Bespokv: Application Tailored Scale-Out Key-Value Stores

Ali Anwar, Yue Cheng, Hai Huang, Jingoo Han, Hyogi Sim, Dongyoon Lee, Fred Douglis, and Ali R. Butt
Role of Distributed KV stores in HPC

• Use of emerging storage technologies has open up new opportunities for the use of KV stored in HPC
• Examples of this use includes dynamic consistency control, coupling applications, and storing intermediate results
• Matching HPC application demands and needs require customizations that existing KV stores do not provide
• As a result, a variety of distributed KV stores have been developed
Fundamental challenges in developing KV stores

1. Developing a new distributed KV store is never an easy task!
   - Redis: 20K LoC – 9 years
   - Cassandra: 390k LoC – 10 years
   - HyperDex: 52k LoC – 7 years

2. Distributed systems are notoriously bug-prone and incorrect implementation causes crash!
   - Cassandra-6023
   - HBase-3380
   - ZooKeeper-335
   - Redis-1381

3. User requirements are changing all the time!

   Last weekend we completed a transition from Flowdock's database of choice, Cassandra, to another NoSQL alternative, MongoDB. Since our technology stack has always generated some interest, I’ll now try to justify our decision in public.

   Some of our users might remember this:

   We're having some database problems. Trying to fix them ASAP.
Fundamental challenges in developing KV stores

1. Developing a new distributed KV store is never an easy task!
   - Redis: 20K LoC – 9 years
   - Cassandra: 390k LoC – 10 years
   - HyperDex: 52k LoC – 7 years

2. Distributed systems are notoriously bug-prone and incorrect implementation causes crash!
   - Cassandra: 6023
   - HBase: 3380
   - ZooKeeper: 335
   - Redis: 1381

3. User requirements are changing all the time!

Trial Run - HBase -> Cassandra

We lean toward reading HBase HFiles directly when we can and avoiding HBase. It’s unnecessary overhead when the entire table is processed anyways. Our plan was to go from HFiles -> SSTables -> Cassandra Cluster.
BespokV: A new paradigm for building distributed KV store services

1: A modular architecture that generalizes common features and can significantly reduce engineering effort

2: A functional partitioning abstraction that is resilient to buggy code with fault isolation

3: A versatile platform that is easy to use, can support a flexible range of deployment options, and reasonably fast
Scale-out a non-distributed KV store

1) Client-side partitioning

2) Proxy-assisted partitioning
Scale-out a non-distributed KV store

3) Proxy-assisted partitioning & replication

4) BespoKV-based flexible partitioning & replication
Comparison with state-of-the-art

<table>
<thead>
<tr>
<th>System</th>
<th>Shard</th>
<th>Replicate</th>
<th>Multiple Backend</th>
<th>Multiple Consistency</th>
<th>Multiple Topology</th>
<th>Automatic Recovery</th>
<th>Programmable</th>
</tr>
</thead>
<tbody>
<tr>
<td>BespoKV</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>NETFLIX</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>DYNAMITE</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>redis</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>BespoKV</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Notes: ✔️ indicates a feature is supported, ✗ indicates it is not.
BespoKV architecture

- **BespoKV controlet**
  - Replication
  - Consistency
  - Topology
  - Recovery

- **BespoKV control plane**
  - Client lib
  - Put(k, ▲)
  - Get(k)

- **BespoKV data plane**
  - Controlet
  - Datalet
  - Storage

- **BespoKV components**
  - Coordinator
  - Shared Log
  - Distributed Lock Manager

- **Developer-defined BespoKV datalets**
  - Files
  - Tree
  - Log
  - Redis
  - LevelDB

- **Operations**
  - Get(k)
  - Put(k, ▲)
Using BespoKV

```cpp
1 void Put(Str key, Obj val) {
2     HashTbl.insert(key, val)
3 }
4
5 Obj Get(Str key) {
6     return HashTbl(key)
7 }
```

```
"Topology": "Master-Slave",
"Consistency": "Strong",
"Replication": 3,
...
```
BespokV flexibility & versatility

BespokV supports wide range of distributed KV services

Topology + Consistency combinations:
1. Master-slave + Strong consistency
2. Master-slave + Eventual consistency
3. Active-active + Strong consistency
4. Active-active + Eventual consistency
5. ...

- Range queries
- Per-request consistency
- Hybrid AA+MS topology
- Heterogeneous configuration
- Dynamic adaptation to consistency/topology changes
- ...

