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

Object store

Swift

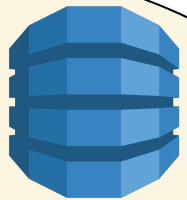
Amazon S3

K-V store

BigTable

Redis

Memcached

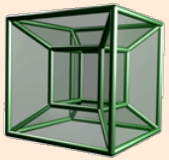


DynamoDB

NoSQL DB

Cassandra

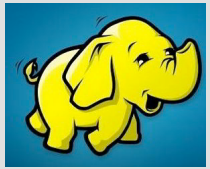
MongoDB



HyperTable

HyperDex

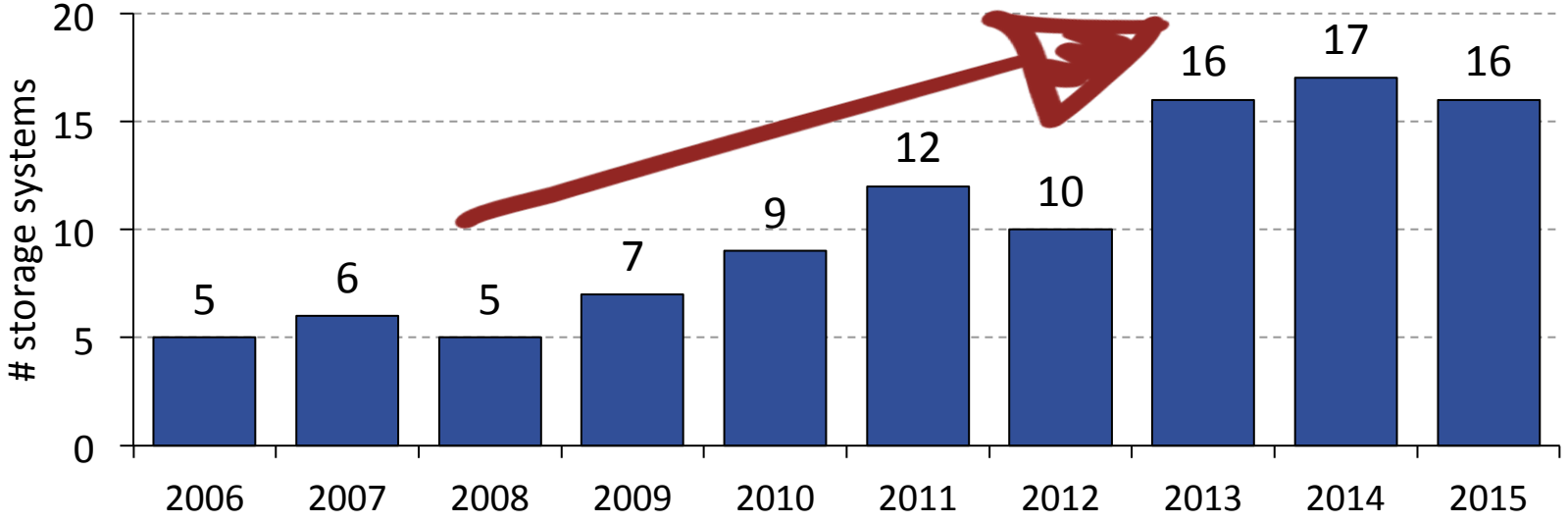
DFS



HDFS

LustreFS

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!

