<https://doordash.engineering/2023/03/14/failure-mitigation-for-microservices-an-intro-to-aperture/>

<https://medium.com/geekculture/design-patterns-for-microservices-circuit-breaker-pattern-276249ffab33>

<https://www.softwaretestinghelp.com/java-architect-interview-questions/>

<https://medium.com/@vivekkadiyanits/tricky-java-architect-interview-question-356d6027d895>

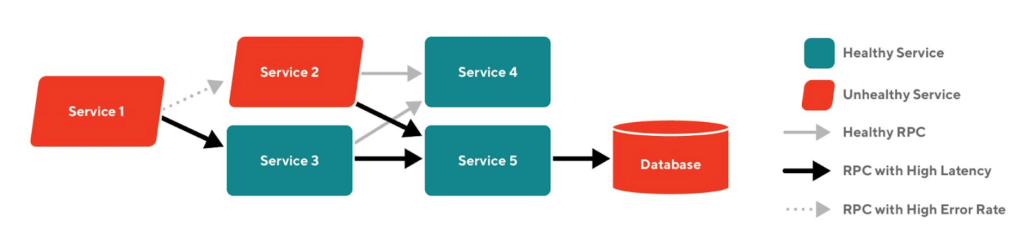
**Classification of MSA Failures**

We can categorize mainly four types of failures in Microservices Architecture –

* **Cascading failure:** a chain reaction of different interconnected services failing.
* **Retry storm:** when retries put extra pressure on a degraded service.
* **Death spiral**: some nodes fail, causing more traffic to be routed to the healthy nodes, making them fail too.
* **Metastable failure**: an overarching term that describes failures that can’t self-recover because of the existence of a positive feedback loop.

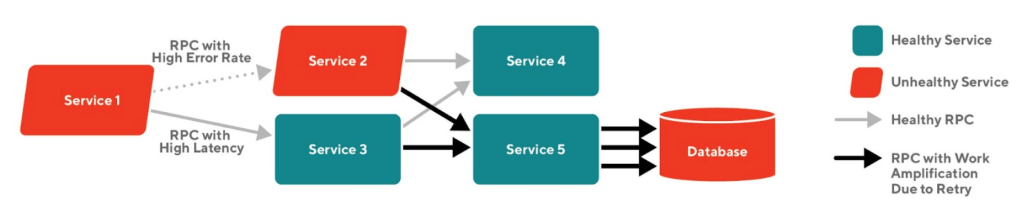
**Cascading failure** refers to the phenomenon where the failure of a single service leads to a chain reaction of failures in other services. We documented[a severe outage of this kind](https://doordash.engineering/2022/05/13/doordashs-may-12th-outage/) in our blog. In that case, the chain of failure started from a seemingly innocuous database maintenance, which increased the database latency. The latency then bubbled up to the upstream services, causing errors from timeouts and resource exhaustion. The increased error rates triggered a misconfigured circuit breaker, which stopped traffic between a lot of unrelated services, resulting in an outage with a wide blast radius.

Cascading failure describes a general phenomenon where failure spreads across services, and there is a wide range of ways a failure can transmit to another. Retry storm is one common mode of transmission among others, which we will dive into next.

Figure 1: The diagram illustrates a microservice system experiencing cascading failure. The issue starts with a degraded database that experiences increased latency. This latency spreads through RPC call chains and triggers a bug in Service 2. This bug causes Service 2 to respond with errors to some requests, which then impacts Service 1.