Practical KV stores

New query types

Flexible configurations
Bespokv use cases

- Hierarchical and heterogeneous storage of HPC
- Distributed cache for deep learning
- Building burst buffer file systems
- Accelerating the file system metadata performance
- Resource and process management
Example 1: Master-Slave Strong Consistency (MS+SC) Write path

1. Put(k,v)
2. putHead(k,v);
3. putMid(k,v);
4. putTail(k,v);
5. Ack; Ack
6. Ack
Example 1: Master-Slave Strong Consistency (MS+SC) Read path

1. Get(k)
2. getD(key)
3. Ack(v)
Example 2: HPC monitoring and Analytics
Master-Slave Eventual Consistency (MS+EC)
BespoKV Implementation

• Prototype implementation using C/C++
• Docker container based cross-platform compatibility
• 5 datalet applications
  • Implemented from scratch (3)
  • Ported from existing standalone KV store applications (2)
• 4 readily available controlets
• 2 custom parsers + Google protobuf support

Welcome to download & try at: https://github.com/tddg/BespokV
### BespoKV Implementation

<table>
<thead>
<tr>
<th>Type</th>
<th>Components</th>
<th># LoC</th>
<th>Sub Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BespoKV</td>
<td>Core IO</td>
<td>5617</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event handler</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messaging</td>
<td>1,772</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log handler</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coordinator</td>
<td>1,031</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock Server</td>
<td>943</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Client lib</td>
<td>1,528</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Google protobuf</td>
<td>262</td>
<td>11,580</td>
</tr>
<tr>
<td></td>
<td>Redis + SSDB</td>
<td>333</td>
<td>595</td>
</tr>
<tr>
<td>Parsers</td>
<td>HT</td>
<td>[966+]</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>Log</td>
<td>[966+]</td>
<td>286</td>
</tr>
<tr>
<td></td>
<td>MT</td>
<td>[966+]</td>
<td>98</td>
</tr>
<tr>
<td>Apps</td>
<td></td>
<td></td>
<td>1,457</td>
</tr>
</tbody>
</table>

Developer provided protocol

Template code shared by datalet applications
### KV store development made easy!

<table>
<thead>
<tr>
<th>Datalet</th>
<th>BespoKV proto</th>
<th>Developer-provided proto</th>
<th>Develop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core IO</td>
<td></td>
<td></td>
<td>2 days</td>
</tr>
<tr>
<td>HT</td>
<td>107</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>Log</td>
<td>286</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>MT</td>
<td>98</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>#LoC Template</td>
<td>966</td>
<td></td>
<td>3 days</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>966</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>966</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controllet</th>
<th>MS+SC</th>
<th>MS+EC</th>
<th>AA+SC</th>
<th>AA+EC</th>
<th>Develop time</th>
</tr>
</thead>
</table>
Experimental Setup

• For scalability, we perform evaluation on Google Cloud Engine
  • 48 nodes
  • Each node has 4 cores and 15 GB memory
  • 1 Gbps connectivity
• For performance testing, we use local testbed
  • 12 nodes
  • Each node has 8 cores and 64 GB memory
  • 10 Gbps connectivity
• We use two workloads obtained from typical HPC services: job launch, and I/O forwarding and three workloads from the Yahoo! Cloud Serving Benchmark (YCSB)
Q1: Are BespoKV-enabled distributed KV stores scalable?

Eventual consistency
Q1: Are BespoKV-enabled distributed KV stores scalable?

Eventual consistency
Q2: How does BespoKV compare to existing proxy-based KV stores?

Throughput (10^3 QPS)

- **Unif 95% GET**
- **Unif 50% GET**
- **Zipf 95% GET**
- **Zipf 50% GET**

<table>
<thead>
<tr>
<th>Setup</th>
<th>Unif 95% GET</th>
<th>Unif 50% GET</th>
<th>Zipf 95% GET</th>
<th>Zipf 50% GET</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS+SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS+EC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA+EC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BespoKV+Redis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynomite+Redis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twemproxy+Redis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q3: How does BespoKV compare to existing natively-distributed KV stores?
Summary

• BespoKV can take a single-server data store and transparently enables a scalable, fault-tolerant distributed KV store service

• BespoKV can significantly reduce the engineering effort to develop interesting KV store services

• Evaluation shows that BespoKV is flexible, adaptive to new user requirements, achieves high performance, and scales horizontally
Thank You!

Questions & contact: Ali Anwar, ali@vt.edu
https://dssl.cs.vt.edu/