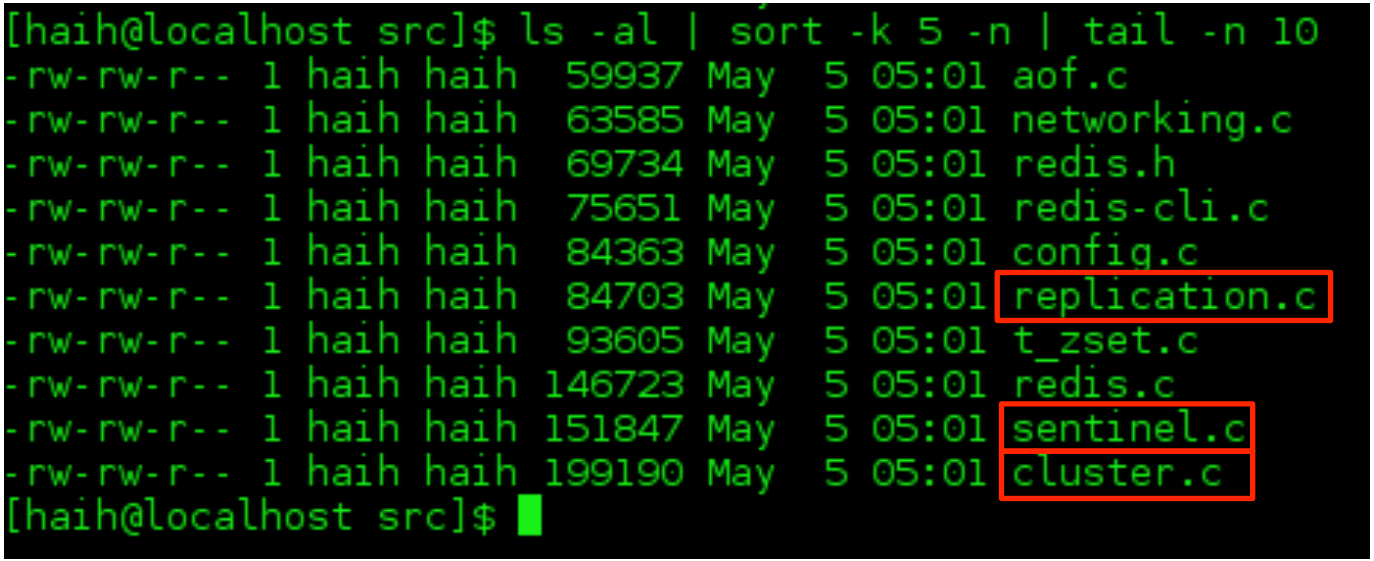

DISTRIBUTED STORAGE
IS
COMING

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

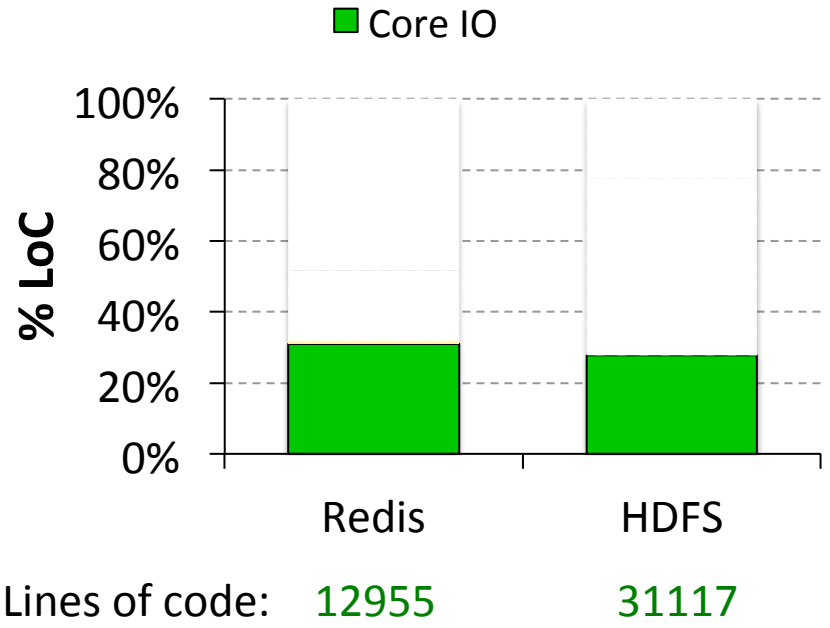


replication.c – replicate data from master to slave, or slave to slave
sentinel.c – monitoring nodes, handle failover from master to slave
cluster.c – support cluster mode with multiple masters/shards

