Cloud and Cluster Data Management

SCALABLE CONSISTENCY AND TRANSACTION MODELS

THANKS TO M. GROSSNIKLAUS

Outline

- Sharding & Replication –
  - What are they?
  - What issues do they introduce?
- CAP Theorem (Consistency, Availability, Partition-Tolerance – can't have it all)
  - Also look at consistency vs. latency
- Eventual Consistency
  - What is it?
  - What are different models of eventual consistency?
  - Also look at configurations of readers and writers for replication and consistency
- Vector clocks: causal consistency

Sharding and Replication

- Sharding (Partitioning)
  - Breaking a database into several collections (*shards*)
  - Each data item (e.g., a document) goes in one shard
- Replication
  - Have multiple copies of a database
  - Each data item lives in several places

Can combine sharding and replication

Why do systems shard and replicate?

Issues with Sharding and Replication

Sharding: If an operation needs multiple data items, it might need to access several shards

Replication: Trying to keep replicas in sync with each other.

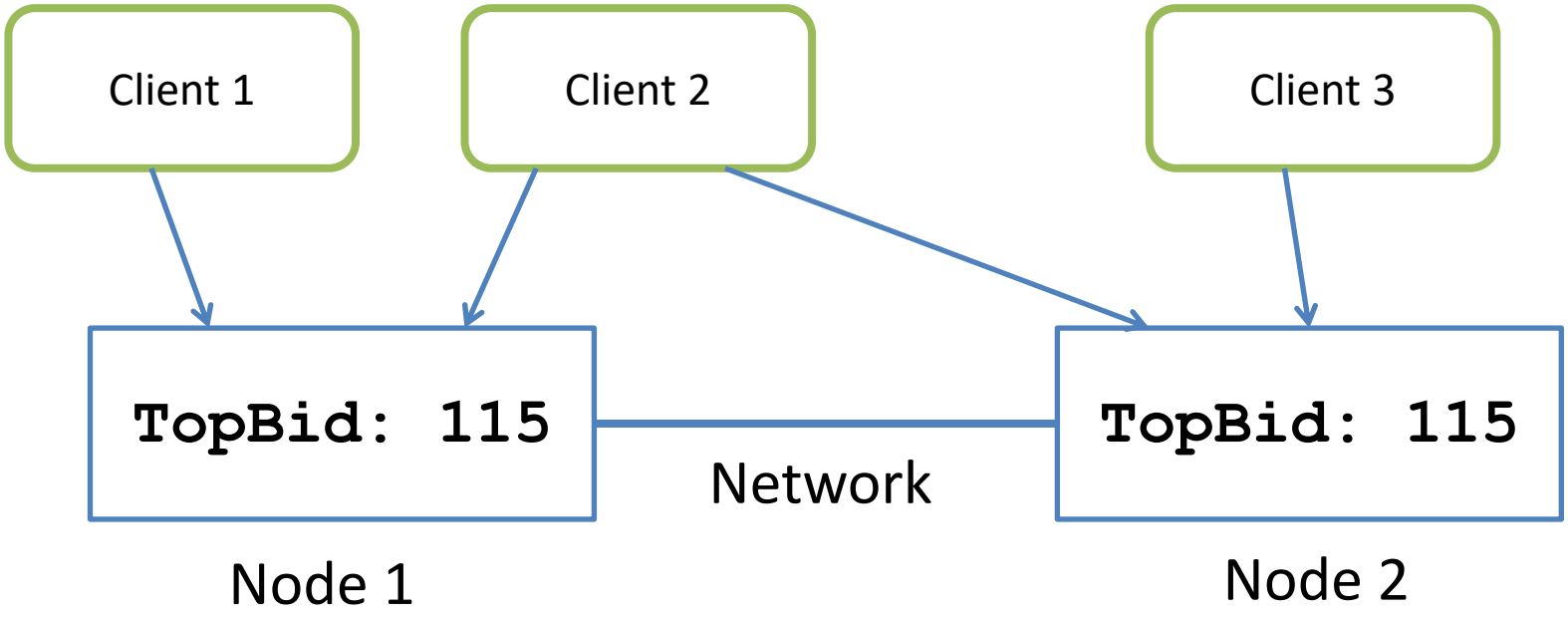
Other issues:

- Replications requires more storage space
- Time of access for both sharding and replications
- If more machines are being added, how would data be redistributed?
- How to locate data among nodes for sharding