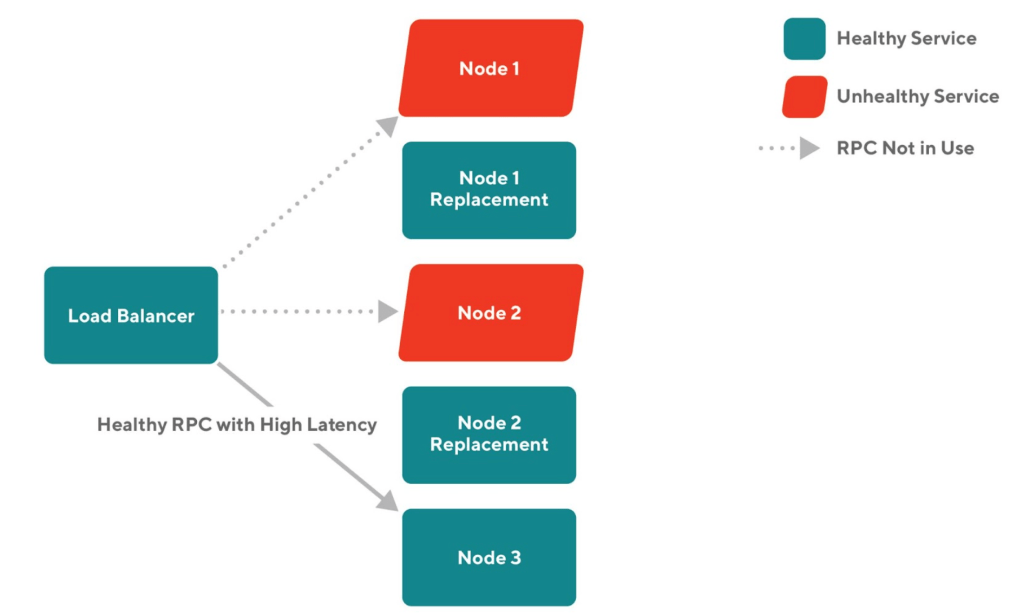
**Retry storm**

Due to the unreliable nature of [Remote Procedure Calls(RPC)](https://doordash.engineering/2022/06/07/improving-fault-tolerance-with-rpc-fallbacks-in-doordashs-microservices/), the RPC call sites are often instrumented with timeouts and retries to make every call more likely to succeed. Retrying a request is very effective when the failure is transient. However, retries will worsen the problem when the downstream service is unavailable or slow since in that case, most requests will end up being retried multiple times and will still ultimately fail. This scenario where excessive and ineffective retries are applied is called work amplification, and it will cause an already degraded service to degrade further. As an example,[this kind of outage](https://doordash.engineering/2021/06/20/our-june-19th-outage-explained/) happened at an early stage of our transition to microservices: a sudden increase in the latency of our payment service resulted in the retry behavior of the Dasher App and its backend system, which exacerbated the situation.

Figure 2: This diagram illustrates a microservice system experiencing a retry storm. The issue starts with a degraded database that experiences increased latency. This latency spreads through RPC call chains and triggers retry in Service 2, 3, and 5, which puts even greater pressure on DB.

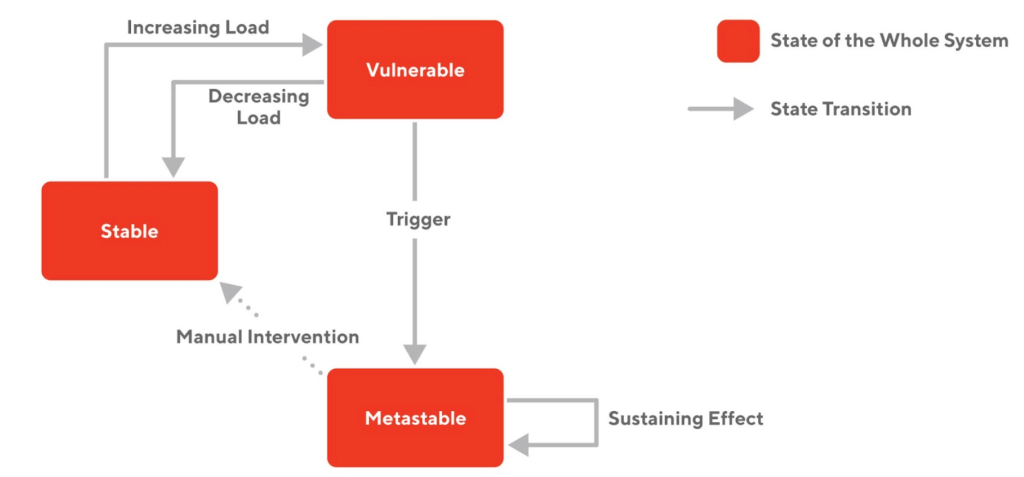
**Death Spiral**

Failures can frequently spread vertically through an RPC call graph across services, but they can also spread horizontally among nodes that belong to the same service. [A death spiral](https://www.couchbase.com/blog/avoiding-death-spirals-distributed-systems/) is a failure that starts with a traffic pattern that causes a node to crash or become very slow, so the load balancer routes new requests to the remaining healthy nodes, which makes them more likely to crash or become overloaded.[This blog post](https://doordash.engineering/2022/08/09/how-to-handle-kubernetes-health-checks/) describes an outage that began with some pods failing the readiness probe and was thus removed from the cluster, and the remaining nodes failed since they were not able to handle the massive loads alone.

