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

Ali Anwar⋆, Yue Cheng⋆, Hai Huang†, and Ali R. Butt⋆
⋆Virginia Tech, †IBM Research – TJ Watson

1) Background

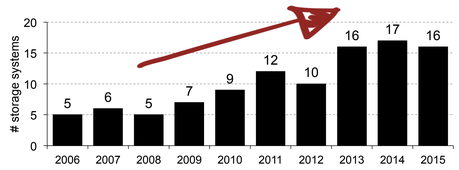
Growing data storage needs is driving the development of an increasing number of distributed storage applications

K-V store: BigTable, Redis, Memcached, DynamoDB

Object store: Swift, Amazon S3

NoSQL DB: Cassandra, MongoDB, HyperTable, HyperDex

DFS: HDFS, LustreFS



Number of storage systems papers in SOSP, OSDI, ATC and EuroSys conferences in the last decade (2006–2015)

2) Motivation

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

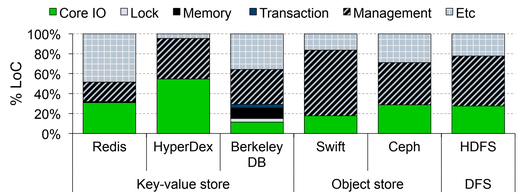
Case Study: Redis V-3.0.1

```
[haih@localhost src]$ ls -al | sort -k 5 -n | tail -n 10
-rw-rw-r-- 1 haih haih  59937 May  5 05:01 aof.c
-rw-rw-r-- 1 haih haih  63585 May  5 05:01 networking.c
-rw-rw-r-- 1 haih haih  69734 May  5 05:01 redis.h
-rw-rw-r-- 1 haih haih  75651 May  5 05:01 redis-cli.c
-rw-rw-r-- 1 haih haih  84363 May  5 05:01 config.c
-rw-rw-r-- 1 haih haih  84703 May  5 05:01 replication.c
-rw-rw-r-- 1 haih haih  93605 May  5 05:01 t_zset.c
-rw-rw-r-- 1 haih haih 146723 May  5 05:01 redis.c
-rw-rw-r-- 1 haih haih 151847 May  5 05:01 sentinel.c
-rw-rw-r-- 1 haih haih 199190 May  5 05:01 cluster.c
[haih@localhost src]$
```

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)

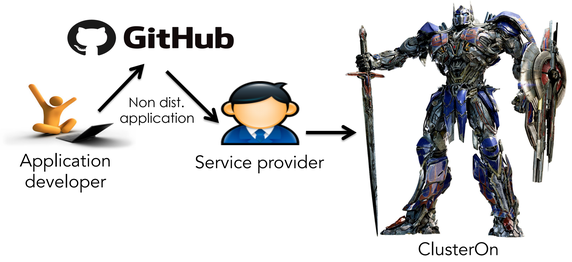
3) Quantifying LoC



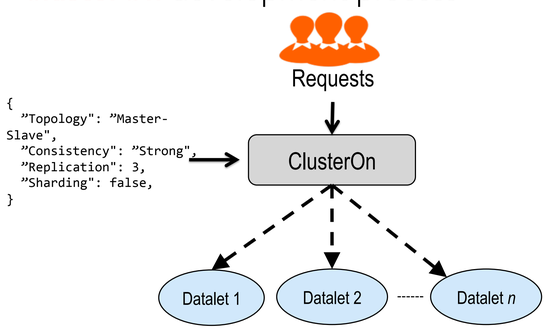
Everything other than CoreIO is **"Messy Plumbing"**

When developing a new distributed storage app, these features can be abstracted and generalized

4) ClusterOn



ClusterOn development process



Datalet = single instance of non dist. app

5) Developing a distributed storage application

(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   ... // Acquirelock
18 }
19
20 void Unlock(Str key) {
21   ... // Releaselock
22 }
23
24 void Sync(Replicas peers) {
25   ... // Updatereplicas
26 }
27
28 void Quorum(Str key) {
29 ... // Select a node
30 }
```

(b) Zookeeper-based