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

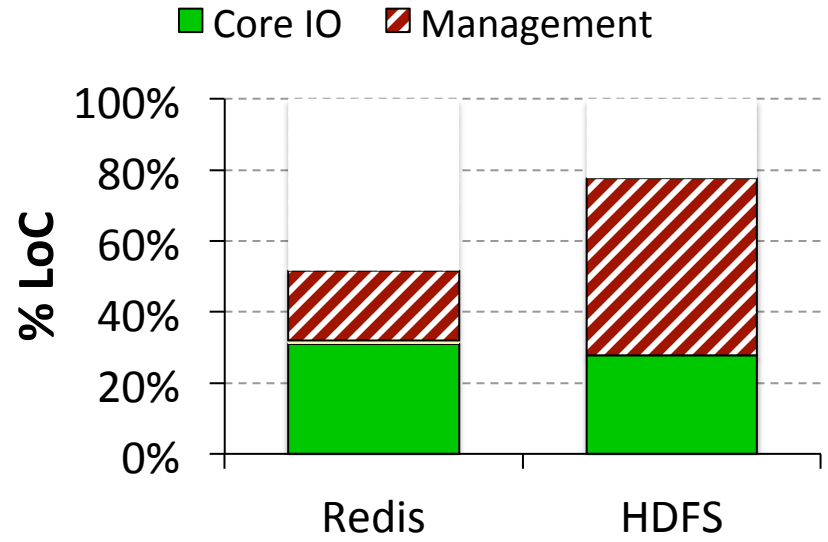
Quantifying LoC

- Core IO



Quantifying LoC

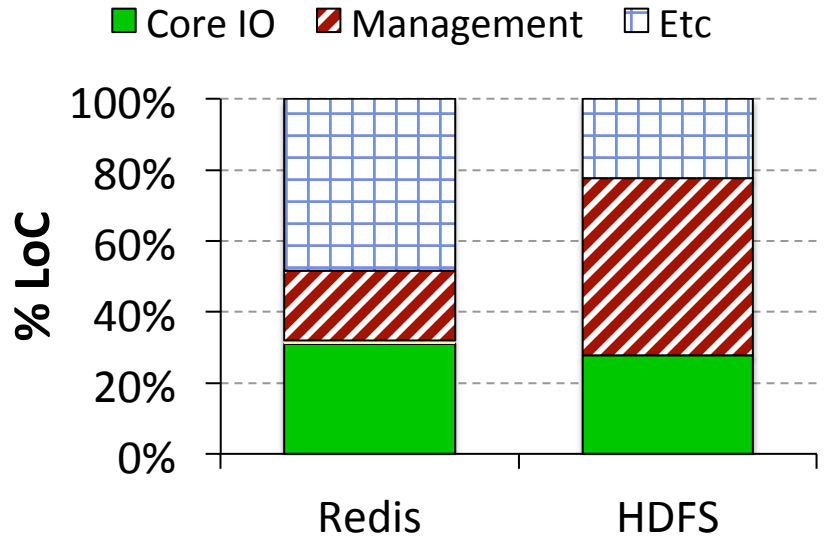
- Core IO
- Distributed management



Lines of code:	Redis	HDFS
Core IO	12955	31117
Management	8237	56208

Quantifying LoC

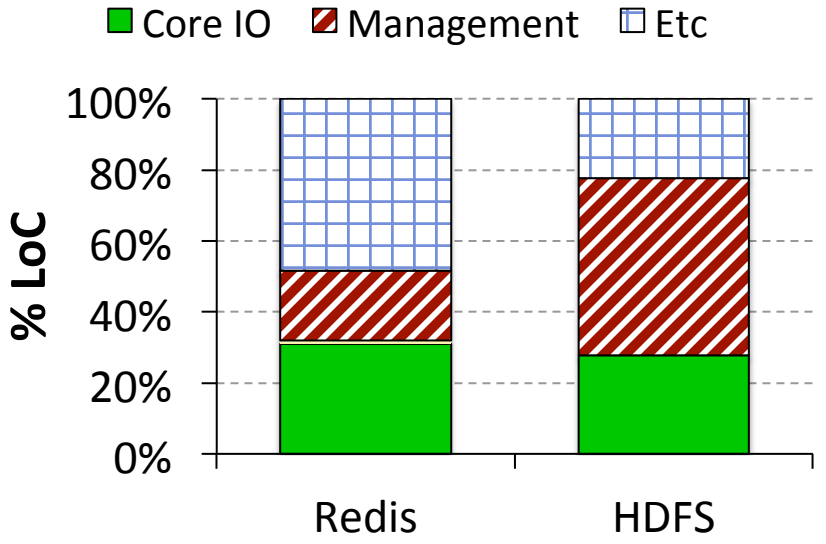
- Core IO
- Distributed management
- Etc: Config/auth/stats/ compatibility …



Lines of code:	Redis	HDFS
Core IO	12955	31117
Management	8237	56208
Etc	20202	25185

Quantifying LoC

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



Lines of code:	12955	31117
	8237	56208
	20202	25185

Abstracting and generalizing *common features/functionality* can simplify development of new distributed storage applications

CLUSTERON

Steps involved in 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 }
```

(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 }
```

