Availability

Learn about availability, how to measure it, and its importance.

We'll cover the following

What is availability?

Availability is the percentage of time that some service or infrastructure is accessible to clients and is operated upon under normal conditions. For example, if a service has 100% availability, it means that the said service functions and responds as intended (operates normally) all the time.

Measuring availability

Mathematically, availability, `A`, is a ratio. The higher the `A` value, the better. We can also write this up as a formula:

$$\text{A (in percent)} = \frac{(Total\ Time - Amount\ Of\ Time\ Service\ Was\ Down)}{Total\ Time} * 100$$

We measure availability as a number of nines. The following table shows how much downtime is permitted when we're using a given number of nines.

The Nines of Availability

Availability Percentages versus Service Downtime			
Availability %	**Downtime per Year**	**Downtime per Month**	**Downtime per**
90% (1 nine)	36.5 days	72 hours	16.8
99% (2 nines)	3.65 days	7.20 hours	1.68
99.5% (2 nines)	1.83 days	3.60 hours	50.4 mi
99.9% (3 nines)	8.76 hours	43.8 minutes	10.1 mi
99.99% (4 nines)	52.56 minutes	4.32 minutes	1.01 mi
99.999% (5 nines)	5.26 minutes	25.9 seconds	6.05 se
99.9999% (6 nines)	31.5 seconds	2.59 seconds	0.605 se
99.99999% (7 nines)	3.15 seconds	0.259 seconds	0.0605 se

Availability and service providers

Each service provider may start measuring availability at different points in time. Some cloud providers start measuring it when they first offer the service, while some measure it for specific clients when they start using the service. Some providers might not reduce their reported availability numbers if their service was not down for all the clients. The planned downtimes are excluded. Downtime due to cyberattacks might not be incorporated into the calculation of availability. Therefore, we should carefully understand how a specific provider calculates their availability numbers.

Reliability

Learn about reliability, how to measure it, and its importance.

We'll cover the following

What is reliability?

Reliability, R, is the probability that the service will perform its functions for a specified time. R measures how the service performs under varying operating conditions.

We often use **mean time between failures (MTBF)** and **mean time to repair (MTTR)** as metrics to measure R.

$$MTBF = \frac{Total\ Elapsed\ Time - Sum\ of\ Downtime}{Total\ Number\ of\ Failures}$$

$$MTTR = \frac{Total\ Maintenance\ Time}{Total\ Number\ of\ Repairs}$$

(We strive for a higher MTBF value and a lower MTTR value.)

Reliability and availability

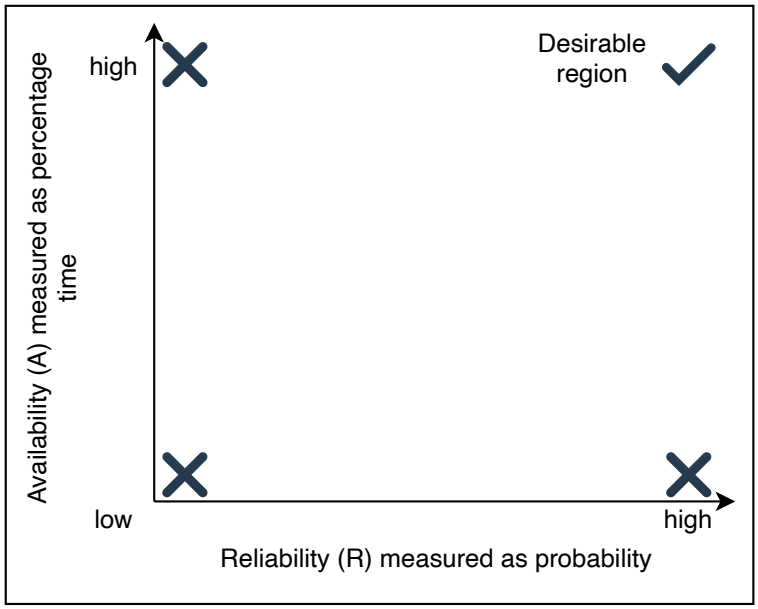
Reliability and availability are two important metrics to measure compliance of service to agreed-upon service level objectives (SLO).

The measurement of availability is driven by time loss, whereas the frequency and impact of failures drive the measure of reliability. Availability and reliability are essential because they enable the stakeholders to assess the health of the service.

Reliability (R) and availability (A) are two distinct concepts, but they are related.

Mathematically, A is a function of R. This means that the value of R can change independently, and the value of A depends on R. Therefore, it's possible to have situations where we have:

- low A, low R
- low A, high R
- high A, low R
- high A, high R (desirable)



Availability as a function of reliability

Point to ponder.

Question

What is the difference between reliability and availability?

