Managing Datastore Locality at Scale with Akkio

Muthukaruppan Annamalai, Kaushik Ravichandran, Harish Srinivas, Igor Zinkovich,
Luning Pan, Tony Savor, David Nagle and Michael Shum

FACEBOOK, UNIVERSITY OF TORONTO*
Strong consistency is not expensive

Better serves workloads with low read-write ratios
Flexible

Supports five different durable storage systems

Scalable

100's of millions of data access requests per second
Metadata DB

- Stores μ-shard → placement mapping
- A cache in front to absorb most reads
• Stale lookups are prevented through means like per-μ-shard Access Control Lists (ACLs)

• Typical footprint is ~300GB
Access DB

- Fundamentally a time-windowed, counter service
- One counter per-\(<\mu\text{-shard, datacenter}>\) tuple
• Typical footprint is ~300GB
• Similar services can share access data
• Routing layer is a complete black box

Routing Layer Load Balancing?  No problem!
Data Placement Service

- Data Placement Service knows list of possible placements and associated data centers
- Receives hints on sub-optimally placed μ-shards from Akkio client
- New μ-shards are provisioned through the Data Placement Service
• Runs a set of policies to determine migration eligibility

• Identifies an optimal location using a configurable scoring policy

**Default Scoring Policy**

• Eliminate any placement possibilities with hot data centers

• Generate per-data center score
  • Number of times a μ-shard was accessed from a data center in the last N days
    • N is configurable
• Generate per-placement score
  • Sum of corresponding data center scores

• Pick placement with highest score
  • If tied, break tie using resource usage information

Fault Tolerance

• Migrations are serialized through a per-\(\mu\)-shard lock on the Metadata DB

• A work item in the Metadata DB is created to track any work
  • Atomically created and deleted with the lock
  • Used to recover migrations
  • Monotonically increasing epoch to fence writes
Migration

- Every backend has a custom Data Placement Service plug-in to enable reads/writes to backend
- Configurable migration strategy
- Depends on capabilities of underlying store
- Native timestamps
- Fine-grained ACLs
- Read-only/modify-write transactions
Simple Migration Strategy

1. [Metadata DB] Atomically grab per-µ-shard lock & create work item
2. [Source] Set per-µ-shard ACL to R/O, drain writes, read data
3. [Destination] Write data, set per-µ-shard ACL to R/O
4. [Metadata DB] Update placement
5. [Source] Delete data & per-µ-shard ACL
6. [Destination] Set per-\(\mu\)-shard ACL to R/W

7. [Metadata DB] Atomically delete lock & work item
Typically its cheaper to move compute

Data does not need to be present everywhere

Savings on WAN bandwidth and latency wins
Locality at fine-granularity works!