FASTER

Efficient state management for the modern edge-cloud

github.com/microsoft/FASTER

Badrish Chandramouli
Sr Principal Research Manager
Microsoft Research
badrish.net | @badrishc

Collaborators: Sebastian Burckhardt, Jose M. Faleiro, Ted Hart, Rohan Kadekodi, Donald Kossmann, Chinmay Kulkarni, Tianyu Li, Hitesh Madan, Samuel Madden, Guna Prasaad, Ryan Stutsman, and our open-source contributors.
Talk Outline

· SimpleStore - Research Project Summary

· FASTER & ecosystem
  · Architecture [SIGMOD 2018]
  · System features & use cases
  · A peek under the hood
    · Epoch protection
    · Async recoverability (CPR) [SIGMOD 2019]

· Multi-node FASTER
  · Architecture [VLDB 2021]
  · Async recoverability [SIGMOD 2021]

· Summary
The SimpleStore Project

Goal
Simplify app view of [storage + cache] at high perf
Create building block components
  • Used by user apps, cloud services, databases, functions
  • Used as storage accelerator or point of truth

• Key Insights
  • Use concurrency & async recovery to create fast resilient components over tiered storage
  • Leverage predicates & temporal patterns at storage to optimize cache-store

• Compute Workloads
  • Cache-store & logging lib: FASTER KV & Log [SIGMOD 2018]
  • Recovery & Scale-out: CPR, DPR, Shadowfax [SIGMOD 2019, SIGMOD 2021, VLDB 2021]
  • Edge-cloud ecosystem: CRA, Surface Fleet, Ambrosia, Netherite [ICDE 2019, VLDB 2020]

• Analytics Workloads
  • Data layout for analytics: Qd-tree [SIGMOD 2020, SIGMOD 2021]
  • Caching for Analytics [work in progress]
  • Secondary Indexing: FishStore [SIGMOD 2019]
The Storage Problem

- Modern apps and services – common requirements
  - Apps access, update, cache huge volumes of state (or objects)
    - E.g., billions of per-device counters, per-ad statistics, sharded app state in actors & serverless
    - May not fit in memory, requires fast access and update, durability
  - Apps need fast reliable logging and messaging as key building blocks
  - Apps need to work in cloud, edge, serverless, multi-tenant environments
What is FASTER

An open-source library to accelerate object storage, indexing, logging
- High performance, concurrent, latch free, shared memory, in C# (also ported to C++)
- Two sub-components

1) FasterLog
- Record log abstraction over tiered storage: enqueue, commit, scan, read, truncate
- In-memory part may be updated/accessed safely as a cache
- Can be used independently as a persistent queue

2) FasterKV
- Hash key-value store over the FASTER record log (hybrid log)
- Shapes the (changing) hot working set in memory → integrated cache
- Performance: >150 million ops/sec on one machine, for YCSB benchmark
  - Exceeds throughput of pure in-memory systems when working set fits in memory, linear scalability with #threads
## System Architecture

Tiered Storage (e.g., local SSD, Azure Premium Storage, RDMA)

Hash Index (no keys)

Read Cache

Virtual Address Space

Hybrid Log
- **Head**
- **ReadOnly**
- **Tail**
- **Begin**

Immovable Region

In Storage & Memory

Threads
Hybrid Log in Detail

- Divide memory into three regions
  - Stable (on disk) → Read-Copy-Update (RCU)
  - Mutable (in memory) → In-Place Update (IPU)
  - Read-only (in memory) → Read-Copy-Update (RCU)
    - Memory: latch-free circular buffer with **epoch protection** for memory safety

- Hybrid concurrency model
  - Read-copy-update (RCU): compare-and-swap on index
  - In-place-update (IPU): user record-level concurrency

- Tail grows → offsets grow as well
  - New records allocated at tail

- New & updated records stay in mutable region for a while → captures temporal locality
Features, Use Cases, Performance
Basic FasterKV Features

- Latch-free basic operations
  - Read, upsert, delete; *no transactions*
  - Atomic read-modify-write (RMW)
    - powerful primitive for aggregation

- Friendly session-based interface
  - Linearizable sequence of operations
  - Prefix recovery within session (later)

