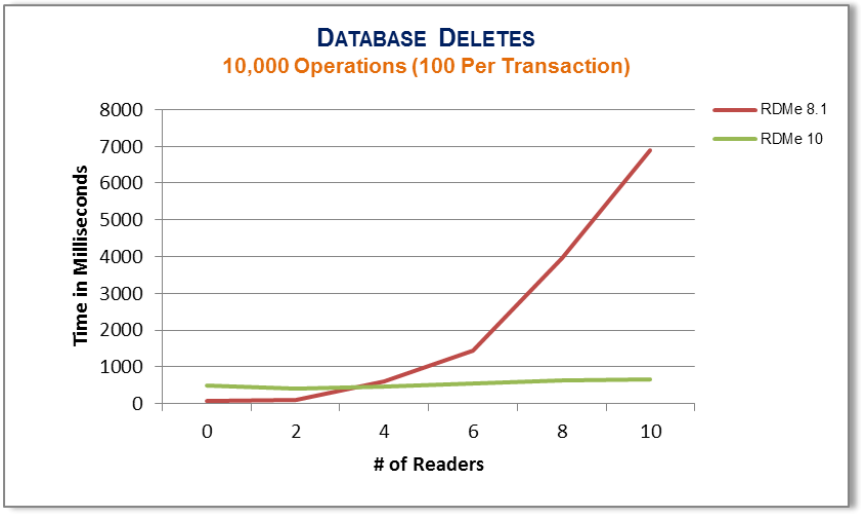
Note that the ability to block operations in transactions depends on the requirements of the application. When blocking one hundred operations in one transaction, none of the operations are visible to other processes until the transaction commits. They may not always be acceptable. One variation of blocking is to begin/end transactions based on the number of operations or a period of time, whichever comes first. For example, a transaction may be told to commit if it performs 100 operations or if 100 milliseconds have passed, and this may satisfy the requirements of a certain type of application.



10,000 operations (100 per transaction) - Inserts		
# Readers	**RDMe 8.1**	**RDMe 10**
0	98	433
2	205	588
4	1004	604
6	4570	631
8	7936	738
10	13205	728



10,000 operations (100 per transaction) -Updates		
# Readers	**RDMe 8.1**	**RDMe 10**
0	269	768
2	305	825
4	1503	958
6	5228	1081
8	10010	1106
10	15505	1211



10,000 operations (100 per transaction) - Deletes		
# Readers	**RDMe 8.1**	**RDMe 10**
0	74	487
2	100	405
4	598	475
6	1433	547
8	3968	631
10	6902	654

Conclusion

RDM Embedded 10.0 has been re-architected from the ground up to support modern hardware and operating system environments. This report demonstrates the circumstances under which RDM Embedded version 10 outperforms version 8.1. With performance improvements in excess of 10 times in high load, high stress testing, RDM Embedded 10 is clearly on the right track for today's scalable performance requirements.

Download our free SDK at www.raima.com/downloads/rdm-embedded and see for yourself how much better RDM Embedded 10 performs.

Contact Information

Website: http://www.raima.com

WORLDWIDE

Raima Inc.
720 Third Avenue, Suite 1100
Seattle, WA 98104
Telephone: +1 206 748 5300
Fax: +1 206 748 5200
E-mail: sales@raima.com

EUROPE

Raima Inc.
Stubbings House, Henley Road
SL6 6QL Maidenhead, United Kingdom
Telephone: +44 1628 826 800
Fax: +44 1628 825 343
E-mail: sales@raima.com