umang goel Follow 5 min read · Oct 15, 2021

Handling Failures in Distributed Systems!!

Failure is inevitable for any system, even 99.99% availability of the system means the unavailability of about 52 minutes in a year. In any distributed system there are large number of components which work

together to complete a task. As the system gets more complex and has more components, the chances of failures will increase resulting in falling reliability.

"No system can be made 100% failure free" doesn't mean that we take this as an excuse to overlook failures but instead this increases the responsibility of system designer to consider various failure points upfront and handle them gracefully and make the system more resilient towards failures.

> ***Fault vs Failure:***
>
> *Fault is the internal incorrect state of the system whereas failure is the inability of the system to complete a work. Faults will lead to failures if they are not properly handled and contained on time. In short, a system is termed as resilient if it can prevent these faults from turning into failures.*

Different components may have different reasons of failures which will require variety of monitoring tools and alerting mechanisms to discover, identify and debug. After the discovery of the failure causes it's important to fix those causes or make the system resilient enough at the time of designing so that it can handle the failures gracefully and recover from such outages as soon as possible. In this article we will discuss on some of the common failure points of distributed systems and possible resolutions.

Failure of application servers

Application servers may crash due to several events like datacenter goes down, CPU usage/ memory usage increases beyond the limit, some faults

in the application code, power failures, natural calamities etc. If the application servers start failing randomly use of monitoring tools can help to understand the reasons of failures and fix them in order to reduce the system downtime.

Alerts can be configured on the application server metrics or on the load balancers and proxies to discover the sudden increase in application failures so that the application team is well informed and the remedial action is taken on time. Some mechanisms that can be used to reduce the downtime can be as follows:

A. Out of Rotation: If a node fails in the application server farm, it should be taken out of rotation and new node can be added . This can be done using some automated scripts or manual intervention.

B. Backup Cluster: In case whole cluster or the application server farm goes down the traffic can be routed to the backup cluster which can be present in some different datacenter within the same region or other region.

Failure of downstream services

In distributed systems, services may directly communicate with each other over network using HTTP/TCP. Failure scenario may happen when service A wants to communicate to service B but service B is unable to respond, this can happen because of numerous reasons like service unavailability, network issues, dependencies failure etc. This can further lead to service A not performing its duties and may lead to failure of the entire system due to this cascading effect.

This type of failure can be detected using the proper logging and alerting in every application. Alerts can be configured on internal errors , timeouts or even the anomalies in response time as compared to historical trends for downstream services. Below are some techniques to recover or handle these integration failures:

A. Implementation of proper Retries : In case of failure try it again based on some retry policy. Retries must be configured in such a way that it doesn't put too much extra load on the already downgraded downstream services and give them some breathing space to recover.

> *Retries reduce the recovery time for intermittent failures but immediate retries might worsen the situation as the downgraded system might take time to recover.*

B. Fallbacks/ Circuit Breakers: Caches can also be used as fallbacks to store the data for the various repeated requests so in case downstream failures eventually consistent data from the cache is served. But the caches may not prove useful in all use-cases, so these failures should be handled gracefully i.e instead of returning the error return the proper degraded response.

Failures of the databases

Databases are the core components of most systems. Although the probability of the the databases failure is low but it's not zero. Databases may be considered failed from application perspective if the application is not able to read/write and this can happen due to various reasons like network issue making the database unreachable, database gets choked due to high CPU/Memory utilisation, database servers itself goes down.

As the data is primary component of every system handling database failures becomes utmost important.

Considering the criticality of DB failure various alerts can be configured both at the application level and database level to detect these failures, these alerts can be configured on increase CPU/memory utilisation, IOPS, throughput, disk space of the DB and also on the application side like increase in the response time from DB or error in reaching the DB servers etc.

There can be different ways to handle the DB failures which can vary from one use-case to other depending on the criticality of the data being handled :

A. Redundancy and Replication : Having a backup DB with all the data replicated from the main DB in it will reduce the probability of single point of failure and in case of the failure this redundant DB can be used to serve the data needs till the main DB is back.