(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 }
```

(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

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

+

ClusterOn

=

Distributed application

Design goals

- Minimize framework overhead
- Enable effective service differentiation
- Realize reusable distributed storage platform components

Design challenges

Diversity of applications

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

CAP tradeoffs

- Latency
- Throughput
- Availability

Consistency

- None
- Strong
- Eventual

Using ClusterOn

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

Non-dist. kvstore

App. developer

kvs

Object store
File system
Database
etc...

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

Bootstrap info

ClusterOn

Metadata server

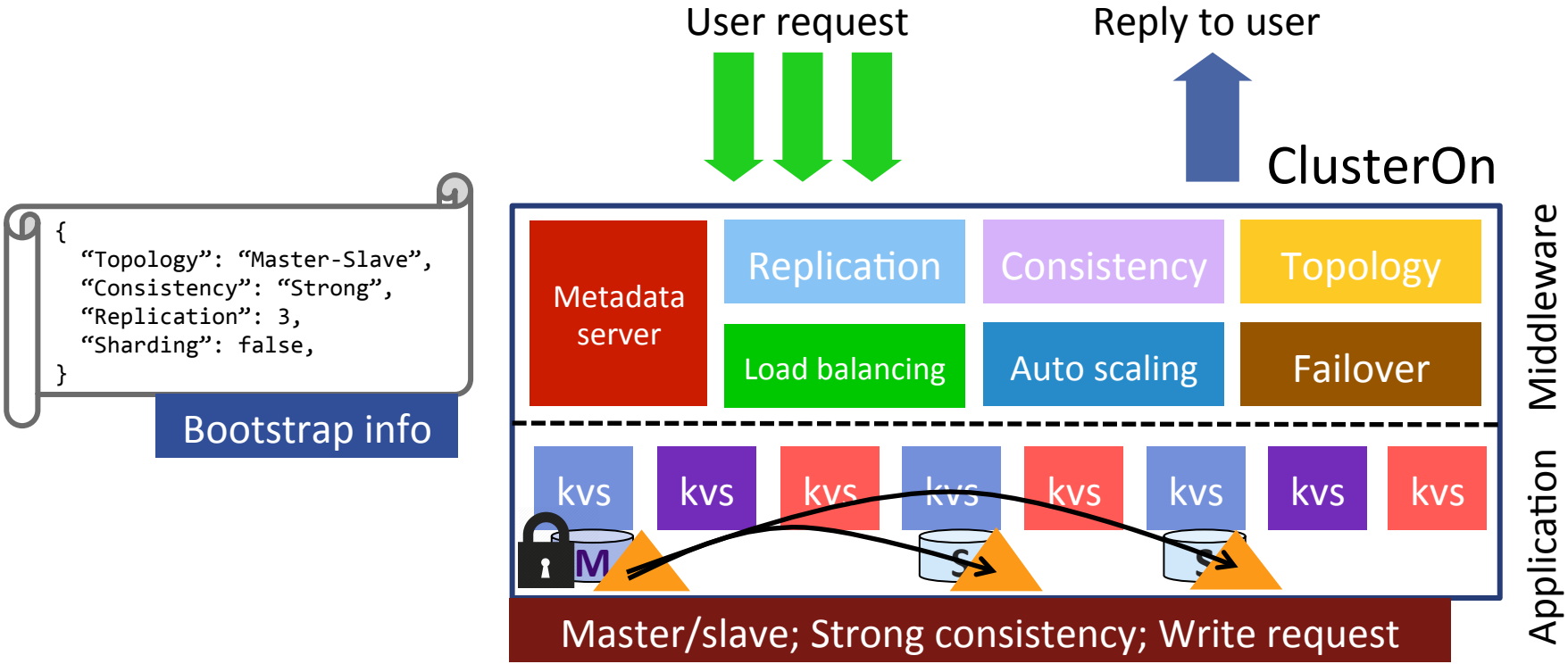
Replication | Consistency | Topology

