Last Transaction Safety on MongoDB with-out compromising on Performance

In this white paper, two pioneers collaborate to bring NoSQL to a new level:

- **MongoDB** is a leader in operational database management systems, and specifically in document-based NoSQL. MongoDB has recently released several key performance optimizations within major version 3 and its newly acquired WiredTiger storage engine.
- **Plexistor** is a pioneer in software for non-volatile memory, designed from the ground up to marry the performance of memory with the capacity and cost of FLASH. This architecture was coined as Software Defined Memory (SDM). Plexistor's SDM is available for download.

Optimized MongoDB Performance at Maximal Durability

Executive Summary

Traditionally, MongoDB users have had to sacrifice performance for assuring that the data has been written to a persistent media. Plexistor's Software Defined Memory (SDM) accelerates performance for MongoDB by liberating it from the overhead of the ordinary Linux operating system's I/O stack and the constraints of decades-old conventional storage architectures which are overdue to be replaced. Through the revolutionary approach of Plexistor's SDM and NVDIMM-N memory cards, MongoDB performance no longer must be sacrificed to ensure data persistency or durability.

As demonstrated in the benchmarks discussed throughout this paper, Plexistor's SDM software accelerates MongoDB performance by 450% operations per second while cutting down their latency by an order of magnitude without putting any data at risk (i.e., with maximal durability).

Table of content:

Introduction

Traditionally, there has always been a tradeoff between two desired properties: performance and the safety of the data being written. Data safety is accomplished by means of synchronous Writes associated with all updates. For example, a fast update to the in-memory image of the working data set will still have to wait for confirmation from one or more other destinations where the update is also recorded to persistent storage. These wait times detract from performance, and the penalty is increased when the storage is inefficient or fundamentally slow. Therefore, the tradeoff between performance and data persistence is a common characteristic affecting both NoSQL in general and MongoDB in particular.

To accommodate this tradeoff, MongoDB supports a range of options:

- MongoDB clients can, per I/O, decide to wait for an acknowledgement from the server, and if that acknowledgement should be after the inserted/updated data is persistent or not;
- MongoDB servers can be configured with parameters set for different levels of data durability, where the increased levels of risk correspond to hypothetically better performance.

United Software Associates wrote a benchmarking paper in March 2015 which quantified this tradeoff across several NoSQL databases ("High Performance Benchmarking: MongoDB and NoSQL Systems")[1]. The main optimization factors that defined traditional performance and durability tradeoff were:

(**T**) "Throughput Optimized" (all data written since last checkpoint is at risk);
(**B**) "Balanced" (some data loss [~100 MB] is risked);
(**D**) "Durability Optimized" (no risk of any data loss whatsoever).

[1] The United Software Associates paper has no reference number, but is available for download on: https://www.mongodb.com/collateral/comparative-benchmarks-mongodb-vs-couchbase-vs-cassandra

United Software Associates denote that configuration (**T**) is considered to be a level of risk much higher than a typical commercial enterprise customer would be likely to commonly implement in production environments. So they focused on configurations (**B**) and (**D**). For the purposes of making an apples-to-apples comparison, we adopted the same classification and focus. We re-ran the experiment in a similar environment and reached similar results and conclusions when we used Flash storage devices and traditional storage software.

