Large-Scale Distributed Systems: Exercise Session 1

Joeri Hermans
joeri.hermans@doct.ulg.ac.be

October 5, 2017

1 Introduction

The core of any distributed system is a set of distributed algorithms that ensures the behaviour and the correctness of the global state. In contrast to single monolithic systems, distributed systems have usually the advantage that other processes are conducting their computations *independently* (asynchronously) from other *entities*, i.e., a different machine or node. However, in order to ensure the correctness of the global goal, information exchange is required. The difficulty in this setting arises from the fact that a physical communication medium is intrinsically lossy due to the fact that other (physical) processes influence the communication signal. Nevertheless, if we construct our infrastructure in such a way that we can minimize the loss, we can assume that with a certain probability some form of *useful* communication might occur between two or more entities, i.e., sometimes, the medium allows for information to pass without being destroyed. In the lecture we saw that using this principle, we can construct relatively complex abstractions, and more importantly, build on top of previous abstractions (layering) to ensure that information always arrives once it is send. Furthermore, communication alone does not contribute to the fact that a distributed system remains stable whenever a entity crashes, or refuses to collaborate. To prevent this from happening, we need to ensure that our distributed system is able to deal with said situations. This is typically done by implementing recovery abstractions.

At this point, we established a method to ensure *reliable* communication. As a result, we are able to coordinate the entities to ensure the correct behaviour of the distributed system. Sadly, reliable *point-to-point* communication is probably not sufficient in a distributed system since more high level features are probably required. For example, let us assume processes p, and q, where p needs to send message m to q. Of course, p could send the message directly to q using our reliable point-to-point abstraction. However, this requires a *direct* connection from p to q. It is trivial to see that point-to-point protocol assumes some abstraction, i.e., a direct connection between p and q is available. As we will see in later courses, distributed systems, some of which you may or may not use (e.g., BitTorrent), usually place additional layers of abstraction on top of existing infrastructure (albeit with the same functionality) to achieve a specific goal, while at the same time, remove the abstraction when something efficient needs to be done.

In this exercise session, we will study and apply some of these abstractions in order to captivate their importance, and to what end they can be utilized.

2 Exercises

Exercise 1

Define liveness and safety formally. Additionally, specify an example for both liveness and safety (the examples in the lecture and the book do not count).

Exercise 2

In class we saw that liveness and safety can be employed to prove the correctness of a distributed system. Let us assume a traditional producer-consumer setting. Which of the following are liveness properties and why?

1. A consumer will eventually consume an element from the queue.
2. Two consumers cannot consume the same head item concurrently.
3. A producer has to stop producing if the queue is full.
4. After a producer stops, it will start again if the queue is not full.

Exercise 3

What is the problem vector clocks solved compared to logical clocks? How do they operate differently? In what type of timing abstraction are vector clocks employed? What is the main limitation of vector clocks?

Exercise 4

Let us assume we have a cluster where users can submit optimization procedures to a processing queue, e.g., for training neural networks. We know that every optimization process will be allocated to a single node. What kind of mechanisms do we need to construct in order to detect the crash of the process (or node), and insure that the neural network does not have to be trained from the start? Make a diagram, and write the procedure down in pseudo-code. Furthermore, identify the abstractions.

Exercise 5

We would like to process a very large dataset $\mathcal{D}$ (e.g., > 4 PB) of meteorological data to obtain the average temperature of a given location in a given time-frame. We define our computation in a acyclic graph structure $\mathcal{G}$ (drawn on the blackboard). This computation graph (this topic is explained in detail in later lectures) will now be submitted to the cluster, where all nodes will participate in the computation. However, a node fails during the computation. Can you come up with several approaches to recover from such a failure? What are the positive and negative aspects of every approach?

Exercise 6

Can you come up with a leader-election abstraction which is resilient against Byzantine adversaries, given the fact that the initial leader is correct? Proof informally (tip: Gibbard–Satterthwaite theorem).