ClusterOn: Building highly configurable and reusable clustered data services using simple data nodes

Ali Anwar★ Yue Cheng★ Hai Huang† Ali R. Butt★

Virginia Tech IBM NSF
Growing data needs → new storage applications
A new storage system is born daily*

Number of storage systems papers in SOSP, OSDI, ATC, and EuroSys conferences in the last decade

* We exaggerate here, but new storage applications are developed fairly regularly!
DISTRIBUTED STORAGE IS COMING
Distributed storage systems are notoriously hard to implement!

“Fault-tolerant algorithms are notoriously hard to express correctly, even as pseudo-code. This problem is worse when the code for such an algorithm is intermingled with all the other code that goes into building a complete system...

Tushar Chandra, Robert Griesemer, and Joshua Redstone, Paxos Made Live, PODC’07
Case study: Redis 3.0.1

replication.c – replicate data from master to slave, or slave to slave
sentinel.c – monitoring nodes, handle failover from master to slave
cluster.c – support cluster mode with multiple masters/shards

These files are 20% of the code base (450K/2100K in size)
Quantifying LoC

- Core IO

<table>
<thead>
<tr>
<th></th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>12955</td>
</tr>
<tr>
<td>HDFS</td>
<td>31117</td>
</tr>
</tbody>
</table>

% LoC

- Redis: 20%
- HDFS: 20%

Core IO
Quantifying LoC

- Core IO
- Distributed management

Lines of code:
- Redis: 12955
- HDFS: 31117

Graph showing % LoC for Core IO and Management.
Quantifying LoC

- Core IO
- Distributed management
- Etc: Config/auth/stats/compatibility ...

<table>
<thead>
<tr>
<th></th>
<th>Redis</th>
<th>HDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code:</td>
<td>12955</td>
<td>31117</td>
</tr>
<tr>
<td></td>
<td>8237</td>
<td>56208</td>
</tr>
<tr>
<td></td>
<td>20202</td>
<td>25185</td>
</tr>
</tbody>
</table>

% LoC

- Core IO
- Management
- Etc
Quantifying LoC

- Core IO
- Distributed management
- Etc: Config/auth/stats/compatibility ...

Lines of code:
- Redis: Core IO 12955, Management 8237, Etc 20202
- HDFS: Core IO 31117, Management 56208, Etc 25185
Abstracting and generalizing common features/functionality can simplify development of new distributed storage applications.
Steps involved in developing a distributed storage application
(a) Vanilla

```c
void Put(Str key, Obj val) {
    if (this.master):
        Lock(key)
        HashTbl.insert(key, val)
        Unlock(key)
        Sync(master.slaves)
    }

Obj Get(Str key) {
    if (this.master):
        Obj val = Quorum(key)
        Sync(master.slaves)
        return val
    }

void Lock(Str key) {
    ... // Acquirelock
}

void Unlock(Str key) {
    ... // Releaselock
}

void Sync(Replicas peers) {
    ... // Update replicas
}

void Quorum(Str key) {
    ... // Select a node
}
```
(a) Vanilla

1 void Put(Str key, Obj val) {
2     if (this.master):
3         Lock(key)
4         HashTbl.insert(key, val)
5         Unlock(key)
6         Sync(master.slaves)
7 }
8
9 Obj Get(Str key) {
10    if (this.master):
11        Objval = Quorum(key)
12        Sync(master.slaves)
13        return val
14 }
15
16 void Lock(Str key) {
17     ...
18 }
19
20 void Unlock(Str key) {
21     ...
22 }
23
24 void Sync(Replicas peers) {
25     ...
26 }
27
28 void Quorum(Str key) {
29     ...
30 }