Load balancing | Auto scaling | Failover

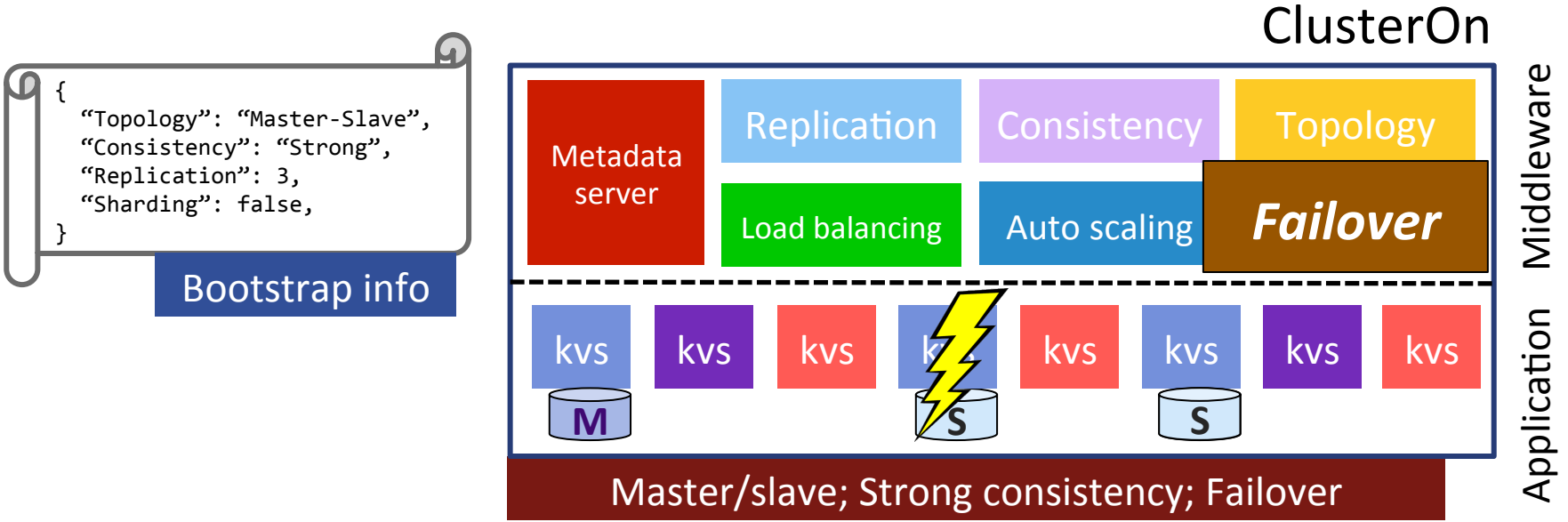
kvs kvs kvs kvs kvs kvs kvs

Datalet

ClusterOn architecture



ClusterOn architecture

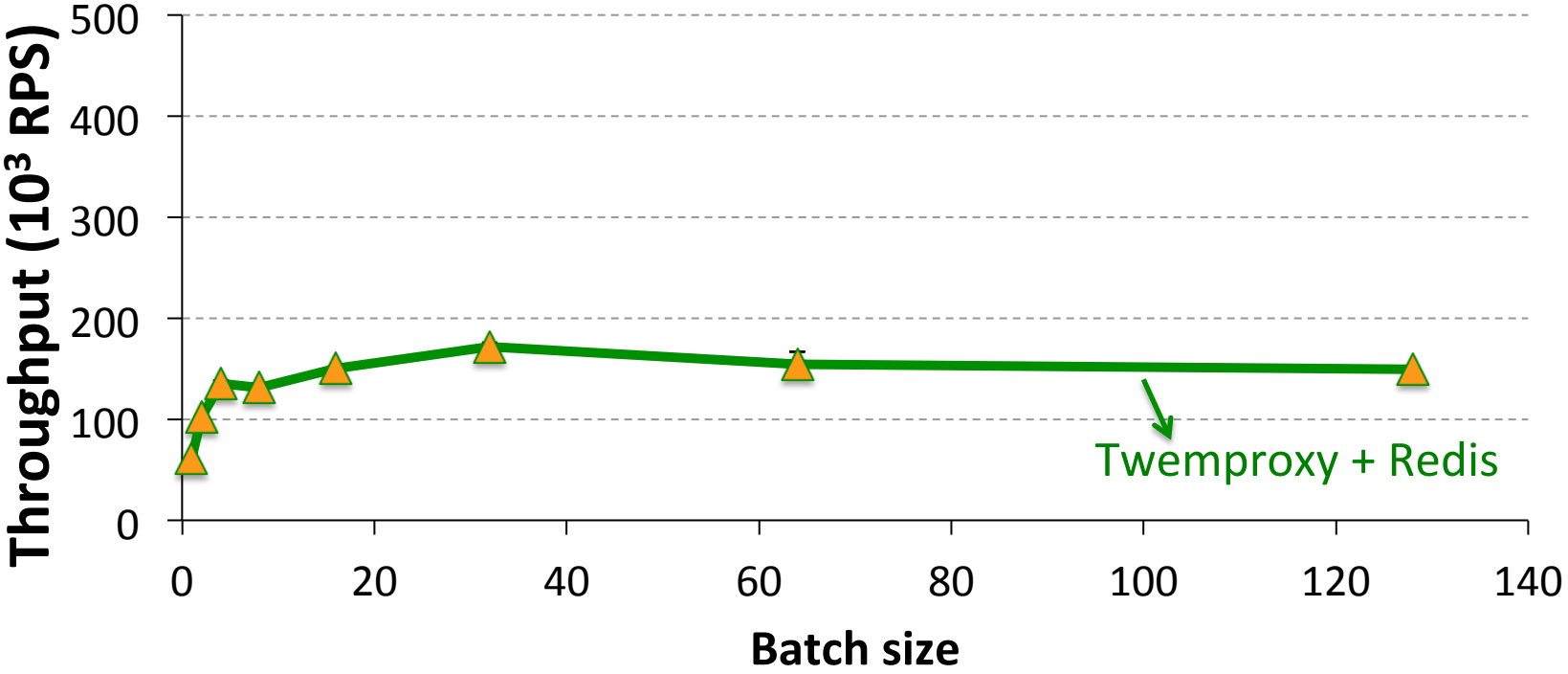


Preliminary evaluation

- 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

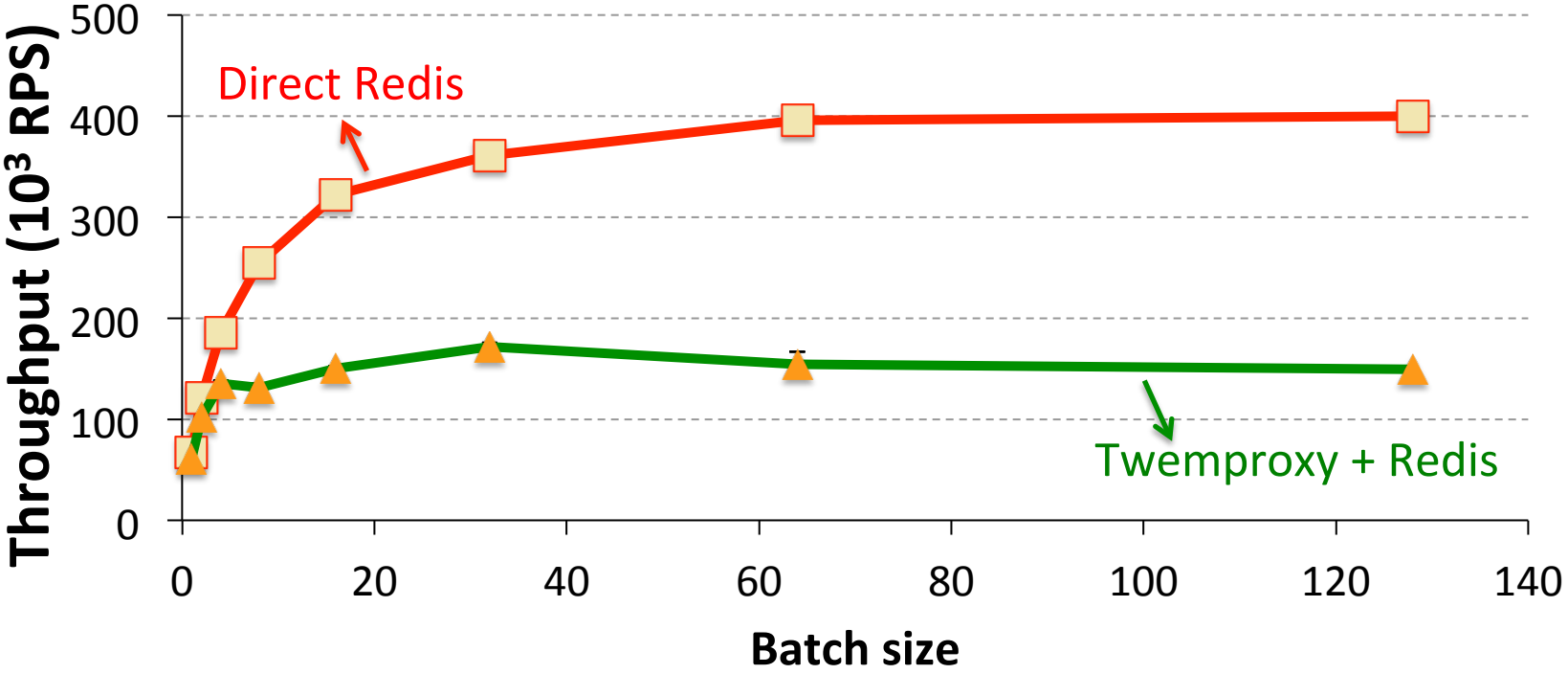
- Measure overhead & scalability

Data forwarding overhead: Redis