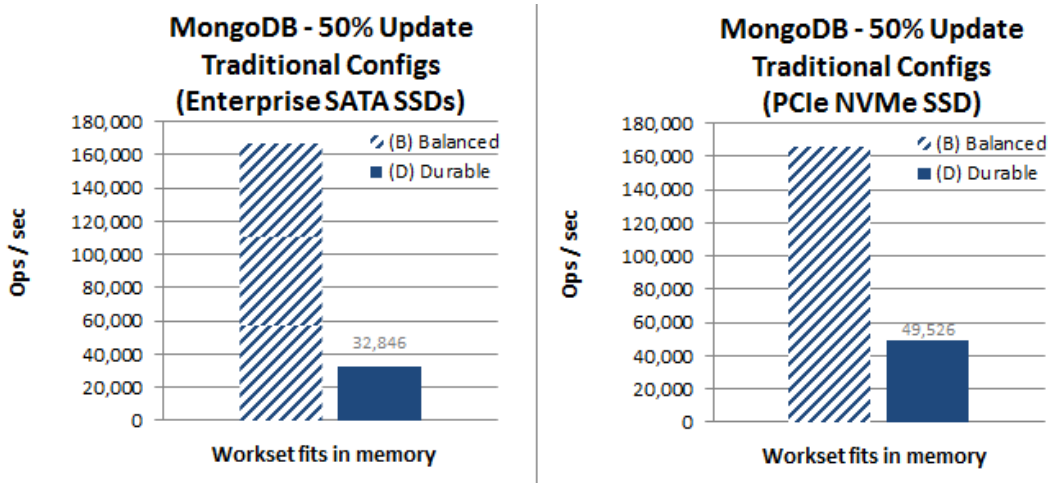

Figure 1 *(A) Enterprise SATA SSD configuration* *(B) PCIe NVMe SSD configuration*

Figures 1(A) and 1(B) illustrate clearly that there is a clear tradeoff between performance and durability when using traditional storage software. These results demonstrate why many database administrators and application developers strategize recovery methods for "tolerable" amounts of data loss, because doing so enables urgently needed performance improvements.

In Figure 1(A), the "Balanced" mode configuration performed 5.1 times better than the "Durability Optimized" configuration when enterprise-grade[2] SATA solid-state drives (SSD) were used with traditional storage software. In Figure 1(B), this ratio is improved, but performance is still 3.4 times greater for Balanced mode, even when the SSD storage hardware was replaced with a recently-released 2nd generation Intel P3608 NVMe SSD.

To resolve these customer challenges, Plexistor has introduced a software-defined memory architecture. We leveraged the previous[1] benchmarking methodology using SDM and have shown that the traditional trade-off between performance and durability is no longer significant. As a result, database administrators can eliminate the risk of losing non-persistent data, and application developers no longer have to accept "tolerable data loss" in exchange for higher performance and can focus on the business logic.

Background on Software Defined Memory

[2] We also measured consumer-grade SSDs (not shown), and the performance gap is an order of magnitude worse.

In the last decade FLASH-based solid-state drives (SSDs) have been widely adopted for accelerating latency-sensitive applications. Recently, new device types were introduced that are orders of magnitude faster and run at memory or at near-memory speeds.

NVDIMM devices reside on the memory interconnect which is designed for low latency at high throughput. NVDIMM-N are NVDIMM devices that are fully mapped to the memory address space and can be accessed at cache-line granularity, unlike SSD devices which require their I/O events to be organized into larger blocks. Also, NVDIMM-N devices respond at near-memory speed, which makes the traditional operating system practice of caching data in main memory redundant.

NVDIMMs are faster, but also more expensive, than FLASH. Building a cost-effective solution requires mixing storage media types. Plexistor's SDM, as shown in Figure 2, is designed from the ground up to marry the performance of persistent memory (PM, e.g. NVDIMM-N) with the capacity of FLASH devices. The software was designed to implement this capability transparently at an architectural level, such that no changes to applications would be required to harness these advantages.

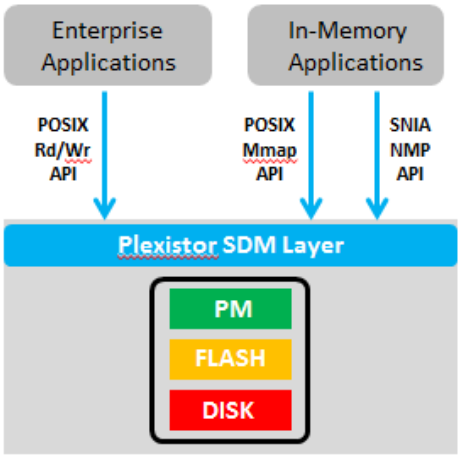

Figure 2 Plexistor SDM – a unified data layer for both POSIX[3] and SNIA NMP1.1[4] models

Figure 2 further illustrates that Plexistor SDM turns off-the-shelf servers into hyper-performing compute, network, and storage and brings together In-memory and traditional applications into single storage architecture.

