DRAMPersist: Making DRAM Systems Persistent

Krishna T. Malladi Manu Awasthi Hongzhong Zheng
Samsung Semiconductor, Inc.
{k.tej,manu.awasthi,hz.zheng}@samsung.com

ABSTRACT

Modern applications exercise main memory systems in different ways. A lot of scale-out, in-memory applications exploit a number of desirable properties provided by DRAM such as high capacity, low latency and high bandwidth. Although DRAM technology continues to scale aggressively, new resistive memory technologies are on the horizon, promising scalability, density and non-volatility. However, they still suffer from longer, asymmetric read-write latencies and have lower endurance as compared to DRAM. Considering these factors, scale-out, distributed applications will benefit greatly from main memory architectures that provide the non-volatility of new memory technologies, but still have DRAM-like latencies. To that end, we introduce DRAMPersist – a novel mechanism to make main memory persistent and complement existing high speed storage, specifically geared for scale-out systems.

CCS Concepts

•Computer systems organization → Processors and memory architectures; •Hardware → Dynamic memory; Non-volatile memory;

Keywords

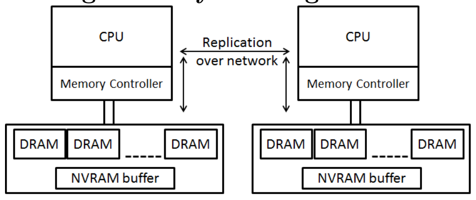
DRAM, NVRAM, Persistent, Replication, Systems

1. INTRODUCTION

Modern datacenters and cloud services have seen massive data growth driven by online services such as web crawling and indexing, generation of application and machine logs, and more recently, sensor data collected from a multitude of devices. Applications processing this data often cache (Ex: Memcached, REDIS) or completely store (Apache Spark, SAP HANA, Oracle in-Memory) datasets in DRAM to meet tight latency requirements from the end user. Multiple prior studies have shown that such applications are bound by memory capacity and I/O [1, 4]. Thus, providing large memory capacity with low latency plays a critical role in building efficient server platforms for scale-out applications [5, 6].

MEMSYS 2016 October 3–6, 2016, Washington, DC, USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4305-3.
DOI: http://dx.doi.org/10.1145/2989081.2989110

Figure 1: System organization



DRAM continues to scale to sub-20nm technology to efficiently enable high-density, low power DRAM chips for high performance memory subsystems [3]. For example, modern DRAM modules can pack as high as 128 GB, often with die-stacked memory chips [7, 2]. However, the scale of data growth has outpaced technology, compelling applications to adapt accordingly. For example, many *in-memory* frameworks still optimize for disk IO to ensure fast data access and analytics.

Currently, most in-memory scale-out platforms have additional, in-built mechanisms to flush DRAM data to disk to maintain data persistence. Furthermore, scale-out applications maintain replicas of data across multiple machines to maintain high availability and achieve load balancing across nodes. Such applications will benefit immensely from an architecture that has (DRAM-like) low latencies, but do not need to frequently flush data to disk. To achieve this, we present DRAMPersist, a persistent memory architecture geared towards scale-out systems.

2. DRAMPERSIST ARCHITECTURE

In this section, we describe the central ideas to build a persistent DRAM tier. We build DRAMPersist based on the observation that inter-node data replication is a stronger guarantee of data availability than a single node data persistence. Most scale out systems, by default, assume system nodes are not reliable and replicate datasets to handle straggler nodes and catastrophic node losses. In addition, this allows for better system scalability, access latency and bandwidth. We reuse this system aspect in order to provide high speed persistent storage, using mostly DRAM, with a little NVRAM support.

System Design The system implementation is shown in Figure 1. The key concept is to develop a DIMM module that has both DRAM and a non-volatile buffer backed up by a super-capacitor. This is fundamentally different from the NVDIMM approach which aims to make the entire DIMM battery-backed. We observe that highly available systems in datacenters almost always use data replication to recover from single system failures. In our current implementation, we propose to provide data persistence only through inter-node replication. However, since there is non-zero network latency, there will be a small time window when the data on a single node is not durable, thus blocking the storage from issuing commit acknowledgments to the host system. Our proposal addresses this problem by coupling the DRAM module with a small NVRAM buffer that is sufficient to hold data transactions in-flight for inter-node replication. The replication in the remote system can also be provided with a similar DRAM module, improving system performance.

