Hazelcast

- The leading open source Java IMDG
- Distributed Java collections, concurrency primitives, ...
- Distributed computations, messaging, ...
In-Memory Data Grids

- Distributed caching
- Keeping data in local JVM for fast access & processing
- Elasticity, availability, high throughput, and low latency
- Multiple copies of data to tolerate failures
Replication

- Putting a data set into multiple nodes
- Fault tolerance
- Latency
- Throughput
Challenges

- Where to perform reads & writes?
- How to keep replicas sync?
- How to handle concurrent reads & writes?
- How to handle failures?
CAP Principle

- Pick two of C, A, and P
- CP versus AP
AP

CLIENT

NODE

REPLICA

NODE

REPLICA

NODE

REPLICA
Consistency/Latency Trade-off
Consistency/Latency Trade-off
PACELC Principle

- If there is a network partition (P), we have to choose between availability and consistency (AC).
- Else (E), during normal operation, we can choose between latency and consistency (LC).
Let’s build the core replication protocol of Hazelcast
Primary Copy

- Operations are sent to primary replicas.
- Strong consistency when the primary is reachable.
Partitioning (Sharding)

- Partitioning helps to scale primaries.
- A primary replica is elected for each partition.
Updating Replicas

map.put(k, v);
Updating Replicas

```java
map.put(k, v);
```

partition id = hash(serialized(key)) % partition count
Updating Replicas

```java
map.put(k,v);
```
Updating Replicas

map.put(k,v);

CLIENT

3

NODE

<k,v>

3

NODE

<k,v>
Async Replication

- Each replica is updated separately.
- High throughput and availability
Anti-Entropy

- Backup replicas can fall behind the primary.
- Non-sync backups are fixed with an active anti-entropy mechanism.
Replicas are not sync

- The client reads a key from the current primary replica.

```java
map.get(k);
```
Network Partitioning

- The client reads the same key.

```java
map.get(k);
```
Split-Brain

- Strong consistency is lost.
Resolving the Divergence

- Merge policies: higher hits, latest update / access, ...
- Merging may cause lost updates.
Let’s classify this protocol with PACELC
Hazelcast is PA/EC

- Consistency is usually traded to availability and latency together.
- Hazelcast works in memory and mostly used in a single computing cluster.
- Consistency - latency trade-off is minimal.
- PA/EC works fine for distributed caching.
Favoring Latency (PA/EL)
Scaling Reads

- Reads can be served locally from near caches and backup replicas.
Favoring Consistency (PC/EC)
Failure Detectors

- Local failure detectors rely on timeouts.
- Operations are blocked after the cluster size falls below a threshold.
Failure Detectors

- It takes some time to detect an unresponsive node.
- Minimizes divergence and maintains the baseline consistency.
Isolated Failure Detectors

- Configure failure detectors independently for data structures
- Phi-Accrual Failure Detector
CP Data Structures

- IDGenerator
- Distributed impls of java.util.concurrent.*
- PA/EC is not the perfect fit for CP data structures.
Flake IDs

- Local unique id generation
- Nodes get a unique node id during join.
- K-ordered IDs
CRDTs

- CRDTs: Conflict-free Replicated Data Types
- Replicas are updated concurrently without coordination.
- Strong eventual consistency
- Counters, sets, maps, graphs, ...
PN-Counter

![Figure showing a PN-Counter system with a client and two nodes (A and B). The client initiates an inc() operation, which is processed by Node A, and a dec() operation, which is processed by Node B. The counters for each node are updated accordingly.](image)
PN-Counter

CLIENT

NODE A

A-INC: 1
A-DEC: 0
B-INC: 0
B-DEC: 1

NODE B

A-INC: 1
A-DEC: 0
B-INC: 0
B-DEC: 1

CLIENT
Sync Replication

- Concurrency primitives imply the true CP behavior.
- Paxos, Raft, ZAB, VR
- Re-implementing Hazelcast concurrency primitives with Raft
Recap

Thanks!

You can find me at
- @metanet
- ebkahveci@gmail.com