Figure 3: This diagram illustrates a service cluster experiencing a death spiral. Node 1 and node 2 become degraded and are replaced by newly launched nodes, which are not ready to take traffic. The load-balancer routes all incoming requests to node 3, making it more likely to degrade too.

**Metastable failures**

[A recent paper](https://www.usenix.org/conference/osdi22/presentation/huang-lexiang) proposes a new framework to study distributed system failures, which is called a “metastable failure.” Many of the outages we experienced belong to this category. This type of failure is characterized by a positive feedback loop within the system that provides a sustaining high load because of work amplification, even after the initial trigger (e.g., bad deployment; a surge of users) is gone. Metastable failure is especially bad because it will not self-recover, and engineers need to step in to stop the positive feedback loop, which increases the time it takes to recover.

Figure 4: This diagram illustrates the lifecycle of metastable failure. The system operates in both stable and vulnerable states as loads fluctuate. When the system is in a vulnerable state, a trigger such as a surge of users can cause it to transition into a metastable state, characterized by a sustained high load resulting from a positive feedback loop within the system. It is then necessary for manual intervention to return the system to a stable state.

1. Can you describe your experience with Java System Architecture and how it has prepared you for this role as a Lead Architect at VISA Inc?

2. How do you approach designing distributed systems and what factors do you consider to ensure they are efficient and reliable?

3. Can you explain how you have used microservices architectures in your previous projects? What were the challenges and how did you address them?

4. As a Lead Architect, how do you ensure that the systems you design are scalable and can handle high volumes of transactions, a requirement that is critical for a company like VISA?

5. Can you provide an example of a complex system you've designed and how you approached the process from start to finish?

6. How do you ensure communication and understanding among different teams working on a project? Can you provide an example of a time when your communication skills helped in the successful completion of a project?

7. What is your approach to system security, especially in the context of financial transactions? Can you provide an example of a security challenge you faced in a previous project and how you addressed it?

8. How do you stay updated with the latest trends and technologies in Java and system architecture? How have you incorporated a recent advancement into your work?

9. Can you describe a time when you had to make a critical decision regarding system architecture and the impact it had on the project?

10. How do you handle failure or setbacks in a project? Can you provide an example of a project that didn't go as planned and how you handled the situation?

Answers –

1. As a Java System Architect, I have spent over 10 years designing and implementing large-scale systems. My experience spans across various industries, including financial services, telecommunications, and e-commerce. This has prepared me for the role at VISA by giving me a deep understanding of how to design robust, scalable, and secure systems.

2. Designing distributed systems is a complex task that requires a deep understanding of certain principles. I always focus on ensuring the system is fault-tolerant, scalable, and consistent. For instance, in my previous role, I designed a distributed system for an e-commerce platform that could handle millions of transactions per day. I used technologies like Apache Kafka and Cassandra to ensure data consistency and fault tolerance.

3. In my previous role at XYZ Corp, we transitioned from a monolithic architecture to a microservices architecture. The main challenge was ensuring the seamless functioning of services during the transition. We addressed this by implementing comprehensive testing and monitoring strategies. The result was a more scalable and maintainable system.

4. Scalability is a critical factor in system design. I ensure scalability by designing systems to be stateless, implementing caching strategies, and using load balancers. For example, in my previous role, I designed a system that could scale to handle sudden spikes in traffic during peak shopping seasons.

5. I once designed a complex payment processing system. The process involved understanding the business requirements, identifying appropriate technologies, designing the system architecture, and overseeing the implementation. The system was designed to be fault-tolerant and scalable, capable of processing millions of transactions per day.

6. I believe in open and frequent communication among teams. In one of my previous projects, I arranged daily stand-ups and weekly meetings to ensure everyone was on the same page. This approach helped us meet our project deadlines and maintain high-quality work.

7. Security is paramount in any system, especially in financial transactions. I always incorporate security measures at every level of system design. In one instance, I implemented encryption and secure APIs to protect sensitive data during transactions.

8. I regularly attend webinars, follow industry experts, and read articles to stay updated with the latest trends. Recently, I incorporated containerization into my work using Docker and Kubernetes, which significantly improved the deployment process and system scalability.

9. In one project, I had to decide between using a SQL and NoSQL database. I chose NoSQL because it provided better scalability and performance for our particular use case. This decision significantly improved the system's performance.

10. Failure is part of the learning process. In one project, we faced significant setbacks due to unforeseen technical issues. However, we learned from the experience, adjusted our approach, and were able to successfully complete the project.