- Async ops within session
  - Works with C# task framework

```csharp
public static void Main()
{
    using var log = Devices.CreateLogDevice("hlog.log"); // backing storage device
    using var store = new FasterKV<long, long>(1L << 20); // hash table size (number of 64-byte buckets)
    new LogSettings { LogDevice = log }; // log settings (devices, page size, memory size, etc.)

    // Create a session per sequence of interactions with FASTER
    // We use default callback functions with a custom merger: RMW merges input by adding it to value
    using var s = store.NewSession(new SimpleFunctions<long, long>((a, b) => a + b));
    long key = 1, value = 1, input = 10, output = 0;

    // Upsert and Read
    s.Upsert(ref key, ref value);
    s.Read(ref key, ref output);
    Debug.Assert(output == value);

    // Read-Modify-Write (add input to value)
    s.RMW(ref key, ref input);
    s.RMW(ref key, ref input);
    s.Read(ref key, ref output);
    Debug.Assert(output == value + 20);

    Console.WriteLine("Success!");
}
```

- Dynamically grow index size
Advanced Features

- Extensible **IDevice** storage interface
  - Local SSD, mounted drives, Azure page blobs, remote mem (RDMA)
  - Composable via Tiered & Sharded meta-device abstractions
- Non-blocking incremental checkpointing, recovery
- Log compaction
- Key iteration
- FasterLog: Enqueue; Commit; Iterators
- **Secondary indexing [coming soon]**
  - Range index over keys or value fields
  - Hash index over value fields: e.g., register `value.pet`; query `value.pet == “dog”`
    - Based on “subset hash indexing” → see FishStore; SIGMOD 2019
    - First target customer: Azure Durable Functions/Netherite
Single-node scalability with # threads

- When current working set “happens to fit” in memory

100% RMW; 8 byte payloads  
100% blind updates; 100 byte payloads
Scalability with Disk-Resident Data

- YCSB, 100% read workload
- Hash table in memory, almost entire log on disk, no caching
- Local NVMe SSD capable of up to 1 million random IOPS.
Sample of Use Cases

- Azure Stream Analytics
  - Uses FasterKV for externalizing reference data and records in large cloud streaming pipelines

- Azure Event Grid
  - Uses FasterLog for fast reliable routing and notification service on edge

- Azure Durable Functions
  - We built a new stateful serverless backend called Netherite
  - System uses FasterKV (for function state) and FasterLog (for fast replayable messaging)
  - See details in their paper: https://arxiv.org/abs/2103.00033

- Many GitHub use cases:

  Jan-2021:
  Using FasterKV.
  I decided to run with Faster in our production system after your help and rewriting things a bit to suit our actual use case. <snip> I think the speed is actually pretty good for 50k+ devices hitting our system in the short term and we can improve later.

  Nov-2020:
  Using FasterLog
  On Azure D2as_v4 machines. We have about 1000 stores but they are all really tiny (less than 1000 messages per store). The largest store is about 1mb and most of the stores are <100kb, for a total of some 200mb across 1k stores.
A Peek Under the Hood

the generalized epoch framework
Multi-Threading: Epoch Protection Basics

• System Requirement
  • avoid any coordination between threads in common case
  • agree on mechanism to synchronize on shared system state

• Solution: epoch protection
  • System maintains shared counter E (current epoch) - can be “bumped” by any thread
  • Each thread keeps a (stale) local epoch counter copied from E
  • An epoch c is “safe” if all thread-local epochs are greater than c

<table>
<thead>
<tr>
<th>Safe Epochs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Thread 2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Thread 3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Thread 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Current Epoch → 2 3 4 5 Increasing Time →
Extending Epochs: Trigger Actions & Marking

- **Trigger Actions**
  - Associate a trigger (function callback) with epoch bump from \( c \) to \( c+1 \)
  - Trigger action will be executed later, when \( c \) becomes safe
  - Simplifies lazy synchronization in multi-threaded systems
  - Example: invoke function \( F() \) when (shared) status becomes “active”
    - Thread updates shared variable status = “active”, then bumps current epoch with trigger = “invoke function \( F() \)”
    - \( \text{BumpEpoch}() \Rightarrow F() \)
  - Guaranteed that all threads have seen “active” status before \( F() \) is invoked

- **Marking**
  - **Mark**: thread “marks” an operation/phase as complete
  - **CheckIfComplete**: thread checks if everyone has completed phase; if yes: advance phase