(b) Zookeeper based

1 void Put(Str key, Obj val) {
2     if (this.master): // zookeeper
3         zk.Lock(key) // zookeeper
4         HashTbl.insert(key, val)
5         zk.Unlock(key) // zookeeper
6         Sync(master.slaves)
7 }
8
9 Obj Get(Str key) {
10    if (this.master):
11        Objval = Quorum(key)
12        Sync(master.slaves)
13        return val
14 }
15
16 void Sync(Replicas peers) {
17     ...
18 }
19
20 void Quorum(Str key) {
21     ...
22 }
### (a) Vanilla

```c
void Put(Str key, Obj val) {
    if (this.master):
        Lock(key)
        HashTbl.insert(key, val)
        Unlock(key)
        Sync(master.slaves)
}

void Lock(Str key) {
    ...
    // AcquireLock
}

void Unlock(Str key) {
    ...
    // ReleaseLock
}

void Sync(Replicas peers) {
    ...
    // UpdateReplicas
}

void Quorum(Str key) {
    ...
    // Select a node
}
```

### (b) Zookeeper based

```c
void Put(Str key, Obj val) {
    if (this.master):
        zk.Lock(key) // zookeeper
        HashTbl.insert(key, val)
        zk.Unlock(key) // zookeeper
        Sync(master.slaves)
}

Obj Get(Str key) {
    if (this.master):
        Obj val = Quorum(key)
        Sync(master.slaves)
        return val
}
```

### (c) Vsync

```c
#include <vsync lib>

void Put(Str key, Obj val) {
    if (this.master):
        zk.Lock(key) // zookeeper
        HashTbl.insert(key, val)
        zk.Unlock(key) // zookeeper
        Vsync.Sync(master.slaves)
}

Obj Get(Str key) {
    if (this.master):
        Obj val = Vsync.Quorum(key)
        Vsync.Sync(master.slaves)
        return val
}
```

(c) Vsync

(a) Vanilla

(b) Zookeeper based
<table>
<thead>
<tr>
<th></th>
<th>Vanilla</th>
<th>Zookeeper based</th>
<th>Vsync</th>
<th>ClusterOn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>void Put(Str key, Obj val) {</code></td>
<td>1</td>
<td><code>#include &lt;vsync.h&gt;</code></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td><code>if (this.master):</code></td>
<td>2</td>
<td><code>void Put(Str key, Obj val) {</code></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td><code>Lock(key)</code></td>
<td>3</td>
<td><code>if (this.master):</code></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td><code>HashTable.insert(key, val)</code></td>
<td>4</td>
<td><code>zookeeper.Lock(key)</code></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td><code>Unlock(key)</code></td>
<td>5</td>
<td><code>zookeeper.Unlock(key)</code></td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td><code>Sync(master.slaves)</code></td>
<td>6</td>
<td><code>Vsync.Sync(master.slaves)</code></td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>}</td>
<td>7</td>
<td>}</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>8</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>9</td>
<td><code>Obj Get(Str key) {</code></td>
<td>9</td>
<td><code>Void Get(Str key) {</code></td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td><code>if (this.master):</code></td>
<td>10</td>
<td><code>ObjVal = Quorum(key)</code></td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td><code>ObjVal = Quorum(key)</code></td>
<td>11</td>
<td><code>sync(master.slaves)</code></td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td><code>return val</code></td>
<td>12</td>
<td><code>return val</code></td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>}</td>
<td>13</td>
<td>}</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td><code>void Lock(Str key) {</code></td>
<td>16</td>
<td><code>Vsync.Quorum(key)</code></td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td><code>... // Acquirelock</code></td>
<td>17</td>
<td><code>Vsync.Sync(master.slaves)</code></td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td><code>}</code></td>
<td>18</td>
<td>}</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td><code>void Unlock(Str key) {</code></td>
<td>20</td>
<td><code>Vsync.Sync(master.slaves)</code></td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td><code>... // ReleaseLock</code></td>
<td>21</td>
<td>}</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td><code>}</code></td>
<td>22</td>
<td>}</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td><code>void Sync(Replicons peers) {</code></td>
<td>24</td>
<td><code>... // Select a node</code></td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td><code>... // UpdateReplicas</code></td>
<td>25</td>
<td><code>}</code></td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td><code>}</code></td>
<td>26</td>
<td>}</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td><code>void Quorum(Str key) {</code></td>
<td>28</td>
<td><code>... // Select a node</code></td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td><code>... // Select a node</code></td>
<td>29</td>
<td><code>}</code></td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>30</td>
<td>}</td>
<td>30</td>
</tr>
</tbody>
</table>