Managing Replicas 1

Strategy 1: Write all

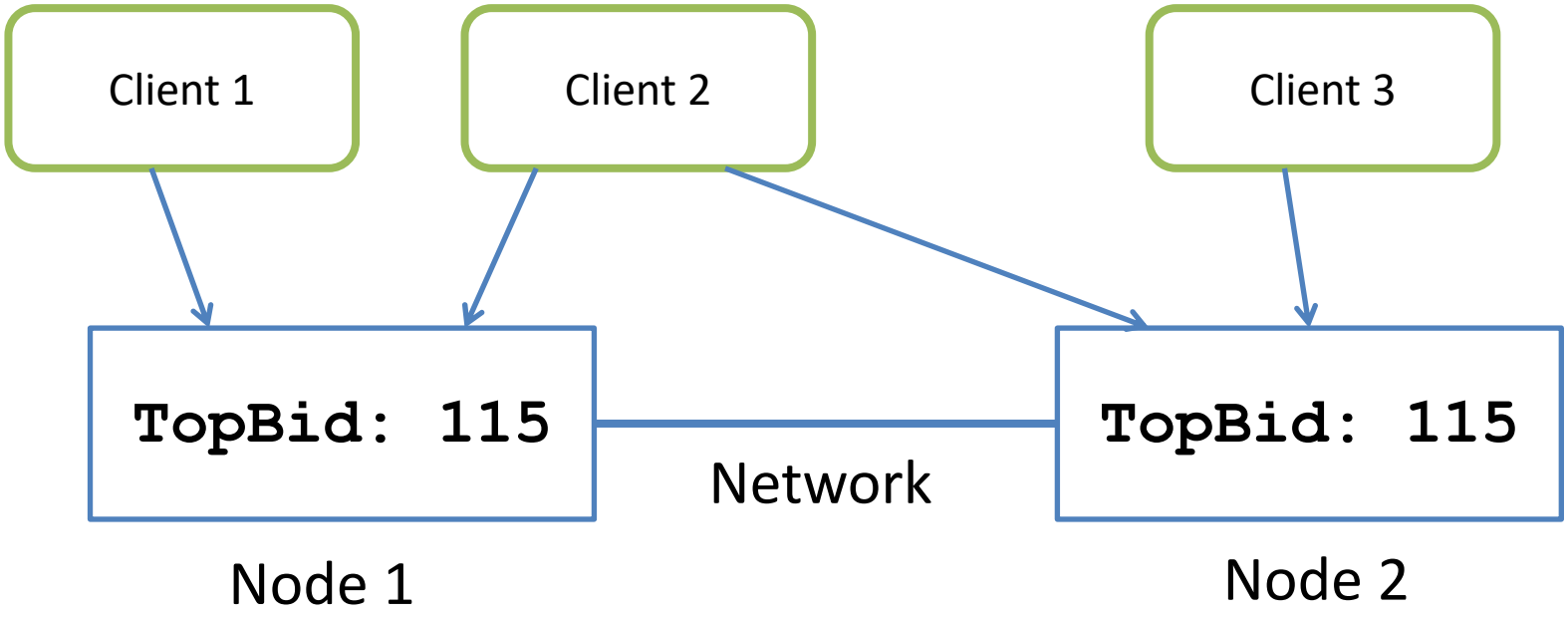
- Write: Update both, synchronously
- Read: Access either



Managing Replicas 2

Strategy 2: Write one

- Write: Update one, propagate update asynchronously
- Read: Access either (or both?)



Brewer's Conjecture

- Three properties that are desirable and expected from real-world shared-data systems
  - **C:** data consistency – you get most recent write or an error
  - **A:** availability – every request gets a (non-error) response
  - **P:** tolerance of network partition
- At *PODC 2000* (Portland, OR), Eric Brewer made the conjecture that only two of these properties can be satisfied by a system at any given time
- Conjecture was formalized and confirmed by MIT researchers Seth Gilbert and Nancy Lynch in 2002
- Now known as the **CAP Theorem**

Data Consistency

- Database systems typically implement ACID transactions
  - **A**tomicity: “all or nothing”
  - **C**onsistency: transactions never observe or result in inconsistent data
  - **I**solation: transactions are not aware of concurrent transactions
  - **D**urability: once committed, the state of a transaction is permanent
- Useful in automated business applications
  - banking: at the end of a transaction the sum of money in both accounts is the same as before the transaction
  - online auctions: the last bidder wins the auction
- There are applications that can deal with looser consistency guarantees and periods of inconsistency

