Design a Distributed System

CS2IDS: Introduction to Distributed Systems

George Hotten (XXXXXX352)

April 28, 2025

Word Count: 1,568

Contents

1 My Distributed System

A distributed system is network of nodes working together to achieve a common goal whilst appearing as a single system to the user (Naik, 2025*a*). Distributed systems typically have the following properties as described by Naik (2025*c*):

- Resource Sharing—resources can be remotely accessed by multiple nodes.
- Openness—node hardware and software should be independent of proprietorship.
- Concurrency—nodes should be able to deal with multiple tasks at the same time without the output of another.
- Consistency—the ability for the system to keep all nodes in the same state of operation.
- Idempotency—the system should be able to detect duplicate requests and discard them.
- Scalability—nodes should scale inline with demand without affecting end-user performance.

My distributed system will be based on a banking system, aiming to meet and exceed the above properties.

2 Peer-to-Peer Distributed System

A structured peer-to-peer system contains a Centralised Lookup Server which is responsible for containing a list of services provided by each node. A client joining the network will register itself with this server, detailing its services. (Naik, 2025*a*)

In an unstructured system, nodes must broadcast a request for the desired service. The node that has the service will then respond. (Naik, 2025*a*)

I would design a structured peer-to-peer system to ensure maximum efficiency and ensure that all requests are answered. With a central server the required service can be quickly found using the hash-based searching method. An unstructured system which relies on flooding the network with broadcast requests makes it harder to find services. This is because response messages could be lost due to the high traffic in the network, meaning that a node may never find the service it is after.

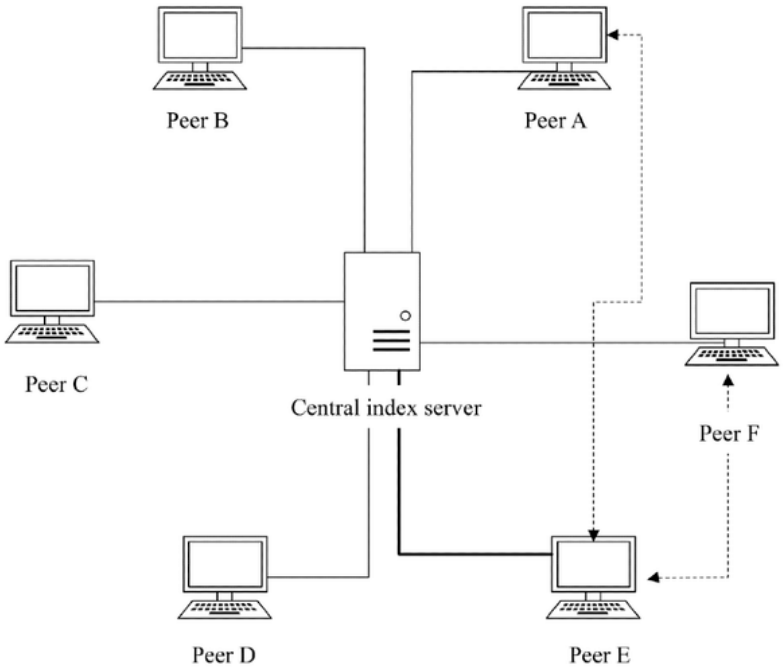

Figure 1: Peer-to-Peer Archiecture (Abdullahi Yari et al., 2021)

3 Client-Server Architecture

As described by Naik (2022), there are numerous advantages and disadvantages to choosing a client-server architecture for the distributed system.

The main reasons for choosing a client-server architecture are:

- Centralisation—data and operations are controlled by a dedicated server, allowing for efficient management and security updates.
- Scalability—new servers and resources can easily be added to cope with increased demand.
- Backup and Recovery—data can be recovered easily as only a single server requires a backup.
- Reduces Data Replication—as data is stored centrally, the amount of data that needs to be replicated is lower.