Read-Write path Consider the I/O write path for scale-out applications. Data writes to the storage layer are first committed to the NVRAM buffer, shown in Figure 2 and to DRAM, both mapped to the storage address space. The corresponding block's valid flag in NVRAM is set. The host driver then issues either a TCP/IP or an RDMA request to copy the data to a memory buffer on a different node on the network. The remote node then sends an acknowledgment in TCP/IP case, that the host uses to mark the transaction complete. The valid bit of NVRAM buffer is then unset, releasing the space to other entries. In RDMA case, the host system polls the queue completion status on the remote node. Priority bits help in replicating critical data first during high storage traffic. Re-writes to the same logical block addresses are updated in both DRAM and NVRAM buffer before being sent to the remote node. Read requests to blocks with both completed and pending remote acknowledgments are served from the local DRAM/NVRAM buffer.

Recovery mechanism In the event of power failure, the local DRAM could potentially lose data. While this node recovers, reads to this data from other systems are served from the remote node(s) with duplicate data. This is similar to standard recovery mechanisms present in distributed systems. Upon recovery, local reads are still served from the remote node, but begin to be cached in the local DRAM, on a per-block demand basis to aid data recovery. In the event of a power failure when data replication acknowledgment is pending, the transaction is secure in the NVRAM buffer and can be flushed to storage. Upon recovery, the data is restored in a mechanism similar to a normal recovery.

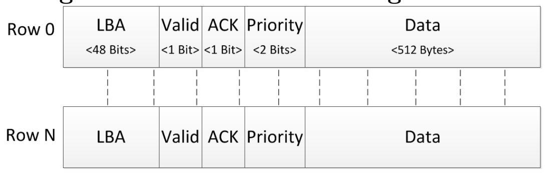
NVRAM buffer size Sizing the NVRAM buffer optimally is important for performance. A small size could limit the number of outstanding entries, while a large buffer increases the cost of the battery or capacitor. To navigate this tradeoff, we provision the buffer with a size that is dependent on network latency. For example, consider a system with a network round trip time of 50us for TCP/IP and performance guarantee to commit a page every 500ns. In this case, the NVRAM buffer needs to hold 100 entries with 4KB data, which is less than 1MB. For low latency RDMA based implementation on high speed networks, the buffer could be even smaller. Overall, the architecture can be further optimized for specific use cases, depending on network latency, read-write ratio, number of systems, latency criticality, data importance and degree of replication.

3. DRAMPERSIST USE CASES

DRAMPersist is envisioned as a use case for systems like RAMCloud [5] and Spark [6]. These are scale-out, primarily main memory distributed systems. Furthermore, since these systems use *volatile* DRAM as the primary data store, they have to take extra precautions in the system stack to ensure data persistence. This is typically done by intermittently flushing the (full or partial) contents of DRAM to disk. Depending on when this flush operation is carried out, it can have different effects. If done on the critical path, the access latency of the applications is increased. Even if the flush operation is not done on the critical path of application execution, precious compute cycles are taken away from the primary task that these systems are designed to do – fast data analysis. Furthermore, as is the case with most distributed systems, these in-memory systems also rely on inter-node replication to combat device, machine and network failures and maintain high availability. DRAMPersist provides hardware support to exploit this feature and provide persistence without flushing overheads.

DRAMPersist solves the scale out problem of in-memory distributed applications. Once system framework developers are provided with main memory that has DRAM like latencies as well as the persistence of storage devices, a new class of distributed system applications can be envisioned. These new class of applications will still rely on inter-node replication to guarantee data persistence and availability. However, these applications don't have to explicitly maintain persistence between main memory and storage on a per node basis as frequently. With DRAMPersist, the common operations of commit logging, write ahead logging and data flush to disk can be made relatively infrequent. This will result in the design of low-overhead distributed systems.

Figure 2: NVRAM buffer organization



4. CONCLUSION

In this paper, we present DRAMPersist, an architecture to make DRAM systems persistent. DRAMPersist is envisioned to be a memory architecture geared towards scale-out, distributed applications. DRAMPersist reduces the overheads of frequent committing of volatile DRAM data to storage and enables new class of distributed applications.

5. REFERENCES

[1] M. Awasthi. Rethinking Design Metrics for Datacenter DRAM. In *Proceedings of MEMSYS*, 2015.

[2] J. S. Kim et al. A 1.2 v 12.8 gb/s 2 gb mobile wide-i/o dram with 4x 128 i/os using tsv based stacking. 2012.

[3] K. Kim. Silicon technologies and solutions for the data-driven world. In *Proceedings of ISSCC*, 2015.

[4] K. T. Malladi et al. Towards Energy-Proportional Datacenter Memory with Mobile DRAM. In *Proceedings of ISCA*, 2012.

[5] J. Ousterhout et al. The Case for RAMCloud. *Commun. ACM*, 54(7), July 2011.

[6] M. Zaharia et al. Spark: Cluster Computing with Working Sets. In *Proceedings of HotCloud*, 2010.

[7] T. Zhang et al. Half-dram: A high-bandwidth and low-power dram architecture from the rethinking of fine-grained activation. In *Proceedings of ISCA*, 2014.