Gaining an Extreme Performance Advantage:

A Benchmark of McObject's *eXtreme*DB In-Memory Database System
& EMC's XtremSF Server-Based PCIe Flash Storage

Financial systems, telecommunications and Cloud-based software-as-a-service (SaaS) are a few of the application types that bump up against the performance limits of disk-based database management system (DBMS) software. In-memory database systems (IMDSs) eliminate much of the latency associated with traditional DBMSs, but some applications require a higher level of data durability (i.e. recoverability if volatile memory is disrupted). As a solution, IMDSs offer transaction logging. But critics object that logging re-introduces the storage-related latency that builds slowness into on-disk DBMSs.

Will an IMDS with transaction logging outperform a traditional DBMS? Will the type of storage used to hold the IMDS's transaction log – hard disk drive vs. state-of-the-art server-based PCIe flash – affect any difference in speed? McObject's benchmark report answers these questions with tests using its own eXtremeDB database system and EMC's XtremSF PCIe flash. It also investigates the hypothesis that flash storage delivers its greatest performance benefit when used with a highly concurrent workload, and examines the impact of database system type on performance at varying levels of concurrency.

McObject LLC
33309 1st Way South, Suite A-208
Federal Way, WA 98092

Phone: 425-888-8505
E-mail: info@mcobject.com
www.mcobject.com

In the past decade, in-memory database systems (IMDSs) have emerged as the database solution for many real-time applications. In-memory databases work with data entirely in main memory, eliminating the file I/O, cache management, data transfer and other overhead that is hard-wired into traditional disk-based database management systems (DBMSs).

Some applications require data durability beyond what RAM-based storage can provide. As a solution, IMDS vendors offer transaction logging, which keeps a record of changes to the database, enabling recovery in the event of system failure. But this logging requires writing to persistent storage. When these "database writes" are reintroduced via transaction logging, do IMDSs retain their performance advantage?

McObject sought the answer to this questions with benchmark tests comparing on-disk database system performance to that of an IMDS with transaction logging. To highlight the effects on performance of today's diverse data storage options, we ran separate tests using hard disk drive (HDD), and high performance server-based PCIe flash, as storage for on-disk database records and for the IMDS transaction log.

McObject's benchmark tests also examined the effect of concurrency on the processing speed of both types of database system described above – IMDS with transaction logging, and traditional on-disk DBMS – when using the high performance server-based PCIe flash, because optimization for highly concurrent workloads is claimed as a benefit of this storage technology. Tests scaling from two to 36 concurrent processes sought to confirm this and examine any interplay with the performance advantage lent by type of database system.

Test hardware and software

The testing platform consisted of the following off-the-shelf elements:

- **Server**. Dell PowerEdge T110 Tower Server with 4GB of 1333 mhz memory. The original equipment hard-disk storage was not used.

- **Hard Disk Drive**. Western Digital VelociRaptor, 600 GB.

- **Server-based PCIe flash storage**. EMC XtremSF™ 2.2 TB MLC installed as a PCIe flash card in the server. Residing in the PCIe interconnect bus enables XtremSF to bypass the overhead of network storage access and alleviate I/O bottlenecks, to maximize read and write performance.

- **In-memory database system**. *eXtreme*DB® IMDS from McObject®.

- **Disk-based DBMS**. McObject's *eXtreme*DB Fusion. With this hybrid DBMS, the developer can combine in-memory and on-disk database storage, or specify entirely persistent

storage. In the tests of on-disk DBMS performance, *eXtreme*DB Fusion implemented the caching, file I/O and other aspects of a disk-based DBMS that affect performance.

Test 1 – Database Operations

Methodology

The test application examined five database operations: record inserts, record updates, record deletes, index searches and table traversals. It measured performance on each operation while looping, with each loop constituting a database transaction and each loop containing at least two instances of the operation, as follows:

Insert records test – 2 record inserts/loop
Update test – 2 record updates/loop
Delete test – 2 record deletes/loop
Index search test – 8 searches/loop
Table traversal test – 8 traversals/loop

The benchmark application recorded the number of loops accomplished per millisecond for each of the two database types (On-disk DBMS and IMDS+TL) with both of the storage devices (HDD and XtremSF).

The *eXtreme*DB database system supports multiple application programming interfaces (APIs) including SQL/ODBC/JDBC and native interfaces for the C/C++, C# (.NET) and Java languages. Code for the operations described above was implemented using the native C/C++ API.

Results

Inserts, Updates & Deletes

The test showed that for insert, update and delete operations, the IMDS performing transaction logging maintained its speed advantage over the traditional on-disk database system. This advantage increased significantly as the IMDS moved from using hard disk to EMC's XtremSF as storage for the transaction log.