On the contrary, the main reasons to not choose this architecture are:

- Availability—the server must be available at all times, if not the entire system could be brought to a halt.
- Cost—central servers need powerful machines, which also come with high maintenance requirements.
- Maintenance—complex maintenance requirements as servers must be online at all times to process requests.
- High Load—as only a centralised server is used for all requests, it can quickly become overloaded and fail to respond.

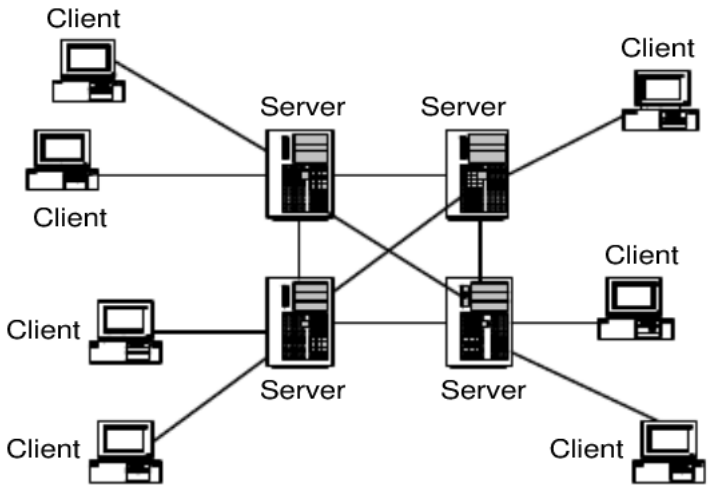

Figure 2: Client-Server Archiecture (Duong and Zhou, 2003)

4 Concurrency or Parallelism

Concurrency refers to a system's ability to handle multiple processes simultaneously without effecting the outcome of the other processes. Parallelism refers to a system's ability to run multiple processes simultaneously. With concurrency, tasks can be in progress at the same time but cannot be executed simultaneously. Parallelism allows for multiple tasks to be executed simultaneously (Naik, 2025*c*).

I would choose to support concurrency over parallelism. The system needs to be able to effectively manage multiple requests whilst ensuring responsiveness and scalability. Distributed systems must commonly handle large numbers of requests, and concurrency allows for a structured and efficient approach to handling these requests. Parallelism is better for systems that benefit from speed but requires tight integration and shared memory. To ensure the highest level of reliability and management, concurrency is the preferred choice.