**ClusterOn** + **Distributed application**
Design goals

• Minimize framework overhead
• Enable effective service differentiation
• Realize reusable distributed storage platform components
# Design challenges

<table>
<thead>
<tr>
<th>Diversity of applications</th>
<th>CAP tradeoffs</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Replication policies</td>
<td>- Latency</td>
<td>- None</td>
</tr>
<tr>
<td>- Sharding policies</td>
<td>- Throughput</td>
<td>- Strong</td>
</tr>
<tr>
<td>- Membership management</td>
<td>- Availability</td>
<td>- Eventual</td>
</tr>
<tr>
<td>- Failover recovery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Client connector</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Consistency options:
  - None
  - Strong
  - Eventual
Using ClusterOn

```cpp
void Put(Str key, Obj val) {
    HashTbl.insert(key, val)
}

Obj Get(Str key) {
    return HashTbl(key)
}
```

ClusterOn

Bootstrap info

```
{  
    "Topology": "Master-Slave",  
    "Consistency": "Strong",  
    "Replication": 3,  
    "Sharding": false,  
}
```

Non-dist. kvstore

App. developer

Object store
File system
Database etc...

Metadata server
Replication
Consistency
Topology
Load balancing
Auto scaling
Failover

Datalet

kvs
kvs
kvs
kvs
kvs
kvs
kvs
kvs
ClusterOn architecture

{  
  "Topology": "Master-Slave",
  "Consistency": "Strong",
  "Replication": 3,
  "Sharding": false,
}

Bootstrap info

User request

Reply to user

ClusterOn

Metadata server

Replication

Consistency

Topology

Load balancing

Auto scaling

Failover

Middleware

Application

kvs kvs kvs kvs kvs kvs kvs kvs

Master/slave; Strong consistency; Write request

User request

Reply to user

ClusterOn architecture

```json
{
  "Topology": "Master-Slave",
  "Consistency": "Strong",
  "Replication": 3,
  "Sharding": false
}
```

- **Metadata server**
- **Replication**
- **Consistency**
- **Topology**
- **Failover**

**Bootstrap info**

**ClusterOn**

**Application**

- **Load balancing**
- **Auto scaling**

**Middleware**

- **kvs**
- **M**
- **S**

**Master/slave; Strong consistency; Failover**
Preliminary evaluation

• Proof-of-concept prototype implementation
  – ClusterOn framework (so far: 5000+ lines of C++; not including header files)
    • Event handling, Protobuf
    • Strong consistency with Zookeeper
  – Storage apps
    • Redis: in-memory KV cache/store
    • LevelDB: persistent KV store

• Measure overhead & scalability
Data forwarding overhead: Redis

Throughput (10^3 RPS) vs. Batch size

1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET
Data forwarding overhead: Redis

1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET
Data forwarding overhead: Redis

1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET
Scaling up: LevelDB

1 ClusterOn proxy + N LevelDB backends; Persistent store SATA SSD; 1 replica; 1 million KV requests
Scaling up: LevelDB

1 ClusterOn proxy + N LevelDB backends; Persistent store
SATA SSD; 1 replica; 1 million KV requests
Scaling up: LevelDB

1 ClusterOn proxy + N LevelDB backends; Persistent store SATA SSD; 1 replica; 1 million KV requests
CHUANG, W.-C., SANG, B., YOO, S., GU, R., KULKARNI, M., AND KILLIAN, C. **Eventwave:** Programming model and runtime support for tightly-coupled elastic cloud applications. In *ACM SOCC’13.*


Vsync: Consistent Data Replication for Cloud Computing
https://vsync.codeplex.com/
Summary

• Modern distributed storage applications share a large portion of common functionalities which can be generalized and abstracted.

• ClusterOn can automatically provide core functionalities, such as service distribution, to reduce development effort:
  – Faster realization of new storage applications
  – Easy code maintenance
  – Flexible and extensible service differentiation

• Questions & contact: Ali Anwar, ali@vt.edu
  http://research.cs.vt.edu/dssl/