```
1 void Put(Str key, Obj val) {
2   if (this.master) :
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   ... // Updatereplicas
18 }
19
20 void Quorum(Str key) {
21 ... // Select a node
22 }
```

(c) Vsync

```
1 #include <vsync lib>
2
3  void Put(Str key, Obj val) {
4    if (this.master):
5      zk.Lock(key) // zookeeper
6      HashTbl.insert(key, val)
7      zk.Unlock(key) // zookeeper
8   Vsync.Sync(master.slaves)
9 }
10
11 Obj Get (Str key)
12   if (this.master):
13     Obj val = Vsync.Quorum(key)
14   Vsync.Sync(master.slaves)
15   return val
16 }
```

(d) ClusterOn-based

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

Design Goal

- Minimize framework overhead
- More effective service differentiation
- Reusable distributed storage platform

Diversity of applications

- Replication policies
- Sharding policies
- Membership management
- Failover recovery
- Client connector

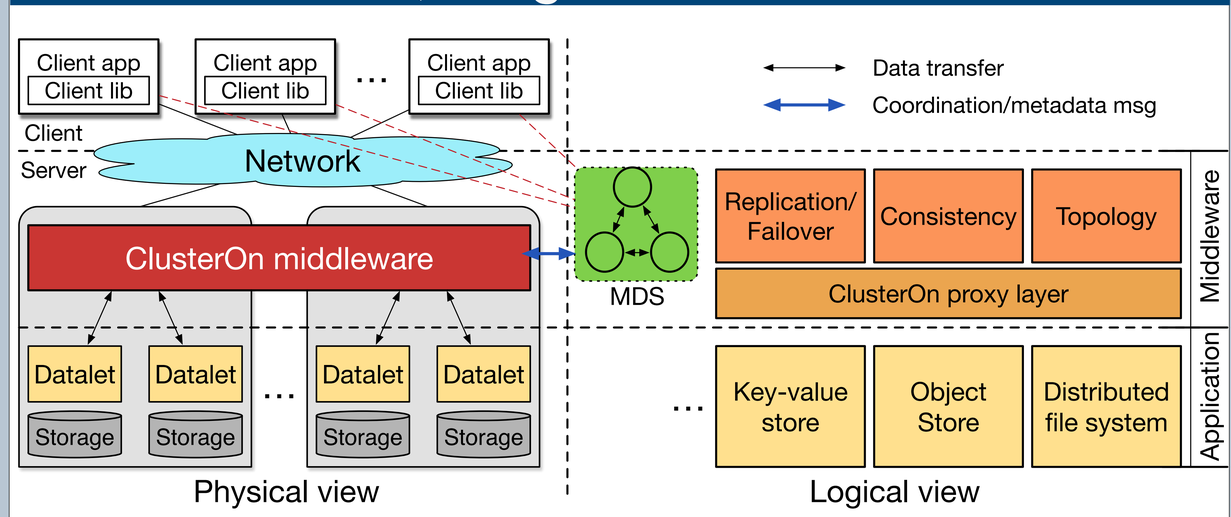
CAP Tradeoffs

- Latency
- Throughput
- Availability

Consistency

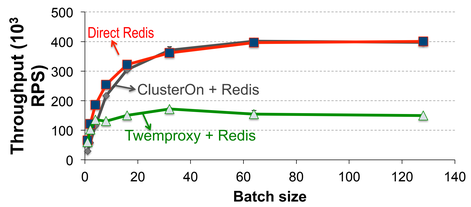
- None
- Strong
- Eventual

6) Design architecture



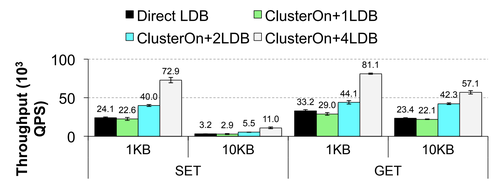
7) Results

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