Distributed Transactions

A short summary by Platon Karageorgis

The following summary focuses, on a subset of the chapter Distributed Transactions, of the book Distributed Systems: Concepts and Design by George Coulouris. A transaction is a core process in an OLTP database system and it is broadly used. The transactions that demand the cooperation of more than one server in order to take place, are called distributed transactions and are the topic of this text.

To begin with, there are two different types of a distributed transaction, the flat transaction and the nested one. A flat transaction is relatively simple, the client contacts more than one servers and the transaction sequentially establishes a connection with each one of them. On the other hand, a nested transaction is a transaction that is able to generate or call sub-transactions, and these sub-transactions can equally do the same until all the servers are reached. The nested technique can be quicker, because according to the author it can manage sub-transactions working in parallel, avoiding the waiting delays that occur in the sequential flat method.

Moreover, a crucial protagonist of a distributed transaction is the coordinator. A coordinator is responsible for the communication of the different servers involved in the transaction. The first task a coordinator faces, is the initialization of the transaction when it is requested by the client. Subsequently, the coordinator sends back to the client an ID usually containing the IP address of the server and a unique number that represent the transaction that has just started. When all is said and done, the coordinator creates a list containing every participant of the transaction and every participant stores the relation with the coordinator. A participant, is a server object that is responsible for handling the part of the transaction that has been assigned to the corresponding server. Ultimately, the coordinator and the participants have to work together, in order to either commit the transaction or to abort it.

In addition, a significant issue in transaction processes concerns the atomic commit protocols. The simplest atomic commit protocol is not discussed in this text, since it is incompetent of completing commits in every scenario. If a communication is lost for example, a transaction that abides by this protocol will never commit or abort and therefore will create a substantial issue. The two phase commit protocol on the other hand, can solve this problem and is the method that the author suggests. The first phase of the protocol is a vote between every participant, while in the second phase the counting takes place, including the coordinator's vote. The same way the coordinator calls the participants to either commit or abort, the participants use specific operations like canCommit that clarify if they can commit. In case everyone agrees on either

commit or abort, the transaction completes with the according action. Contrastingly, if the vote is not unanimous, then the transaction is aborted.

There are many different cases that can be tricky to deal with in a distributed transaction, and the two phase protocol seems to be able to deliver in every one of them. Firstly, if during the process a server crashes or there is a loss of communication, the server is being replicated by a new process that has direct access to the server's permanent storage and therefore solves the issue. Furthermore, there are some cases where a participant refuses to reply, until the outcome of the vote is known, and performs an operation called getDecision, forcing the coordinator to provide the participant with the result. This case is also handled by the protocol, but it can cause significant delays, especially in cases where there are more than one participants having this demand or when the coordinator crashes and needs to be replaced. Additionally, the case where a participant never receives the canCommit question, could be problematic. The protocol has a plan here as well, as there is a timestamp assigned to each participant, allowing them to abort the transaction if the waiting time exceeds the limit. It should be noted that the opposite case is dealt with as well, since a coordinator has the ability to abort if there is a substantial delay from one of the participants. In case this happens, the transaction is aborted and overtime replies are not being considered.

Last but not least, the performance of the protocol is discussed. The author mentions that the cost of the operation is 3N, where N is the number of the participants. This cost is an outcome of three total rounds of messages. Moreover, he outlines that regardless of the issues that can come up during a transaction, the protocol always comes through with the only side effect being that there could be a long completion time, with respect to the failures that appeared. The author finishes, by informing the reader that an enhanced protocol called three phase commit protocol exists, which faces the delaying issues but requires a higher budget to be implemented.

Which questions aren't answered by this text?

The text is written extremely well. It has a great flow, it includes the according illustrations that help the reader comprehend the topic better and sometimes repeats terms, like the getDecision method in order to make sure it is clear. I feel I understood fully the topic and the writer answered all my questions.

What has changed since this was written?

The text is from a book written in 2005, which is too long ago for the modern technological era. Although, the protocol discussed cannot be debunked since it is a method that had been working efficiently at the time. The one thing that could change

is that the method is no longer applicable, but from a quick search online I see that it is still functional and implemented in modern distributing systems.