Peer-to-peer Streaming System
Distributed Systems Project

Gabriele Voltan, Gianluca Zavan
DMIF, University of Udine, Italy

March 12, 2024

Abstract

The aim of this document is to describe how to write a report for the project-exam for the course Distributed Systems, at the University of Udine. It is also a guideline for the development of the project itself.

This document is very brief and succint, and it is by no means comprehensive of all the informations that should be given about a software project. You are invited to supplement these guidelines as needed to best describe your work.

Chapter 1

Introduction

In this chapter the problem and the goal of the project are presented, with the aim of introducing the work done.

1.1 Problem

The problem which this project is meant to solve is the following:
"N peers want to share data with each other in pairs (P_1, P_2), using the available communication channels, each having a bandwidth B. When a peer P_1 wants to establish a connection with another peer P_2 it is necessary to establish the channels it will use. These channels must be chosen in such a way that the final communication channel has maximum bandwidth."

Problem description

Chapter 2

Analysis

In this chapter, we describe in detail functional and non-functional requirements of a solution for the problem.

2.1 Functional requirements

Which functions must be offered to users / other programs? Which are the input data and the output data? Which is the expected effect?

2.2 Non functional requirements

Everything about mode and transparencies: availability, mobility, security, fault tolerance, etc.

Are there execution time bounds? Minimum data rates?

If requested, specific platforms/languages/middlewares requirements for the implementation can be decided here. (E.g.: if the project is on a SOA, we may request that functions are offered via SOAP or RESTful services).

Chapter 3

Project

This chapter is devoted to the description of the general architectures, and specific algorithms.

3.1 Logical architecture

Describe the components of your systems: modules/objects/components/services. For each component, describe the functionalities it implements, and by who is used.

3.2 Protocols and algorithms

Communication between components. UML sequence diagrams go here.

Also, put here a detailed description of distributed algorithms used to solve specific problems of the project.

3.3 Physical architecture and deployment

Which nodes and platforms involved, and where each component is deployed.

3.4 Development plan

Since it is difficult to predict just how hard implementing a new system will be, you should formulate as a set of "tiers," where the basic tier is something you're sure you can complete, and the additional tiers add more features, at both the application and the system level.

Chapter 4

Implementation

Details about the implementation: every choice about platforms, languages, software/hardware, middlewares, which has not been decided in the requirements.

Important choices about implementation should be described here; e.g., peculiar data structures.

Chapter 5

Validation

Check if requirements from Chapter 2 have been fulfilled. Quantitative tests (simulations) and screenshots of the interfaces are put here.

Chapter 6

Conclusions

What has been done with respect to what has been promised in Chapters 1 and 2, and what is left out.

Appendix A

Appendix

In the Appendix you can put code snippets, snapshots, installation instructions, etc.

Evaluation

Your system will be judged mainly on how it operates as a distributed system. The primary evaluation will be according to whether your system has the following attributes:

- It should be an interesting distributed system, making use of some of the algorithms we have covered in class for distributed synchronization, replication, fault tolerance and recovery, security, etc.
- The software should be well designed and well implemented, in terms of the overall architecture and the detailed realization.
- You should devise and apply systematic testing procedures, at both the unit and systems levels.
- The system should operate reliably and with good performance, even in the face of failures.

Important, but secondary considerations include:

- Time taken to do the project (the sooner the better, but do not miss details in order to end sooner)
- How nice is the application's appearance: does it have a nice interface or a compelling visual display?