Lectures on Agreement Part I

Uri Abraham

June 2, 2014

Abstract

Part one: why do we need agreement protocols?

1 Introduction

I like the following passage that Brian Masao Oki wrote in the Introduction to his thesis (M.I.T. 1988), because usually you do not see any personal touch in technical theses.

> "High availability is essential to many computer-based services. For example, imagine trying to withdraw money from your savings account at your local savings and loan bank. As you present your passbook to the teller, he tells you, red-faced with embarrassment, that he cannot process your request because the system is down and will be for some time. I actually found myself in the teller's position some years ago, apologizing profusely to customers. The entire customer database resided on a single mainframe computer located somewhere in Beverly Hills, California. All branches in Los Angeles County were connected directly to this computer, and its problems affected everybody. In a fit of rage, some customers later closed out their accounts and took their money elsewhere, much to the bank's chagrin".

Replication is the obvious direction in which increased availability and dependability and thus a remedy to Brian's embarrassment can be found. This is a familiar idea both in everyday life and in engineering. For example, I have two pairs of eye glasses so that I can continue with my usual activities in case one breaks or get lost, and every car has an additional hand breaks in case the braking system fails.

So a natural construction one can suggest to the savings and loan bank is to have several centers located in different places, say for example in three places, so that these centers are completely coordinated and all keep copies of the same information. A local branch can contact any of these centers for making transactions, and if a communication line from the branch to the nearest center fails, then the branch can try another connection without any trouble to the customers. For this scheme to work well it is necessary that any transaction that one of the costumers makes with one of the branches is immediately and reliably copied at all other branches. All branches must keep the same information and allow changes to this information that are exactly replicated at each branch. Unfortunately such a completely reliable system relies on an assumption that experience has shown to be impossible to implement. Even the best communication centers that one can dream of may one day collapse in some unpredictable way. No mater how well they are build, it is impossible to build computer centers that are failure free. Electricity line may fail leaving a center without power (and the alternative local supply break), or a scheduled maintenance somehow went wrong etc. etc.

The aim of our informal discussion is to explain that reaching consensus, that is agreement between different branches on the transaction to execute, is an important and nontrivial problem. Our reader is encouraged to think and suggest solutions to this problem in order to be ready to invest in understanding a complex algorithm such as the Paxos agreement algorithm that we are going to present next. We must first define exactly what is it that we expect from an agreement protocol to achieve. There are two technical notions that we shall use: consensus and agreement. We shall see that achieving consensus is too good to be possible (this is the famous FLP impossibility result) and agreement is the name that we give to the weaker agreement property that Paxos algorithm for example achieves. ("Consensus" is an accepted term in the literature, but "agreement" is not. Still I find it useful to have such a notion to use in class.)

To formulate the consensus problem we assume serial processes $P_0, \dots, P_{N-1}$, and a set of values V. Each process P_i starts its algorithm with some input value v_i and there are three possible types of resulting executions:

1. P_i may reach a "decision" stage with some output value $w_i \in V$. The value of w_i will not change after a decision stage is reached.

2. P_i may fail-stop before reaching a decision stage and never again make any more steps.

3. P_i may run endlessly its algorithm without ever reaching its decision stage.

An algorithm solves the "consensus problem" if it satisfies the following properties.

1. No two processes decide differently (**unanimity of agreement**). That is, whenever the algorithm runs, starting with arbitrary input values, if P_i and P_j both reaches their decision stages then $w_i = w_j$.

2. The output value is the input value of some process (**validity**). That is, in any run, if P_i reaches agreement then $w_i = v_j$ for some j (possibly the index of a failed process).

3. Each correct process eventually decides (**termination**). That is, no process can run endlessly without reaching consensus: it either fail-stop or reaches it decision final stage. (A process is correct if it does not fail.)

An algorithm that ensures the first two conditions is said to be an *agreement algorithm*[1] and an algorithm that ensures all three conditions is said to be a consensus algorithm. The FLP impossibility result is that there is no consensus algorithm. That is, agreement algorithm fails to satisfy termination. It turns out that there are agreement algorithms that although do not (and cannot) satisfy termination have nonetheless immense practical importance. The Paxos algorithm which we shall describe later is the best known such agreement algorithm.

On the difference between the multi-stage and single-stage (single shot) agreement problem. In applications, we need to solve the multi-stage agreement problem. For example suppose that the processes $P_0, \ldots, P_{N-1}$ are the bank branches discussed above that have to work in unison. That is, they must start with the same state and execute the same

[1]This is not an established terminology.

transactions. The P_i processes must decide for every $n \in \omega$ (ω is the set of natural number) which value from V is the common value of step n. That is, the processes have to agree on a function γ from ω to V. The idea is that if $\gamma(n) = v$, then all active processes agree on value v for their common step n. Now if we concentrate on that single stage n, then we get a single-stage agreement. Because of its complexity, usually textbooks and courses concentrate on the single shot consensus problem as we do here. It must be understood however that a multi-stage run of the algorithm is not a serial application of the single-stage part. That is, it is conceivable that while some processes are active trying to get the agreed value for stage 4 for example, some other processes have already moved to higher stages and perhaps even agreed on later stages.