Modular Composition of Coordination Services

Kfir Lev-Ari[1], Edward Bortnikov[2], Idit Keidar[1,2], and Alexander Shraer[3]

Technion Israel Institute of Technology

YAHOO! RESEARCH

Google

Coordination Services

Apache Zookeeper

etcd

Doozer

CONSUL

Used for configuration & metadata storage, global locks, leader election, service discovery, and more…

Who uses coordination services?

Google

hadoop

facebook

pied piper

in

airbnb

Goldman Sachs

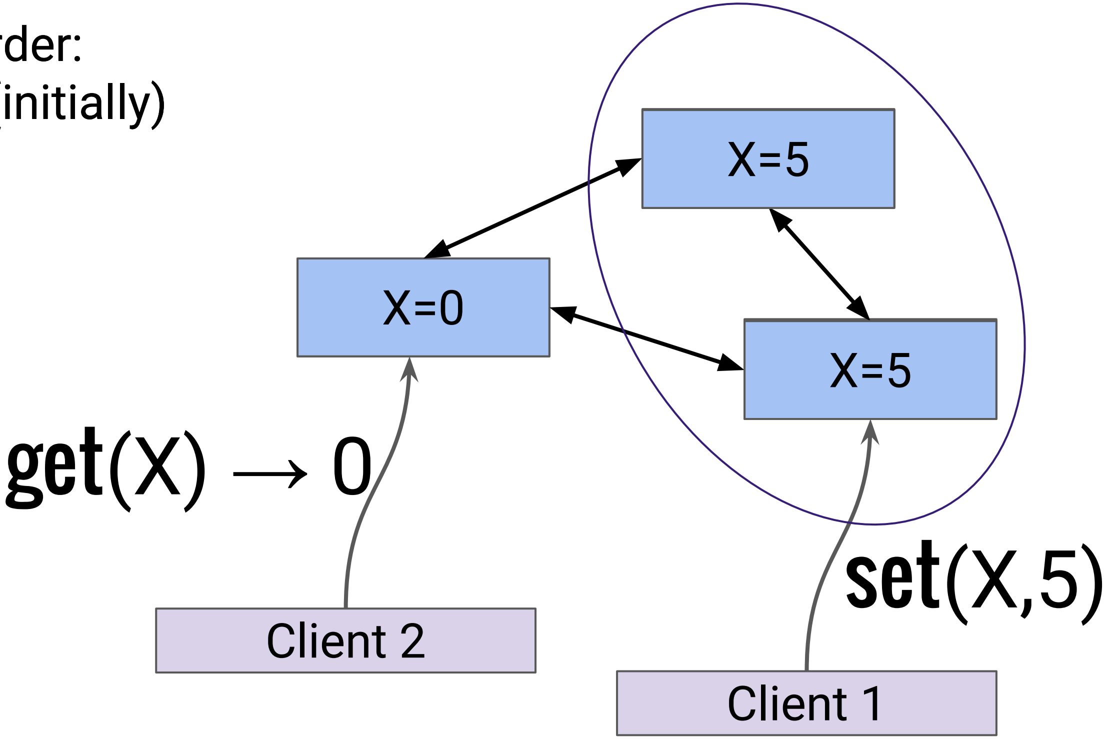
IBM

YAHOO!

Coordination Services Structure

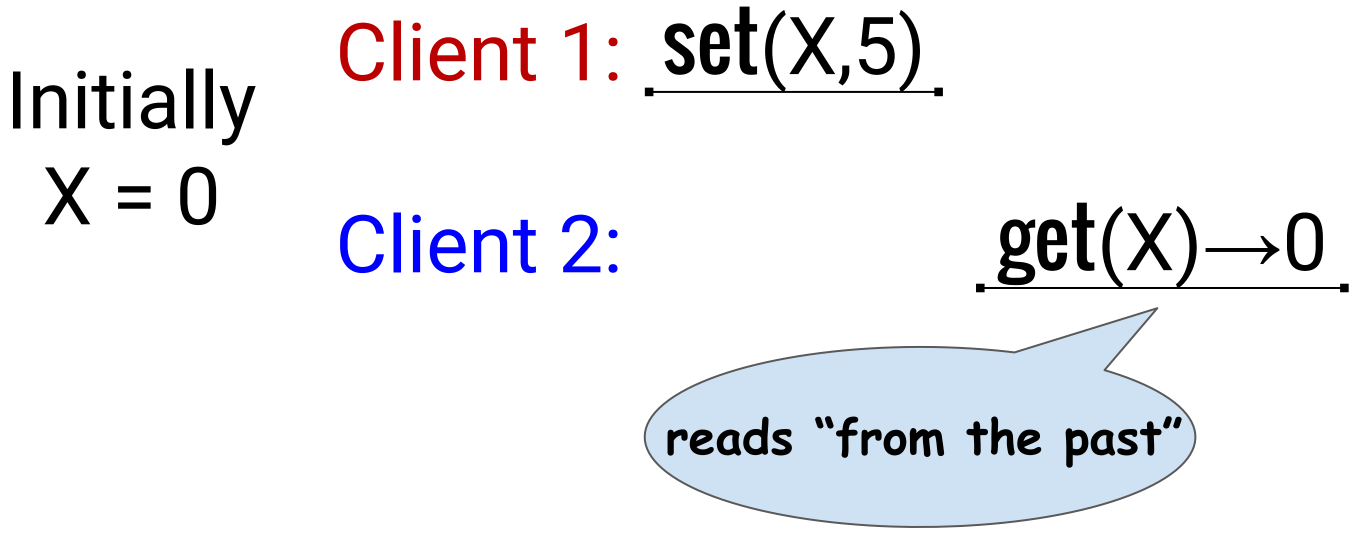
Updates order:

1. X = 0 (initially)
2. X = 5

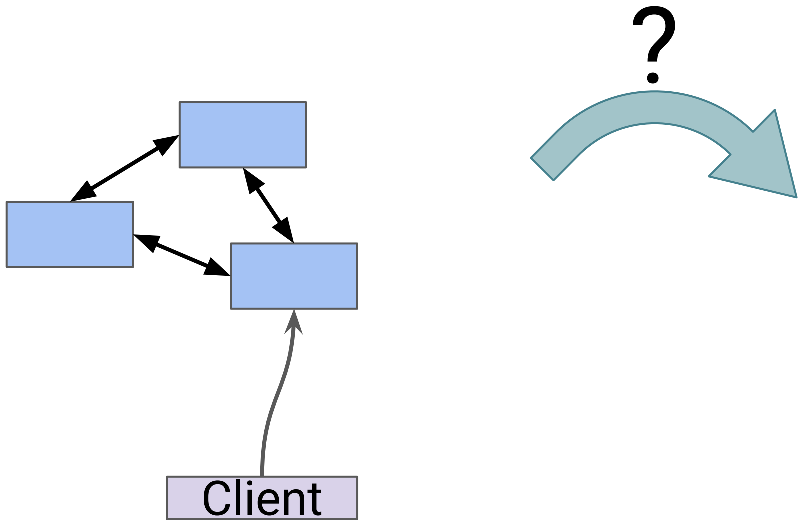


Coordination Services Semantics

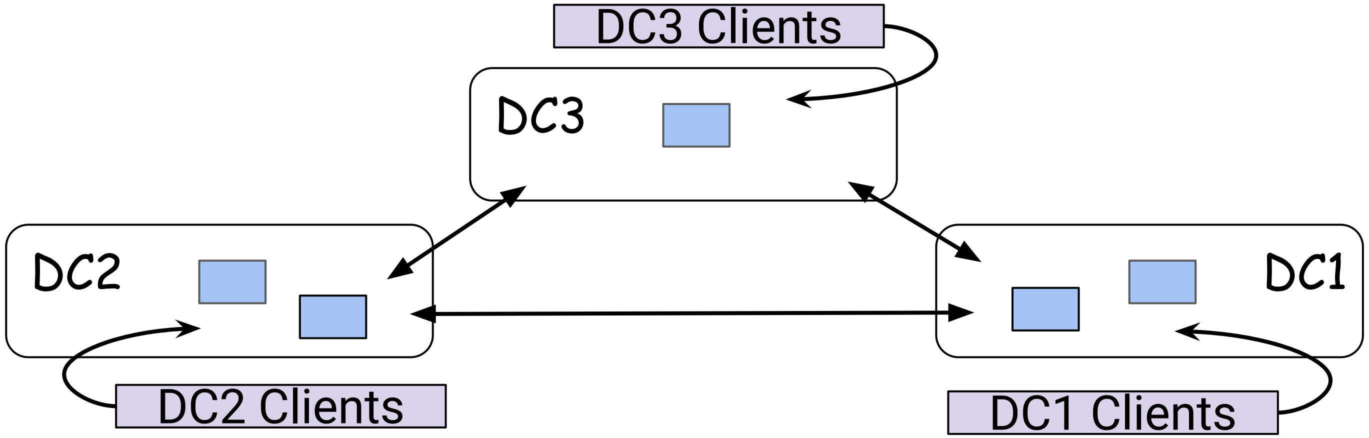
1. Clients see the same order of updates (linearizable updates)
2. Reads might be served from the past



