Two Phase Commit

Definition: A two-phase commit is a standardized protocol that ensures that a database commit is implementing in the situation where a commit operation must be broken into two separate parts

Problem: In database management, saving data changes is known as a commit and undoing changes is known as a rollback. Both can be achieved easily using transaction logging when a single server is involved, but when the data is spread across geographically-diverse servers in distributed computing (i.e., each server being an independent entity with separate log records), the process can become more tricky.

Solution

A special object, known as a **coordinator**, is required in a distributed transaction. As its name implies, the coordinator arranges activities and synchronization between distributed servers. The two-phase commit is implemented as follows:

Phase 1 - Each server that needs to commit data writes its data records to the log. If a server is unsuccessful, it responds with a failure message. If successful, the server replies with an OK message.

Phase 2 - This phase begins after all participants respond OK. Then, the coordinator sends a signal to each server with commit instructions. After committing, each writes the commit as part of its log record for reference and sends the coordinator a message that its commit has been successfully implemented. If a server fails, the coordinator sends instructions to all servers to roll back the transaction. After the servers roll back, each sends feedback that this has been completed.

SAGA

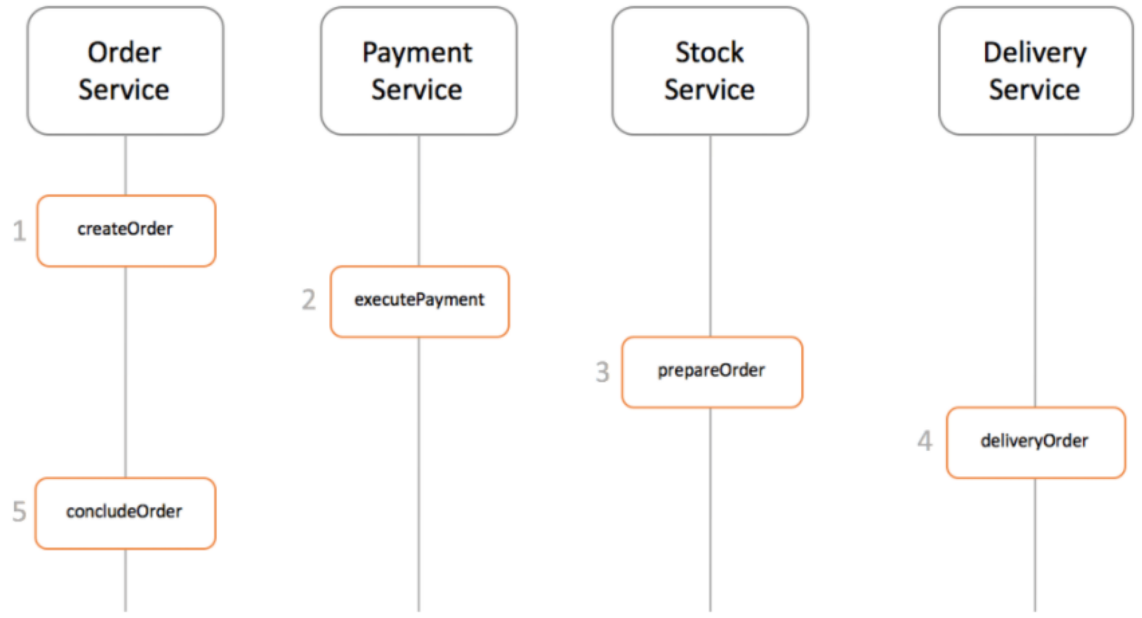
One of the most powerful types of transactions is called a Two-Phase Commit, which is in summary when the commit of a first transactions depends on the completion of a second. It is useful especially when you have to update multiples entities at the same time, like confirming an order and updating the stock at once.

However, when you are performing Microservices orchestration, for example, things get more complicated. Each service is a system apart with its own database, and you no longer can leverage the simplicity of local two-phase-commits to maintain the consistency of your whole system.

When you lose this ability, RDBMS becomes quite a bad choice for storage, as you could accomplish the same “single entity atomic transaction” but dozens of times faster by just using a

NoSQL database like Couchbase. That is why the majority of companies working with microservices are also using NoSQL.

A saga pattern is a sequence of local transactions where each transaction updates data within a single service. The first transaction in a saga is initiated by an external request corresponding to the system operation, and then each subsequent step is triggered by the completion of the previous one.



Summary

The Saga architecture pattern provides transaction management using a sequence of local transactions. A local transaction is the unit of work performed by a saga participant. Every operation that is part of the Saga can be rolled back by a compensating transaction. Further, the Saga pattern guarantees that either all operations are complete successfully or the corresponding compensation transactions are run to undo the work previously completed.