The Bottleneck Problem

Distributed Systems - Project 2019

Edoardo Lenzi, Talissa Dreossi
DMIF, University of Udine, Italy

Version 0.1, May 21, 2019

Abstract

The aim of this project is the analysis, the implementation and testing of a distributed solution for the **bottleneck problem** (also know as *The Monkeys Problem*, in operative systems theory).

Basically the problem consists in a two-way road with a **bridge** (bottleneck) in the middle, the bridge has a certain **maximum capacity** and can be crossed in only **one direction at a time**.

The bridge is **very risky** cause it is located in a remote place where there is nothing that can prevent **car crashes/congested traffic** but cars.

Note that **cars** here are **autonomous systems** without any human driver inside and can send messages with others adjacent cars with some wireless technologies (ie. bluetooth, wifi, ...) in order to solve the situation.

The projet requires the implementation of a **simulator/business logic** for the environment setup and a **web server** that will expose the simulation state with some API for any further **UI application**.

Chapter 1

Introduction

In this chapter you describe the main problem, and an idea of the solution. It is not necessary to be very detailed or formal, but it is important to explain which are the main aims and issues from the point of view of Distributed Systems:

- A description of the application.
- The overall structure of the implementation: how resources are deployed, which are the players, the rôles.
- The distributed system features (and the transparencies) and algorithms you intend to implement.
- Your plan for testing the system.
- A schedule for how you plan to carry our your design and implementation

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 youre 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

Conclusions

What has been done with respect to what has been promised in Chapters **??** and **??**, 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?