From DRAM to SSDs
Challenges with caching @ Facebook

Agenda:
1. Introduce CacheLib & hybrid caching
2. Hybrid caching challenges
3. Call to action
## Caching is important for social media

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Recency</th>
<th>Scale &amp; Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users share and consume content through overlapping social circles</td>
<td>Users interact with recently shared content</td>
<td>Efficiency is important to scale products to billions of users</td>
</tr>
</tbody>
</table>
### Evolution of FB infrastructure

<table>
<thead>
<tr>
<th></th>
<th>2005 - 2010</th>
<th>2010 - 2015</th>
<th>2015 - today</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Users</strong></td>
<td>10s of millions</td>
<td>100s of million</td>
<td>billions</td>
</tr>
<tr>
<td><strong>Product</strong></td>
<td>likes, comments, profile</td>
<td>+ timeline, messaging, videos, photos, location..</td>
<td></td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td>Memcached, MySQL</td>
<td>fb-Memcached, TAO MySQL, CDN, Haystack</td>
<td>hundreds of services</td>
</tr>
</tbody>
</table>
# Diverse caching needs @ Facebook

<table>
<thead>
<tr>
<th>Application Domain</th>
<th>Architecture</th>
<th>Data format</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>General purpose KV cache</td>
<td>Look-aside</td>
<td>Blob</td>
<td>DRAM</td>
</tr>
<tr>
<td>Social graph cache</td>
<td>Read &amp; write through</td>
<td>Structured</td>
<td>SSD</td>
</tr>
<tr>
<td>CDN caches</td>
<td>Write back</td>
<td>Semi-structured</td>
<td></td>
</tr>
<tr>
<td>Ranking/ML systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer infrastructure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analytics systems</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Should we generalize or specialize?
CacheLib – Embedded C++ cache engine

Aggregation point for innovation

CacheLib solves:

1) What to cache? - Heuristics
2) How to cache? – Storage design

CacheBench – workload benchmarking tool
CacheLib @ Facebook

Online Data
- Social graph
- Look aside KV Cache

Traffic
- CDN
- Proxygen

Storage
- HDD file storage
- HDD video storage

AI/ML
- Training
- Ranking

Dev Infra
- Data analytics
- Source control
- Stream processing

70+ services, XXX K+ instances, XX PB DRAM, XXX PB NVM

Open sourced by Facebook
www.cachelib.org
CacheLib: Item

Unit of object managed in the cache

- Sequence of bytes (<4MB) identified by a key (<255 bytes)
- Exposed through an ItemHandle (shared_ptr<Item>)
CacheLib: API

ItemHandle `allocate(key, size, ttl)`

void* Item::getMemory()

ItemHandle `insertOrReplace(handle)`

ItemHandle `find(key)`

bool `remove(key)`

Features

- variety of cache policies
- memory pools & isolation
- resource adaptiveness
- zero-copy access
- high concurrency
- structured data cache
- hybrid cache
Hardware trade-offs

Explore cost and power tradeoffs within a cache system

- marginal cost of an additional hit goes up
  - Hot objects in DRAM
  - Warm objects in NVM
Hybrid Cache goals

**Fluid usage** of storage mediums based on cost-sensitivity
- DRAM, NVM technologies etc.

**Portability** of caching applications
- Hide the complexity of hardware technologies behind the API
Hybrid caches @ Facebook

Social graph: 30GB DRAM, 1TB SSD
Look-aside key-value cache: 40GB DRAM, 1TB SSD
CDN: 30–120 GB DRAM, 2–8 TB SSD
HDD Storage: 10GB DRAM, 400GB SSD
Hybrid Cache efficiency in practice

256GB DRAM -> 60GB DRAM + 2TB SSD

- 10x more cache capacity
- 50% reduction in misses
- 25% reduction power and cost of ownership
Hybrid cache: Challenges

**DRAM**
- mutable bytes
- small working set
- infinite R/W BW
- low & steady latency
- infinite endurance

**NVM**
- mutable blocks
- large working set
- limited R/W BW
- latency is complicated
- limited endurance
Hybrid cache: Design

- **Portability**: Applications always see a DRAM cache
- Items are allocated in DRAM and then evicted to NVM
  - Evicted items can be rejected by NVM admission policy
Latency: accessing Items through find()

- **Portability**: `getMemory()` blocks until Item is in DRAM
- Async interface `ItemHandle`
  - `isReady()` - is Item in DRAM?
  - `wait()` - wait until Item is in DRAM
  - SemiFuture compatibility
Navy
NVM storage engine

Goal: **CPU & IO efficient** point & negative lookups with **low DRAM overhead**

Asynchronous IO

Optimized for caching tiny objects (100s of bytes)

Optimized for caching large objects (>4KB)

File system/Block interfaces
Small Item engine design

Accessing billions of objects per TB
  - With single IO
  - Low DRAM overhead

Design
  - Set-associative index (no DRAM)
  - Optional DRAM bloom filters
  - FIFO eviction (no DRAM)
  - No space amp, but large write-amp

Due to endurance constraints, caches can’t write all the objects to SSD -> poor miss ratios
Improving small object caching

Log structured storage?

To be compute and IO efficient for lookups, log structured cache needs a full DRAM index.

Set-Associative Cache  Log-Structured Cache

- DRAM overhead
  - Low
  - High
- Write amplification
  - High
  - Low

Kangaroo: Realizing the best of both worlds

Collaboration with Carnegie Mellon University, SOSP’21
Kangaroo: SOSP’21

Design

1) Insert to Klog after buffering in DRAM
2) Periodically flush objects from KLog to KSet
3) Move all objects in KLog that map to the same bucket in KSet

Reducing the write amplification improves miss ratios.
Summary: call to action

Hybrid caching is gaining popularity, lots of new challenges

Caching small objects on block devices
- With lower DRAM overheads
- With lower endurance overheads
- With more IO efficiency

New hardware technologies on the horizon

www.cachelib.org

CacheLib offers an OSS platform to collaborate and explore solutions

need innovative solutions

collaborate among industry & academia