DM883 Project

Exam 2022

Description

In this group project you will solve one of the themes listed in this document. The themes are intentionally open ended in terms of technologies (e.g., programming languages, libraries, middleware). Your first task is to refine a theme of your choosing by writing a short specification (key features, technologies, etc.) and reach an agreement with the instructor by the project selection deadline (see below). Consider this an interactive process and seek feedback from the instructor before the deadline. You can update this specification after the deadline provided changes are discussed with the instructor and approved.

You need to sign up for a project group via itsLearning by the deadline specified below. Groups must have three members, two if it is not possible to form a group of three. The instructor can reorganise groups to meet this criterium.

Important Dates

- Group Registration: 2022-05-05 23:59 CEST
- Project Selection: 2022-05-12 23:59 CEST
- Final Submission: 2022-06-02 23:59 CEST

Any changes will be announced on itsLearning.

Policies

No late submissions will be accepted/graded unless the instructor is contacted in advance with a reasonable explanation for the delay. If the instructor approves late acceptance of a submission, the submission will incur a late penalty. Submissions received after the due date will not be graded. The actual late penalty is at the instructor's discretion and may vary (based on the reason for the delay and how early the instructor was informed).

All submissions must be your own work. No collaboration is allowed on projects (except with your project partners).

We will default to SDU's academic policies unless stated otherwise above.

Submission

You must hand in a zip file with following:

1. A PDF document named "`report.pdf`" consisting of your report.
2. A folder "`artefact`" with your artefact.

The name of the zip file must be the name of your group e.g., "`Group C18.zip`" (capitalisation is immaterial).

The artefact consists of your implementation of the system described in the project theme of your choosing together with a `README` text file with the instructions for running it. Contact the instructor in advance if your project requires a particular configuration besides a standard linux environment with Erlang/OTP and Docker.

The report should include the following:

- A description of the architecture design, the main challenges it poses and your algorithmic solutions (for algorithms seen in class or found in the literature it is enough to give a brief discussion to motivate the choice and any challenges/changes needed to use them in your settings).
- A discussion of any added assumption (besides the ones listed in the project description) together with a brief motivation.
- A reflection on the correctness of your solution.

The report must be in English or Danish and delivered as a single PDF file printable in black and white. The report must list all members of the group (full name and SDU email).

Project Themes

Project 1: Decentralised Chat

Design and implement a distributed chat that supports multiple groups. The system must offer:

1. Listing users in a group chat and search for users based on their name.
2. Searching for groups based on their name.
3. Causal ordering of messages.
4. Message history (you may assume a time or length limit).
5. An interactive user interface.

The system must use a decentralised or semi-decentralised structure. There are failures but you are free to assume a failure model of your choosing.

Project 2: Decentralised hierarchical storage system

Design and implement a decentralised storage system (a key-value database, a document storage, or a rudimentary file system) and implement a prototype of your design. Your solution should prioritise fault tolerance and scalability. The system must support a folder hierarchy for resources (you may assume that the hierarchy is a tree) together with an API for its manipulation and navigation.

The system must use a decentralised or semi-decentralised structure. You may decide to allow nodes to act purely as clients (instead of full peers) and consume the services of the storage system without joining it.

Project 3: Peer-to-Peer Web

Design and implement a decentralised web based on a p2p architecture where clients are peers. Your solution must focus on scalability, trust, and fairness.

Project 4: Distributed media streaming

Design and implement a decentralised streaming solution. Your solution must support live streaming and focus on scalability and quality of service.

Project 5: Peer-to-Peer Algorithms

Select a structured P2P DHT (Chord, CAN, Tapestry, …). Assume that there are failures and (if necessary) adapt the algorithm to achieve fault-tolerance. Extend the DHT with at least one of the features below.

1. *Load balancing.* The load on the nodes in a DHT can vary drastically due to the varying popularity of resources and keys or the sheer number of keys a node must manage. Develop a mechanism for load balancing like e.g., trading keys between nodes based on their relative workload.

2. *Federation*. Working on heterogeneous or geographically spread networks can be challenging for P2P systems as the overlay network may not reflect administrative or performance critical properties of the underlying network. DHT Federation is the practice of combining various DHTs into a single system. Under this practice the single DHTs of the federation are usually restricted to a local area or an organisation (e.g., to control data placement) but can interact with other DHTs when necessary (e.g., to retrieve a document hosted on another DHT).
3. *Advanced queries*. Extend your DHT to support advanced queries like queries by attribute, range, or approximated search (at least one of these cases). For queries by attribute, you may assume a fixed list of supported attributes.

Implement a prototype of your solution.

Project 6: A simulator for Algorithms on Graphs

Design and implement a workbench for algorithms on graphs (e.g., those in Chapter 5 of the course book). The system must support:

1. Running a simulation while visualising the graph and the status of its processes (you may assume that processes need to push updates of their states using an API you provide).
2. Injection of node and link failures.
3. Running multiple simulations with different initialisation parameters and reporting theirs results.
4. Utilities for generating graphs (e.g., common topologies, random graphs).

The algorithms for the simulation are written in Erlang (e.g., a module exposing an API of your choosing or implementing a custom OTP Behaviour of your choosing) and you may assume they rely only on the STDLIB. Validate your solution by evaluating it on a suite of algorithms from the course literature (to be agreed with the instructor)

Project *: Student ideas

Anything relevant for this course but not covered above. Bear in mind that the instructor might request major changes or reject your suggestion altogether, so it is advisable to start the discussion as soon as possible.