**Lecture 13 - Fault tolerance**

**Failure types**

* Crash failure — the process executes normally until some point, then it stops completely (it doesn't send any message afterwards).
* Byzantine failure — any behavior is possible on the part of the failing process.
* Communication failure — some messages are lost (however, they are not altered, duplicated, or created). Furthermore, it is usually assumed that there is no point in time from which all the messages are lost; this is often phrased as *if infinitely many messages are sent, then infinitely many messages are delivered*.

**Synchronous vs asynchronous system**

* Asynchronous — there is no upper limit on the processing time or on the message transit time. However, note that any computation and any message delivery takes a finite time.
* Synchronous — there is a known upper limit on the processing times and on the message transit times. Consequently, it is possible to arrange the computations in *synchronous rounds*: in each round, each process performs some computations based on the inputs and on the messages received from peers during the previous rounds, and then it sends messages to peer processes. If, during a round, a process is supposed (according to the protocol) to receive a message from some other process, but it doesn't, it reliably detects this as a missing message (and deduce that either the message was lost or the source process has failed).

**Consensus and other closely related problems**

Background: consider a system consisting of *n* functionally identical process computers, each having a complete set of sensors. Each computer evaluates the readings from the sensors and decides what to do to controll the process. If they command the same action to tha actuators, the action is performed; otherwise, the action is based on the command from the majority of the process computers and an alarm is raised. To do this, however, all the computers must have all the input data from the sensors and must agree on the content of that data. This leads to the following *consensus problem*.

**Consensus problem**

We assume we have *n* processes, and each one has an input value *xi*. Those values should be the same but, occasionally, they can differ. All processes must agree on a value to be used as input for further processing. If all inputs are equal, the agreed value must be equal to this input; if the inputs are not all equal, the agreed value can be the input of any process, but it must be the same for all processes.

The requirements are:

* Each correct process must decide an output value *yi* in a finite time;
* All correct processes must decide the same output value;
* The decided value for the correct processes must be the input of some process; consequently, if all processes have the same input value, then all correct processes must decide that value as output.

**Interactive consensus**

Like the consensus, but the processes are supposed to find a vector containing the inputs of all processes. The requirements are:

* Each correct process must decide an output vector in a finite time;
* All correct processes must decide the same output vector;
* The decided value for the position of a correct processes must be the input of that process.

**General's problem (reliable broadcast)**

A source process (the general) must send a message to other processes (its lieutenants). However, the source process may be faulty; even in that case, all the (correct) recipients must agree on the transmitted message. Essentially, this is the interactive consensus problem, but we need only the component corresponding to one process (the source).

A diagram of a line with arrows

Description automatically generated with medium confidence

**Asynchronous case**

The problem is unsolvable, even if there is at most one process that can fail and the only failure is crash-failure.

Intuitively, the problem is that we cannot distinguish between a failed process and a slow process. However, the formal proof relies on assuming the existence of a solution, examining the decision tree and finding a contradiction.

**Synchronous case, byzantine failures**

The solution for interactive agreement is the following:

For at most one failed process, we need *n≥4*. In the first round, each process sends its input to everybody else. In the second round, each process sends everybody else the values received in the first round. In the third round, each process *p* computes, for each other process *q*, the corresponding value in the output vector as follows: it looks at the reported values for *q*'s input (the one received directly in round 1, and the two reported by the others in round 2). If at least 2 of them are equal, that value is set in the output vector; otherwise, a default (*null*) value is set.

The algorithm generalizes to any number *n* of processes, as long as the number *t* of faulty processes is strictly less than one third (*3t < n*). This limit is proven necessary.

A diagram of a line with lines and arrows

Description automatically generated with medium confidence

**Two generals problem**

Two processes must reach an agreement (like in the consensus problem). The two processes are supposed correct, but the communication may lose messages. However, it is supposed that the communication does not become permanently faulty. This means that, at any point, if one process sends enough messages, one of them will be delivered (however, the source cannot know how many messages to send).

It is proven that no solution exists. The idea is to suppose that a protocol exists, to remove all unnecessary messages, and then to notice that for the last message, while it is necessary for the receiver in order to decide, the sender cannot know if it was delivered or not.