Fault Tolerance in Distributed Systems

K V SUBRAMANIAM

Basic Concepts

- *Fault Tolerance* is closely related to the notion of "Dependability". In Distributed Systems, this is characterized under a number of headings:

- *Availability* – the system is ready to be used immediately.
- *Reliability* – the system can run continuously without failure.
- *Safety* – if a system fails, nothing catastrophic will happen.
- *Maintainability* – when a system fails, it can be repaired easily and quickly (and, sometimes, without its users noticing the failure).

Availability

- Availability: expresses the fraction of time a system is operational
  - a 0.999999 availability means the system is not operational at most one hour in a million hours
  - a system with high availability may in fact fail -> its recovery time and failure frequency must be small enough to achieve the desired availability
  - high availability is important in airline reservations, telephone switching etc, in which every minute of downtime translates into revenue loss…

Reliability

- Reliability: the likelihood that a system will remain operational for the duration of a mission
  - requirement might be stated as 0.999999 availability for a 10 hr mission -> the probability of failure during the mission must be at most 10^{-6}
  - Very high reliability is most important in critical applications - space shuttle, industrial control, in which failure could mean loss of life!

But, What Is “Failure”?

- Definition:

- A system is said to “fail” when it *cannot meet* its promises.

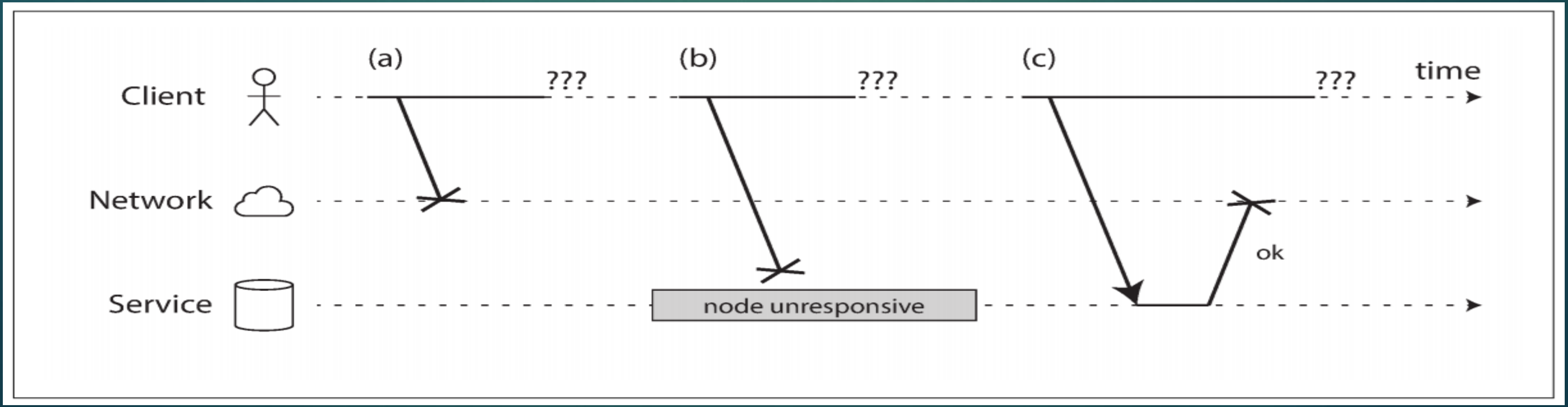
- A failure is brought about by the *existence* of “errors” in the system.

- The *cause* of an error is called a “fault”.

Issues with distributed systems

- Sender issues
  - Cable unplugged? High load
- Network issues
  - Cable/port faults
  - Can cause temporary faults
- ?

- Receiver issues
  - Failure in hardware or software
- Response not received



Types of Fault

Transient Fault

- appears once, then disappears.

Intermittent Fault

- occurs, vanishes, reappears; but: follows no real pattern (worst kind).

Permanent Fault

- once it occurs, only the replacement/repair of a faulty component will allow the DS to function normally.

Classification of Failure Models

Type of failure	**Description**
Crash failure	A server halts, but is working correctly until it halts.
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests. - A server fails to receive incoming messages. - A server fails to send outgoing messages.
Timing failure	A server's response lies outside the specified time interval.
Response failure *Value failure* *State transition failure*	The server's response is incorrect. - The value of the response is wrong. - The server deviates from the correct flow of control.
Arbitrary failure	A server may produce arbitrary responses at arbitrary times.

Common Metric - Failure

- MTBF – Mean Time Between Failures
  - Avg operating time between failures
  - =Total uptime/#failure
- Typically as #components increase, MTBF increases
- Prob of failure of system = Prob that any one can fail

- MTTR – Mean Time To Repair
  - Avg time it takes to restore a failed system
  - =Total Downtime/#repairs

Running Example

- Let us consider our BookKarts example. We needed at a point 50 compute servers to run our application. In most cloud computing solutions, we have a centralized controller and compute nodes as a workers.
  - How to deal with failure of worker nodes?
  - How to deal with failure of controller nodes?

