Distributed Systems (5DV147)
Distributed Agreement
Fall 2015

Processes often need to coordinate their actions and/or reach an agreement

- Which process gets to access a shared resource?
- Has the master process crashed? Elect a new one!
- Failure detection – how to decide that a node has failed (e.g., crashed)?
- Agreement is maybe the most fundamental problem in distributed computing

One solution would be to use a master-slave relationship?
... but

- we want our systems to keep working correctly even if failures occur
- we need to avoid single points of failure

Failure model

- Processes and communications channels may fail from incorrect behavior
- Failure model defines ways in which failures may occur in order to understand their effects
- Fault tolerance
  - Failure of a system component
    - from a part of a system listener that may lead to failure
    - from a part of a system that operates on that part
    - Failure of the master
    - Fault tolerant – a system that can still operate even in the presence of faults
  - Fault-tolerant failure: failure of a system component but with the failure not propagated

Arbitrary failures

- Byzantine or malicious failures
  - Difficult to catch
  - The execution of a process deviates arbitrarily from what it should do
  - A channel may corrupt, duplicate, or deliver non-existent messages
How to determine that a process has crashed?

- Correct process
  - Exhibits no failures at any point
- Keepalive messages every X time units
  - No message: the process has failed
  - Crashing immediately after sending?
- Message delays, re-routing, etc.
- Failure detector
  - Detects if process has failed
  - Unreliable failure detector
  - Reliable failure detector

A fault was detected, what can we do now?

- Mask the failure by either hiding it or converting it into a more acceptable type of failure
  - Arbitrary failure
  - Omission failure
- Masking by redundancy
  - Information redundancy - e.g., send an extra bit
  - Time redundancy - e.g., redo an action e.g., transactions
  - Physical redundancy - e.g., add extra hardware or software

Consensus and related problems

Agreement...

- Processes need to agree on a value after proposed by one or more processes ... even in the presence of faults (crash and arbitrary)
  - Consensus
  - Any process can suggest a value, all have to reach consensus on which is correct
  - Byzantine Generals Problem (BGP)
  - One process can suggest a value, others have to agree on what that value was

General requirements

- Termination: Each non-faulty process must eventually decide on a value.
- Agreement: all non-faulty processes must agree on the same value.
- Integrity: If all correct processes propose the same value, any correct process in the decided state has chosen that value
Consensus

Processes need to agree on a single value from values proposed by all processes
- Every process begins in an undecided state
- A process propose one of $D$ possible values
- Processes exchange values
- Each process decides on one of the proposed values
  - Once choosing a value, processes enters a decided state
  - Processes can’t change their chosen value once in a decided state

Simple if processes can’t fail

- Collect all processes in a group
- Each process multicast its proposed value to the members of the group
- Each process waits for $N$ messages (including own)
  - Evaluates $\text{majority}(v_1, v_2, ..., v_D)$
  - If no majority exists, majority returns a special value

but if processes can fail...

A simple algorithm using B-multicast (No crash)

Each processor:
1. B-multicast value to all processors
2. Decide on the minimum

(only one round is needed)

Without failure

Multicast

Send a message to all processors in one round

At the end of round: everybody receives a
Start- all sending messages

B-multicast values
0,1,2,3,4
0,1,2,3,4
0,1,2,3,4
0,1,2,3,4
0,1,2,3,4

All processes send their values

Start
fail

The failed processor doesn’t multicast its value to all processors

Multicasted values
0,1,2,3,4
1,2,3,4
0,1,2,3,4
1,2,3,4
0,1,2,3,4

An f-resilient algorithm (crash failure in sync systems)

Round 1:
B-multicast my value

Round 2 to round f+1:
Multicast any new received values

End of round f+1:
Decide on the minimum value received
Example: \( f=1 \) failures, \( f+1 = 2 \) rounds needed

Start

Round 1

B-multicast all values to everybody

Example: \( f=1 \) failures, \( f+1 = 2 \) rounds needed

Round 2

B-multicast all new values to everybody

Example: \( f=2 \) failures, \( f+1 = 3 \) rounds needed

Start

Another example execution with 3 failures

Example: \( f=2 \) failures, \( f+1 = 3 \) rounds needed

Round 1

Multicast all values to everybody
Consensus in synchronous systems

Byzantine Generals Problem (BGP)

BGP in synchronous systems (3 processes)

BGP with 4 processes, 1 faulty, 2 rounds

It is impossible to derive a solution if \( N < 3f \)

It is possible to derive a solution if \( N \geq 3f + 1 \)
Final notes
- Solutions rely on system being synchronous
- Message exchanges take place in rounds
- No timing constraints
- Fischer’s impossibility result
- Solutions to consensus and BG problem exist in synchronous systems
  - Can’t distinguish between crash process and a slow one
  - There is always some continuation of the processes’ execution that avoids consensus being reached.
- Still, we manage to do quite well in practice, how can that be?

How to cope with the impossibility result...
- Mask the faults
  - Use persistent storage and allow process restarts
- Use failure detectors
  - No reliable detectors, but good enough, agree that process is crashed if it takes too long to receive a message (fail-silent)
  - Eventually weak failure detector, reaches consensus while allowing suspected processes to behave correctly instead of excluding them
- Randomization
  - Introduces an element of chance that affects the adversary's strategy

Paxos algorithm
- How to reach consensus/data consistency in distributed system that can tolerate non-malicious failures in asynchronous system?
- Classes of agents:
  - Proposers
  - Acceptors
  - Learners

Paxos Example
- Paxos Example
- Paxos Example
Paxos Example

Acceptor Z ignores proposer A’s request because it has already seen a higher numbered proposal (4 > 2). Acceptors X and Y respond to proposer B’s request with the previous highest request that they acknowledged, and a promise to ignore any lower numbered proposals. (Taken from: http://angus.nyc/2012/paxos-by-example/)

Paxos Example

Proposer B sends an accept request to each acceptor, with its previous proposal number (4), and the value of the highest numbered proposal it has seen ([n=2, v=8]). (Taken from: http://angus.nyc/2012/paxos-by-example/)

Paxos Example

All processes reach consensus on value v=8.

Summary:
- Applications
  - Chubby lock service.
  - Petal: Distributed virtual disks.
  - Frangipani: A scalable distributed file system.
- Read about multi-paxos

Summary
- Unreliable failure detectors
  - Inaccurate and incomplete
- Reliable failure detectors
  - Require the system to be synchronous
- The problem of agreement is for processes to agree on a value after one or more of the processes has proposed values (even in the presence of faults)
  - Consensus, Byzantine Generals problem, Interactive consistency...
Next Lecture

Replication