Flex-KV: Enabling Flexible And High-Performance Key-Value Systems

Amar Phanishayee, David Andersen
Carnegie Mellon University

Himabindu Pucha
Violin Memory
Anna Povzner, Wendy Belluomini
IBM Research
put(key, value)

get(key)    value

delete(key)
KV Systems - Popular & Widely Deployed

Web Services

Infrastructure Services
# Today’s Space: Many Point Solutions

<table>
<thead>
<tr>
<th>KV Systems</th>
<th>Hardware</th>
<th>Durability</th>
<th>Application Needs</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem</td>
<td>Disk</td>
<td>Flash</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cache</td>
<td>Store</td>
</tr>
<tr>
<td>memcached</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td></td>
</tr>
<tr>
<td>repcached</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>memcachedb</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td></td>
</tr>
<tr>
<td>Dynamo</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Hibari</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>FAWN-KV</td>
<td>🟢</td>
<td></td>
<td>🟢</td>
<td>🟢</td>
</tr>
</tbody>
</table>
We propose one technique (*Ouroboros*+) that can be used to build a KV System to

- run on different hardware configurations
- support a variety of application needs

---

* Mem, Disk, Flash, ...
* Wimpy & brawny nodes
* Durability - cache, store
* Replication for availability/performance
* Consistency - strong, eventual/weak
* Performance - query/update recovery speed

---

Opportunity To Fix Impeding Disaster
Goals For Our KV Systems

• High Performance
  ‣ Steady state (no churn)
  ‣ Churn (node additions and failures)
  ‣ Use storage tech effectively (memory, Flash, Disk)
  ‣ Load balancing

• Fault tolerance
  ‣ Provably correct replication
  ‣ Maintain replication factor
  ‣ Balance roles/load

• Flexibility - Hardware configs, App requirements
Achieving KV Goals

- Consistent Hashing with Virtual Nodes
  - Partitioning scheme for load balancing
  - Limits global data movement & local load on churn
  - High performance on Flash (*log-based updates and transfers*)

**Ouroboros+**: generalized chain-based replication

- Node additions to any position in the chain
- High performance (*log-based replication, non-blocking protocols*)
- Self healing, Strong Consistency (*Invariant: if* $i < j$, $H_i \supseteq H_j$)
- Balance backend roles/load (*all nodes play all roles*)
- Relaxed consistency for higher performance (different update, query protocols)
Replication For Fault-Tolerance: Replicate Along The Ring

Consistent Hashing

Replica Chain Along The Ring
(for green range)

[Chord: Stoica-2001] [Dynamo: Vogels-2009] [CFS: Dabek-2001]
**Chain Replication**

Updates To Head, Queries From Tail

- **Put** operation: `put(k, v, id)`
  - Processed sequentially from **head** to **tail**.
  - Acknowledged by each replica.
  - **If** `i < j`, `Hi ⊇ Hj`.

- **Get** operation: `get(k, id'')`
  - Initiated by **tail**.
  - Response received by **head**.

- **Pending** and **Last_ACK** states
  - 4, 3, 2, 1

- Background (vanRenesse-2004)
How To Replicate On A Ring?

**Consistent Hashing**
Add node in the middle
B: Owner of Green Range

**Chain Replication**
Add node at end
But B should be head!
Ouroboros: Enabling Node Additions To Any Position In The Chain

Challenge 1: Quick non-blocking node add/leave
Challenge 2: Maintain strong consistency
Ouroboros: All Backends Share Similar Roles

- Configurable replication factor (R)
- Each node part of R overlapping replica chains
  - Head for 1, Mid for (R – 2), Tail for 1
Ouroboros: Generalized Chain-based Replication

- Allows node additions to any chain position
  - High performance (fast & effective)
    - log-based replication
    - non-blocking node addition and removal protocols
  - Provably strong data consistency (per key linearizability)
    - Query, Update, Replication Guarantees
    - Chain Invariant: if $i < j$, $H_i \supseteq H_j$
  - Balance backend roles (all nodes play all roles)

Detailed protocol, event loops, and complete proof in TR# CMU-CS-12-139
Flex-KV uses Ouroboros+ to build a KV System that can
• run on different hardware configurations
• support a variety of application needs

1. Fast node addition to any chain position
2. Provably strong data consistency
   (per key linearizability)

Generalize to support weaker consistency for higher performance, lower latency
1. Update plumbing (SU, AU, AUSI)
2. Query node selection

Use Homogeneous Chains
or Heterogeneous Replica Chains for faster recovery & memory efficiency

