Distributed Systems

ECE428

Lecture 20

Adopted from Spring 2021

Distributed Transactions

- Transaction processing can be *distributed* across multiple servers.
  - Different objects can be stored on different servers.
  - An object may be replicated across multiple servers.
    - Our focus today.
- Case study: Google's Spanner System

Distributed Transactions

- *Sharding:* objects can be distributed across multiple (1000's of) servers
  - Primary reason: load balancing and scalability.

- *Replication:* the same object may be replicated among a handful of nodes.
  - Primary reason: fault-tolerance, availability, durability.

Replication: Natural way to handle failures

- Node failures are common.

In each cluster's first year, it's typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will "go wonky," with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span. And there's about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover.

-- Jeff Dean (Google), source: cnet.com

Replication: Natural way to handle failures

- Node failures are common.
- What could happen if a node fails?
  - Objects unavailable until recovery.
  - 2-phase commit
  - “stuck” after coordinator failure
- Even worse: what happens if the drive fails.
  - no recovery!
- Replication provides greater availability and robustness to failures.
  - Geo-replication (spanning datacenters across the world) for greater robustness.

Replication

- Replication = An object has identical copies, each maintained by a separate server.
  - Copies are called “replicas”

- With *k* replicas of each object can tolerate failure of any (*k-1*) servers in the system

Replication: Availability

- If each server is down a fraction *f* of the time
  - Server's failure probability

- With no replication, availability of object =
  = Probability that single copy is up
  = $(1 - f)$

- With *k* replicas, availability of object =
  = Probability that at least one replicas is up
  = 1– Probability that all replicas are down
  = $(1 - f^k)$

Replication: Availability

- With no replication, availability of object =
  = Probability that single copy is up
  = $(1 - f)$
- With k replicas, availability of object =
  = Probability that at least one replicas is up
  = 1 - Probability that all replicas are down
  = $(1 - f^k)$

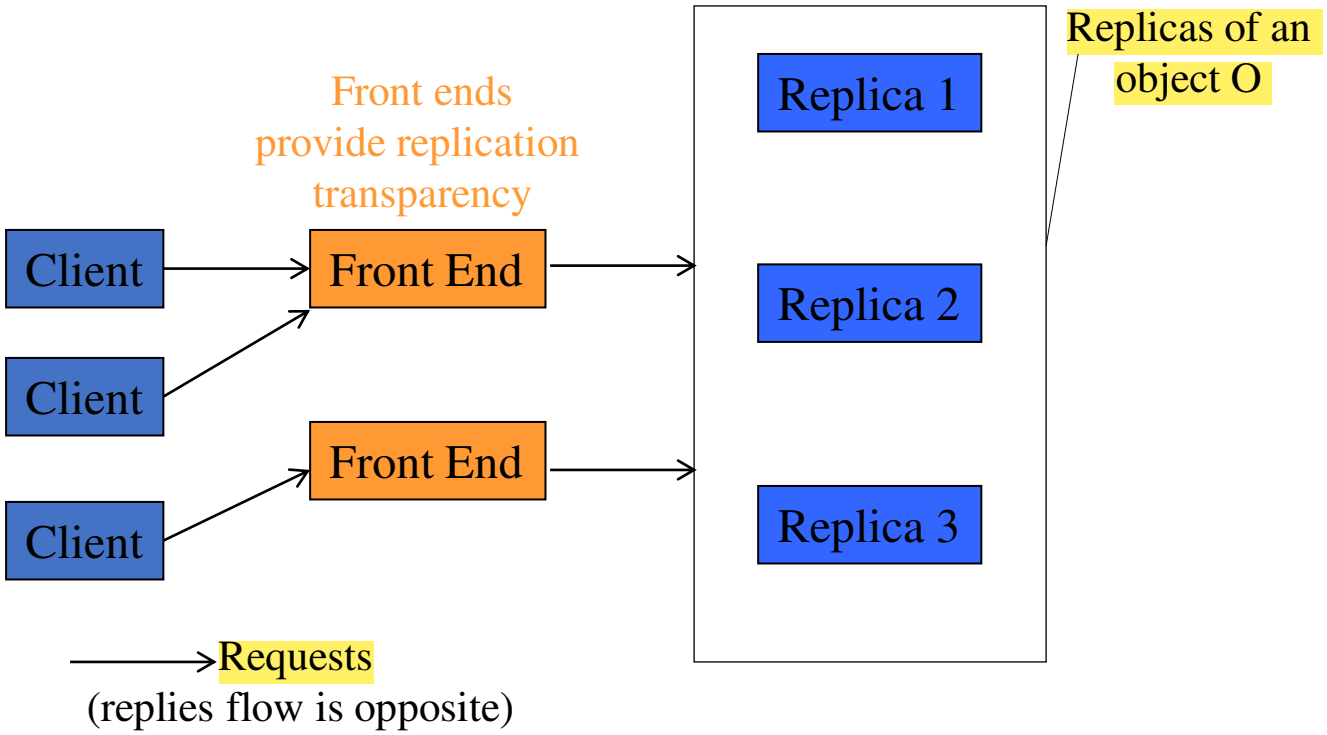
f=failure probability	No replication	*k*=3 replicas	*k*=5 replicas
0.1	90%	99.9%	99.999%
0.05	95%	99.9875%	6 Nines
0.01	99%	99.9999%	10 Nines

Replication Challenges

1. Replication Transparency
   - A client ought not to be aware of multiple copies of objects existing on the server side

2. Replication Consistency
   - All clients see single consistent copy of data, in spite of replication
   - For transactions, ACID is guaranteed.