Availability

- Services are expected to be highly available
  - every request should receive a response
  - if you can read a data item, you can update it
  - it can create real-world problems when a service goes down
- Realistic goal
  - service should be as available as the network it runs on
  - if any instance of a service on the network is available, the service should be available

Partition-Tolerance

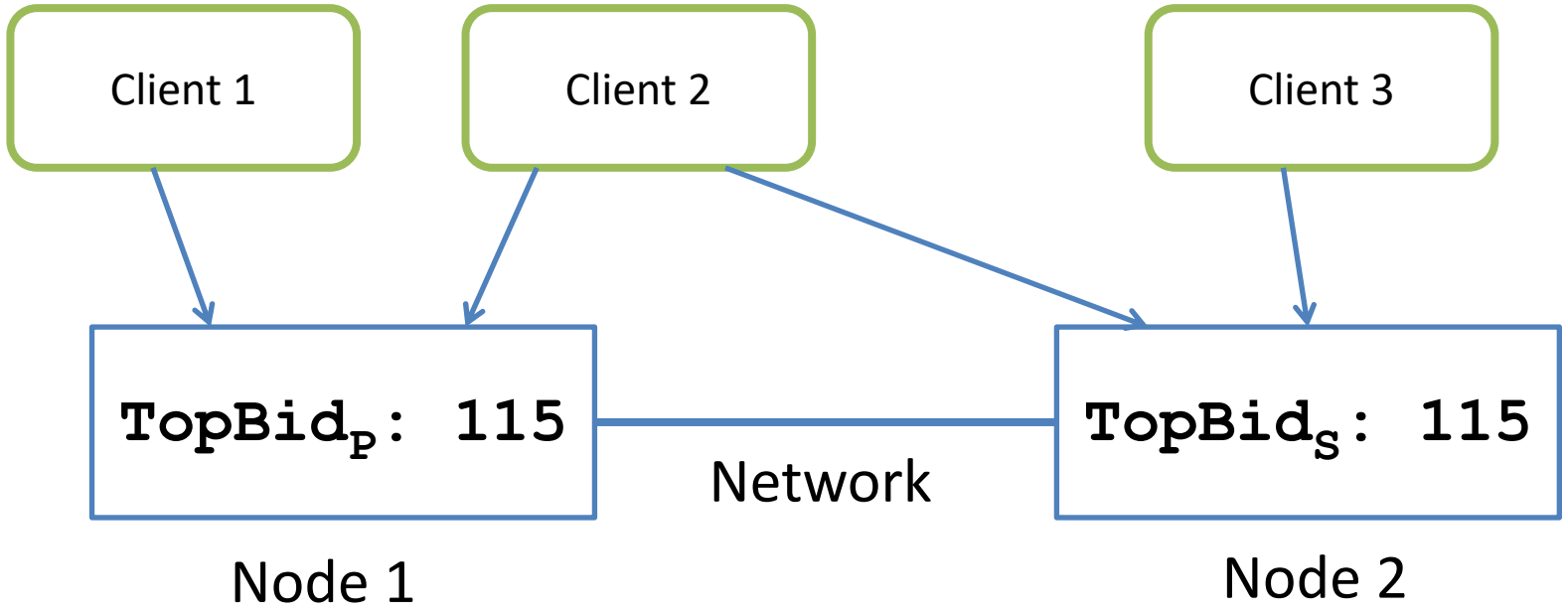
- A service should continue to perform as expected
  - if some nodes crash
  - if some communication links fail
- One desirable fault tolerance property is resilience to a network partitioning into multiple components
- In cloud computing, node and communication failures are not the exception but everyday events

Managing Replicas: Activity

Strategy 3: Primary-Secondary

1. Write: Update primary, propagate to secondary asynchronously
2. Read Strong: Access primary
3. Read Weak: Access either

For each op, does it give consistency or availability in a partition?



Participation Question 1

In the primary-secondary strategy, for each of the following operations, does it give consistency or availability in a partition?

1. Write: Update primary, propagate to secondary asynchronously
2. Read Strong: Access primary
3. Read Weak: Access either