CANOPUS: A SCALABLE AND MASSIVELY PARALLEL CONSENSUS PROTOCOL

Joint work with Sajjad Rizvi and Srinivasan Keshav
CONSENSUS PROBLEM

Agreement between a set of nodes in the presence of failures

- Asynchronous environment

Primarily used to provide fault tolerance

Node A: W(x=1)

Node B: W(x=2)

Replicated log

W(x=3)  W(y=1)  W(z=1)  ?
A BUILDING BLOCK IN DISTRIBUTED SYSTEMS

System applications:
- Hadoop
- HBase
- Kafka
- Akka
- Mesos
- BookKeeper
- Spanner
- ...

Coordination services:
- ZooKeeper
- Chubby
- etcd
- Chef
- Consul
- Puppet
- ...

Consensus and atomic broadcast:
- ZAB
- Paxos
- Raft
- ...
- EPaxos
- Mencius
- SPaxos
- ...
- NetPaxos
- AllConcur
- NOPaxos
- ...
Current consensus protocols are not scalable

However, most applications only require a small number of replicas for fault tolerance
PERMISSIONED BLOCKCHAINS

A distributed ledger shared by all the participants

Consensus at a large scale

- Large number of participants (e.g., financial institutions)
- Must validate a block before committing it to the ledger

Examples

- Hyperledger, Microsoft Coco, Kadena, Chain …
CANOPUS

Consensus among a large set of participants
- Targets thousands of nodes distributed across the globe

Decentralized protocol
- Nodes execute steps independently and in parallel

Designed for modern datacenters
- Takes advantage of high performance networks and hardware redundancies
SYSTEM ASSUMPTIONS

Non-uniform network latencies and link capacities
- Scalability is bandwidth limited
- Protocol must be network topology aware

Deployment consists of racks of servers connected by redundant links
- Full rack failures and network partitions are rare

![Diagram](image-url)
CONSENSUS CYCLES

Execution divided into a sequence of consensus cycles

- In each cycle, Canopus determines the order of writes (state changes) received during the previous cycle
SUPER-LEAVES AND VNODES

Nodes in the same rack form a logical group called a super-leaf.

Use an intra-super-leaf consensus protocol to replicate write requests between nodes in the same super-leaf.
SUPER-LEAVES AND VNODES

Nodes in the same rack form a logical group called a super-leaf.

Use an intra-super-leaf consensus protocol to replicate write requests between nodes in the same super-leaf.

Represent the state of each super-leaf as a height 1 virtual node (vnode).
ACHIEVING CONSENSUS

Members of a height 1 vnode exchange state with members of nearby height 1 vnodes to compute a height 2 vnode

- State exchange is greatly simplified since each vnode is fault tolerant

h rounds in a consensus cycle

A node completes a consensus cycle once it has computed the state of the root vnode
CONSENSUS PROTOCOL WITHIN A SUPER-LEAF

- Exploit low latency within a rack
  - Reliable broadcast
  - RAFT
1. Nodes prepare a proposal message that contains a random number and a list of pending write requests.
2. Nodes use reliable broadcast to exchange proposals within a super-leaf
3. Every node orders proposals
These three steps make up a consensus round.

At the end, all three nodes have the same state of their common parent.
CONSENSUS PROTOCOL BETWEEN SUPER-LEAVES
CONSENSUS PROTOCOL BETWEEN SUPER-LEAVES

1. Representatives send proposal requests to fetch the states of vnodes
2. Emulators reply with proposals
3. Reliable broadcast within a super-leaf
Representative

Consensus cycle ends for a node when it has completed the last round
Read requests can be serviced locally by any Canopus node

- Does not need to disseminate to other participating nodes

Provides linearizability by

- Buffering read requests until the global ordering of writes has been determined
- Locally ordering its pending reads and writes to preserve the request order of its clients

Significantly reduces bandwidth requirements for read requests

Achieves total ordering of both read and write requests
ADDITIONAL OPTIMIZATIONS

Pipelining consensus cycles
- Critical to achieving high throughput over high latency links

Write leases
- For read-mostly workloads with low latency requirements
- Reads can complete without waiting until the end of a consensus cycle
EVALUATION: **MULTI DATACENTER CASE**

3, 5, and 7 datacenters

- Each datacenter corresponds to a super-leaf

3 nodes per datacenter (up to 21 nodes in total)
- EC2 c3.4xlarge instances

100 clients in five machines per datacenter
- Each client is connected to a random node in the same datacenter

### Latencies across datacenters (in ms)

<table>
<thead>
<tr>
<th></th>
<th>IR</th>
<th>CA</th>
<th>VA</th>
<th>TK</th>
<th>OR</th>
<th>SY</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>133</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VA</td>
<td>66</td>
<td>60</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TK</td>
<td>243</td>
<td>113</td>
<td>145</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>154</td>
<td>20</td>
<td>80</td>
<td>100</td>
<td>0.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SY</td>
<td>295</td>
<td>168</td>
<td>226</td>
<td>103</td>
<td>161</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>22</td>
<td>145</td>
<td>89</td>
<td>226</td>
<td>156</td>
<td>322</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Regions: Ireland (IR), California (CA), Virginia (VA), Tokyo (TK), Oregon (OR), Sydney (SY), Frankfurt (FF)
CANOPUS VS. EPAXOS (20% WRITES)
EVALUATION: SINGLE DATACENTER CASE

3 super-leaves of sizes of 3, 5, 7, 9 servers (i.e., up to 27 total servers)
- Each server has 32GB RAM, 200 GB SSD, 12 cores running at 2.1 GHz

Each server has a 10G to its ToR switch
- Aggregation switch has dual 10G links to each ToR switch

180 clients, uniformly distributed on 15 machines
- 5 machines in each rack
ZKCANOPUS VS. ZOOKEEPER

The diagrams compare the median request completion time for ZKCCanopus and ZooKeeper with different write intensities across various throughput levels. The graphs illustrate how ZKCCanopus performs better in terms of request completion time compared to ZooKeeper, especially under high write intensities and increased node counts.
LIMITATIONS

We trade off fault tolerance for performance and understandability
- Cannot tolerate full rack failure or network partitions

We trade off latency for throughput
- At low throughputs, latencies can be higher than other consensus protocols

Stragglers can hold up the system (temporarily)
- Super-leaf peers detect and remove them
ON-GOING WORK

Handling super-leaf failures
- For applications with high availability requirements
- Detect and remove failed super-leaves to continue

Byzantine fault tolerance
- Canopus currently supports crash-stop failures
- Aiming to maintain our current throughput
Emerging applications involve **consensus at large scales**
- Key barrier is a **scalable consensus protocol**

**Addressed by Canopus**
- Decentralized
- Network topology aware
- Optimized for modern datacenters