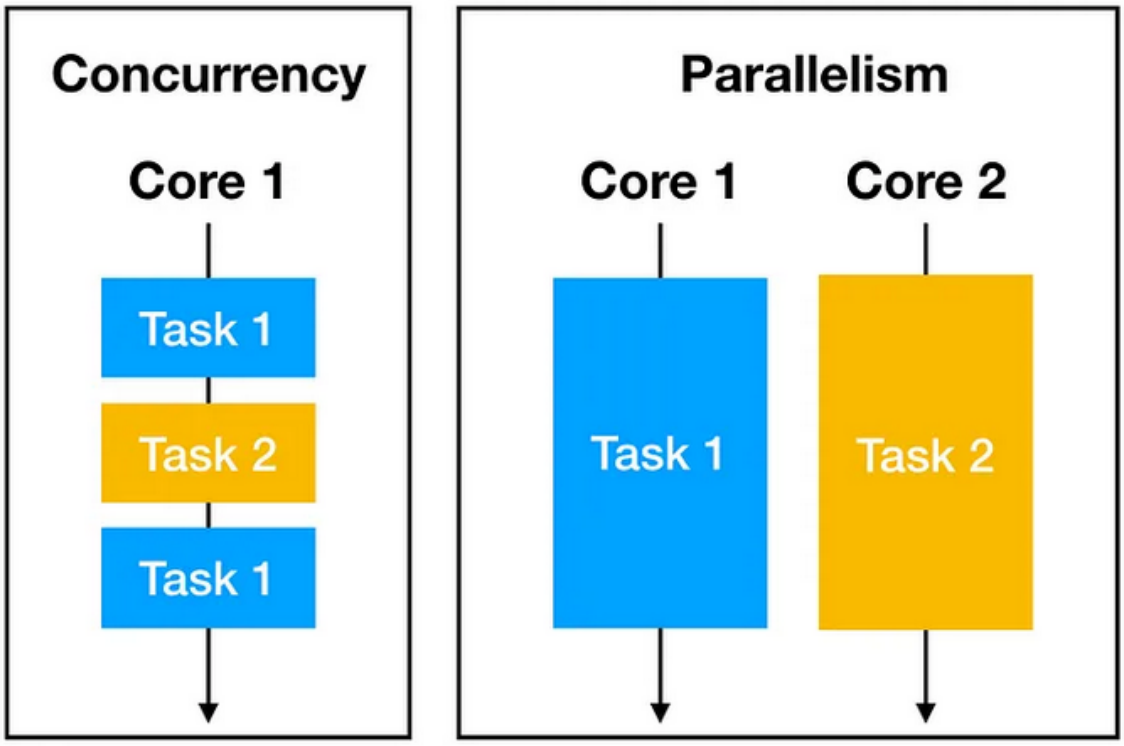

Figure 3: Concurrency vs Parallelism (Singh, 2024)

5 Strong or Weak Consistency

The decision on which level of consistency to use depends on the distributed system's use case. A system with strong consistency ensures that all nodes have identical data. As soon as an update is requested, all nodes coordinate the synchronisation of the data. A downside of strong consistency is that nodes will be unavailable during the synchronisation process, which may cause high latency and a performance penalty (Naik, 2025*c*). This model would be best suited for systems such as banks, finances and inventory management.

A system with weak consistency does not guarantee that data would be up to date on all nodes, however, it does ensure they will eventually be consistent with each other. The time taken to become consistent is not always defined. When a node is updated, it performs the request and sends the update to other nodes, but there is no synchronisation for order of operations. This model offers better performance and faster responses at the expense of accurate data (Naik, 2025*c*). Weak consistency would be best suited for systems such as multiplayer games, social networks and notification systems.

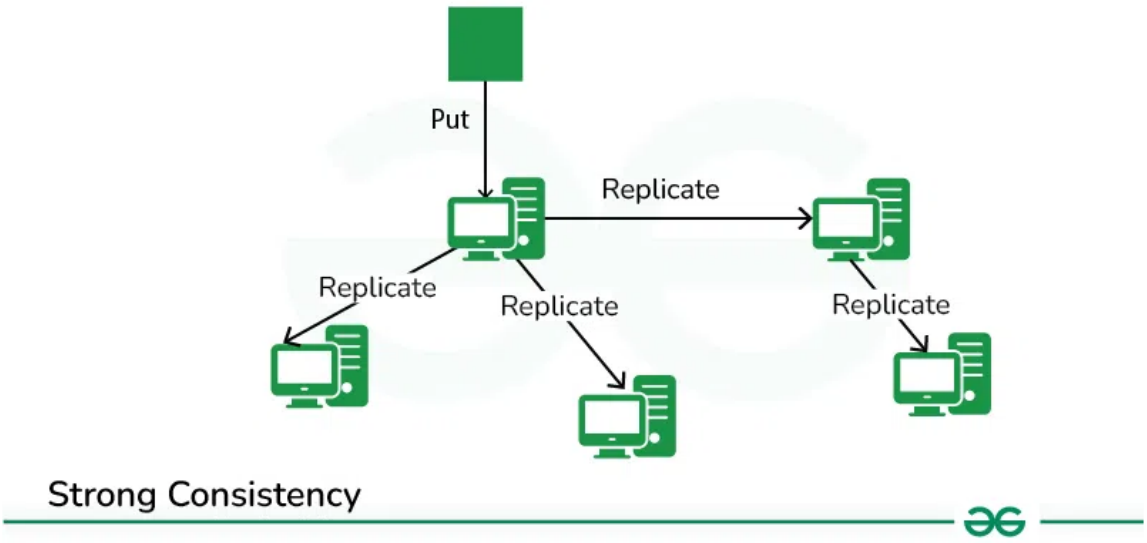

Figure 4: Example of Strong Consistency (GeeksforGeeks, 2024*b*)

6 Idempotency in Distributed Systems

Idempotency is a system used to prevent clients from submitting duplicate identical requests. The system will only action the request once, preventing inconsistency (Naik, 2025*c*).

