Two-Phase Commit

*It is widely used pattern for distributed transaction. In this, there is a co-ordinator component which control all the transactions and has a logic to manage transactions.*

*The other components are nodes i.e. microservices which are participating in transaction.*

*As the name suggests, 2-pc works in 2 phases.*

1. *Prepare phase – The co-ordinator asks all participants, whether they are ready to commit.*
2. *Commit phase – If all participants respond as “yes”. Co-ordinators ask them to commit. If any of the participant respond to “No”, co-ordinator ask them to roll back.*

***Problems in 2-PC –***

* *The overall responsibilities are dependent on co-ordinator, hence there will be problem if co-ordinator fail.*
* *All nodes need to wait for all other nodes to respond, even the slowest one.*
* *It cannot support NoSQL db*
* *Data Inconsistency: In the event of a service failure during the commit phase, it can be challenging to ensure that all services roll back to their previous state, leading to potential data inconsistency.*
* *Complex Error Handling: Handling errors and implementing rollback mechanisms in microservices can be complex due to the distributed nature of the services.*
* *Resource Locking: During the prepare phase, resources may be locked, which can reduce system performance and responsiveness.*

***When to use -***

*The 2PC protocol is useful in situations where all participants of the distributed transaction must commit or roll back the transaction together. It ensures atomicity and consistency of the transaction.*

*2PC guarantees atomicity and read/write isolation. It ensures the data is consistent and all the data is synchronized all the time. Despite its benefits, it’s not recommended in many situations due to its synchronous nature. 2PC protocol is blocking protocol and has a single point of failure too. If transaction coordinator fails, the DB will be in an inconsistent state, also due to multiple transactions, latency is high, and the complete system will be slow as the slowest transaction is the whole process.*

*Problems With 2PC*

*Although 2PC is useful to implement a distributed transaction, it has the following shortcomings:*

*The onus of the transaction is on the coordinator node, and it can become the single point of failure.*

*All other services need to wait until the slowest service finishes its confirmation. So, the overall performance of the transaction is bound by the slowest service.*

*The two-phase commit protocol is slow by design due to the chattiness and dependency on the coordinator. So, it can lead to scalability and performance issues in a microservice-based architecture involving multiple services.*

*Two-phase commit protocol is not supported in NoSQL databases. Therefore, in a microservice architecture where one or more services use NoSQL databases, we can’t apply a two-phase commit.*

*When to use 2pc and when to use SAGA*

*E-commerce Platforms:*

*Platforms like Amazon and eBay have complex workflows. When a user places an order, it involves inventory checks, payment processing, and shipping. Using 2PC can lead to performance bottlenecks, especially during sales. SAGA, with its decentralized approach, offers a more scalable solution.*

*Financial Systems:*

*Banks and financial institutions often rely on 2PC for transactions, given the need for immediate consistency. However, with the rise of microservices in fintech, many are exploring SAGA as an alternative.*

*Social Media Platforms:*

*Platforms like Facebook and Twitter, which deal with massive amounts of data and require high scalability, often employ variations of the SAGA pattern.*