Replication Transparency

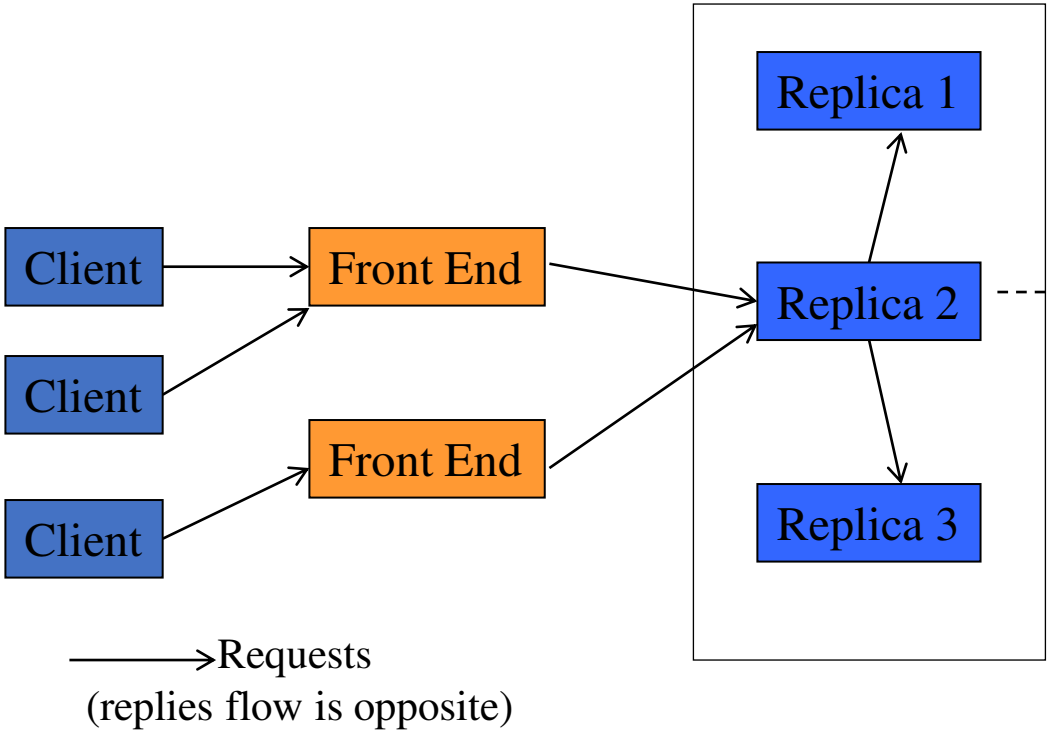


Replication Consistency

- Two ways to forward updates from front-ends to replica group
  - Passive Replication: uses a primary replica (leader)
  - Active Replication: treats all replicas identically

- Both approaches use the concept of
- "Replicated State Machines"
  - Each replica's code runs the same state machine
  - *Multiple copies of the same State Machine begun in the Start state, and receiving the same Inputs in the same order will arrive at the same State having generated the same Outputs.* [Schneider 1990]