This is an excellent perk for data-based requests, such as banking. For example, if a user attempted to transfer £500 and the client mistakenly sent two requests, Idempotency will prevent two transactions being actioned. However, for systems such as multiplayer games, this would most likely not affect the user or the data stored within the system.

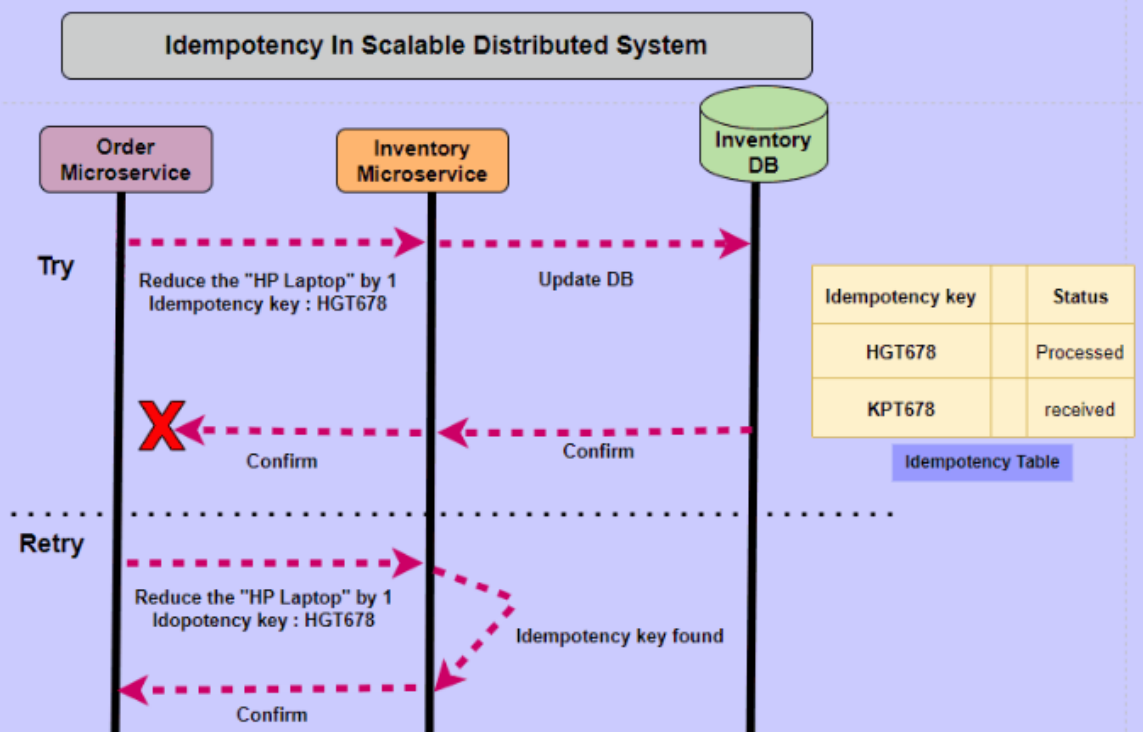

Figure 5: Example flow within an idempotent system (Das, 2023)

7 Horizontal or Vertical Scalability

Scaling is the ability to increase or decrease resources to meet demand from clients. Vertical scaling is the method of upgrading a specific node by adding more resources directly to it. For example, more RAM, storage and processing power. Horizontal scaling is the method of adding more systems to the network, duplicating the specified node (Naik, 2025*c*).

For larger applications, horizontal would be considered most appropriate. This is because adding more nodes allows for more connections to the application. It also improves performance as individual servers are less likely to become overloaded. For smaller applications, vertical scaling would be most appropriate. This is because implementation is typically easier to implement and maintain as it is only a single server (GeeksforGeeks, 2024*a*).

