Hathi: Durable Transactions for Memory using Flash

Mohit Saxena
U. Wisconsin-Madison

Mehul A. Shah, Stavros Harizopoulos
Nou Data

Michael M. Swift
U. Wisconsin-Madison

Arif Merchant
Google

We describe Hathi*, a lightweight, high-performance and ACID-compliant transactional store based on* durable memory transactions*.*

1 Introduction

Transactions serve two purposes that make it easier to build robust applications. First, transactions enable fine-grained concurrency control, allowing developers to scale applications more easily across multiple processors [1, 6]. Second, transactions provide a simple interface for managing the durability and consistency of application state in the face of failures [4]. However, the commonly used transactional stores – DBMSs or distributed transaction systems – are a poor fit for many modern workloads, mainly due to the cost and complexity of managing such systems even for lightweight operations. As a result, many applications, particularly web services, sacrifice strong consistency for simpler storage models, such as key-value stores [3].

Hathi is a lightweight, high-performance and ACID-compliant transactional store based on *durable memory transactions*: a program updates an in-memory data structure that is persistent and consistent across failures. We leverage three recent architectural trends in the design. First, DRAM prices have dropped to a point that even mid-tier servers support up to 4 TB of memory. At these sizes, many workloads can execute in core rather than from disk — an observation also made by others [5, 8]. Second, power considerations have driven processor manufacturers away from uni-processors towards multi-core chips, so concurrency between threads becomes a key concern. Third, flash-based solid-state drives (SSDs) provide scalable bandwidth and 1-2 orders of magnitude lower latency than the fastest disks.

Hathi presents an in-memory transactional heap interface that is automatically made persistent on fast SSDs. Thus, programs can create and manipulate in-memory data structures, but ensure that the data is durable with little extra effort. To do so, Hathi combines the simple and highly concurrent interface ("ACI") of transactional memory [6] with an SSD-optimized write-ahead logging and checkpointing scheme for durability ("D"). Thus, a programmer can wrap a section of program in a transaction to make updates durable and consistent. As flash storage is fast but still much slower than memory, Hathi implements two approaches to reduce and eliminate the overhead of persistence.

First, Hathi provides options at transaction commit to control whether the program blocks, similar to asynchronous file I/O. This control has not previously been available for memory transactions, and allows developers to leverage application knowledge for increased performance. Specifically, Hathi exposes *split-phase* commit, which decouples the installation of in-memory updates from the flush of transaction log records. Using this interface, applications can continue with other tasks and later check for completion of the commit, thereby overlapping computation and commit I/O.

Second, Hathi uses partitioned logging, in which each thread maintains a separate log. Partitioned logging leverages the SSD's internal parallelism and avoids contention on in-memory log buffers that hold the tail of the transaction log. For full consistency, Hathi ensures that all preceding transactions from all threads are durable before marking a transaction as durable. However, Hathi also provides *partitioned commit*, which allows application threads to commit transactions that operate on independent data structures without coordinating with the logs from other threads.

Finally, Hathi provides incremental checkpointing in small chunks of the memory heap and ensures consistency with concurrent transactions, without the need to pause execution because both chunks and log records share the same log sequence number space. After a crash, Hathi performs recovery by first loading the checkpoint in memory and then replaying the logs. It merges the log records from the different log partitions and then replays

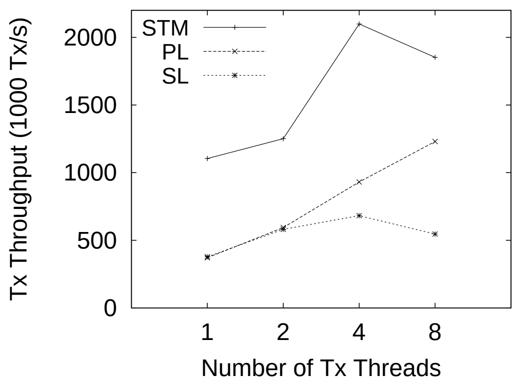

Figure 1: **Durability Cost**: Comparing software transactional memory (STM), Hathi with single (SL) and partitioned logs (PL).

them in log sequence order.

With these new interfaces and mechanisms, we find that Hathi provides significant improvement in transaction throughput over the traditional mechanisms for providing durability and the synchronous commit interface. At these speeds, we believe Hathi is suitable for building not only user-facing applications, but also infrastructure applications like distributed file systems, key-value stores, massively multi-player online games and social-network graphs.

2 Results

We evaluate Hathi to measure the cost of durability and the value of partitioned logging. We use two setups in these experiments. To emulate a high-end system, we use a 3.0 GHz Intel Xeon HP Proliant dual quad-core server with 8 GB DRAM and a 80 GB PCIe FusionIO ioDrive. It runs different transaction threads that copy six memory words at random offsets within the heap to minimize the cost of STM contention and only measure the cost of durability. To emulate a mainstream system, we use a 2.5 GHz Intel Core 2 quad with 1 GB heap and a consumer-grade SSD, an Intel X-25M. It runs the travel reservation workload (vacation) from STAMP transactional memory benchmark suite [7] that we ported to Hathi.

Figure 1 shows that Hathi reaches 1.25 million txns/sec on a high-end FusionIO drive on the high-end system. Similarly, on the mainstream setup, Hathi achieves nearly 200 K txns/sec on the consumer-grade Intel X-25M SSD. This is less than 38% and 15% short of the peak STM throughput respectively. With the use of separate logging threads Hathi could eliminate blocking and further improve the CPU and I/O utilization, and potentially reaching the same throughput as the STM with durability. Thus, Hathi provides durable transactions at little additional cost over non-durable transactional memory. These results also indicate that the added cost of durability with Hathi is low for these workloads. As a comparison, recent work on persistent memory using future projections of phase-change memory performance achieved only 1.3-1.6 M txns/sec with 4 cores [2, 9], not much better than Hathi in the high-end configuration.

Figure 1 also compares the transaction throughput for single log (SL) and partitioned log (PL) on the FusionIO device. At best, SL achieves only 45% the performance of PL, with only 20% CPU utilization and 10% of peak FusionIO bandwidth. Thus, the single log is clearly the bottleneck. Single log performance degrades after 4 threads because of write serialization to ensure sequential I/O. In contrast, as we increase the number of threads and log partitions with PL, the throughput increases almost linearly. We note that partitioned logging depends on parallelism and low access latencies of the storage device, as the seek costs in a disk would make partitioned logging more expensive.

References

[1] Aguilera, M. K., Merchant, A., Shah, M. A., Veitch, A. C., and Karamanolis, C. T. Sinfonia: a new paradigm for building scalable distributed systems. In *SOSP* (2007).

[2] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson, S. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *ASPLOS* (2011).

[3] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: amazon's highly available key-value store. In *SOSP* (2007).

[4] Gray, J. The transaction concept: Virtues and limitations. In *VLDB* (1981).

[5] Kallman, R., et al. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow. 1*, 2 (2008), 1496–1499.

[6] Larus, J. R., and Rajwar, R. *Transactional Memory*. Morgan & Claypool Publishers, 2006.

[7] Minh, C. C., Chung, J., Kozyrakis, C., and Olukotun, K. STAMP: Stanford transactional applications for multi-processing. In *IISWC* (2008).

[8] Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., Rumble, S. M., Stratmann, E., and Stutsman, R. The case for ramclouds: scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev. 43* (January 2010), 92–105.

[9] Volos, H., Tack, A. J., and Swift, M. M. Mnemosyne: Lightweight persistent memory. In *ASPLOS* (2011).