Hide Answer

Reliability measures how well a system performs its intended operations (functional requirements). We use averages for that (Mean Time to Failure, Mean Time to Repair, etc.)

Availability measures the percentage of time a system accepts requests and responds to clients.

Example 1: A certain system may be 90% available but only reliable 80% of the time.

Example 2: Suppose we consider our "system" the stuff inside a data center (hardware + software). Let's assume this data center suffers a network failure such that no outsider traffic is coming in and no insider traffic is going out. In this case, instantaneous availability might be zero (because clients cannot reach the service) even though inside the data center, all systems are perfectly functioning (instantaneous reliability 100%).

We use both of them (reliability and availability) in different contexts. For example, storage vendors often quote MTTF for their disks. Most online services use uptime (as a measure of availability) in their SLAs. For example, the uptime of EC2 virtual machines is 99.95%.

Back

Mark As Completed

Next

Availability

Scalability

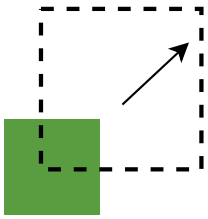
Scalability

Learn about scalability and its importance in system design.

We'll cover the following

What is scalability?

Scalability is the ability of a system to handle an increasing amount of workload without compromising performance. A search engine, for example, must accommodate increasing numbers of users, as well as the amount of data it indexes.



The workload can be of different types, including the following:

- **Request workload**: This is the number of requests served by the system.
- **Data/storage workload**: This is the amount of data stored by the system.

Dimensions

Here are the different dimensions of scalability:

- **Size scalability**: A system is scalable in size if we can simply add additional users and resources to it.
- **Administrative scalability**: This is the capacity for a growing number of organizations or users to share a single distributed system with ease.

- **Geographical scalability**: This relates to how easily the program can cater to other regions while maintaining acceptable performance constraints. In other words, the system can readily service a broad geographical region, as well as a smaller one.

Different approaches of scalability

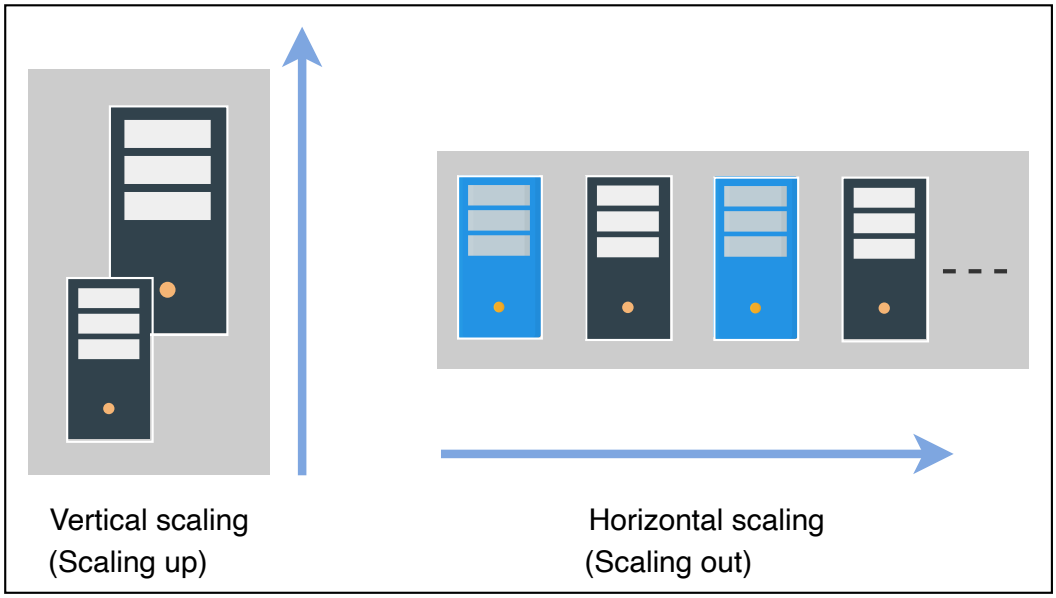
Here are the different ways to implement scalability.

Vertical scalability—scaling up

Vertical scaling, also known as "**scaling up**," refers to scaling by providing additional capabilities (for example, additional CPUs or RAM) to an existing device. Vertical scaling allows us to expand our present hardware or software capacity, but we can only grow it to the limitations of our server. The dollar cost of vertical scaling is usually high because we might need exotic components to scale up.

Horizontal scalability—scaling out

Horizontal scaling, also known as "**scaling out**," refers to increasing the number of machines in the network. We use commodity nodes for this purpose because of their attractive dollar-cost benefits. The catch here is that we need to build a system such that many nodes could collectively work as if we had a single, huge server.