B. Fallbacks and retries : The application can use fallback mechanisms for the upcoming requests till the DB is ready to take up the load again. Reads can be served either from cache or from redundant DB. Writing data to files on disk or pushing the write payload to streams will give the flexibility to the retry writing in the DB when DB is up and running. Even the redundant DB can be used for writing the data and once the main DB is up data can be synced between the two.

Note:- These techniques only work if the data is not transactional. In case of the transactional data whole transaction needs to be rolled back.

Failure of message queues/streams

Queues and streams are important component used for delivering the messages and events. These failures may happen due to infrastructure failures, multiple nodes are reachable, minimum in-sync replica count is not achieved etc. Monitoring and alerting on the hardware and software infrastructure will ensure the early detection of the failures. Some mechanisms that can be used to deal with such scenarios are:

A. Build redundancy : Push the message to the redundant stream or queue. This was even the transactional messages will not be lost. Redundancy can be best achieved by creating the resources in different data-centres and availability zones.

B. Transactional-logs and retries : In case the message is tier 3 message it can put in some transactional log temporarily. Application can retry to push the messages in transactional logs periodically till the stream recovers.

C. Dead Letter Queues : In case the message was not pushed to stream on time even after retries it can be placed in the DLQ for further analysis.

Conclusion

System cannot be made 100% resilient but use of some common design considerations like redundancy, retries, timeouts will reduce the time to recover from failures and improve the overall resiliency.

Written by umang goel

851 followers · 8 following

Follow

Engineering Lead@Microsoft For sessions: https://topmate.io/umang_goel

No responses yet

I Ishu

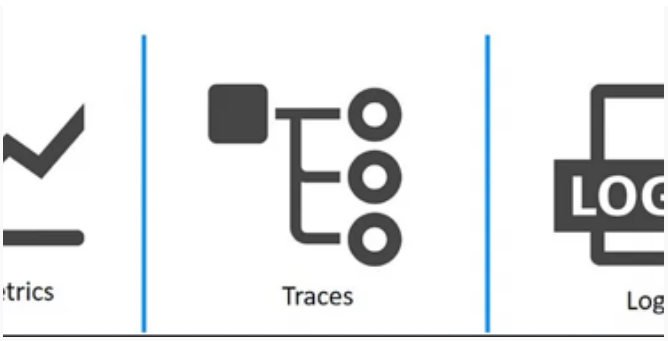
What are your thoughts?

More from umang goel

umang goel

System Design: Amazon Flash Sale!

Jun 3, 2023 260 5

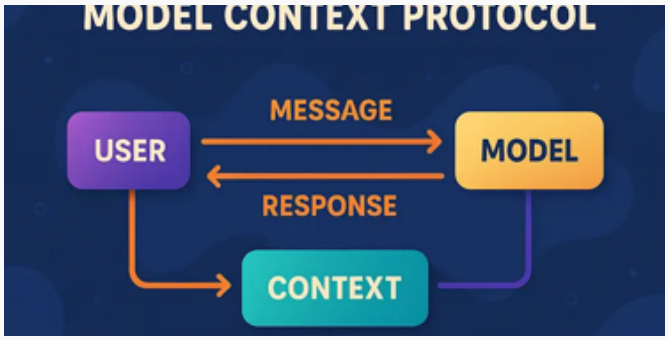


umang goel

Observability : Logs vs Traces vs Metrics!

Software systems have become the integral part of any organisation. As the...

Dec 10, 2021 201 6

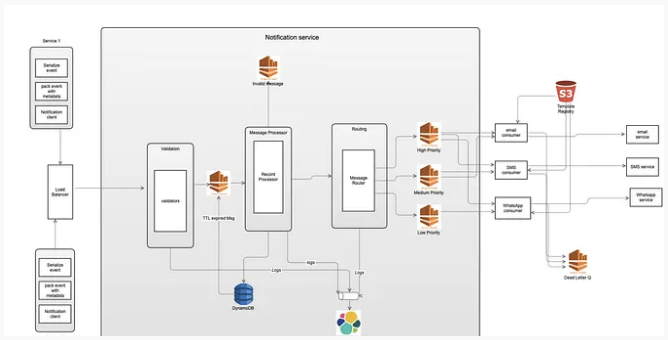


umang goel

How to setup MCP Client

In Part 2 of this series, we learned how to setup MCP server from scratch. If you...