Table 1 shows results in loops/ms for each of the configurations, as well as the performance multiple exhibited by each of the IMDS+TL configurations over the baseline on-disk DBMS with hard disk. For example, in the test of database deletes, the IMDS+TL using XtremSF as storage for its transaction log was 21.55 times faster than the on-disk DBMS using HDD.

Inserts	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	1.60	1.00
HDD - IMDS+TL	5.11	3.20
XtremSF - IMDS+TL	29.22	18.27

Updates	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	3.00	1.00
HDD - IMDS+TL	5.32	1.77
XtremSF - IMDS+TL	34.60	11.53

Deletes	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	1.50	1.00
HDD - IMDS+TL	5.31	3.55
XtremSF - IMDS+TL	32.26	21.55

Table 1.

Why is an in-memory database system with transaction logging so much faster than a disk-based DBMS for inserts, updates and deletes? First, on-disk DBMSs cache large amounts of data in memory to avoid disk writes. The algorithms required to manage this cache are a drain on speed and an IMDS (with or without transaction logging) eliminates the caching sub-system.

Second, widely-used b-tree indexes can greatly improve random lookups, and retrieval of database content in sorted order, but they are also expensive for an on-disk DBMS to maintain during insert/update/delete operations, and become more burdensome as database size increases. (B-tree lookups are less costly with an IMDS because they impose no cache processing, accesses happen at in-memory speed, and trees are shallower because they contain no duplicate index data.)

Third, the writes imposed by transaction logging are sequential, i.e. they append to the end of the log file, adjacent to the previous write. DBMSs write to a transaction log file sequentially, too, but whenever pages must be flushed from a DBMS cache, they are written to random locations on disk. Thus the disk head potentially travels far between write locations, adding mechanical overhead during these flushing operations.

Index Searches & Table Traversals

Database index searches and table traversals showed no significant performance change when moving from on-disk DBMS to IMDS+TL, or when changing storage device from hard disk to server-based PCIe flash storage. This result was expected, because such database "reads" are typically much less costly, in performance terms, than inserts, updates and deletes. Because they do not change the database contents, reads impose little overhead, and hence have less to gain from moving from disk to main memory for data storage, or from disk to server-based PCIe flash storage to store the transaction log. The server's ample memory ensured that all or most of the on-disk database was cached, so there were few (if any) cache misses that would cause a read from the database storage device.

Storing the *Entire* Database on XtremSF flash storage

So far this paper has compared storing the entire DBMS on hard disk drive, to scenarios using an in-memory database system with transaction logging enabled, and storing the log to HDD or XtremSF. Another option worth examining is using the on-disk DBMS to store the *entire* database (not just the transaction log) on EMC's flash storage. McObject performed such tests. In Table 2, below, these results are inserted, in red, into the chart shown earlier.

Inserts	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	1.60	1.00
XtremSF - On-disk DBMS	**5.69**	**3.56**
HDD - IMDS+TL	5.11	3.20
XtremSF - IMDS+TL	29.22	18.27

Updates	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	3.00	1.00
XtremSF - On-disk DBMS	**16.17**	**5.39**
HDD - IMDS+TL	5.32	1.77
XtremSF - IMDS+TL	34.60	11.53

Deletes	Loops/ms	Perf. Multiple
HDD - On-disk DBMS	1.50	1.00
XtremSF - On-disk DBMS	**5.69**	**3.80**
HDD - IMDS+TL	5.31	3.55
XtremSF - IMDS+TL	32.26	21.55

Table 2.

Again, “performance multiple” is the speed advantage lent by the combination of a particular database type and storage device over using a conventional on-disk DBMS with “spinning disk” storage. The traditional DBMS performance is markedly better when using the more advanced storage device (XtremSF) instead of HDD. But by far, the highest performance is seen using XtremSF flash storage with the IMDS plus transaction logging.

Test 2 – Effects of Concurrency

The tests discussed above used a single application process to interact with the database system and storage. However, EMC states that XtremSF will show its highest performance advantage with highly concurrent workloads. “Mechanical drives in the storage array have only one or two read/write heads, which means that only a limited number of I/Os can be processed at any one point in time from one disk. So when there are multiple threads in the application trying to access data from the storage array, response times tend to go up because the I/Os need to wait in the queue before they are processed. However, storage and caching devices using Flash technology typically have multiple channels internally that can process multiple I/Os at the same time…Because of this, applications that can utilize multiple threads will yield the greatest performance benefit,” EMC’s product white paper states.

McObject’s second benchmark test sought to confirm this benefit and to observe any additional effect of database system type. If XtremSF performed better with higher concurrency, would replacing the traditional DBMS with IMDS+TL increase that advantage?

