CSC469 – Tutorial

Fault Tolerance

Stathis Maneas
Friday, November 24, 2017
Consensus

• Processes in a distributed system must “agree” on some value in the presence of failures.
• Individual processes begin with an initial value.
• Individual processes “propose” their initial values and “decide” on their final values.
• The algorithm must be able to tolerate faults.
Failure Model

• The system model is critical:
  ▪ Timing assumptions.
  ▪ Failure assumptions.

• Timing assumptions:
  ▪ **Synchronous**: shared clock, known bounds on message delivery.
  ▪ **Asynchronous**: no global clock, no time bounds.
  ▪ **Partial Synchrony**: fixed upper bounds on processor speeds and message delivery.

• Failure assumptions:
  ▪ Fail-stop (crash).
  ▪ Send-omission.
  ▪ Receive-omission.
  ▪ ...
  ▪ Byzantine (arbitrary, malicious).
Properties of Consensus

• Apply to “correct”, i.e., non-faulty processes:
  ▪ Agreement: all non-faulty processes decide on the same value.
  ▪ Validity: the decided value was proposed by a process.
  ▪ Termination: processes eventually decide.

• Safety vs. liveness properties.
Terminology

• Algorithms are described as tolerating $f$ failures:
  ▪ That is, if at most $f$ processes fail, we can still reach consensus.

• Obviously, we can’t control how faulty processes behave; that’s why we need majority.

• Intuitively, for majority, we need at least $f+1$ correct processes and thus, $2f+1$ in total (considering only fail-stop faults).
Fail-stop Algorithm

• Processes communicate in *rounds* (remember we’re in a synchronous environment).

• Intuitively, to tolerate $f$ fail-stop failures, the algorithm must have $f+1$ rounds of communication.

• If processes fail, they simply stop sending and receiving messages.
Byzantine Generals

• Slight difference initial setup, but same principles apply.

• Unsigned (oral) messages.

• Generals send their initial value, then send a vector of all values they’ve received from others.
Byzantine Generals

Figure 8-6. The same as Fig. 8-5, except now with two correct process and one faulty process.
Byzantine Generals

• Having just $f+1$ correct (non-faulty) processes is not enough since they can be misled.

• In other words, we need more correct processes to “counter out” the faulty votes.

• Need $2f+1$ non-faulty processes, $3f+1$ in total.
Byzantine Generals

Figure 8-5. The Byzantine agreement problem for three nonfaulty and one faulty process. (a) Each process sends their value to the others. (b) The vectors that each process assembles based on (a). (c) The vectors that each process receives in step 3.

From Tanenbaum and Van Steen, *Distributed Systems: Principles and Paradigms*
Why?

• Consensus is required for a lot of things:
  • Electing a master/coordinator.
  • Transaction commit.
  • Granting locks.
  • Work distribution.
  • Serialization.

• Individual hardware/software components fail, risking the correctness of the system.
Context

• Always consider the underlying system and the assumptions in effect:
  • Synchronous vs. asynchronous.
  • Signed vs. unsigned messages.
  • Failure types?
  • Bounded/Unbounded message delays?

• Different versions with different assumptions:
  • Reliable broadcast, atomic commit, etc.

• This is not a made-up problem!
  • Paxos, Chubby, Zookeeper, ...
FLP Impossibility Result (1/3)

• Distributed consensus is impossible in a completely asynchronous environment:
  ▪ Cannot differentiate between failed and slow processes.

• Both safety and liveness cannot be guaranteed in such a setting.

• Literature contains several techniques that circumvent the FLP impossibility result.
Partial Synchrony [1]:
- Lies between synchronous and asynchronous systems.
- In synchronous systems there are fixed upper bounds on:
  - Message delivery time.
  - Relative speed of processors.
- In asynchronous systems, no fixed upper bounds exist.
- Different variations:
  - The bounds exist but are not know \textit{a priori}.
  - The bounds are know but are guaranteed to hold starting at some \textit{unknown} time $T$.

Randomization:
- Involve some uncertainty, such as \textit{coin toss}, etc.
- Achieve consensus with \textit{probability 1}.
- May require more than one executions of the algorithm.
FLP Impossibility Result (3/3)

Failure Detectors [1]:
- They are characterized using two different properties, namely *completeness* and *accuracy*:
  - Strong/Weak completeness.
  - Strong/Weak/Eventual Strong/Eventual Weak accuracy.
- $\diamondsuit W$ is the weakest failure detector that solves Consensus in asynchronous distributed systems subject to crash failures:
  - Weak completeness.
  - Eventual weak accuracy.

Wormholes [2]:
- A hybrid distributed system model, where a part of the system consists of “good” properties otherwise not guaranteed by the “normal” weak environment.

[1] Unreliable Failure Detectors for Reliable Distributed Systems by Chandra et al.
Broadcast Mechanisms

• Nodes in a distributed system make use of broadcast protocols in order to exchange messages.

• Applications (algorithms) are described and implemented on top of these primitives.
Properties of Reliable Broadcast

• A Reliable Broadcast protocol must satisfy the following properties:

  • **Validity**: if a correct process broadcasts a message $m$, then all correct processes eventually deliver $m$.
  
  • **Agreement**: if a correct process delivers a message $m$, then all correct processes eventually deliver $m$.
  
  • **Integrity**: for any message $m$, every correct process delivers $m$ at most once, and only if $m$ was previously broadcast by a process.
Message Order

- **Unordered**: no guarantees on message order.
- **FIFO Order**: if a process broadcasts a message $m$ before it broadcasts a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$.
- **Causal Order**: If the broadcast of a message $m$ causally precedes the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$.
- **Total Order**: All correct processes deliver messages in the same order.
Broadcast Taxonomy

- Reliable Broadcast
- Atomic Broadcast
- FIFO Broadcast
- FIFO Atomic Broadcast
- Causal Broadcast
- Causal Atomic Broadcast

Ordering:
- FIFO Order
- Total Order
- Causal Order
Broadcast Algorithms

- **Reliable Broadcast (Diffusion):**
  - Cannot handle network partitions.
  - Failures assumed to be *fail-stop*.
  - Floods the network.

- **FIFO Broadcast:**
  - Implemented on top of Reliable Broadcast.
  - Each message also contains the process’s sender and sequence number.
  - Each process maintains the next sequence number for each other process in the system.
  - Requires buffering of messages.

- **Causal Broadcast:**
  - Implemented on top of FIFO broadcast.
  - Prepend list of messages upon which message $m$ causally depends.
  - Requires buffering of messages.
Atomic Broadcast

• A broadcast primitive where all correct processes in a distributed system deliver the same set of messages in the same order.

• Enforce total ordering.

• Distributed Consensus and Atomic Broadcast have been proved to be equivalent in asynchronous distributed systems subject to crash failures.

• Synchronous algorithms for various models exist.