Introduction

What is Distributed Computing?

In the last few 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 appears in quite diverse application areas: Typical "old school" examples are parallel computers, or the Internet. More recent application examples of distributed systems include peer-to-peer systems, sensor networks, and 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 may have their own hardware, their own code, and sometimes their own independent task. Nevertheless, the nodes may share common resources and information, and, in order to solve a problem that concerns several—or maybe even all—nodes, coordination is necessary.

Despite these commonalities, a peer-to-peer system, for example, is quite different from a multi-core architecture. Due to such differences, many different models and parameters are studied in the area of distributed computing. In some systems the nodes operate synchronously, in other systems they operate asynchronously. There are simple homogeneous systems, and heterogeneous systems where different types of nodes, potentially with different capabilities, objectives etc., need to interact. There are different communication techniques: nodes may communicate by exchanging messages, or by means of shared memory. Sometimes the communication infrastructure is tailor-made for an application, sometimes one has to work with any given infrastructure. The nodes in a system sometimes work together to solve a global task, occasionally the nodes are autonomous agents that have their own agenda 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 that can be considered: nodes may just crash, or they might exhibit an arbitrary, erroneous behavior, maybe even to a degree where it cannot be distinguished from malicious (also known as Byzantine) behavior. It is also possible that the nodes follow the rules indeed, however they tweak the parameters to get the most out of the system; in other words, the nodes act selfishly.

Apparently, there are many models (and even more combinations of models) that can be studied. We will not discuss them in greater detail 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 basic principles of distributed computing, highlighting common themes and techniques. In particular, 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 so 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. In this course, we will address the fundamental question in distributed computing whether a local solution is possible for a wide range of problems.
- Parallelism: How fast can you solve a task if you increase your computational power, e.g., by increasing the number of nodes that can share the workload? How much parallelism is possible for a given problem?
- Symmetry breaking: Sometimes some nodes need to be selected to orchestrate the computation (and the communication). This is typically achieved 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 that fittingly describes this course, it is probably "uncertainty". As the whole system is distributed, the nodes cannot know what other nodes are doing at this exact moment, and the nodes are required to solve the tasks at hand despite the lack of global knowledge.

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. Examples for such topics are distributed programming, software engineering, as well as security and 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!

Chapter Notes

Many excellent text books have been written on the subject. The book closest to this course is by David Peleg [Pel00], as it shares about half of the material.

A main focus of Peleg's book are network partitions, covers, decompositions, spanners, and labeling schemes, an interesting area we will only touch in this course. There exist a multitude of other text books that overlap with one or two chapters of this course, e.g., [Lei92, Bar96, Lyn96, Tel01, AW04, HKP+05, CLRS09, Suo12].

Some chapters of this course have been developed in collaboration with (former) Ph.D. students, see chapter notes for details. Many students have helped to improve exercises and script. Thanks go to Philipp Brandes, Raphael Eidenbenz, Roland Flury, Klaus-Tycho Förster, Stephan Holzer, Barbara Keller, Fabian Kuhn, Christoph Lenzen, Thomas Locher, Remo Meier, Thomas Moscibroda, Regina O'Dell, Yvonne Anne Pignolet, Jochen Seidel, Stefan Schmid, Johannes Schneider, Jara Uitto, Pascal von Rickenbach (in alphabetical order).

Bibliography

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition).* John Wiley Interscience, March 2004.

[Bar96] Valmir C. Barbosa. *An introduction to distributed algorithms.* MIT Press, Cambridge, MA, USA, 1996.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009.

[HKP+05] Juraj Hromkovic, Ralf Klasing, Andrzej Pelc, Peter Ruzicka, and Walter Unger. *Dissemination of Information in Communication Networks - Broadcasting, Gossiping, Leader Election, and Fault-Tolerance.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.

[Lei92] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Suo12] Jukka Suomela. *Deterministic Distributed Algorithms.* 2012.

[Tel01] Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, New York, NY, USA, 2nd edition, 2001.