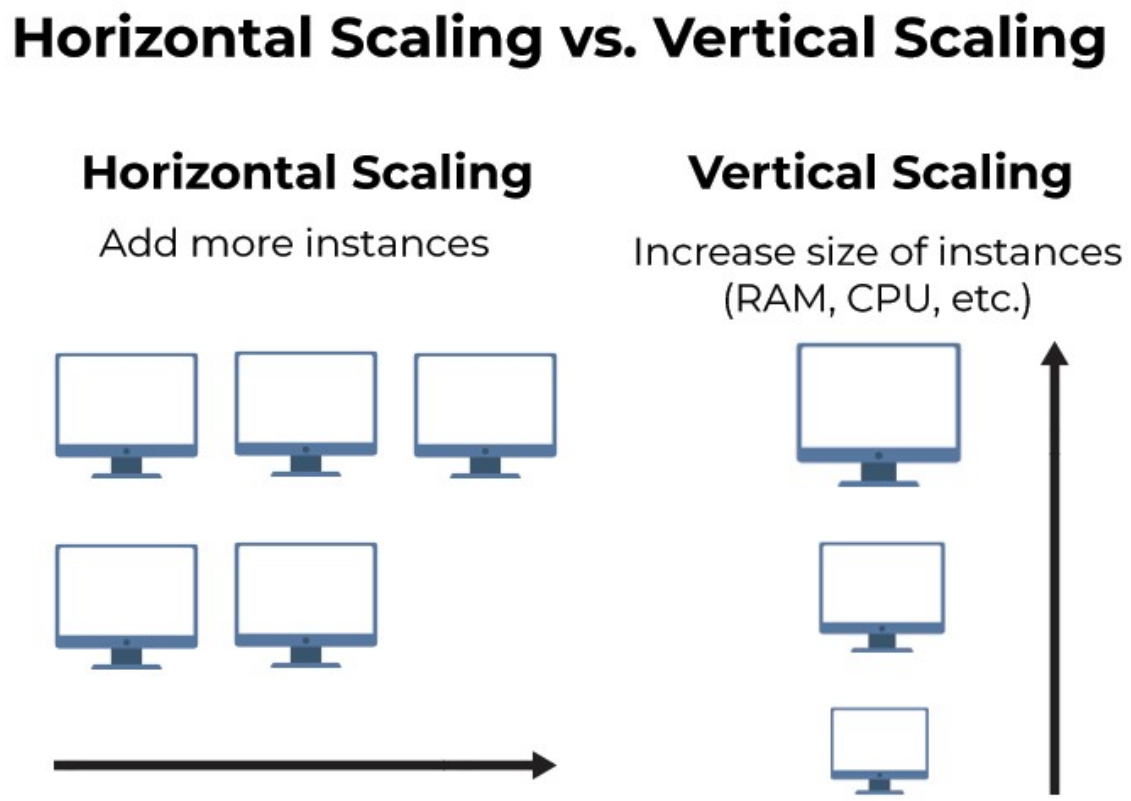

Figure 6: Example flow within an idempotent system (Ashtari, 2021)

8 Availability or Reliability

Availability is the measurement of the time a system is accessible compared to the amount of time it should be accessible. It is ideal for a system to have 100% availability, however, this is typically not practical due to costs and higher-latency. Reliability is the measurement of a system's ability to complete its tasks within a specified time frame without failure. It is normally measured as the average time between faults (Naik, 2025*c*).

A system can still be deemed available even if the applications within it are not functioning properly. Because of this, it would be most beneficial to provide reliability over availability. End users are typically more interested in whether the system can complete the tasks required of it as opposed to the system being online and unable to function. Therefore, reliability is a better measure to provide.

9 Transparency

Using the example of a banking application, I would provide the following three transparencies as described by Naik (2025*c*):

- Location—access nodes without knowing their physical location.
- Concurrency—allow several processes to operate concurrently without interfering with each other.
- Replication—permit accessing replica resources without knowledge of it.