SDM eliminates the redundant block abstraction layer and collapses the multiple storage software layers into a single layer, which is tailored for modern multi-core processors with available persistent memory. It further cuts latency and saves resources by allowing users to directly access the storage media without creating an additional copy of the data in volatile memory. Most importantly, with SDM and NVDIMMs, write accesses are immediately made persistent, so application developers do not have to consider the historical storage tradeoffs.

[3] POSIX stands for Portable Operating System Interface and is IEEE standards for defining (among other things) traditional storage system calls and APIs.
[4] NPM is SNIA NVM Programming Model, currently on version 1.1: www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.1.pdf

Consider the contrast between complexity and simplicity illustrated in Figure 3 below, comparing the numerous layers involved in handling I/O for Linux versus Plexistor's streamlined single layer. Plexistor's SDM eliminates redundant software caching, enables byte-level addressing, and implements persistence in memory. The net result simplifies application development and delivers unsurpassed performance without any tradeoffs in data persistency.

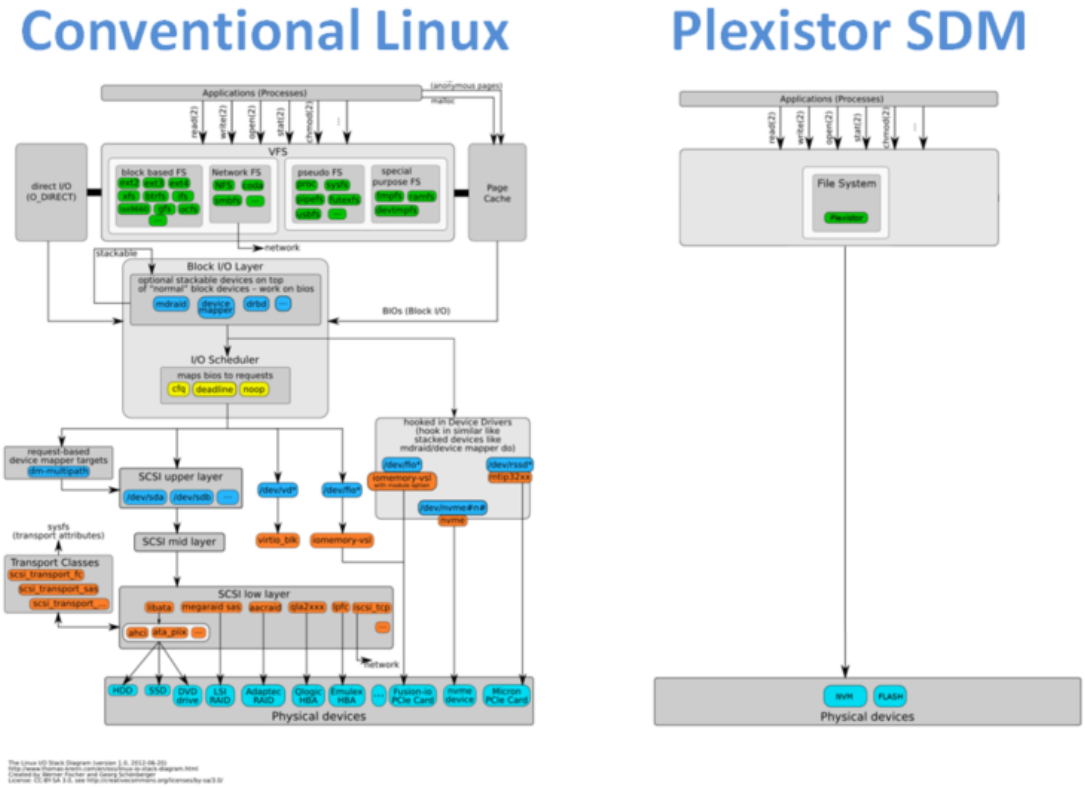

Figure 3 Linux conventional I/O Stack versus Plexistor SDM*
(Example based on Linux kernel 3.3)*

Benchmarking Methodology

The methodology used follows the United Software Associates in their NoSQL (and MongoDB) performance and durability analysis[1]. As depicted in Figure 4, this includes the following.