Challenge: Coordination Service over WAN

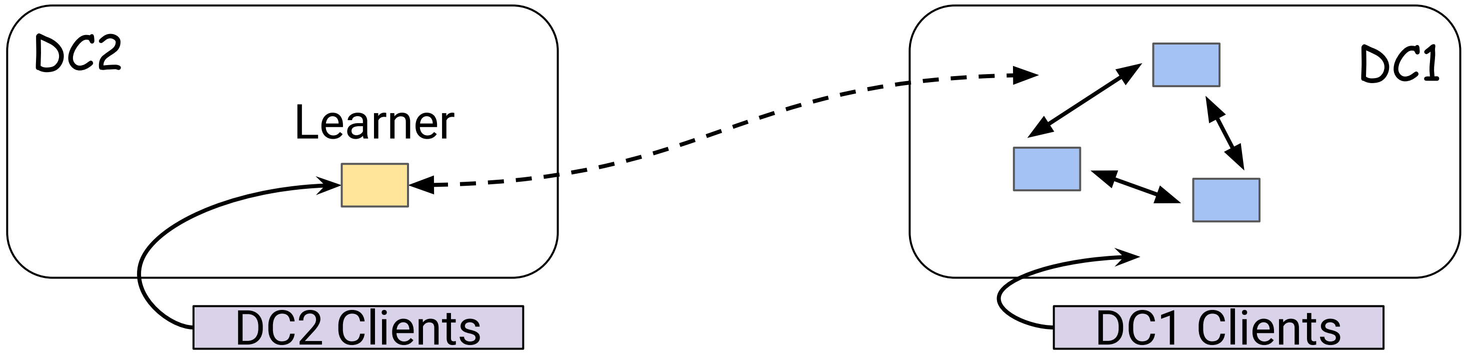


Coordination Services over WAN



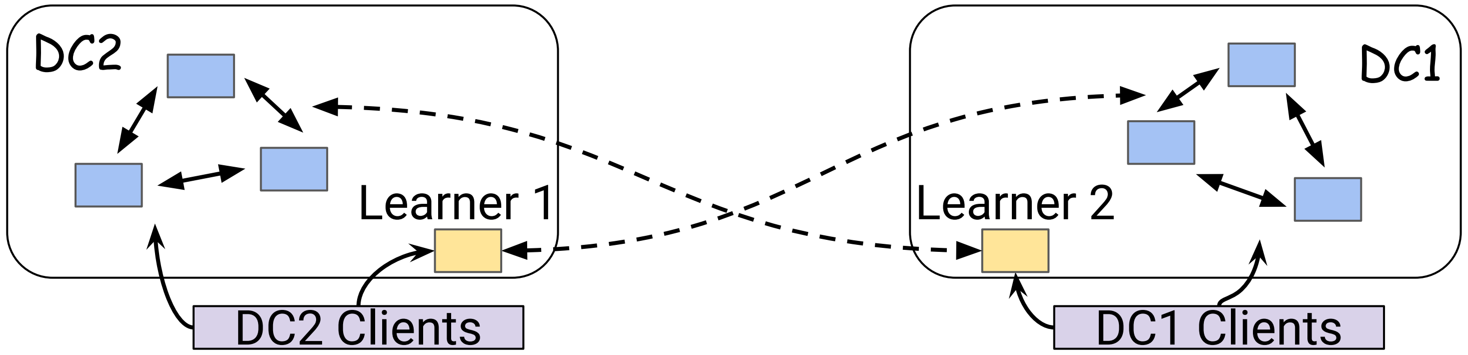
	Updates	Reads	Correctness	Example
Distributed Service	Very slow	Fast	Yes	ACMS, Zeus, Megastore

Coordination Services over WAN



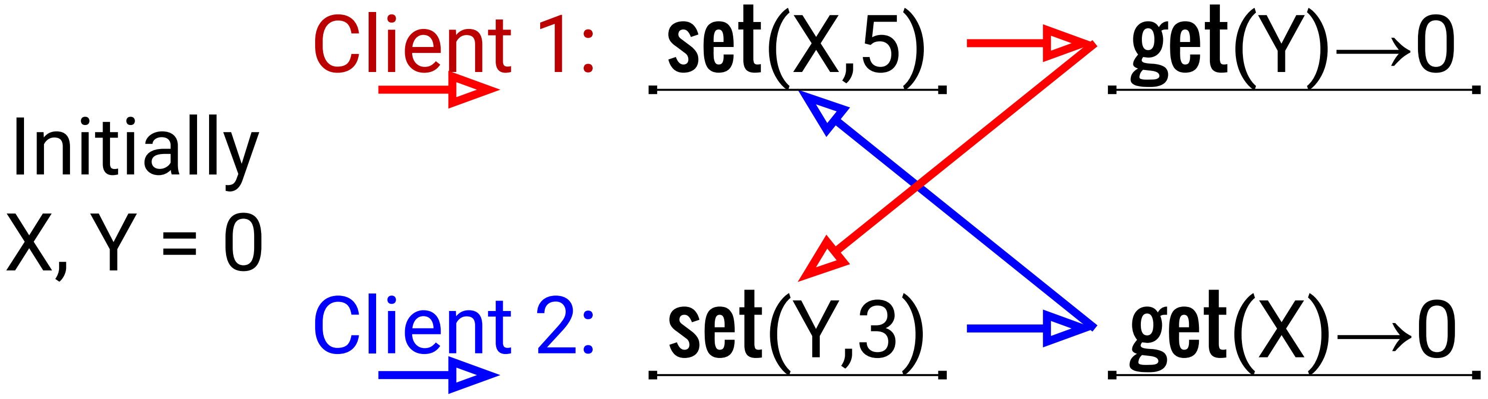
	Updates	Reads	Correctness	Example
Distributed Service	Very slow	Fast	Yes	ACMS, Zeus, Megastore
Co-located Service + Learners	Slow	Fast	Yes	ZooKeeper, Consul

Coordination Services over WAN



	Updates	Reads	Correctness	Example
Distributed Service	Very slow	Fast	Yes	ACMS, Zeus, Megastore
Co-located Service + Learners	Slow	Fast	Yes	ZooKeeper, Consul
Multiple Co-located services + Learners	Fast	Fast	No	Global service discovery