Common data storage interface
- FAWN-DS
- SSDs, Hard Disks
- DRAM-DS
- Tape-DS
- Your-DS
Flex-KV And Three Example Configurations

memcached

M

Replication Factor = 1
Cache

Core2, 1GB DRAM
100,000 queries/sec
Matches memcached
(no multi-get)

FAWN-KV

D

Replication Factor = R
Store
Strong Data Consistency

Core2, 500GB HDD
289 queries/sec

M - Memory Replica, D - Disk Replica
Challenge: Availability

Presentation Tier

Logic Tier

get(key)

memcache

Database

Storage
Challenge: Availability

![Twitter Overload Image]

Presentation Tier

Logic Tier

get(key)

Overload

memcache

Database

Storage

Slow recovery from disk

[reddit-2010] [Facebook-2010] [Twitter-2010]
Memory Overhead vs Recovery Time Tradeoff

M : Disk seeks on recovery = bad!

M-M (higher memory overhead)

M - Memory Replica
D - Disk/Flash Replica
1. Different replica types (M, D)
2. Heterogeneous chains (M-M, M-D)
3. Query node selection

M - Memory Replica
D - Disk/Flash Replica
Consistency vs Performance Tradeoff

Update -> M -> D -> D

Response ← M

Synchronous Updates (SU)

M - Memory Replica
D - Disk/Flash Replica
Consistency vs Performance Tradeoff

- **Strong Consistency**
  - Synchronous Updates (SU)

- **Weak Consistency**
  - Asynchronous Updates (AU)

**Update**

**Response**

- **M** - Memory Replica
- **D** - Disk/Flash Replica
Consistency vs Performance Tradeoff

Strong Consistency

Synchronous Updates (SU)

Asynchronous Updates (AU)

Weak Consistency

Update

Response

M - Memory Replica
D - Disk/Flash Replica
Mi - Memory Invalidation Map
Consistency vs Performance Tradeoff

1. Different replica types (Mi...D)
2. Update mechanisms (SU, AU, AUSI)
3. Query node selection

M - Memory Replica
D - Disk/Flash Replica
Mi - Memory Invalidation Map
Proactive Recovery Using M-D Cache

Flex-KV Cache with M-D chains

get(key)

put (key, value)

Small % of misses when using AUSI

Presentation Tier

Logic Tier
Perf., Memory Overhead, Recovery Time Tradeoff

1. Different replica types (M...D)
2. Update mechanisms (SU, AU, AUSI)
3. Query node selection

M - Memory Replica
D - Disk/Flash Replica

M : Disk seeks on recovery = bad!
M-D: streaming from disk = good
M-M (higher memory overhead)
Flex-KV Knobs To Control KV Configuration

- Replication Factor (R)
- Replica Type (Mem, Flash, Disk)
- Heterogeneous chains (M-M, D-D, M-D)
- Query Node Selection
- Update schemes (SU, AU, AUSI)
## Flex-KV Coverage (Cache)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SU (Consistent)</th>
<th>AUSI (Consistent)</th>
<th>AU (Weakly Consistent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M - M + DB</td>
<td>Gear6</td>
<td>repcached</td>
<td></td>
</tr>
<tr>
<td>(memory inefficient)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M - D + DB</td>
<td></td>
<td></td>
<td>redis</td>
</tr>
<tr>
<td>(memory efficient)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Flex-KV Coverage (Store)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SU (Consistent)</th>
<th>AUSI (Consistent)</th>
<th>AU (Weakly Consistent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M - M (memory inefficient)</td>
<td>Gear6</td>
<td></td>
<td>repcached</td>
</tr>
<tr>
<td>M - D (memory efficient)</td>
<td>RAMCloud (Mb...D)</td>
<td></td>
<td>redis</td>
</tr>
<tr>
<td>D - D (memory efficient)</td>
<td>FAWN-KV Hibari</td>
<td></td>
<td>Tokyo Tyrant</td>
</tr>
</tbody>
</table>
Flex-KV uses *Ouroboros*+ to build a KV System that can
- run on different hardware configurations
- support a variety of application needs

1. Fast node addition to any chain position
2. Provably strong data consistency
   (per key linearizability)

Generalize to support weaker consistency for higher performance, lower latency
1. Update plumbing (SU, AU, AUSI)
2. Query node selection

Use Homogeneous Chains
or Heterogeneous Replica Chains for faster recovery & memory efficiency

Common data storage interface

- FAWN-DS
  - SSDs, Hard Disks
- DRAM-DS
- Tape-DS
- Your-DS