- Use of "Yahoo! Cloud Serving Benchmark" (YCSB) tool to generate the load, while using the database-specific branch of the YCSB client.
- Set the number of clients that provide maximal throughput without increasing latency.
- Balanced and durable client requests.
- MongoDB version 3.2 with WiredTiger storage engine and separated mount points for data and journal.
- Disabled transparent huge pages and use of Numactl interleave.

- Active work set that fits within the memory size. We used a smaller RAM size and thus proportionally scaled down the number of records and operations as well (from 20M to 15M). This should have no effect on the conclusions.
- Two physical machines connected with 10G Ethernet. The hardware and platform specification used for the MongoDB server is depicted in Table 1.

Figure 4 The two physical machines used for MongoDB client and server

Below are example command lines for running the benchmarks regardless of whether that data landed on XFS or Plexistor SDM:

MongoDB server:

```
numactl --interleave=all
mongod --dbpath=/mnt/dev0 --journal --storageEngine=wiredTiger
  --wiredTigerJournalCompressor=none
```
(This flag had negligible effect)

YCSB client:

```
/opt/ycsb/10gen/ycsb-mongodb/bin/ycsb run mongodb -P <workload>
 -s -threads 40
  -p recordcount=<recordcount>   -p operationcount=<opcount>
   -p mongodb.url=…:27017 -p mongodb.database=ycsb
    -p mongodb.writeConcern= journaled or acknowledged
```
(for the (D) or (B) configurations respectively)

	Baseline	Plexistor SDM
Compute	RHEL 7.1 Linux running on a dual socket XEON E5-2650 v3	
Storage	XFS using: 1. 64GB DDR4 DIMM (Linux Page Cache) 2. SanDisk CloudSpeed SATA SSD	Plexistor's SDM using 1. 64GB[5] DDR4 NVDIMM-N 2. SanDisk CloudSpeed SATA SSD

Table 1 Hardware and platform configurations of the MongoDB server

Results

[5] We used 56GB for data and 8GB for journal, which may be further optimized.

Performance of maximal durability configurations were evaluated using three types of write-intensive workloads:

1. YCSB Workload A — a balanced 50% read, 50% update workload;
2. YCSB Workload F — a balanced 50% read , 50% read-modify-write workload;
3. Insert-Mostly Workload — a write-biased 95% insert workload.

Performance results are compared across four storage-related configurations. These include both throughput (operations per second) and latency.

Workload A ("50% Update" workload)

As shown in Figure 5, all MongoDB operations are fully durable (D). Throughput (operations per second) with Plexistor's SDM was over 4.5 times better than the conventional XFS stack.

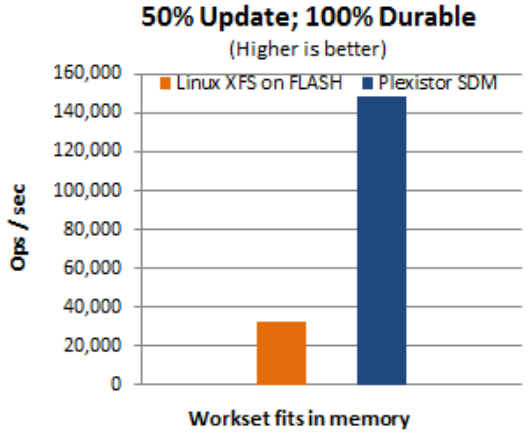

Figure 5 Workload A – Throughput

Comparing the results presented in figures 1 and 5 reveals that Plexistor's SDM fully durable (D) results are within 10% of the Balanced configuration (B) results, thus eliminating the risk of losing up to 100MB of data for Balanced configuration.