Methodology

Test hardware consisted of an HP ProLiant DL380p Generation8 with 16 cores and 384 GB RAM. Database software and XtremSF storage were unchanged from earlier tests. The application used *eXtreme*DB’s SQL ODBC interface to perform the following five SELECT and UPDATE statements per loop:

```
UPDATE accounts SET Abalance=Abalance+value WHERE Aid=value

SELECT Abalance FROM accounts WHERE Aid=value

UPDATE tellers SET Tbalance=Tbalance+value WHERE Tid=value

UPDATE branches SET Bbalance=Bbalance+value WHERE Bid=value

INSERT INTO account_history(Tid, Bid, Aid, delta, htime, filler) VALUES
(value1,value2,value3,value4,now,value5)
```

Each of these statements was implemented as a single, very short transaction (resulting in five transactions per loop). The benchmark measured speed (loops/ms) as the application added processes two at a time and scaled from a total of two processes up to 36 processes. Tests were conducted using both the on-disk DBMS storing the entire database to XtremSF, and the in-memory database system storing its transaction log to XtremSF. Results are shown in Tables 3, 4 and 5, below.

Results

As in the test of individual database operations in the first section of this report, the IMDS+TL and XtremSF combination proved to be fastest, delivering maximum speed of 48.4 loops/ms when using 36 processes. In comparison, at 36 processes, the on-disk DBMS with XtremSF achieved 12.21 loops/ms. Table 3 shows the relative performance of the two solutions as concurrency increased.

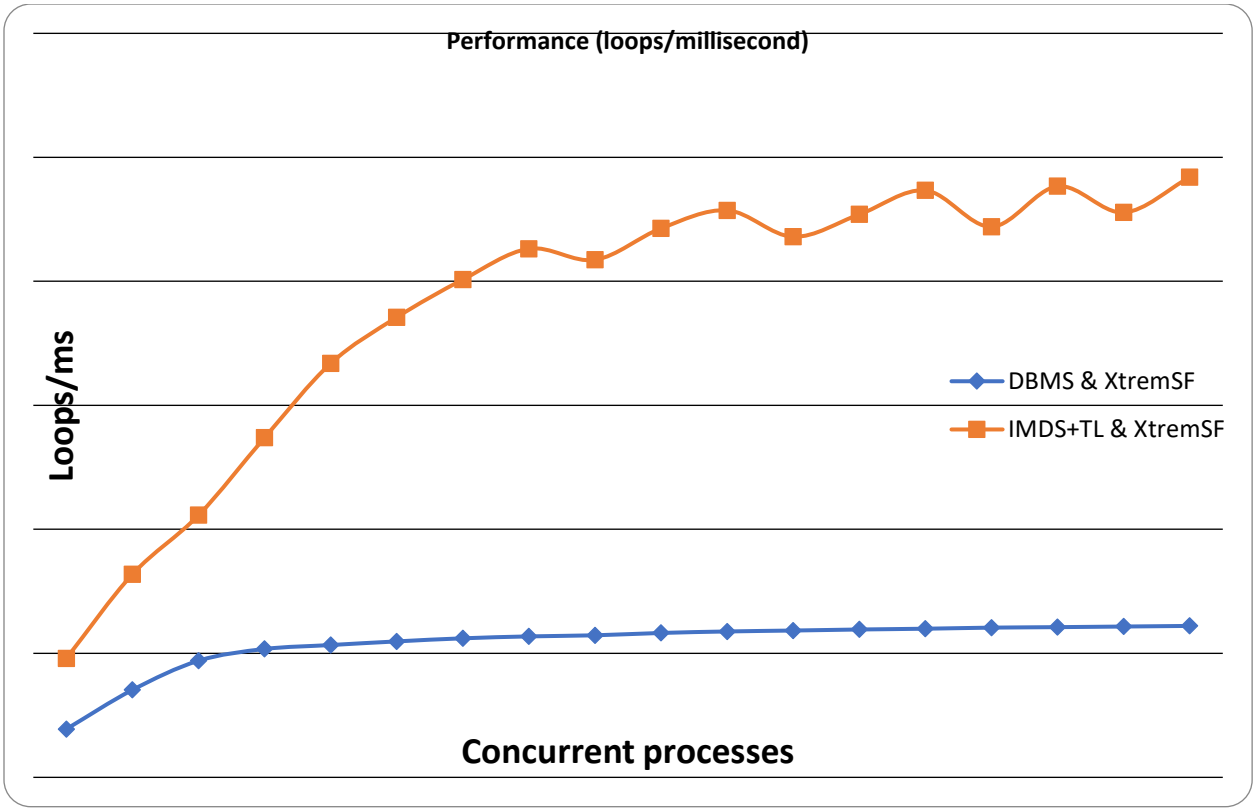

Table 3. Performance of on-disk DBMS & XtremSF (blue line) vs. IMDS+TL & XtremSF (red line) as benchmark application scales from 2 to 36 concurrent processes.