Multiple Services Deployment - Correctness

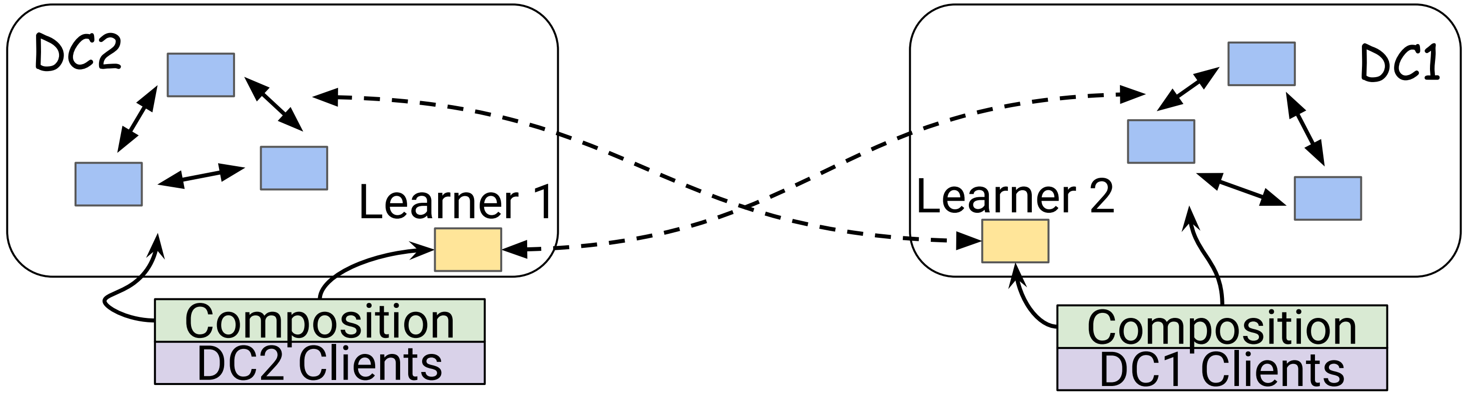


- Clients see different order of updates:

  Client 1: x=0 →x=5 →y=0 →y=3

  Client 2: y=0 →y=3 →x=0 →x=5

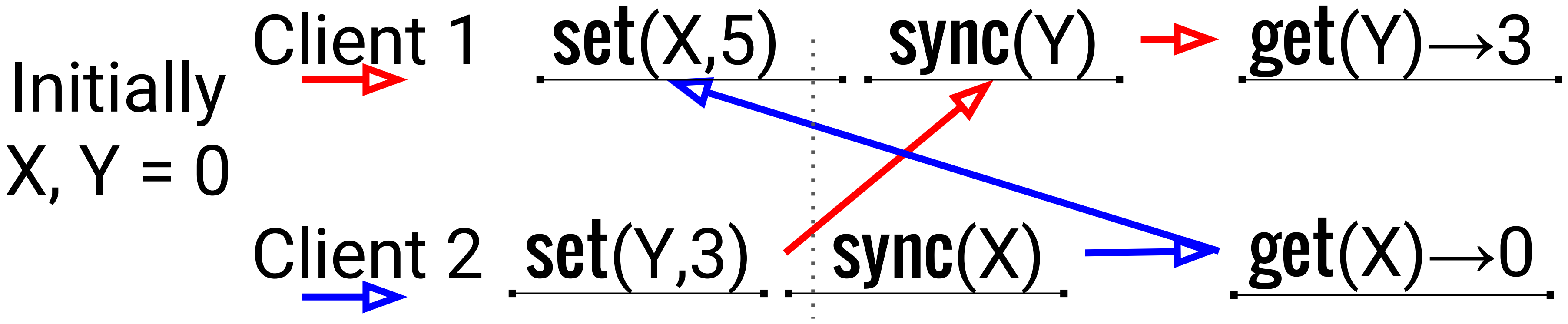
Our Solution: Modular Composition



	Updates	Reads	Correctness	Example
Distributed Service	Very slow	Fast	Yes	ACMS, Zeus, Megastore
Co-located Service + Learners	Slow	Fast	Yes	ZooKeeper, Consul
Multiple Co-located services + Learners	Fast	Fast	No	Global service discovery
Modular Composition	Fast	Fast	Yes	Our implementation: ZooNet

