Rethinking Storage for the Cloud, Edge, Serverless, and Big Data Era

With the advent of the cloud-edge system architecture, along with streaming, micro-services, actor frameworks, and serverless applications, there is an unprecedented demand for simple, easy-to-use, and high-performance storage solutions. Storage in these settings is characterized by a hierarchy, consisting of main memory, possibly non-volatile memory, local SSD, and cloud storage. Applications (usually sharded) usually exhibit *temporal locality* of data reference, along with complex data formats and high ingestion and update rates.

For example, consider an ads pipeline that maintains per-user models or aggregates based on page clicks and ad views, over a week-long hopping window. When run entirely in main memory, e.g., using Trill [4] – our recent open-source in-memory steaming engine – we may need to scale out across hundreds of machines and possibly take expensive and synchronous memory checkpoints for durability. Fortunately, the hot working set of users actively updating and reading counters at a given moment is usually a small fraction. At the edge, applications often need to run at low memory footprint, while offloading cold data to their local storage or the cloud. Serverless functions in the cloud can benefit from the ability to embed a storage library that enables best-effort caching of sharded state without user involvement. Finally, with big data, there is a tremendous growth in raw and evolving flexible-schema data being ingested (e.g., the Twitter firehose), which complicates storage and retrieval needs as well.

In this paper, we argue for a new suite of storage and indexing artifacts that satisfy such workloads. Our artifacts are architected as libraries that applications may easily embed and use. They provide a unified view of the tiered storage hierarchy, exploit concurrent access to shared memory, and provide new durability models and techniques that better expose the commit latency vs. throughput trade-off to applications. We briefly overview them below:

(1) ***FASTER*** [1] is a new latch-free embedded persistent hash key-value store + cache over tiered storage that provides performance of more than 150M ops/sec when the hot working set fits in main memory. It supports point reads, (blind) updates, and atomic read-modify-write operations. It is based on a novel hybrid log abstraction that allows in-place updates directly on the log, and uses a new epoch protection framework for memory safety. It is available publicly as open source in C# and C++, with more than 3400 stars on GitHub.

(2) ***CPR*** (concurrent prefix recovery) [2] is a new recovery model that is suited to our target applications, and is also compatible with reliable messaging systems such as Kafka. The CPR model can avoid the scalability bottleneck of a write-ahead log by unifying the notions of group commit and incremental checkpointing, with fast non-deterministic in-place memory updates between commit points. CPR is the durability model used in FASTER.

(3) ***FishStore*** [3] is a new log storage and retrieval system extending FASTER, that provides extremely fast ingestion speeds for raw flexible-schema data, saturating SSD write speeds (2GB/sec) using just 2-4 CPU cores. It features a novel dynamic predicate registration interface that enables fast partial parsing of data, indexing using multiple hash chains, and speedy data retrieval using an adaptive mix of hash chain traversals and full scans.

We conclude by making the case for simplifying storage system interfaces. We believe applications should not have to make the hard choices of a priori schema specification, indexes, and durability techniques. Rather, applications should embed storage as a library, to which they provide key application requirements in terms of memory utilization and commit latency goals. Then, they simply ingest data and impose a dynamic workload mix. The system should make the hard storage, data layout, and indexing choices, learning and optimizing for the workload over time.

References

[1] FASTER: A Concurrent Key-Value Store with In-Place Updates. In SIGMOD, 2018. [link] Available as open-source at https://github.com/Microsoft/FASTER.

[2] Concurrent Prefix Recovery: Performing CPR on a Database. In SIGMOD, 2019. [link]

[3] FishStore: Faster Ingestion with Subset Hashing. In SIGMOD, 2019. [link]

[4] Trill: A High-Performance Incremental Query Processor for Diverse Analytics. In VLDB, 2014. [link] Available as open-source at https://github.com/Microsoft/Trill.

About the Speaker

Badrish Chandramouli is a principal researcher in the database group at Microsoft Research. He is interested in creating technologies to perform near real-time and offline big and raw data processing, as well as resilient state management for cloud, edge, and serverless applications. His initial work on stream processing first shipped commercially in 2010 with Microsoft SQL Server, as the StreamInsight engine. Starting 2012, he built Trill, a streaming analytics engine (now open source) that is widely used at Microsoft, for example, in the Bing advertising platform and as part of the Azure Stream Analytics service. More recently, he has been looking at the storage interface to applications, and built FASTER, a popular high-performance concurrent key-value store, and FishStore, a new high-speed ingestion and querying layer for raw data. Learn more about his research at https://badrish.net/ and at https://github.com/badrishc.