EMC's claim that XtremSF delivers maximum benefit with a highly concurrent workload is clearly supported by the results in Tables 4 and 5. In the tests using the on-disk DBMS and XtremSF, for example, four processes delivered 1.81x times the performance of using two processes; moving to 22 processes more than tripled the performance of two processes.

DBMS w/ XtremSF Storage

Processes	loops/ms	Perf. Multiple
2	3.89	
4	7.05	1.81
6	9.40	2.41
8	10.35	2.66
10	10.67	2.74
12	10.96	2.82
14	11.21	2.88
16	11.37	2.92
18	11.46	2.94
20	11.64	2.99
22	11.77	3.02
24	11.84	3.04
26	11.92	3.06
28	11.99	3.08
30	12.07	3.10
32	12.11	3.11
34	12.16	3.12
36	12.21	3.14

Table 4. Performance benefit gained by adding concurrent processes when using on-disk DBMS and XtremSF storage. The third column shows the performance multiple gained by using X processes, compared to using 2 processes (e.g. 28 processes is 3.08 times faster than 2 processes).

IMDS+TL w/ XtremSF Storage

Processes	loops/ms	Perf. Multiple
2	9.59	
4	16.38	1.71
6	21.14	2.20
8	27.39	2.86
10	33.38	3.48
12	37.10	3.87
14	40.14	4.18
16	42.63	4.44
18	41.74	4.35
20	44.27	4.62
22	45.73	4.77
24	43.59	4.54
26	45.40	4.73
28	47.33	4.93
30	44.40	4.63
32	47.66	4.97
34	45.55	4.75
36	48.40	5.05

Table 5. Performance benefit gained by adding concurrent processes when using an in-memory database system with transaction logging active and the log stored to XtremSF. The third column shows the performance multiple gained by using X processes, compared to using 2 processes (e.g. 28 processes is 4.93 times faster than 2 processes).

The results also show that choice of database system (IMDS+TL vs. DBMS) enhances the performance benefit of higher concurrency. Not only did the IMDS+TL outperform the on-disk DBMS on an absolute basis (with greater speed at any number of concurrent processes), but the advantage widened as concurrency increased. At 36 processes, this performance multiple peaked at 5.05x for IMDS+TL, and at 3.14x for the DBMS. These findings confirm that the strengths of an in-memory database system with transaction logging play into those of server-based PCIe flash storage, resulting in a durable, high performance data management solution that is "greater than the sum of its parts."

It should be noted that these tests of concurrency portray *eXtreme*DB's and XtremSF's potential conservatively. First, the test server has 16 cores, so any processes added after the 16th must contend for a time slice on a core. Adding cores would accelerate performance at the higher levels of concurrency. Second, each loop in test application includes a database read operation (SELECT) in addition to four database writes. As discussed previously, reads impose less latency

than writes. A test loop consisting entirely of writes would impose more latency, and of a type that the IMDS+TL can ameliorate. This would result in a higher performance multiple for the IMDS+TL over the DBMS.

Finally, while SQL ODBC is widely used, it carries higher overhead than the C/C++ database API used in the first round of tests. The need to parse and optimize SQL statements imposes proportionately more latency than a C/C++ API, of a type that is not directly addressed via IMDS architecture. Instead, this kind of latency is solved by moving to an interface that is inherently faster. As they say, "your results may vary" – but in this test the outcome would likely be even more impressive if the server hardware and database API were more optimal.

Conclusion

The first and most obvious conclusion from these tests is that if an application demands both fast database processing and data durability, and is expected to modify the database (i.e. the software is not read-only), then an in-memory database system with transaction logging outperforms an on-disk DBMS. Equally clear is that type of storage device matters greatly. In the test implemented using the database system's C/C++ interface (a more likely choice of tool than SQL in high performance applications), the IMDS+TL/hard drive combination accelerated processing by as much as 3.55x over the traditional on-disk DBMS with HDD, but substituting EMC's XtremSF for the IMDS's transaction log storage pushed its speed advantage to as high as 21.55x over DBMS+HDD. The second test confirmed that server-based PCIe flash storage delivers its greatest benefit with a highly concurrent workload, and that an in-memory database system works hand-in-glove with that storage to accelerate performance as concurrency rises. Given the performance documented in these tests, likely uses for IMDS+TL and server-based PCIe flash storage include applications for capital markets (e.g. risk management, automated trading and order execution), software-as-a-service (SaaS), real-time ad targeting and telecommunications. Other scenarios that demand maximum data management speed and high concurrency along with data durability/recoverability could be added to the list.