Back

Mark As Completed

Next

Reliability

Maintainability

Maintainability

Learn about maintainability, how to measure it, and its relationship with reliability.

We'll cover the following

What is maintainability?

Besides building a system, one of the main tasks afterward is keeping the system up and running by finding and fixing bugs, adding new functionalities, keeping the system's platform updated, and ensuring smooth system operations. One of the salient features to define such requirements of an exemplary system design is **maintainability**. We can further divide the concept of maintainability into three underlying aspects:

1. **Operability**: This is the ease with which we can ensure the system's smooth operational running under normal circumstances and achieve normal conditions under a fault.
2. **Lucidity**: This refers to the simplicity of the code. The simpler the code base, the easier it is to understand and maintain it, and vice versa.
3. **Modifiability**: This is the capability of the system to integrate modified, new, and unforeseen features without any hassle.

Measuring maintainability

Maintainability, `M`, is the probability that the service will restore its functions within a specified time of fault occurrence. `M` measures how conveniently and swiftly the service regains its normal operating conditions.

For example, suppose a component has a defined maintainability value of 95% for half an hour. In that case, the probability of restoring the component to its fully active form in half an hour is 0.95.

Maintainability

> **Note:** Maintainability gives us insight into the system's capability to undergo repairs and modifications while it's operational.

We use (mean time to repair) MTTR as the metric to measure `M`.

$$MTTR = \frac{Total\ Maintenance\ Time}{Total\ Number\ of\ Repairs}$$

In other words, MTTR is the average amount of time required to repair and restore a failed component. Our goal is to have as low a value of MTTR as possible.

Maintainability and reliability

Maintainability can be defined more clearly in close relation to reliability. The only difference between them is the variable of interest. Maintainability refers to `time-to-repair`, whereas reliability refers to both `time-to-repair` and the `time-to-failure`. Combining maintainability and reliability analysis can help us achieve availability, downtime, and uptime insights.

Fault Tolerance

Learn about fault tolerance, how to measure it, and its importance.

We'll cover the following

What is fault tolerance?

Real-world, large-scale applications run hundreds of servers and databases to accommodate billions of users' requests and store significant data. These applications need a mechanism that helps with data safety and eschews the recalculation of computationally intensive tasks by avoiding a single point of failure.

Fault tolerance refers to a system's ability to execute persistently even if one or more of its components fail. Here, components can be software or hardware. Conceiving a system that is hundred percent fault-tolerant is practically very difficult.

Let's discuss some important features for which fault-tolerance becomes a necessity.

Availability focuses on receiving every client's request by being accessible 24/7.

Reliability is concerned with responding by taking specified action on every

client's request.

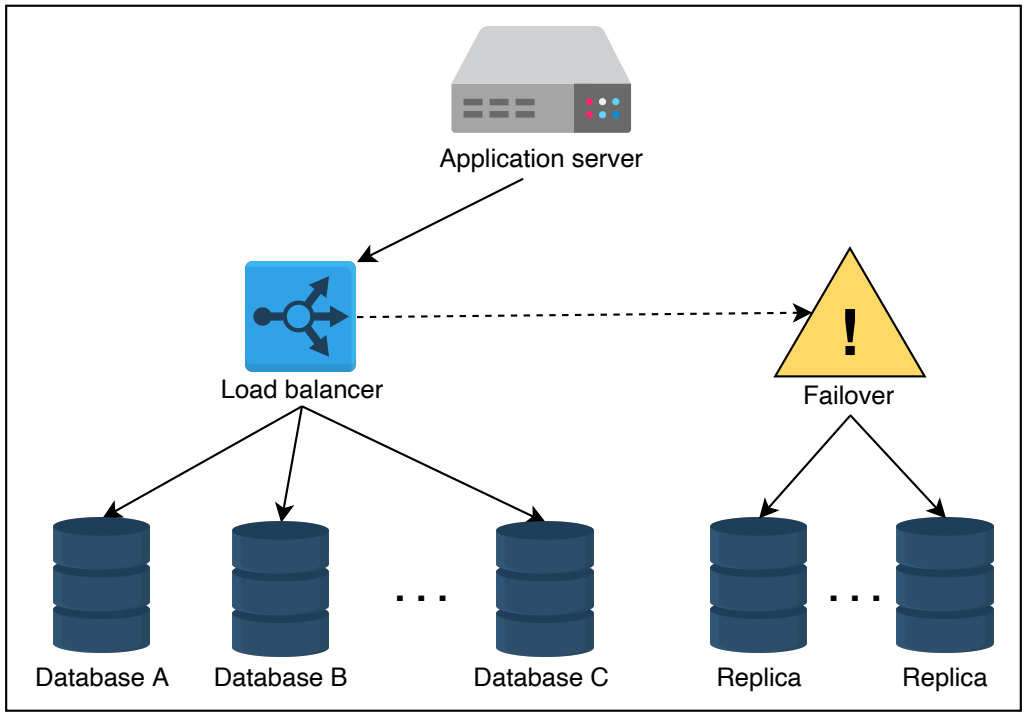
Fault tolerance techniques

Failure occurs at the hardware or software level, which eventually affects the data. Fault tolerance can be achieved by many approaches, considering the system structure. Let's discuss the techniques that are significant and suitable for most designs.