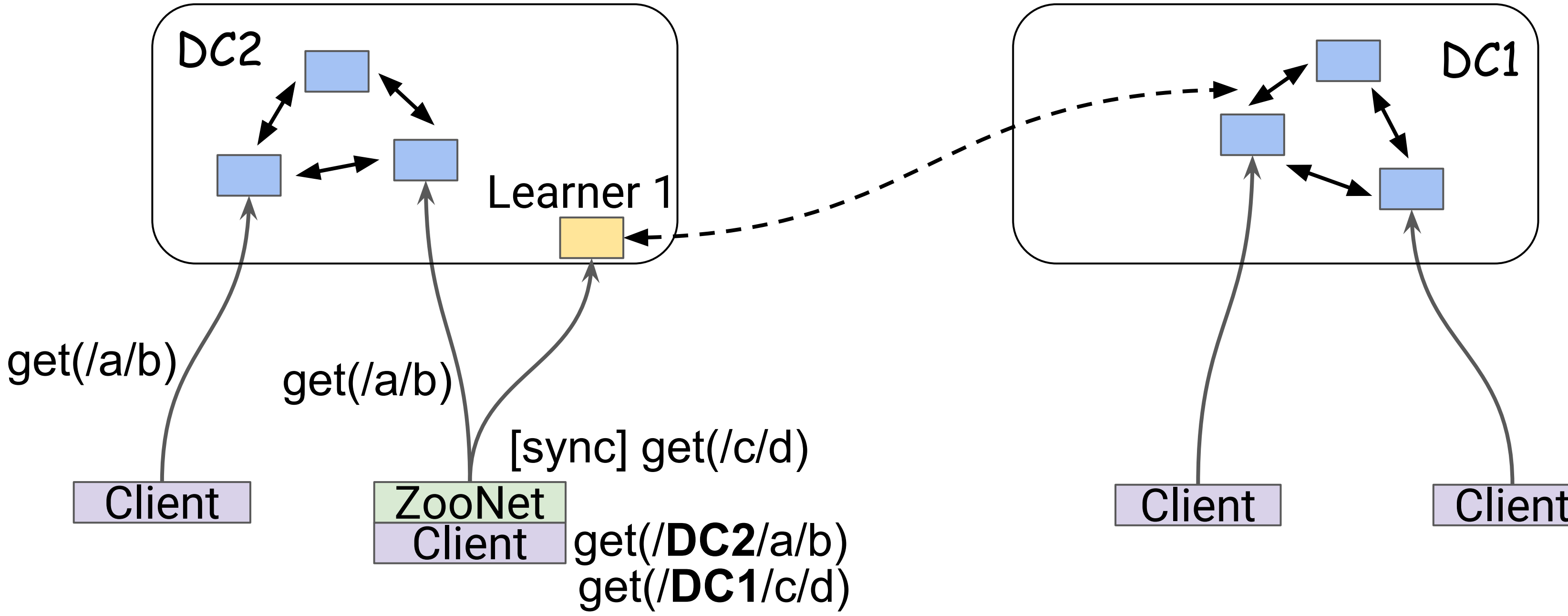
Modular Composition - Algorithm

Linearizable operation (sync) upon switching service instance

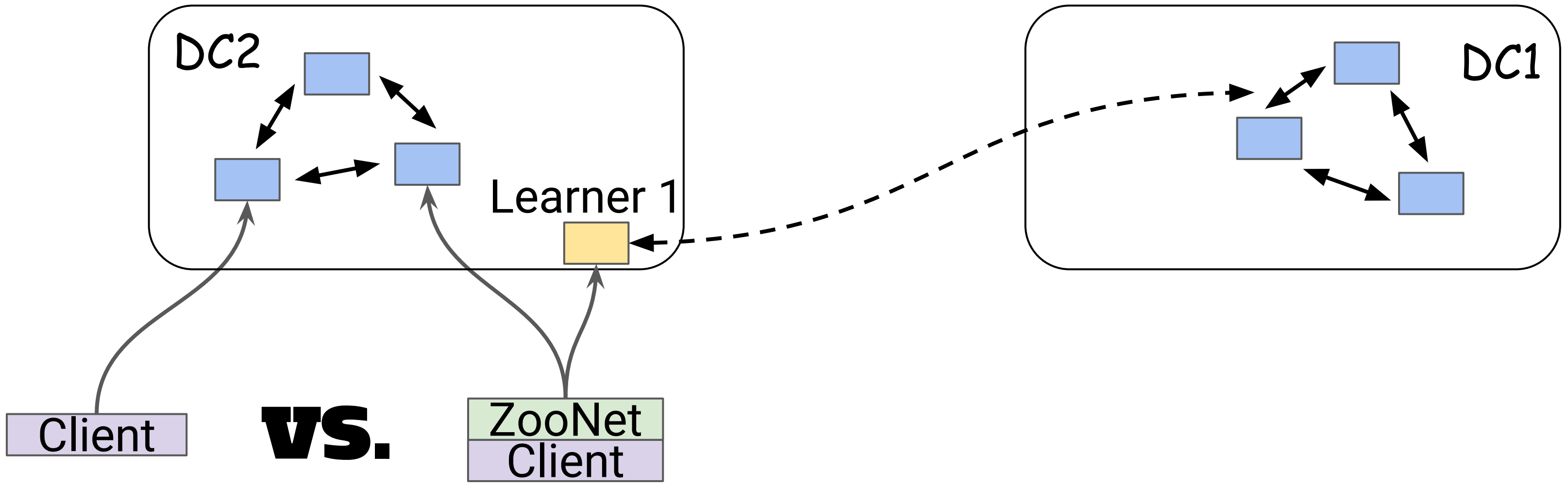


- Clients see same order of updates :
  y=0 → y=3 → x=0 → x=5

ZooNet - Modular Composition of ZooKeepers



ZooNet - Cost of Consistency

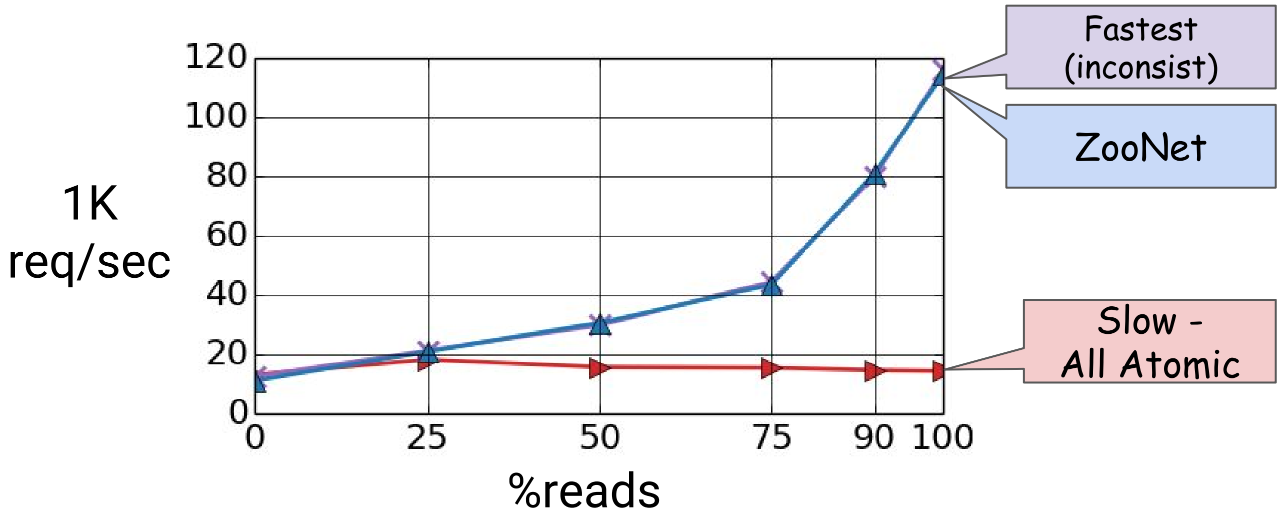