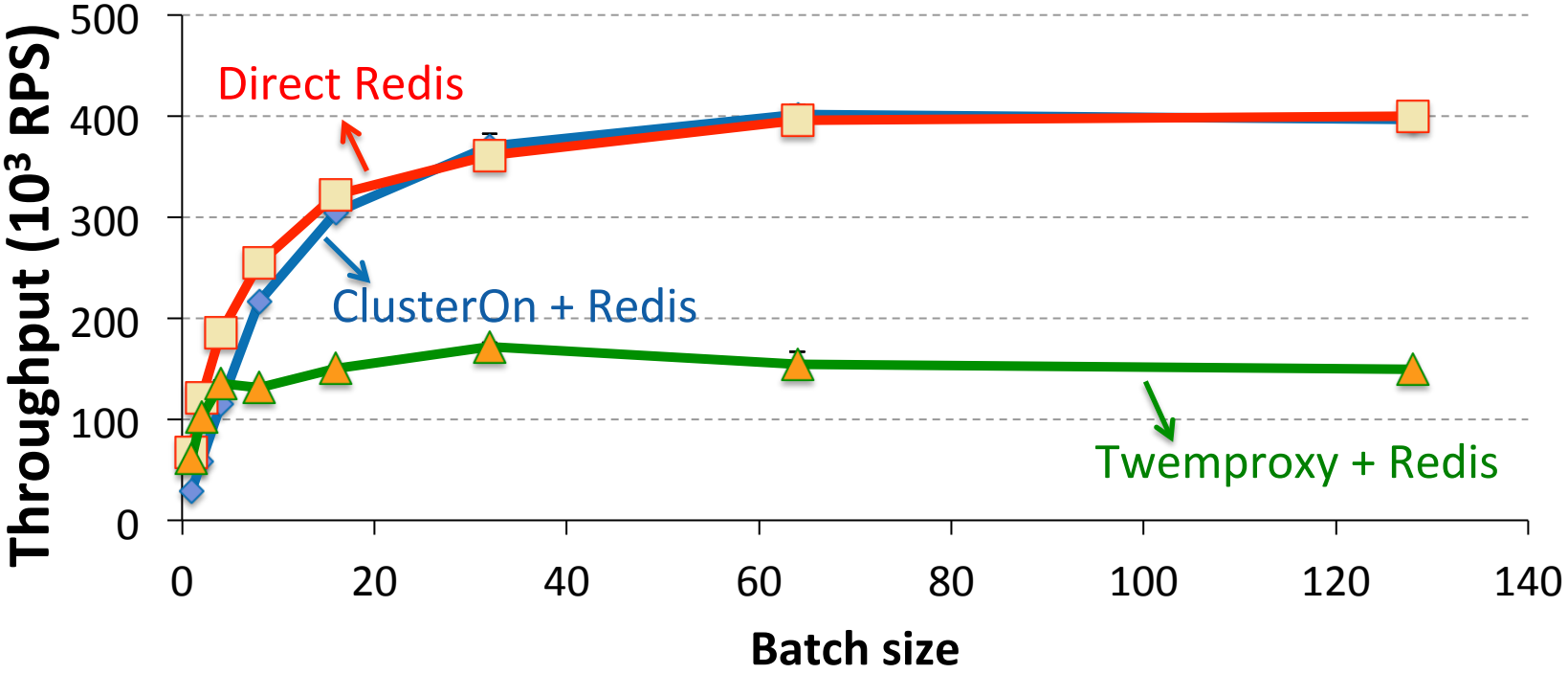
1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET

Data forwarding overhead: Redis



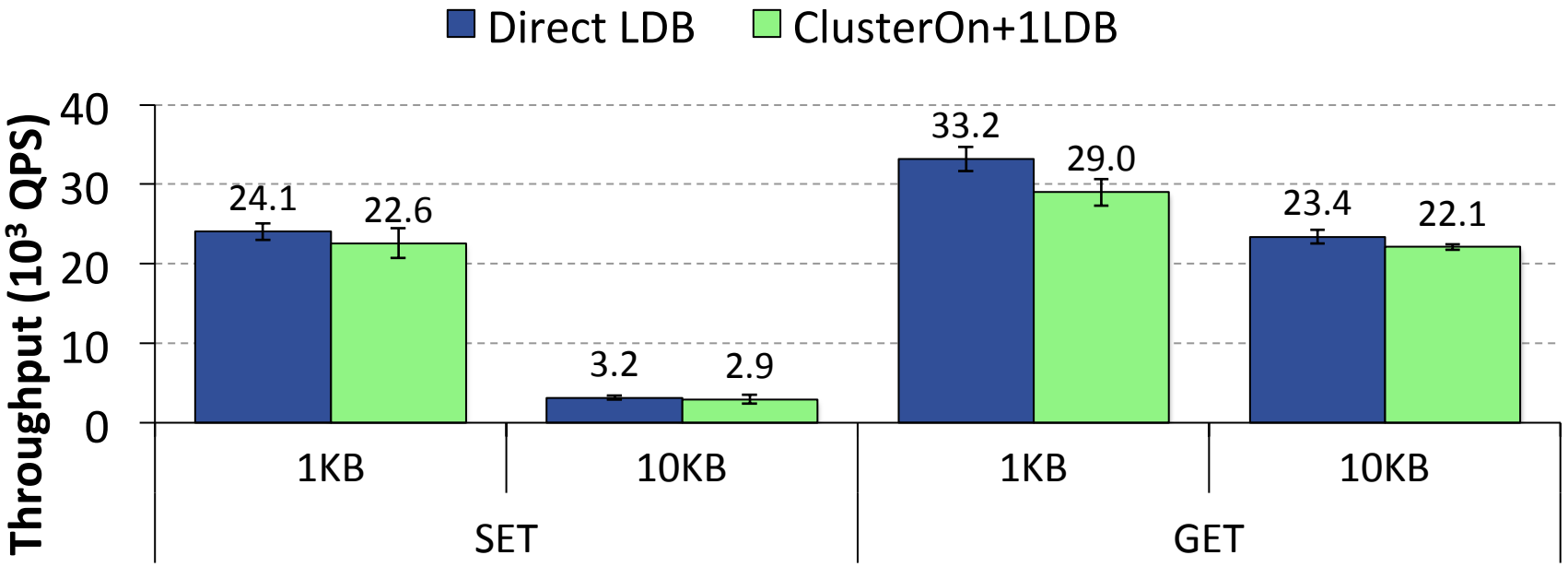
1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET

Data forwarding overhead: Redis



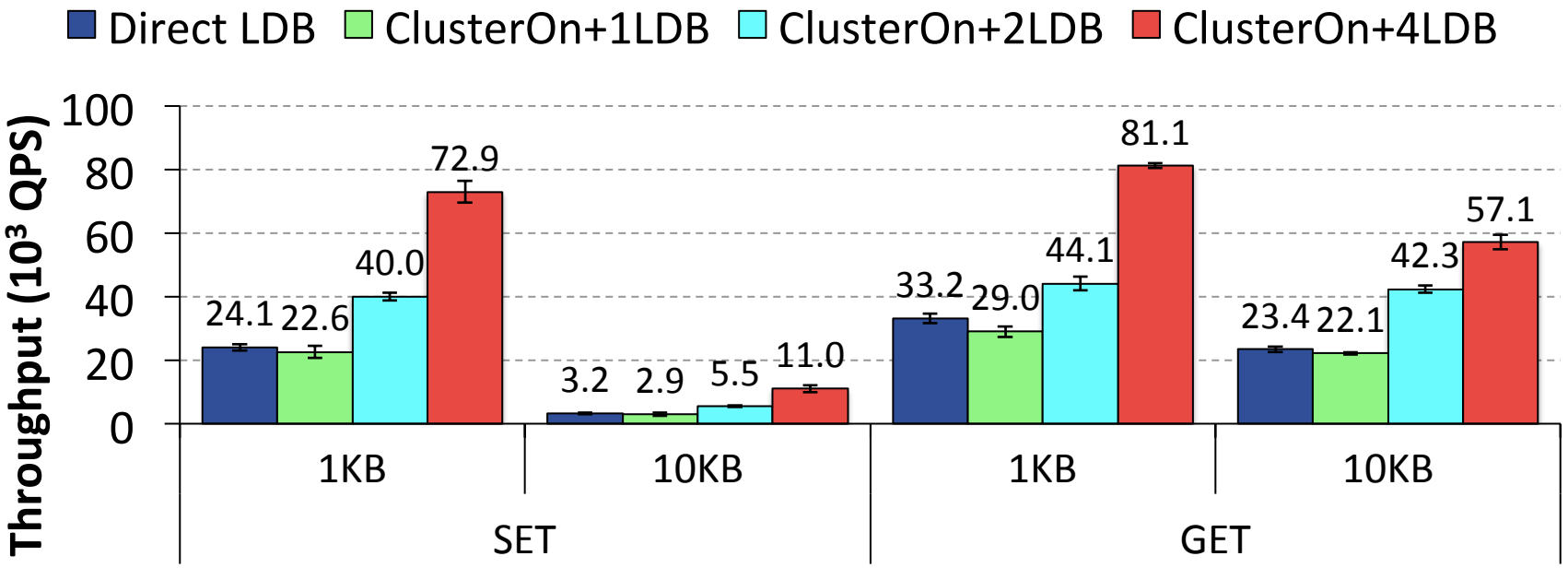
1 ClusterOn proxy + 1 Redis backend; Memory cache
16B key 32B value; 10 millions KV requests; YCSB: 100% GET

Scaling up: LevelDB



