Large-scale Data Systems

Fall 2018

Prof. Gilles Louppe
g.louppe@uliege.be

Logistics

This course is given by:

- Theory: Prof. Gilles Louppe [g.louppe@uliege.be]

- Exercises and projects: Joeri Hermans (joeri.hermans@doct.ulg.ac.be)

Feel free to contact any of us for help!

Slides

Slides

- are available at https://github.com/glouppe/info8002-large-scale-data-systems.

- are available in HTML and in PDFs.

- will be posted online the day before the lesson.

- will be slightly different from previous years.

Slides are partially adapted from CSE 486/585 Distributed systems (University at Buffalo) and CS425 Distributed systems (University of Illinois UC).

Textbook

Christian Cachin
Rachid Guerraoui
Luis Rodrigues

Introduction to

**Reliable and
Secure Distributed
Programming**

Second Edition

Springer

The core content of this course is based on the following textbook:

Christian Cachin, Rachid Guerraoui, Luis Rodrigues, "Introduction to Reliable and Secure Distributed Programming", Springer.

This textbook is recommended, although not required.

Philosophy

Solid ground

- Understand the foundational principles of distributed systems, on top of which distributed databases and computing systems are operating.

Practical

- Exposition to industrial software.
- Fun and challenging course project.

Critical thinking

- Assess the benefits and disadvantages of distributed systems.
- No hype!

Lectures

- Theoretical lectures

- Exercise sessions

Outline

- Lecture 1: Data deluge

- Lecture 2: Basic distributed abstractions

- Lecture 3: Reliable broadcast

- Lecture 4: Shared memory

- Lecture 5: Consensus

- Lecture 6: Blockchain

- Lecture 7: Cloud computing

- Lecture 8: Distributed hash tables

- Lecture 9: Distributed file systems

Projects

Reading assignment

Read, summarize and criticize a major scientific paper in Large-Scale Data Systems. (Paper to be announced later.)

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then

Programming project

Implement a secure and distributed service using Blockchain technology.
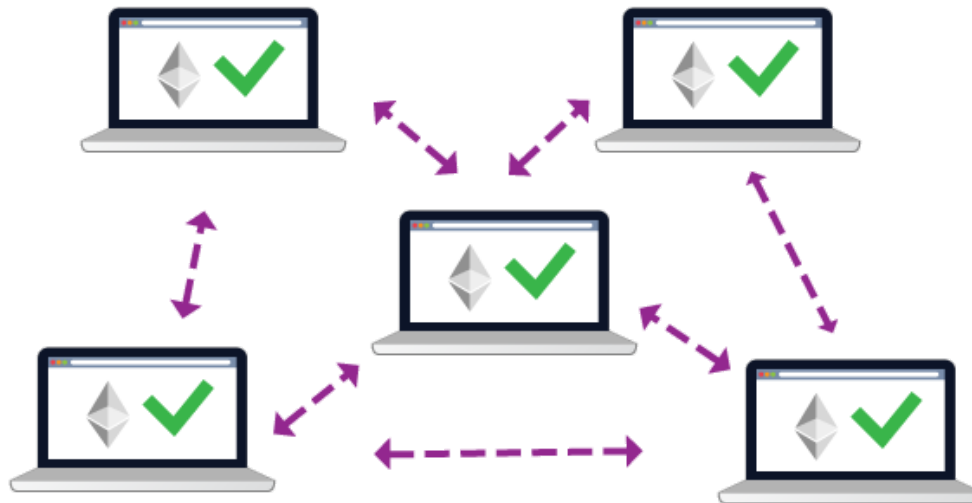
Evaluation

- Oral exam (60%)

- Reading assignment (10%)

- Programming project (30%)

Projects are mandatory for presenting the exam.

Let's start!