Location transparency allows users to connect to the closest node to their location, providing faster response timers and a better experience. The user would not be aware of this, and to them, it would seem as if they are accessing the same resource.

Concurrency would be crucial for a banking application to allow users globally to manage and move their money simultaneously. Without it, users would potentially have to wait an extended period of time for their transactions to be processed.

Replication transparency is also important for the sustained uptime of the application. Accessing replica resources combats against downtime from other nodes by ensuring the user is always able to access their data. This is especially important in a banking scenario as users may need instant access to their money and make transactions at all times.

10 Crash Fault or Byzantine Fault

A crash fault is a common fault which can fall under the following three categories (Naik, 2025*c*):

- Crash-Stop—a node stops working without warning and cannot recover.
- Crash-Recovery—a node stops working temporarily and later recovers.
- Networking Partition—a node is separated from the network due to network errors or topology changes.

A byzantine fault occurs when a node displays malicious behaviour and broadcasts conflicting or contradictory information to other nodes (Naik, 2025*c*).

For our banking application, byzantine fault tolerance would be most appropriate as it protects against malicious nodes from manipulating data within the system.

10.1 Preferred Consensus Algorithm

To ensure the highest level of security and integrity, I would choose the PBFT algorithm. PBFT can reach a consensus even when nodes maliciously or fail to respond. The algorithm uses a three-stage consensus process, which makes it more communication heavy at the cost of being more secure. It can function on the assumption that there are no more than a third of malicious nodes. This means the more nodes, the safer the system (Naik, 2025*b*).

For the banking system, the slight increase in latency is a small trade-off for higher security against malicious nodes posing a risk to the system's integrity.

References

Abdullahi Yari, I., Dehling, T., Kluge, F., Geck, J., Sunyaev, A. and Eskofier, B. (2021), 'Security engineering of patient-centered health care information systems in peer-to-peer environments: Systematic review', *Journal of Medical Internet Research* **vol. 23**.

Ashtari, H. (2021), 'Horizontal vs. vertical cloud scaling: Key differences and similarities'. Accessed: 28 April 2025.
URL: *https://www.spiceworks.com/tech/cloud/articles/horizontal-vs-vertical-cloud-scaling/*

Das, D. (2023), 'Idempotency in scalable distributed architectures — example'. Accessed: 28 April 2025.
URL: *https://medium.com/@debasisdasnospdii/idempotency-in-scalable-distributed-architectures-example-e23a12c70048*

Duong, T. and Zhou, S. (2003), A dynamic load sharing algorithm for massively multiplayer online games.

GeeksforGeeks (2024*a*), 'Horizontal and vertical scaling | system design'. Accessed: 23 April 2025.
URL: *https://www.geeksforgeeks.org/system-design-horizontal-and-vertical-scaling/*

GeeksforGeeks (2024*b*), 'Strong consistency in system design'. Accessed: 28 April 2025.
URL: *https://www.geeksforgeeks.org/strong-consistency-in-system-design/*

Naik, N. (2022), 'Client-server architectural model: 3-layer 3-tier architectures, advantages and disadvantages'. 27 Febuary 2022. Accessed: 16 April 2025.
URL: *https://www.youtube.com/watch?v=OTCYvBPHcdo*

Naik, N. (2025*a*), 'Distributed systems and its architectural models'. Lecture Notes on CS2IDA, Aston University.

Naik, N. (2025*b*), 'Pbft (practical byzantine fault tolerance): A distributed consensus algorithm'. Lecture Notes on CS2IDA, Aston University.

Naik, N. (2025*c*), 'Properties of distributed systems'. Lecture Notes on CS2IDA, Aston University.

Singh, S. (2024), 'Understanding concurrency and parallelism: A beginner's guide'. Accessed: 28 April 2025.
URL: *https://medium.com/@suyashsingh.stem/understanding-concurrency-and-parallelism-a-beginners-guide-4cb3f22b90cf*