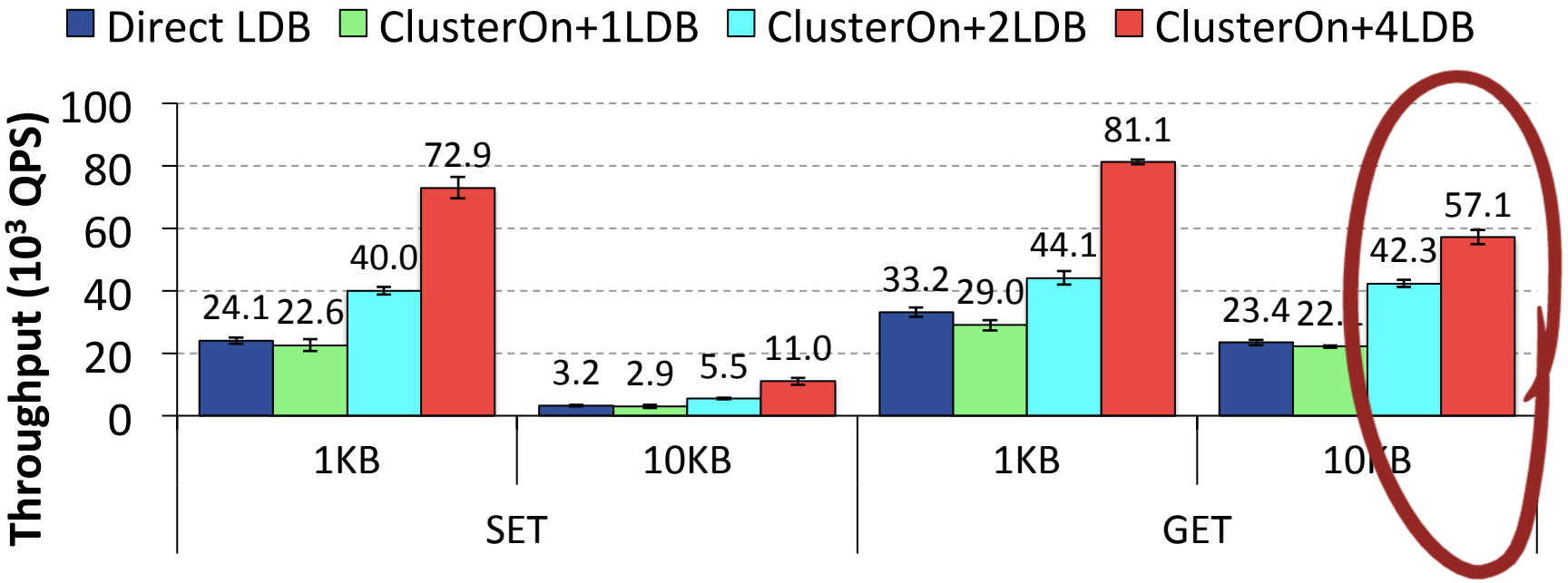
1 ClusterOn proxy + N LevelDB backends; Persistent store
SATA SSD; 1 replica; 1 million KV requests

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

Related work

MESOS

kubernetes

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**.

HUNT, G. C., SCOTT, M. L., ET AL. The coign automatic distributed partitioning system. In **USENIX OSDI'99**.

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/