LTAT.06.007 Distributed Systems
Lecture 8 – Consensus

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia, Delta building (auditorium 1021) 28/03/2022
Recap

- Replication
  - Reason for replication
  - Reconciling differences in replicas, e.g., anti-entropy protocols
- Quorum
  - Deciding valid data in replicas
Think about

Back-up nodes

Online book shop

Server 1
Service payment

Server 2
Service payment

Server N
Service payment

Leader

Replica 1...
Replica...
N

Charge

Success

Agreeing on the same values/operations over time
Agenda

- **Goal:** To study the importance of consensus in distributed systems

- **Content:**
  - State machine replication
  - Consensus
  - Raft (Modern implementation of consensus)
  - Chain replication – “alternative to consensus”

**After this lecture, you should be able to:**

- To apply consensus to any distributed problem
- To understand the benefits and complexity of consensus
State machine replication

Essence

• The main idea behind is that a single process *(the leader)* broadcasts the operations that change its state to other process, the followers *(replicas)*

• **Total order broadcast:** every node delivers the *same messages* in the *same order*

• The followers execute the same sequence of operation as the leader, then the state of each follower will match the leader

State machine replication *(SMR)*: - *every replica acts as SM*

• FIFO-total order broadcast: every update to all replicas

• Replica deliver update message: apply to to own state

• Applying an update is deterministic – even errors

• Replica is a state machine: starts in fixed initial state, goes through same sequence of state transitions in the same order → all replicas end up in the same state
State machine replication

Essence

Closely related ideas:
• Serializable transactions (execute in delivery order) – Active/Passive replication
• Blockchains, distributed ledgers, smart contracts

Limitations:
• Cannot update state immediately, have to wait for delivery through broadcast
• Need fault-tolerant total order broadcast

[source] Distributed Systems course given by Dr. Martin Kleppmann (University of Cambridge, UK)
Database leader replica

Client 1

Client 2

Leader

Follower

\{ x \rightarrow (t_1, \text{true}) \}

T_1 \rightarrow \text{ok} \rightarrow \text{commit} \\
T_2 \rightarrow \text{ok} \rightarrow \text{commit} \\
T_1 \rightarrow \text{commit} \\
T_2 \rightarrow \text{commit}
Other broadcasts

<table>
<thead>
<tr>
<th>Broadcast</th>
<th>Assumptions about state update function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total broadcast</td>
<td>Deterministic (SMR)</td>
</tr>
<tr>
<td>Causal</td>
<td>Deterministic, concurrent updates commute</td>
</tr>
<tr>
<td>Reliable</td>
<td>Deterministic, all updates commute</td>
</tr>
<tr>
<td>Best-effort</td>
<td>Deterministic, commutative, idempotent, tolerate message loss</td>
</tr>
</tbody>
</table>

Observation

When updates are **commutative**, replicas can process updates in different orders and still end up in the same state.
## Consensus

### Essence

A fundamental problem studied in distributed systems, which requires a set of processes to agree on a value in fault tolerant way so that:

- Every non-faulty process eventually agrees on a value
- The final decision of every non-faulty process is the same everywhere
- The value that has been agreed on has been proposed by a process

### Consensus has a large number of practical applications

- Commit transactions, Decisions in general (votes are involved)
- Hold a lock (Mutual exclusion), Failure detection (Byzantine)

Any problem that requires consensus can be solved with a **state machine replication**
Consensus and total order broadcast

More about consensus

- Leader regulates the consensus with the nodes via total order broadcast
  - Single point of failure
  - **Failover**: human operator chooses a new leader, e.g., databases
- Election algorithms can automate the selection of the leader (properties?)
- Consensus and total order broadcast are formally equivalent

**Common consensus algorithms:**
- **Paxos**: single-value consensus
- **Multi-paxos**: generalization to total order broadcast
- **Raft, Viewstamped replication, Zab**: FIFO-total order broadcast
Consensus system models

Essence

• Paxos, Raft, etc., assume a partially synchronous crash-recovery model.
• Why not asynchronous?
  ▪ FLP result (Fisher, Lynch, Paterson): There is no deterministic consensus algorithm that is guaranteed to terminate in an asynchronous crash-stop system model
  ▪ Paxos, Raft and others, use clocks only used for timeouts/failures detector to ensure progress. Safety (correctness) does not depend on timing

There are also consensus algorithms for a partially synchronous Byzantine system model (used in blockchains)

Practical considerations

ZooKeeper (https://zookeeper.apache.org/); etcd (https://etcd.io/)
Observations

Some consensus uses a leader to sequence messages

- Use a **failure detector** (timeout) to determine suspected crash or unavailable leader
- On suspected leader crash, a new leader is elected
- Prevent two leader at the same time “split-brain”

Elects a leader

Cannot elect a different leader as C already voted
Leader in consensus

Observations

Ensure <= leader per term:
• Term is incremented every time a leader election is started
• A node can only vote once per term
• Require a quorum of nodes to elect a leader in a term

A
B
C
D
E

Elects a leader
Cannot elect a different leader as C already voted
A single leader?

Can guarantee unique leader per term?