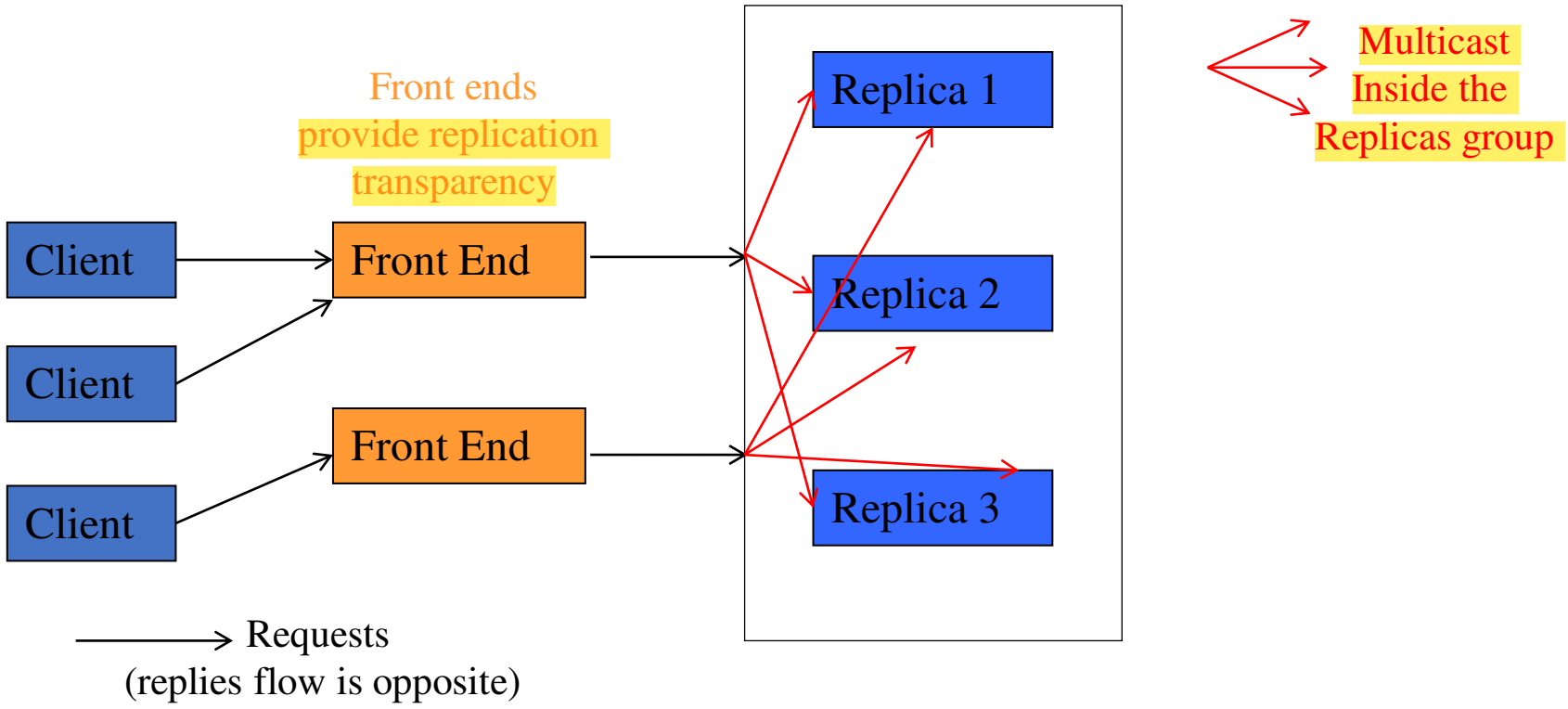
Passive Replication



- Leader => total reliable ordering of all updates
- On leader failure, run leader election

Elected leader

Active Replication



Transactions and Replication

- One-copy serializability
  - *A concurrent execution of transactions in a replicated database is one-copy-serializable if it is equivalent to a serial execution of these transactions over a single logical copy of the database.*
  - (Or) The effect of transactions performed by clients on replicated objects should be the same as if they had been performed one at a time on a single set of objects (i.e., 1 replica per object).
- In a non-replicated system, transactions appear to be performed one at a time in some order.
  - Correctness means serial equivalence of transactions
- When objects are replicated, transaction systems for correctness need one-copy serializability.

Transactions and Replication

- Objects distributed among 1000's cluster nodes for load-balancing (sharding)

- Objects replicated among a handful of nodes for availability and durability.
  - Replication across data centers, too

- Two-level operation:
  - Use transactions, coordinators, 2-phase commit per object
  - Use Paxos / Raft among object replicas

- Consensus needed across object replicas, e.g.
  - When acquiring locks and executing operations
  - When committing transactions

2-phase commit and Paxos

- Example workflow:
  - Coordinator leader sends Prepare message to leaders of each replica group
  - Each replica leader uses Paxos to commit the Prepare to the group logs
  - Once commit Prepare succeeds, reply to coordinator leader
  - Coordinator leader uses Paxos to commit decision to its group log.
  - Coordinator leader sends Commit message to leaders of each replica group.
  - Each replica leader uses Paxos to process the final commit.
  - Replica leader send the “commit ok / have committed” message back to coordinator.

Spanner: Google's Globally-Distributed Database

- First three lines from the paper:

  - Spanner is a scalable, globally-distributed database designed, built, and deployed at Google.

  - At the highest level of abstraction, it is a database that shards data across many sets of Paxos state machines in datacenters spread all over the world.

  - Replication is used for global availability and geographic locality; clients automatically failover between replicas.

Google Spanner Basics

- Building distributed databases with ACID properties is extremely challenging.
- Use GPS and atomic clock timestamps to decide if the locally available data are up to date.
- Transparent, scalable, SQL semantic.

Data update

- Send notification of update to every other location of that data together with the timestamp.
- Requests for the data are served from the most up to date location until the updated data are fully replicated.

Google Spanner Basics (contd)

TrueTime

- Highly available, distributed clock
- Guarantees monotonically increasing timestamps (new timestamp generated after the previous one), i.e., also guarantees ordering

Use in Spanner

- Consistent reads without blocking writes
- Weaker guarantees: writes not blocked by read-only transac's, exploits multiple versions of data
- Strong read: committed data up until now, may need to contact other locations for the most recent data at cost of extra latency
  Stale read: most recent committed data available with low latency
- Consistency == same as serially executed transactions, i.e., If T1 completes before T2, the client will see the effect of T1 before T2