Lecture notes: About failures in the Cloud

M2 MOSIG: Cloud Computing, from infrastructure to applications

Thomas Ropars

2025

These lecture notes discuss failures in the Cloud.

1 Some definitions

1.1 Fault, error, and failures

An **error** is the part of the system state that may cause a subsequent **failure**: a failure occurs when an error reaches the service interface and alters the service. A **fault** is the adjudged or hypothesized cause of an **error**

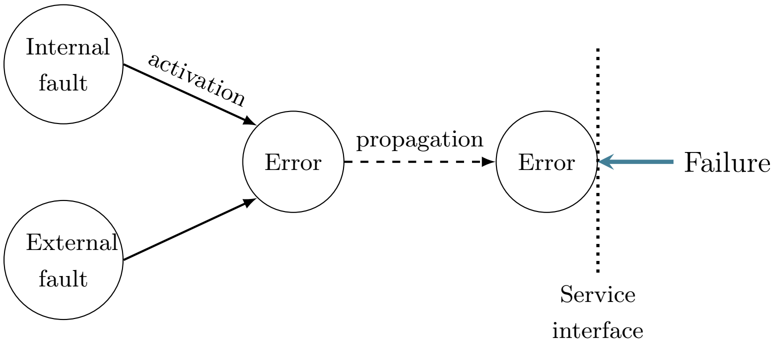

Figure 1: Fault/error/failure

The failure of a component is a fault from the perspective of the component using it, as illustrated in Figure 2.

Failures and Cloud applications When building cloud applications, we would like to avoid that a failure impacting an internal component of the system, lead to a failure for a user service.

2 A focus on availability

Availability can be defined as the degree to which a component is operational and accessible when required for use. Said differently it is the portion of time for which a component is able to perform its function.

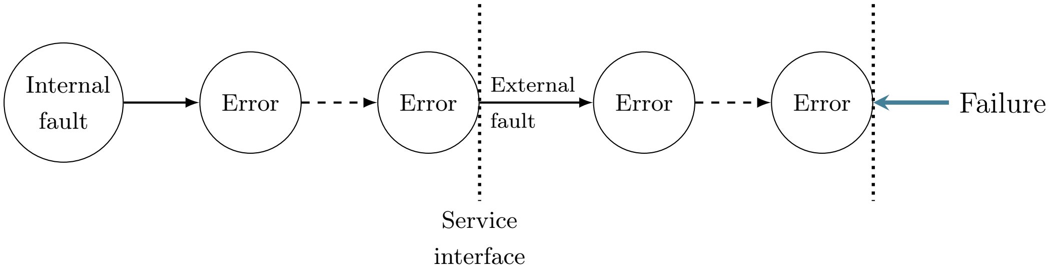

Figure 2: About errors and failures

To compute availability, we need to introduce two parameters: MTBF (Mean Time Between Failures), MTTR (Mean Time To Repair). Hence we can compute the availability of a system as:

$$A = \frac{Uptime}{Uptime + Downtime} = \frac{MTBF}{MTBF + MTTR}$$

Table 1 presents availability classes. Mission critical systems should be *ultra available.*

System type	Unavailability (min/year)	Availability
Managed	5256	99 %
Fault tolerant	53	99.99 %
Highly available	5	99.999 %
Ultra available	0.05	99.99999 %

Table 1: About nines classes

A large majority of applications running in the Cloud are online services that we cannot afford to stop and restart. We need to target *high availability* (*5 nines*).

3 Failures in Data Centers and Clouds

3.1 Anatomy of the Cloud

We list here some characteristics of Cloud environments. These characteristics contribute to the risk of failures in the Cloud and/or to the difficulty of managing failures.

- Very large number of nodes (from thousands to millions)
- Geo-distributed
  - Multiple data centers in different geographical area (different continents)
- Huge number of users
  - Shared physical resources

- Complex hardware/software stacks
  - High-end hardware
  - Virtualized/contenerized environment
  - Data center resource management / Orchestration
  - All kind of services (File systems, databases, message brokers, etc.) with different designs
  - User applications
- Aggressive optimization for resource management
  - Computing and power oversubscription.
    - Assign more work to a node than it is theoretical able to deal with. As long as not all jobs have their peak consumption at the same time everything goes find. If we reach the limits, some jobs need to be killed. See [5] for an example.
  - Access to resource based on bidding (e.g., spot instances in AWS)
    - Notification 2 minutes in advance in case of interruption (but most often treated as a failure – no dedicate procedure to handle these cases)