• Cannot prevent having multiple leaders from different terms
• Example: Node 1 is leader in term $t$, but due to a network partition it can no longer communicate with node 2 and 3

Node 2 and 3 may elect a new leader in term $t + 1$
Node 1 may not even know that a new leader has been elected!
Checking if a leader has been voted out

Leader

Am I still be leader in term $t$?

Follower 1

Can we deliver message $m$ next in term $t$?

Follower 2

Right, now deliver $m$ please

For every decision (message to deliver), the leader must first get acknowledgements from a quorum.

[source] Distributed Systems course given by Dr. Martin Kleppmann (University of Cambridge, UK)
Raft

Essence

• Modern solution the problem of consistency
• An algorithm that guarantees the strongest consistency possible
• Raft is based on state machine replication
• In Raft, time is divided into election term
• A term is depicted by a logical clock and just increases forward
• The term starts by an election to decide who becomes a leader
• Raft guarantees that for any term there is at most one leader
State machine replication (in Raft)

- Follower
- Candidate
- Leader
State machine replication (in Raft)

Raft’s algorithm

Follower

- Starts up
- Discovers current leader or new term

Candidate

- Times out, start election
- Receives vote from majority
- Discovers new term

Leader

- Times out, new election
- Starts up

Notes:

- Follower
- Candidate
- Leader
Raft

Algorithm (overview)

• Every process starts as **follower**
• A follower expects to receive **a periodic heartbeat** from the leader containing the election term the leader was elected in.
• If the follower does not receive any heartbeat within a certain period of time, **a timeout** fires and the leader is presumed dead.
• The follower start a new election by increment the current **election term** and transitioning to candidate state.
• It then votes for itself and sends a request to all processes in the system to vote for it, stamping the request with the current election term.
Raft

Outcome

- **The candidate wins the election:** the candidate becomes a leader and starts sending out heartbeats to the other processes.

- **Another process wins the election:** In this case, terms between process are compared, if another process claims to be the leader with a term greater or equal the candidate’s term, it accepts the new leader and returns to the follower state.

- **A period of time goes by with no winner:** very unlikely, but if it happens, then candidate will eventually time-out and starts a new election process.

One single leader guarantee is enough?

On way to avoid dynamic leaders is by using a fencing token (a number that increases every time a distributed lock is acquired - a logical clock).
Raft

Log replication

• The leader is the only one that can make changes to the replica states
• A log is created inside the leader and then replicated across the followers (log replication)
• When the leader applies an operation to its local state, it appends a new log entry into its own log (operation is logged)
Raft

Leader

Followers

Committed entries

[source] Raft's paper
Raft

Data storing in a single node
Raft
Raft

Client
Raft

Client

3

3
Raft

Data storing in a multiple nodes
We cannot expect to communicate with all of them (although we could)
We need a protocol structure to handle data consistency across multiple nodes
Raft
Raft
Raft
Raft
Raft
Raft
Raft

set 3

AppendEntries
Raft

3

set 3

set 3

set 3
Raft
Raft
Leader wait only for a majority (Quorum) of followers to commit
Raft
Raft in action

Leader Election

Continue →

https://raft.github.io/
Chain replication

Essence

• Chain replication uses a very different topology than leader based replication protocols.

• In chain replication, processes are arranged in a chain. The leftmost process is referred to the chain’s head, while the rightmost one as the chain’s tail.

• In the absence of failures, the protocol is strongly consistent as all writes and reads are processed one at a time.

• What happens if a node fails? There is a control panel component, which monitors the health of the chain. The control panel implements consensus.

• The chain can tolerate up to $N - 1$ processes failing, where $N$ is the chain length. The control panel can just tolerate $C/2$ failures, where $C$ is the number of replicas that make up the control panel.
Chain replication

Write

Read

Head

Tail
Chain replication

Write

[Diagram showing a chain of servers with arrows indicating data flow]
Chain replication

Write

[Diagram showing chain replication process]
Chain replication

Write
Chain replication

Write
Chain replication
Chain replication
Chain replication

Write(k, v2)

(dirty)
Read(k)

(clean)
Read(k)

k -> v1, v2

k -> v1, v2

k -> v1

k -> v1
Chain replication

Write($k$, $v_2$)

(dirty)

Read($k$)

(k?)

(clean)

Read($k$)

$k$ -> $v_1$, $v_2$

$k$ -> $v_1$, $v_2$

$k$ -> $v_1$

$k$ -> $v_1$
Chain replication

Write(k, v2)

(dirty) Read(k)

(k -> v1, v2)

Read(k)

v1

(k -> v1, v2)

Write(k, v2)

(k -> v1)

Read(k)

v1

(k -> v1)

Read(k)

v1

(k -> v1)
Summary

• Studied a fundamental issue in distributed systems (Consensus)
• Studied state machine replication and an actual implementation that uses it (Raft)
• Alternative solutions, Chain replication
References

Part of this material is inspired by:

- Understanding Distributed Systems, Version 1.1.1., Roberto Vitillo, 2021
- Distributed Systems course given by Dr. Martin Kleppmann (University of Cambridge, UK)
- Raft - Understandable Distributed Consensus, http://thesecretlivesofdata.com/raft/
Next lecture

Consistency models
Questions?

E-mail: huber.flores@ut.ee