1. Can you explain the concept of CAP theorem and how it impacts the design of distributed systems?

2. How do you handle data consistency in a microservices architecture?

3. Can you discuss your experience with containerization technologies like Docker and Kubernetes?

4. What strategies do you use for error handling and fault tolerance in distributed systems?

5. Can you explain how you would secure a microservice and its communication with other services?

6. How do you approach the design of APIs in a microservices architecture to ensure they are robust and scalable?

7. Can you describe your experience with cloud platforms like AWS, Azure, or Google Cloud? How have you leveraged these platforms in your architecture designs?

8. How do you ensure that the systems you design can handle high volumes of data and traffic?

9. What is your approach to load balancing in a distributed system?

10. How do you monitor and log activities in a distributed system? Can you discuss any tools or practices that you use?

Sure, here are some interview questions for a similar role at Wells Fargo, accompanied by sample answers.

1. **Questions** Can you describe your experience with Java System Architecture and how it has prepared you for this role at Wells Fargo?

**Answer** I have over 15 years of experience working with Java and designing system architectures. I've worked in various sectors, including banking, which has given me a deep understanding of the unique challenges and requirements in this field. My experience has equipped me with the skills to design secure, robust, and scalable systems, which I believe are crucial for a financial institution like Wells Fargo.

2. **Questions** How do you approach designing distributed systems and what factors do you consider to ensure they are efficient and reliable?

**Answer** When designing distributed systems, I consider factors like scalability, fault tolerance, and data consistency. I use tools like Apache Kafka for real-time data processing and Apache Cassandra for managing large amounts of data across many commodity servers. For instance, at my previous job, I designed a distributed system that efficiently handled millions of transactions per day.

3. **Questions** Can you explain how you have used microservices architectures in your previous projects? What were the challenges and how did you address them?

**Answer** At XYZ Corp, we transitioned from a monolithic architecture to a microservices architecture. The main challenge was ensuring minimal downtime during the transition. We addressed this by implementing a comprehensive testing and monitoring strategy. As a result, we had a more scalable and maintainable system.

4. **Questions** As a Lead Architect, how do you ensure that the systems you design are scalable and can handle high volumes of transactions, a requirement that is critical for a company like Wells Fargo?

**Answer** Scalability is a critical factor in system design. I ensure scalability by designing systems to be stateless, implementing caching strategies, and using load balancers. For example, in my previous role, I designed a system that could scale to handle sudden spikes in traffic during peak shopping seasons.

5. **Questions** Can you provide an example of a complex system you've designed and how you approached the process from start to finish?

**Answer** I once designed a complex payment processing system for a large retail company. I started by understanding the business requirements, then identified the appropriate technologies. I designed the system architecture and oversaw the implementation. The system was designed to be fault-tolerant and scalable, capable of processing millions of transactions per day.

6. **Questions** How do you ensure communication and understanding among different teams working on a project?

**Answer** I believe in open and frequent communication among teams. In one of my previous projects, I arranged daily stand-ups and weekly meetings to ensure everyone was on the same page. This approach helped us meet our project deadlines and maintain high-quality work.

7. **Questions** What is your approach to system security, especially in the context of financial transactions?

**Answer** Security is paramount in any system, especially in financial transactions. I always incorporate security measures at every level of system design. In one instance, I implemented encryption and secure APIs to protect sensitive data during transactions.

8. **Questions** How do you stay updated with the latest trends and technologies in Java and system architecture?

**Answer** I regularly attend webinars, follow industry experts, and read articles to stay updated with the latest trends. Recently, I incorporated containerization into my work using Docker and Kubernetes, which significantly improved the deployment process and system scalability.

9. **Questions** Can you describe a time when you had to make a critical decision regarding system architecture and the impact it had on the project?

**Answer** In one project, I had to decide between using a SQL and NoSQL database. I chose NoSQL because it provided better scalability and performance for our particular use case. This decision significantly improved the system's performance.

10. **Questions** How do you handle failure or setbacks in a project?

**Answer** Failure is part of the learning process. In one project, we faced significant setbacks due to unforeseen technical issues. However, we learned from the experience, adjusted our approach, and were able to successfully complete the project.

Sure, here are a few coding problems that are relevant to the role of a Lead Architect, especially one focused on Java, Distributed Systems, and Microservices:

1. **Problem:** Design a basic microservice in Java that performs CRUD operations on a database.

**Suggestion:** Use Spring Boot to create the microservice. Utilize Spring Data JPA to interact with the database. Make sure to handle potential exceptions and return appropriate HTTP status codes.

2. **Problem:** Implement a distributed caching solution.

