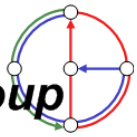
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group



Principles of

Distributed Computing

Roger Wattenhofer
wattenhofer@tik.ee.ethz.ch

Head Assistant: Christoph Lenzen (lenzen@tik.ee.ethz.ch)
Assistant: Thomas Locher (locher@tik.ee.ethz.ch)

Spring 2009

Introduction

What is Distributed Computing?

In the last two decades, we have experienced an unprecedented growth in the area of distributed systems and networks; distributed computing now encompasses many of the activities occurring in today's computer and communications world. Indeed, distributed computing may appear in quite diverse application areas, in networking as well as in distributed systems. Typical "old school" examples are parallel computers or the Internet. More recent application examples of distributed computing include peer-to-peer systems, sensor networks, or multi-core architectures.

These applications have in common that many processors or entities (often called nodes) are active in the system at any moment. The nodes have certain degrees of freedom: they have their own hardware, their own code, and sometimes their own independent task. Nevertheless the nodes are sharing common resources and information. In other words, coordination is necessary.

Despite these commonalities, a peer-to-peer system, for example, is quite different from a multi-core architecture. And indeed, the area of distributed computing needs to deal with many different models and parameters. There are systems where the nodes operate synchronously, and systems where they operate asynchronously. There are simple homogeneous systems, and heterogeneous systems where different types of nodes need to interact. There are different ways of communication, nodes may communicate by exchanging messages, or through shared memory. Sometimes the communication infrastructure is tailor-made for an application, sometimes one has to work with an infrastructure that has grown naturally. The nodes in a system sometimes work together to solve a global task, occasionally the nodes are autonomous agents that all have their own task and compete for common resources. Sometimes the nodes can be assumed to work correctly, at times they may exhibit failures. In contrast to a single-node system distributed systems may still function correctly despite failures, as other nodes can take over the work of the failed nodes. There are different kinds of failures, nodes may just crash, or they might exhibit an erroneous behavior, maybe even to a degree where it cannot be distinguished from malicious (also known as Byzantine) behavior. Maybe the nodes do follow the rules, however they tweak the parameters to get most out of the system; in other words, the nodes are selfish. As you see, there are many models (and even more combinations of models) that we could study in class. We will not discuss any of these models now, but simply define them when we use them. Towards the end of the course a general picture should emerge. Hopefully!

This course introduces the principles of distributed computing, highlighting common themes and techniques. We study some of the fundamental issues underlying the design of distributed systems:

- Communication: Communication does not come for free; often communication cost dominates the cost of local processing or storage. Sometimes we even assume that everything but communication is free.
- Coordination: How can you coordinate a distributed system that it performs some task efficiently?
- Fault-tolerance: As mentioned above, one major advantage of a distributed system is that even in the presence of failures the system as a whole may survive.
- Locality: Networks keep growing. Luckily global information is not always needed to solve a task, often it is sufficient if nodes talk to their neighbors. Whether a local solution is possible is a fundamental question in distributed computing that we will address in this course.
- Parallelism: How fast can you solve a task if you throw more hardware at the problem? How much parallelism is possible for a given problem?
- Symmetry breaking: Sometimes some nodes need to be selected to orchestrate the others. This is done by a technique called symmetry breaking.
- Synchronization: How can you implement a synchronous algorithm in an asynchronous system?
- Uncertainty: If we need to agree on a single term describing this course, it probably is "uncertainty". As the whole system is distributed no node knows what other nodes are doing at this exact moment.

Finally there are also a few areas that we will not cover in this course, mostly because these topics have become so important that they deserve and have their own courses. One example is distributed programming and software engineering, another example is security or cryptography.

In summary, in this class we explore essential algorithmic ideas and lower bound techniques, basically the "pearls" of distributed computing and network algorithms. We will cover a fresh topic every week.

Have fun!