Handling worker node failure

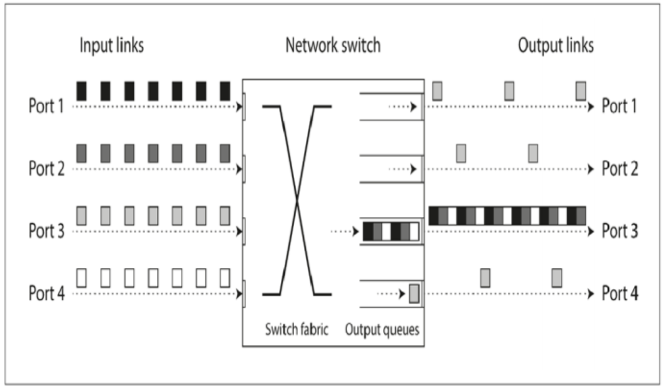
- If one VM/Container fails
- Detection:
  - Monitor the VM using a heartbeat mechanism
- Corrective action:
  - And start another VM/Container

Solution: Coordinator failure strategy

- Simple solution
  - Why don’t we keep multiple copies of the coordinator node?
- Challenges:
  - If we have a cluster of nodes
    - How do we find out which one should be active?
    - How do we detect failure?

Why do failures occur?

- Failure of a node
- Unreliable communication/congestion in network



If several machines send network traffic to the same destination, its switch queue can fill up. Here, ports 1, 2, and 4 are all trying to send packets to port 3.

Failure Masking by Redundancy

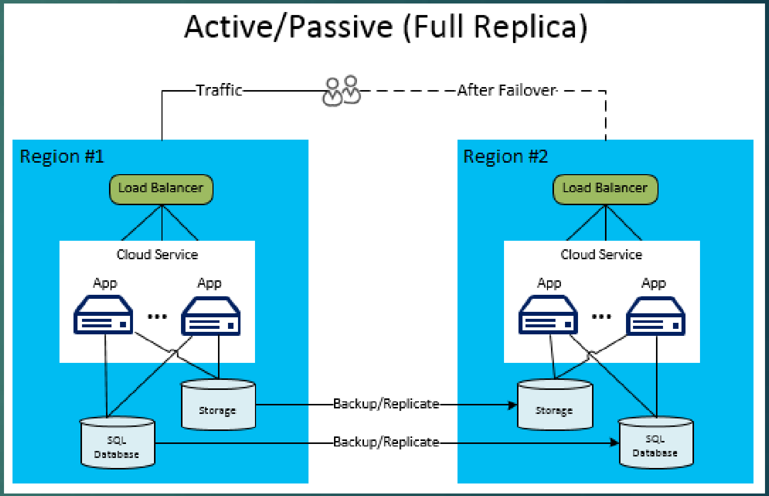
- **Strategy**: hide the occurrence of failure from other processes using *redundancy*.
  - *Information Redundancy* – add extra bits to allow for error detection/recovery (e.g., Hamming codes and the like).
  - *Time Redundancy* – perform operation and, if needs be, perform it again. Think about how transactions work (BEGIN/END/COMMIT/ABORT).
  - *Physical Redundancy* – add extra (duplicate) hardware and/or software to the system.

Process Resilience – active/standby

Processes can be made fault tolerant by arranging to have a group of processes, with each member of the group being *identical*.

Messages are sent to active node only

Other is in standby; ready to take over when there is a failure



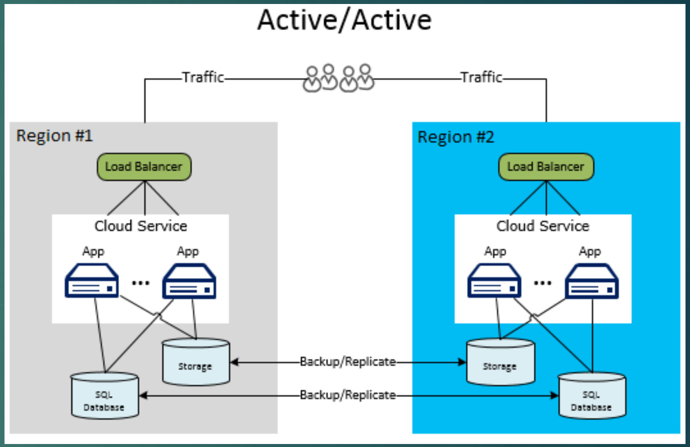
https://docs.microsoft.com/en-us/azure/architecture/resiliency/disaster-recovery-azure-applications

Process Resilience – active/active

Processes can be made fault tolerant by arranging to have a group of processes, with each member of the group being *identical*.

A message sent to the group is delivered to all of the "copies" of the process (the group members), and then *only one* of them performs the required service.

If one of the processes fail, it is assumed that one of the others will still be able to function (and service any pending request or operation).



https://docs.microsoft.com/en-us/azure/architecture/resiliency/disaster-recovery-azure-applications

Ask the leader..

- If multiple coordinators are active
  - How to determine which one should answer?
  - Need mechanisms of leader election