The huge throughput improvements achieved with SDM are matched with even greater latency reduction. Figure 6 reveals that despite serving more operations per second per server, SDM improved operation latency by 6.4 – 8.8 times compared to the conventional stack.

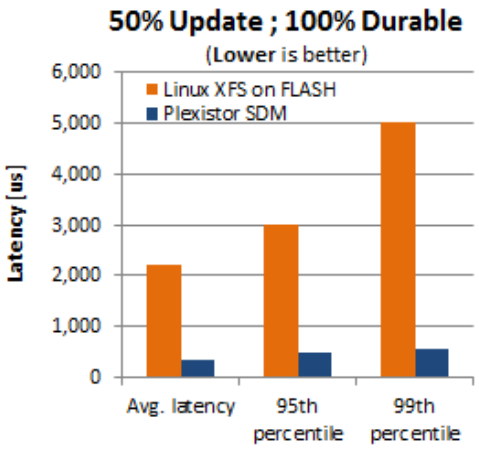

Figure 6 Workload A – Latency[6]

With three times throughput improvement and seven times reduced latency, SDM enables high-performance for fully-durable MongoDB-based applications. Today, these results include many redundant operations such as saving the data and MongoDB's journal in a persistent manner. This redundancy derives from MongoDB using a storage engine that was designed for traditional storage I/O stacks. Clearly, since the data is already persistent on the NVDIMMs, making an extra copy to another persistent storage location is not necessary. Future optimizations to MongoDB's storage engine have the potential for further improving the results presented here (refer to the Discussion section, below).

Workload F ("50% Read-Modify-Write" workload)

Similar to Workload A, with Workload F, all MongoDB configurations have been optimized for durability. Therefore, writes are acknowledged only after they are persistent, and doing so eliminates any possibility of data loss.

Figure 7 shows that for workload F, 50% read-modify-write, MongoDB on Plexistor's SDM delivers 3.8 times more throughput, while reducing the latency by a factor of 7.2 – 9.1, compared to the conventional software stack.

[6] The YCSB tool reported latency for Plexistor's SDM as "0ms" because it was not designed to measure latencies below 1000µs. Thus, the SDM results shown in Figure 6 for 95th and 99th percentile are estimated.

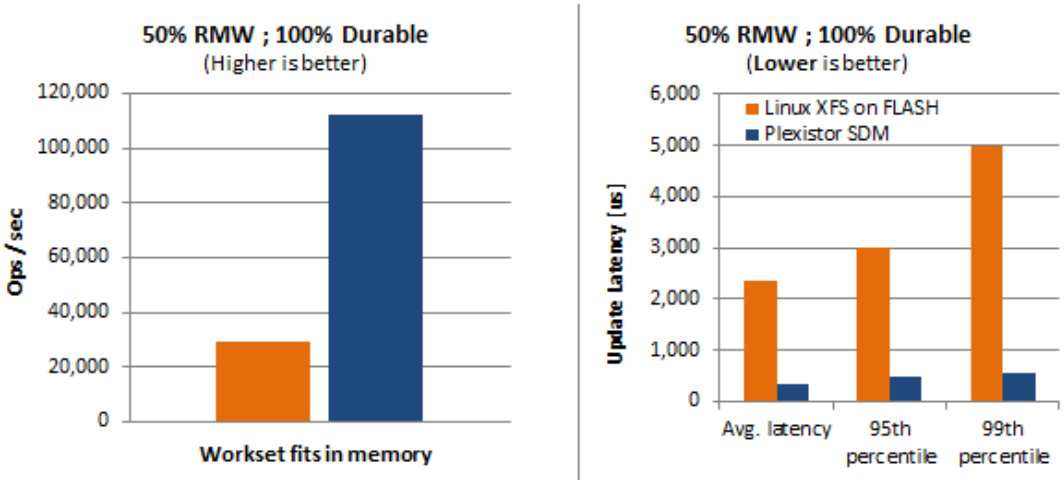

Figure 7 Workload F (RMW) – Throughput and Latency

Insert-Mostly Workload ("95% Insert" workload)

Thus far workloads have mimicked balanced applications. Many applications however are insert intensive. Figure 8 shows that for the Insert-Mostly Workload ("95% Insert"), MongoDB using Plexistor's SDM achieves over 3 times better throughput and latency compared to the baseline.