Vary locality:

- Spatial: % local access
- Temporal: # consecutive accesses to same DC

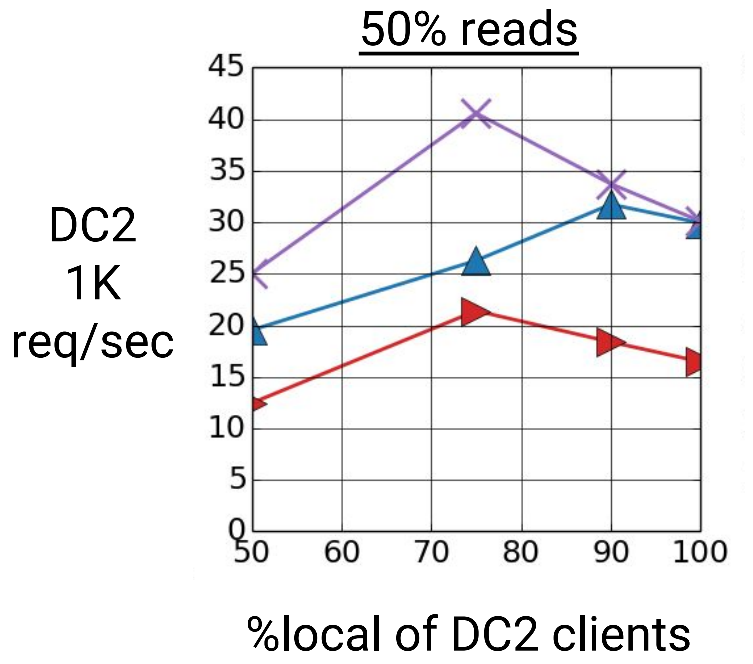
ZooNet Evaluation - Cost of Consistency

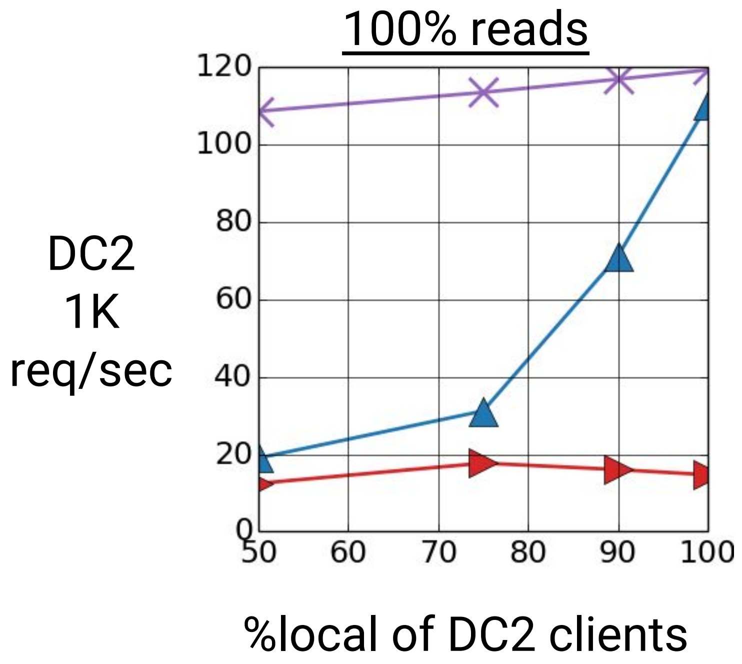
100% spatial locality:



ZooNet Evaluation - Cost of Consistency

No temporal locality, varying spatial locality:





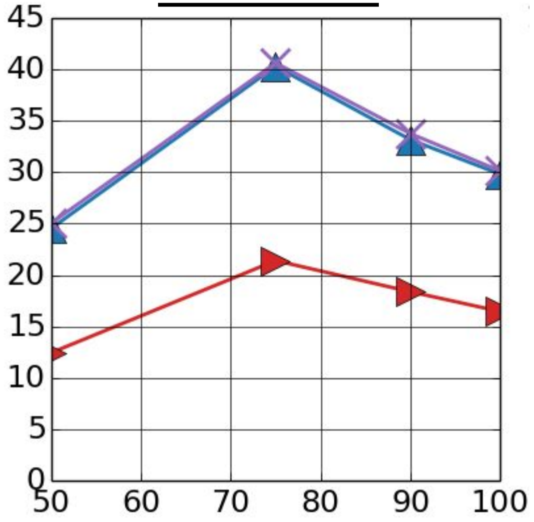
Fastest (inconsist)
ZooNet
Slow - All Atomic

ZooNet Evaluation - Cost of Consistency

With temporal locality, varying spatial locality:

50% reads

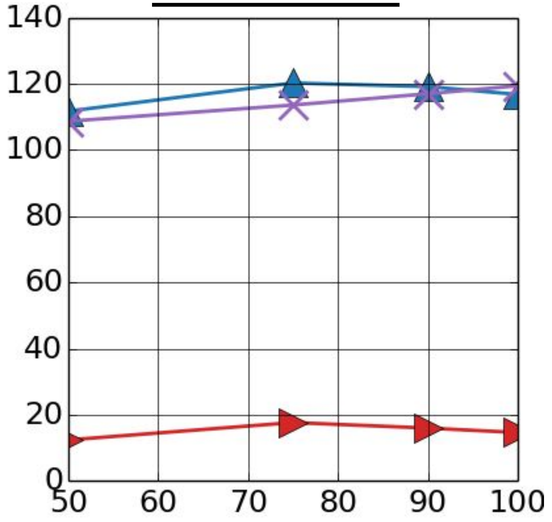
DC2 1K req/sec



%local of DC2 clients

100% reads

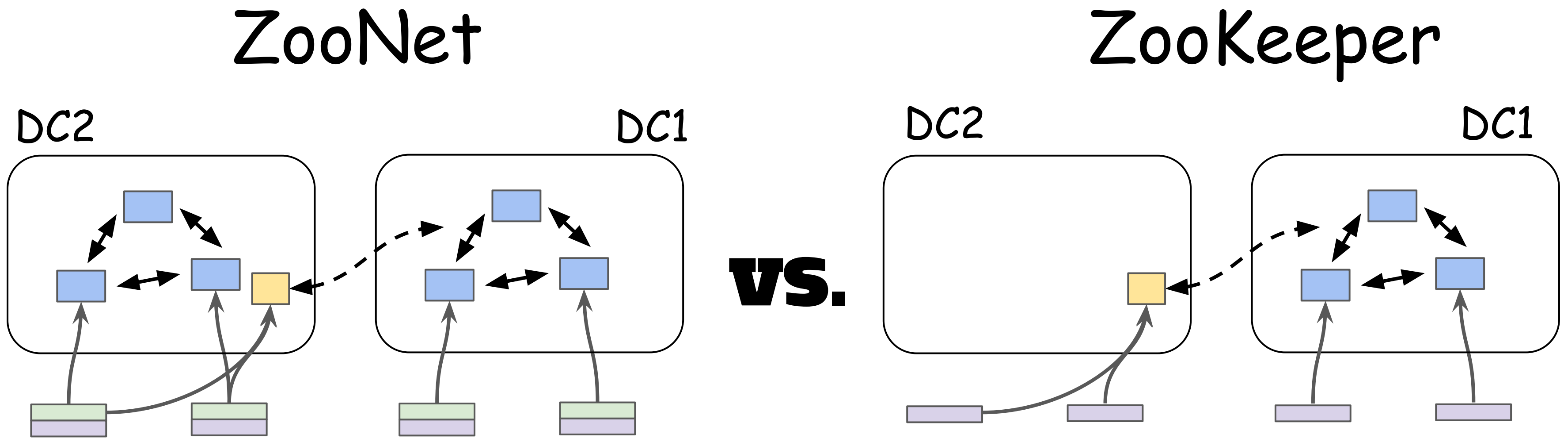
DC2 1K req/sec



%local of DC2 clients

Fastest (inconsist)
ZooNet
Slow - All Atomic

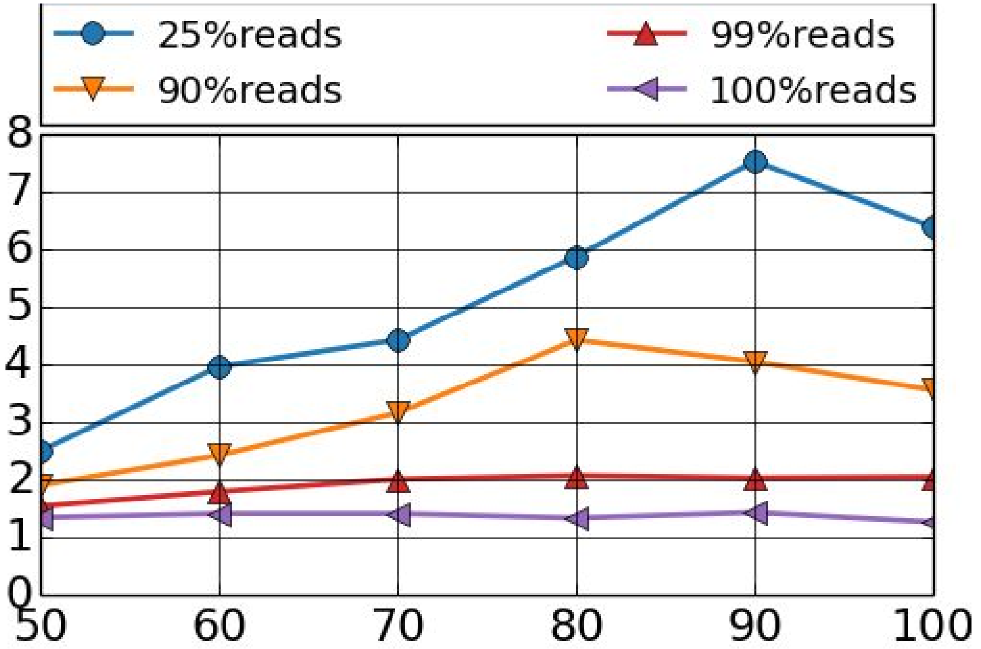
ZooNet vs. ZooKeeper Evaluation



ZooNet vs. ZooKeeper Evaluation

With temporal locality, varying spatial locality of DC2 clients:

DC2
Throughput
Speedup
ZooNet/ZooKeeper



%local of DC2 clients

Zookeeper - Server Side Improvement

- We improved ZooKeeper:
  - **Performance** - reads blocked for no reason
  - **Starvation** in read-intensive workloads

- In a nutshell:
  - 2 clients connecting to same server blocked each other
  - Not required by semantics
  - We isolated clients

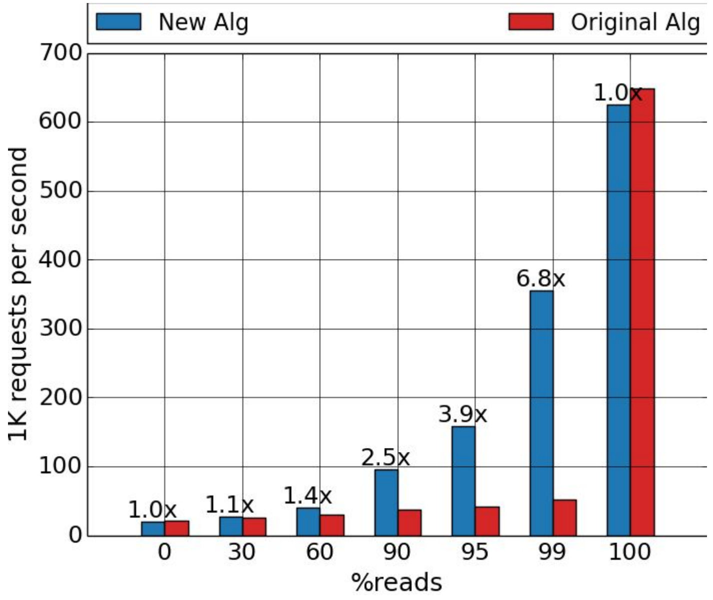
- Committed into ZooKeeper trunk

See Jira ZooKeeper-2024 for more experiments and details

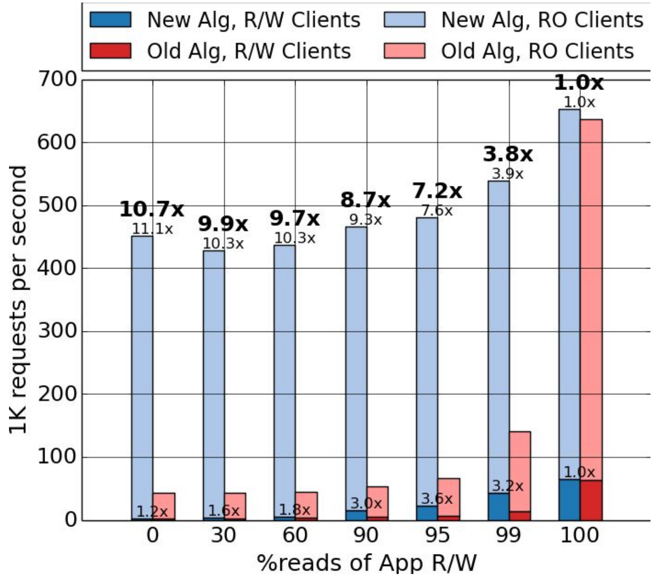
ZooKeeper Improvment: Evaluation

Single ZK of 5 servers, 900 clients:

100% R/W clients:



10% R/W, 90% RO clients:



Conclusion

- ✓ Performance
  ✓ Simplicity
  ✓ Correctness
- Small change in the client side
- Backward compatible
- Higher locality ⇒ Lower cost

+Improving ZooKeeper up to x10

Thank you!