- **FASTER uses epochs, triggers, marking extensively**
  - Threads agree to respect global system state at epoch refresh boundaries
  - Memory safety, index resizing, log buffer maintenance, checkpoint state machine
Recovery: **Concurrent Prefix Recovery**

- In memory part of hybrid log is lost on update
- New recovery model for concurrent DB/KVS
  - Persist all ops until some point in input op sequence, per thread, none after
- Combines worlds of DB group commit and in-memory incr. checkpointing
- Admits scalable non-blocking implementation using epochs
- Fully implemented in FASTER
- See SIGMOD 2019 paper
  - https://aka.ms/FASTER
Remote FASTER

from embedded to clients → servers
Remote FASTER

- Goal: access FASTER as mid-tier cache/store from remote clients

- Brief Summary
  - Get the same FASTER throughput scalable with #threads, from **remote client sessions**
  - Low round-trip latencies (<100microsec for one request in datacenter)
  - Uses standard TCP protocol; also supports RDMA
  - Server inherits rich FASTER features (tiered storage, checkpoints, RMW, …)
Shadowfax

- Elastic client-server prototype FASTER C++
- Uses standard TCP on cloud VMs
  - We get high performance (100+ Mops/sec per server)
  - Low latency <100 μsec round-trip in Azure data center
  - Also supports RDMA access

- Key ideas in Shadowfax
  - Eliminate “shuffle” at network layer
  - Use “asynchronous global cuts” across machines + epochs within machine for elastic state migration

- Details: paper at VLDB 2021
Distributed Prefix Recoverability (DPR)

_prefix recovery for client sessions talking to multiple shards_
The Compute-Storage Pattern

- Read & write ops served from durable storage
- Reads: served from cache
- Writes:
  - Go through to durable storage, or
  - In 1 failure domain: ops complete → commit

Azure Function Workflow Step
Client App Framework

Azure Blobs, SSD, DB, KVS, FASTER Disk Log (local or remote)

Redis
Main memory DB (local or remote)
FASTER - mutable region

compute
storage

$msec$

compute
cache
storage

$nsec$, $\mu$sec
Let’s Shard the Data

- Shared-nothing state

- Global shared state
Goal: Distributed Compute on Sharded Cache-Stores

- Normal behavior
  - Return operations immediately; before commit
  - Including **writes, not just reads**
  - Preserve client’s notion of operation dependencies via **lazy prefix commits**
  - At low cost with no cross-shard overheads

- Behavior on shard failure
  - Rollbacks will happen due to multiple failure domains (unlike CPR)
  - Limit effect of rollback to true dependencies
  - Make rollbacks non-blocking
  - Notify affected client sessions of rollback of uncommitted ops
Basic Idea: **Client Session** to Capture Dependencies

- Clients issue ops to cache-store shards
  - Op status can be \{ issued, completed, committed \}
- **Client Session** captures op dependencies
  - Issued op depends on all previous “completed” ops in session (transitive)
  - Issued \(=\) completed:
    - ![Diagram of issued and completed dependencies](image)
  - Issued \(!=\) completed:
    - ![Diagram of issued and completed dependencies](image)
- Periodically commit prefixes of **session order**
  - An order of ops that respects partial order of op dependencies
- Implemented DPR for FASTER & Redis
- Details: see SIGMOD 2021 (to appear)
Recoverability Summary

- Clients talk to sharded cache-stores via linear sessions
- Separate op completion from commit
- We provide a prefix recovery guarantee within each session
  - Requires dependency tracking mechanism
- Client rollback in case of failure
- Applicable to stores and workflows
- Details in paper at SIGMOD 2021
Talk Summary
Summary

- SimpleStore project aims to simplify the use of storage for apps, workflows, services, analytical databases, serverless
- The FASTER project offers
  - A concurrent latch-free embedded library for managing memory and tiered storage
  - Two concrete artifacts: FasterKV and FasterLog
  - Secondary indexing for log analytics & range queries
  - Remote access without performance loss
  - Novel recovery techniques for single- and multi-node DB
- DB techniques are generally applicable beyond artifacts
- More details at https://aka.ms/FASTER
  - Link to research papers: https://microsoft.github.io/FASTER/docs/td-research-papers/
Thank you!

https://aka.ms/FASTER