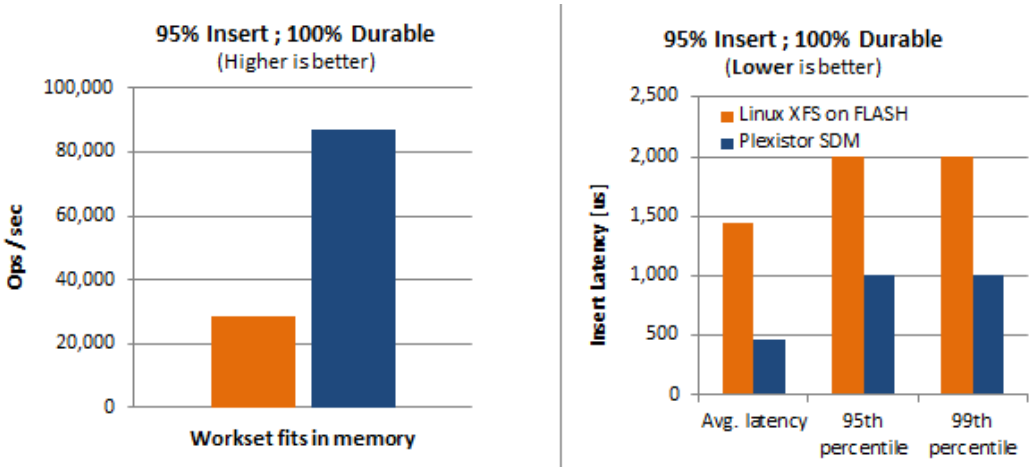

Figure 8 Insert-intensive workload – Throughput and latency

Discussion – Further Speedup Potential

The test methodology used in this paper focused on showing the value of SDM “out of the box”, using legacy storage engine code and configurations.

MongoDB’s storage engine, WiredTiger, was optimized for legacy storage. As such, its code and configuration inherently reflects outdated assumptions that have become obsolete due to SDM. Data writes, for instance, are assumed to be buffered in volatile memory because storage is expected to be orders of magnitude slower than memory — and therefore a second copy of the data (i.e., the MongoDB journal) is maintained for persistency. However, with SDM there is no longer a requirement for such redundant “safety net” writing of data to slow media. In the future, if MongoDB is modified to allow the elimination of journaling when SDM is used, we estimate that there is an additional 19% speedup available (beyond what was presented in Figure 5). Nevertheless, the assumption that a journal is required for production environments is so fundamental that a careful review of the storage engine is required to certify such a durable configuration.

Another opportunity for additional speedup derives from the low CPU utilization observed. The processor was measured to be mostly in the idle state while the benchmarks were running. This may be derived from inefficiencies inherent to asynchronous I/O access patterns, but it will require further investigation to understand completely. Even if such access patterns have been historically important for conventional high-latency storage, they are certainly redundant and counter-productive in the context of SDM.

Should SDM-optimized versions of MongoDB storage engine become available, then it will be of great interest to revisit these tests at that time, to test the hypotheses above with regard to additional performance enhancement opportunities.

Summary

In conclusion, the test results presented in this paper demonstrate unequivocally that Plexistor's SDM makes it possible for MongoDB to achieve maximal durability with no performance compromise. As shown in Figure 8, this is clearly true even for the most write-intensive workloads, and requires no modification to MongoDB or its storage engine.

MongoDB administrators and application developers who have become accustomed to reluctantly risking "some" data loss in their pursuit for improved performance now with modern server using Plexistor's SDM have a viable alternative to deliver excellent performance without sacrificing data durability. This is only the beginning. Further data flow optimizations may unleash even greater potential via SDM and significantly accelerate MongoDB beyond the "as is" configurations presented in this paper.