3.2 Failures in the Cloud

Failures cannot be neglected in the Cloud. There are plenty of sources of failures:

- Failure of hardware components
  - Each component of a distributed system has a very high MTBF (more than 10 years for HDDs). However, with a very high number of components, the risk of experiencing a failure becomes much higher.
    - Simple computation assuming a uniform distribution of failure probability: $100K$ hardware components with MTBF of 10 years $\Rightarrow$ one failure every 1h45.
  - Some failures are very difficult to anticipate (See *"Sharks attack the Internet"*)
- Network failures are common.
  - In the Google infrastructure about 1 network failure per week can be observed [2].
- Catastrophic failures can also occur (e.g., OVH fire in March 2021)

Other major source of failures are :

- Software failures
  - Any software component in a system includes bugs.
  - Hardware failures may also lead to software failures, even in cases where softwares are supposed to be able to deal with failures [1].
- Human errors

  - Platforms and software are designed and operated by humans. Humans make mistakes.
  - Many services outages come from human errors. The human errors will translate into software or hardware failures.
  - Solutions to limit the human errors: automatizing (infrastructure-as-code), devops, etc.
  - The study of network failures at Google [2] show that more than 60% of the failures occur during *Management Operations* despite the fact that these operations are partially automated.

It should also mentionned that not all failures in large scale infrastructures have a *fail-stop* behavior. Hardware *performance faults* can also occur [3]:

- Such faults are characterized by the fact that a hardware device becomes slower than expected, but continues working.
- Such faults can be very difficult to deal with because they can have various causes, varying symptoms, long chains of root causes, and it can take very long time to detect them (months?).
  - Example taken from [3]: A fan stopped working, making other fans run at maximal speeds, causing heavy vibration that degraded the disk performance.

4 Detecting failures

Dealing with failures requires being able to detect failures. Some fault tolerant systems may continue working despite hardware or software failures that would impact a subset of the components of the system. However, ultimately failed components need to be replaced to maintain the same level of availability for the services.

On a single machine, it is most of the time easy to know if everything works correctly or not, especially because most errors lead to a crash. In a distributed system, things can be more complex.

Definition 1 (Distributed systems – by L. Lamport) *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*

Detecting failures in distributed systems is difficult for two main reasons:

Partial failures: Some parts of the system may stop working in unpredictable ways (but the other parts of the system might not be aware of it).

Asynchronous system: The time for messages to travel across the network may significantly vary.

When discussing about data centers and clouds, we are considering mostly **shared-nothing** infrastructures. It means that the only way to interact between the nodes is by sending/receiving messages over the network.

The network in a data center should be considered as unreliable and asynchronous. When a packet is sent, it is not guaranteed that it will arrive, and we don't know when it will arrive.

In such a system, discovering if something is going wrong and what is going wrong is difficult. Imagine that a node `A` sends a request to another node `B` and does not receive an answer. What could be the reasons?

- The request was lost and never reached B.
- The request is delayed and has not been delivered yet.
- Node B is down (maybe because it crashed or because it shut down)
- Node B may be experiencing a transient issue (slow-down) and will respond later
- Node B may have processed the request but the answer was lost on the way back
- Node B may have processed the request but the delivery of the answer is delayed

At the end, it is impossible to tell why no answer was received. It is not even possible to know whether the request has been processed or not. All of these reasons make it challenging to implement fault tolerance in this kind of infrastructure.

About *gray* failures There are multiple evidences that some failures in the Cloud might even be more difficult to detect. They are sometimes called *gray* failures and are characterized by the fact that they are perceived differently by different entities (for instance, some entities do not detect any failures while others are negatively impacted) [4].

To go further

Some references can complement the material presented in these lecture notes:

- Chapter 8 of *Designing Data-Intensive Applications* by Martin Kleppmann

References

[1] A. Alquraan et al. An analysis of network-partitioning failures in cloud systems. In *OSDI*, 2018.

[2] R. Govindan et al. Evolve or die: High-availability design principles drawn from googles network infrastructure. In *SIGCOMM*, 2016.

[3] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)*, 14(3):1–26, 2018.

[4] P. Huang et al. Gray failure: The achilles' heel of cloud-scale systems. In *HotOS*, 2017.

[5] V. Sakalkar et al. Data center power oversubscription with a medium voltage power plane and priority-aware capping. In *ASPLOS*, 2020.