Replication

One of the most widely-used techniques is **replication-based fault tolerance**. With this technique, we can replicate both the services and data. We can swap out failed nodes with healthy ones and a failed data store with its replica. A large service can transparently make the switch without impacting the end customers.

We create multiple copies of our data in separate storage. All copies need to update regularly for consistency when any update occurs in the data. Updating data in replicas is a challenging job. When a system needs strong consistency, we can synchronously update data in replicas. However, this reduces the availability of the system. We can also asynchronously update data in replicas when we can tolerate eventual consistency, resulting in stale reads until all replicas converge. Thus, there is a trade-off between both consistency approaches. We compromise either on availability or on consistency under failures—a reality that is outlined in the CAP theorem.

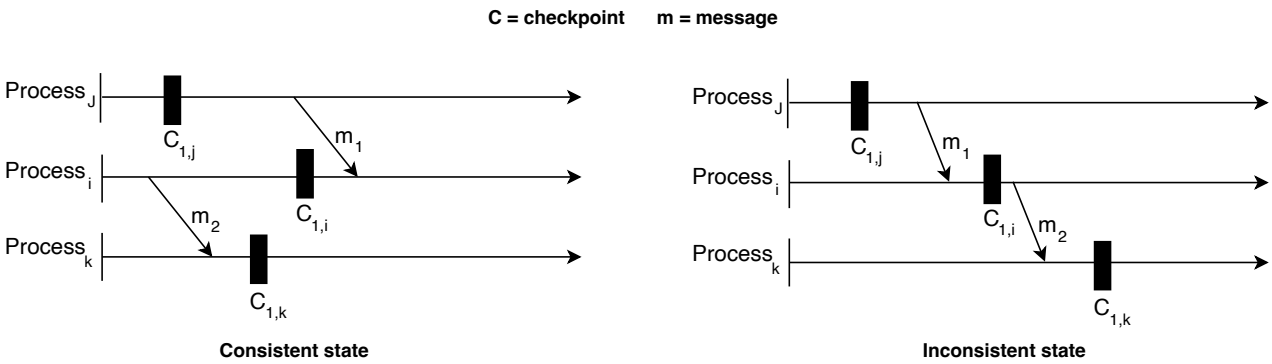


Replication-based fault tolerance

Checkpointing

Checkpointing is a technique that saves the system's state in stable storage when the system state is consistent. Checkpointing is performed in many stages at different time intervals. The primary purpose is to save the computational state at a given point. When a failure occurs in the system, we can get the last computed data from the previous checkpoint and start working from there.

Checkpointing also comes with the same problem that we have in replication. When the system has to perform checkpointing, it makes sure that the system is in a consistent state, meaning that all processes are stopped except read processes that do not change the state of the system. This type of checkpointing is known as **synchronous checkpointing**. On the other hand, checkpointing in an inconsistent state leads to data inconsistency problems. Let's look at the illustration below to understand the difference between a consistent and an inconsistent state:



Checkpointing in a consistent and inconsistent state. In the left illustration, the first checkpoints at process J and i are consistent because m1 was sent and received after the checkpoints. On the contrary, in the right hand illustration first checkpoint in process J does not know about m1 while the first checkpoint at process i recorded the reception of message m1. Hence the inconsistent state.

Consistent state: The illustration above shows no communication among the processes when the system performs checkpointing. All the processes are sending or receiving messages before and after checkpointing. This state of the system is called a consistent state.

Note that on the left side illustration above, the sending and reception of any message are recorded in respective checkpoints. For example, message m1's sending and reception are recorded in the left illustration, while on the right illustration, the checkpoint on process J does not record sending it, while Process i does record its reception. If both systems fail at that point, and we use checkpoints for recovery, then one checkpoint will know about m1's reception, while the other will have no record of sending it.

Inconsistent state: The illustration also displays that processes communicate through messages when the system performs checkpointing. This indicates an inconsistent state, because when we get a previously saved checkpoint, *Process i* will have a message (m_1) and *Process j* will have no record of message sending.

← Back

Mark As Completed

Next →

Maintainability

Put Back-of-the-envelope Numbers ...