Jul 18



umang goel

System Design : Notification Service

Problem Statement:

Oct 21, 2021 554 5

See all from umang goel

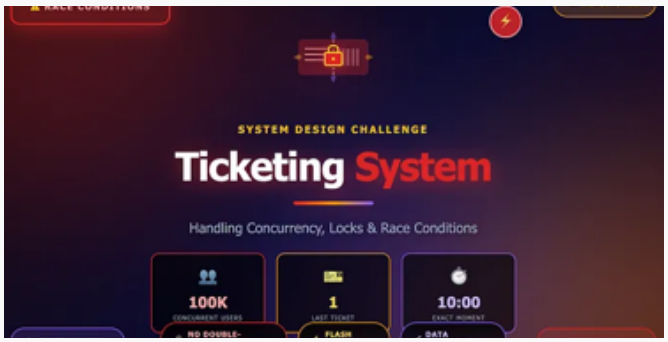
Recommended from Medium

In Nerd For Tech by Nikhil Bhatnagar

System Design of a Ride Booking System (Uber/Ola/Rapido)

This article deals with the Hld and Lld of a ride booking system where we cover the...

Jun 15 6 1

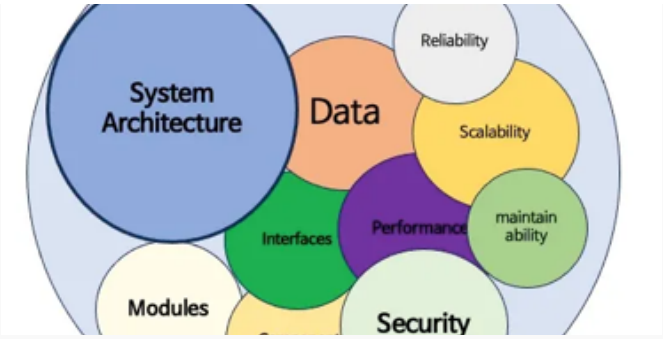


Arvind Kumar

Building a Ticketing System: Concurrency, Locks, and Race...

What happens when 100,000 fans try to book the same concert ticket at exactly...

Oct 30 240 4

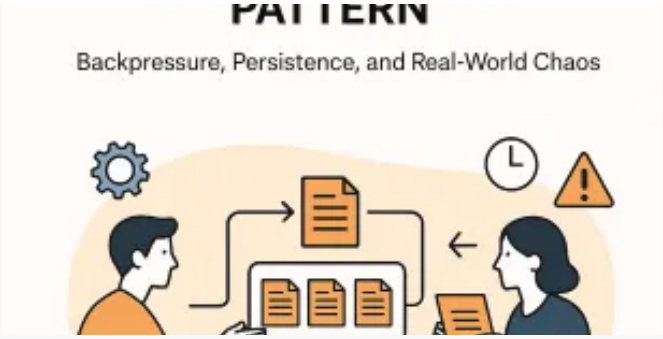


Observability Guy

Scalable Solutions for Double Booking: System-Design...

Double booking—two people getting the same seat, room, or calendar slot at once...

Oct 17



Alpesh Dhamelia

Mastering the Producer-Consumer Pattern:...

In today's era of microservices and distributed systems, data flow...

Jul 30

Ben Meehan

Understanding Change Data Capture (CDC): The Backbone o...

Modern software systems generate and consume massive volumes of data at ever...

5d ago 7

Mitali Choubisa

Hashing System

When dealing with large amounts of data, one common challenge is how to quickly...

Sep 3 5

See more recommendations