**Suggestion:** You could use Redis as a distributed cache. Write a service that interacts with Redis to store and retrieve data. Make sure to handle potential issues like cache misses and synchronization.

3. **Problem:** Write a function to handle inter-service communication in a microservices architecture.

**Suggestion:** You could use HTTP/REST or messaging queues for communication between services. Implement a method to send a request to another service and handle the response. Make sure to handle potential errors and implement appropriate retry logic.

4. **Problem:** Implement a load balancing strategy.

**Suggestion:** There are many strategies for load balancing like Round Robin, Least Connections, etc. Choose one and implement it. If you're using a cloud platform, you could also use their load balancing services.

5. **Problem:** Design and implement a secure API.

**Suggestion:** Use Spring Security to secure your API. Implement authentication and authorization mechanisms. Make sure to validate inputs and handle potential security risks like SQL Injection, XSS, etc.

Remember, these problems are quite complex, and a complete solution would be quite large and beyond the scope of this platform. However, these suggestions should give you a starting point.

Certainly, here are a few more coding problems relevant to a Lead Architect role:

1. **Problem:** Design a rate limiter to prevent abuse of your APIs.

**Suggestion:** Consider using a sliding window log or fixed window counter algorithm. You could also use a third-party library or a service like Redis.

2. **Problem:** Implement a circuit breaker pattern to prevent a network or service failure from cascading to other services.

**Suggestion:** You could use libraries like Hystrix or Resilience4j in a Spring Boot application. The idea is to stop all requests to a particular service if the error rate goes beyond a threshold, thus preventing system-wide failures.

3. **Problem:** Write a program that can perform health checks on all your microservices.

**Suggestion:** Implement a service that sends requests to the health endpoints of your microservices. It could log the status or send alerts if any service is down.

4. **Problem:** Implement data sharding in a distributed database to improve performance.

**Suggestion:** Data sharding can be complex and depends on your specific use case. A simple round-robin or hash-based sharding might be a good place to start.

5. **Problem:** Write a program that can serialize and deserialize a binary tree.

**Suggestion:** This is a common problem in system design interviews. The idea is to convert the tree into a format that can be easily stored or transmitted (like a string), and then convert it back into the original tree structure.

Remember, these problems are meant to be high-level and complex, mirroring real-world scenarios a Lead Architect might encounter. The solutions will depend on many factors, including the specific requirements, the existing system architecture, and the available tools and technologies.

**Question** - How to execute 3rd thread after completion of first two threads?

Approach 1 –

In Java, you can use the `join()` method to ensure that a thread does not start executing until the ones it has joined have completed. Here is an example:

public class Main {

public static void main(String[] args) {

Thread t1 = new Thread(() -> {

System.out.println("First thread is completed");

});

Thread t2 = new Thread(() -> {

System.out.println("Second thread is completed");

});

Thread t3 = new Thread(() -> {

System.out.println("Third thread is completed");

});

try {

t1.start();

t1.join();

t2.start();

t2.join();

t3.start();

t3.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

In this code, the `main` method starts `t1` and then calls `t1.join()`, which causes the main thread to wait until `t1` has finished executing. It then does the same for `t2`. Finally, it starts `t3`, ensuring that `t3` does not start until both `t1` and `t2` have finished.

Approach 2 - If you want to execute the third thread only after the completion of the first two threads, you can use Java's `CountDownLatch` class from the `java.util.concurrent` package. This class allows one or more threads to wait until a set of operations being performed in other threads completes.

Here's how you can use a `CountDownLatch`:

import java.util.concurrent.CountDownLatch;

public class Main {

public static void main(String[] args) {

CountDownLatch latch = new CountDownLatch(2); // Initialize it to number of threads, which is 2 in this case

Thread t1 = new Thread(() -> {

System.out.println("First thread is completed");

latch.countDown(); // Decrements the latch count

});

Thread t2 = new Thread(() -> {

System.out.println("Second thread is completed");

latch.countDown(); // Decrements the latch count

});

Thread t3 = new Thread(() -> {

try {

latch.await(); // Causes the current thread to wait until the latch has counted down to zero

System.out.println("Third thread is completed");

} catch (InterruptedException e) {

e.printStackTrace();

}

});

t1.start();

t2.start();

t3.start();

}

}

```

In this code, `latch.countDown()` is called when each of the first two threads finishes, decrementing the count of the `CountDownLatch`. The third thread calls `latch.await()`, causing it to block until the count of the `CountDownLatch` has reached zero, indicating that the first two threads have completed.