

Final Lecture

- Fault tolerance in distributed systems

Today's papers

- "The Byzantine General's Problem", Leslie Lamport
- "Practical Byzantine Fault Tolerance", Miguel Castro, Barbara Liskov,
- Reliable Broadcast and Consensus - "This has resulted in a voluminous literature which, unfortunately, is not distinguished for its coherence. The differences in notation and the haphazard nature of the assumptions obfuscates the close relationship among these problems" – Hadzilacos & Toueg, Distributed Systems.

Distributed Algorithms

- Distributed system is comprised of n processes
- A process executes a sequence of events
 - Local computation
 - Sending a message m
 - Receiving a message m
- A distributed algorithm is an algorithm that runs on more than one process.

Properties of Distributed Algorithms

- Safety
 - Means that some particular "bad" thing never happens.
- Liveness
 - Indicates that some particular "good" thing will eventually happen.

Assumptions for Distributed Algorithms

- Timing
 - Synchronous
 - Asynchronous
 - Partially Synchronous
- Failure
 - Models
 - Errors
 - Fail-stop
 - Byzantine
 - Fault/Failure Detectors

Synchronous timing assumption

- Processes share a clock
- Timestamps mean something between processes
- Communication can be guaranteed to occur in some number of clock cycles

Asynchronous timing assumption

- Processes operate asynchronously from one another.
- No claims can be made about whether another process is running slowly or has failed.
- There is no time bound on how long it takes for a message to be delivered.

Partial synchrony assumption

- "Timing-based distributed algorithms"
- Processes have some information about time
 - Clocks that are synchronized within some bound
 - Approximate bounds on message-deliver time
 - Use of timeouts

Errors as failure assumptions

- Specific types of errors are listed as failure assumptions
 - Link may lose messages
 - Link may duplicate messages
 - Link may reorder messages
 - Process may reset

Fail-Stop failure

- A failure results in the process, p , stopping
- p does not send anymore messages
- p does not perform actions when messages are sent to it
- Other processes can detect that p has failed

Byzantine failure

- Process p fails in an arbitrary manner.
- p is modeled as a malevolent entity
 - Can send the messages and perform the actions that will have the most detrimental impact on other processes
 - Can collaborate with other "failed" processes
- Common constraints on Byzantine assumption
 - Incomplete knowledge of global state
 - Limited ability to coordinate with other Byzantine processes
 - Restricted to polynomial computation (i.e., assume $P \neq NP \dots$)

Fault/failure detectors

- A perfect failure detector
 - Guaranteed not to give false positives (only reports actual failures).
 - Eventually reports failures to all processes.
- Heartbeat protocols
 - Assumes partially synchronous environment
 - Processes send "I'm Alive" messages to all other processes regularly
 - If process i does not hear from process j in some time $T = T_{\text{delivery}} + T_{\text{heartbeat}}$ then it determines that j has failed
 - Hinges on assumption about T_{delivery} being known and accurate

Distributed Consensus

- Distributed Consensus has many names (depending on the assumptions and the application...)
 - Reliable multicast
 - Byzantine Generals Problem
 - Interactive agreement
 - Atomic broadcast

Setup of Distributed Consensus

- Distinct processes having to agree on a single value.
- Example applications of consensus:
 - Performing a commit in a replicated/distributed database.
 - Sensor fusion (i.e. deciding car location based on different sensor readings)
- There are many system models
 - Processes co-located, synchronous, & fail-stop
 - Processes widely distributed, asynchronous, & Byzantine
 - And everything in between...

Properties of Distributed Consensus

- *Agreement*
 - If any correct process believes that V is the consensus value, then all correct processes believe V is the consensus value.
- *Validity*
 - If V is the consensus value, then some process proposed V .
- *Termination*
 - Consensus protocol terminates at some time with the consensus value V .
- *Agreement* and *Validity* are **Safety** Properties
- *Termination* is a **Liveness** property.

Synchronous/Byzantine Consensus Algorithm

- Byzantine Generals paper presents one algorithm for consensus
- Each process sends all messages it has seen so far (including which process sent it and in which round) during each round.
- Then the reasoning/analysis each process must do, albeit convoluted, leads to a result that meets the three properties
 - Agreement
 - Validity
 - Termination

Asynchronous Distributed Consensus

- Fail-Stop/Byzantine ® IMPOSSIBLE!
- FLP impossibility result
 - Fischer, Lynch and Patterson impossibility result
 - Asynchronous assumption makes it impossible to differentiate between failed and slow processes.
 - Therefore *termination (liveness)* cannot be guaranteed.
 - If an algorithm terminates it may violate *agreement (safety)*.
 - A slow process may decide differently than other processes thus violating the agreement property

Partially Synchronous Consensus Algorithms

- Relies on a Fault-Detector
- Synchronous/Fail-stop distributed consensus algorithms can be transformed to run in the partially synchronous environment
- Byzantine is still a problem though...
 - DoS attacks on correct processes result in the identification of correct processes as failed processes, thus reducing the number of processes that must be compromised to breach the *safety* property

Castro – BFT

- Relies on partially synchronous assumption to guarantee **liveness**.
- Therefore attacks on system can only slow it down – **safety** is guaranteed.
- Assumes that an attack on **liveness** can be dealt with in a reasonable amount of time.
- Suitable for wide area deployment (e.g., internet)
- Performance measures taken with $f=1$ and $N=4$.
- Recommends that N processes implementing the BFT be heterogonous for security reasons.