Distributed Systems - Project 2023

For 2023: Projektin kuvauksen pitää ohjata suunnittelemaan riittävän monimutkainen hajautettu järjestelmä, josta sitten toteututetaan vain pieni osa. Vuonna 2022 useampi ryhmä suunnitteli niin pienen järjestelmä, että noiden erilaisten piirteiden toteutus ei ollut tarpeellista sovelluksen näkökulmasta.

Overview

The goal of this group work is to a) design a complex-enough distributed system and then b) implement a simplified prototype of it. Typical group size is three (or four) students. The grading is done assuming that the group had three active students. You can do the task also individually or in pairs, but you may have to use more hours per person than a larger group.

The prototype must be running on at least three (3) separate nodes. You may use the computers: svm-11.cs.helsinki.fi, svm-11-2.cs.helsinki.fi and svm-11-3.cs.helsinki.fi or you can choose some other environment feasible to your group. The simple client node used to test your system cannot be counted in the three nodes. The nodes can be physical or virtual ones. Each node must have its own IP-address and the communication must be done via message passing; sockets and remote procedure calls (RPC) are both fine.

Your distributed application's functionality can be anything. Some examples include a multi-player game, a collaboration tool, distributed user interface, distributed data processing for sensing devices, etc. Each application instance on a separate node must exchange data with at least two other instances to form a shared state across the distributed nodes.

Some topic examples:

1. A multiplayer game
2. An online store.
3. Synchronized music player.
4. Distributed analysis of some publicly available datasets.
5. Toy version of content delivery system.
6. Your own idea.

Do not focus your work on the application area, facy interface or functionalities for the client, but the required distribution features and how they support the selected application area.

About the distributed application

When selecting the topic, keep in mind, that the system must have a shared distributed state and that each node must communicate with at least two other nodes. In addition to shared state, your system should have at least three of the following functionalities: a) naming and node discovery, b) synchronization and consistency, c) fault tolerance, d) consensus.

Your system might have different phases and it might remind a centralized service for some purposes and be very decentralized for some others. It might also change the set of functionalities, if some events happen.

When planning your system, think big, because only then would some of the features be actually needed. So plan for a large-scale service and then think how you can demonstrate the key features with a tiny, functionally-limited prototype.

Web services

For web services the project requirements mean that you need to focus on the server side and make it complex enough. So that the group of nodes forming the server side share the state and implement the required features. On these applications using a reverse proxy is strongly recommended.

You are not allowed to count the node running the client browser interface, if it only communicates with the www server /reverse proxy. You need at least three nodes on the server side.

Others (like P2P)

All distributed applications that are not based on www servers and clients have a large option of architectural choices and communication options. Here the key focus is that each node must communicate with at least two other nodes. The system might not even have a human client. Think about the DNS system as an example. On these applications having a leader election is recommended.

Requirements for the nodes in the system

The planned system typically has plenty of nodes. The implemented prototype must have at least 3 participant nodes, but can have more (or you can run multiple processes simulating extra nodes on some of these. Each of the minimum 3 participant nodes in the prototype must:

1. be running on a separate computer / virtual machine
2. have its own IP-address
3. communicate with at least two other nodes only by using Internet protocol based message exchange
4. be able to express their state and/or readiness for sessions towards other nodes

If your design needs, you can have e.g. client node that communicates only with one server node. Because it only has one communication partner it cannot be counted to the required minimum number of nodes.

All nodes must log all important events of their activity either locally or to a central location. This feature is useful for debugging and demonstrating the correct functionality of your system.

Nodes can have different roles as long as they meet the requirement of communicating with at least two other nodes. They do not have to be identical. It is very typical in a distributed system that nodes have different functionalities. For example, one can act as a monitor/admin and another one as a sensor/actuator.

Please note that details of session and state are dependent on the application implemented; thus, they depend on design and are not identical across implementations.

Some technical issues

You can choose programming language yourself. You may even use different languages on different nodes, if you wish.

Grading focuses on the distribution aspect of your system, not on the user interface or application logic. There has to be some application logic and user interface for you to demonstrate the system behavior, but they can be very coarse.

Do not overcomplicate things, but do not plan too small either – a good initial design will help a lot. Remember to design the communication protocol, that is the messages and how to use them between the nodes.

We recommend you to use virtualization techniques on your own computer at least in the development phase

1. Virtualization technologies can be used for emulating several individual machines on a single computer (e.g., VMWare, Virtual Box)
2. Virtual machines running on a single computer are usually much easier during development than real distinct physical computers
3. When using virtual machines you need to configure the virtual network so that the processes on different virtual machines can communicate with each other.

It is also recommended to deploy the prototype to the provided three computers for the final tests.

We also recommend you to use a code repository (preferably GitHub or version.helsinki.fi) during the project to help in the software production and version handling, but also to make the final submission easier.

If you want to use containers as your implementation technique, you can do so. Please note, that you cannot use one container to implement all nodes, because each node must be placed in a separate physical or virtual computer. With containers the recommendation is to use one container for each separate node and place them manually to different virtual machines. Alternatively you can use kubernetes or similar technique for the placement, but that is not required here. With few nodes the manual placement works just fine. If you are not familiar with them already, please do not waste your time in learning containers or kubernetes for this project. It is not needed here.

Simple python, C or java application using datagrams in communication between the nodes works just fine for this project.

About the required functionalities

The implemented prototype must have

1) A shared distributed state
2) Synchronization and consistency
3) Consensus
4) And either a) naming and node discovery or b) fault tolerance or c) both

The originally planned system should also contain features related to scalability and performance, but these are typically not included in the tiny prototype.

Shared distributed state

Shared distributed state means that each node maintains some information about the previous events for correct functionality. If node can react to each arriving message without any former information about the history, then the node does not have a state.

Synchronization and consistency

Synchronization and consistency have a close relationship with system state, but they focus specifically on the data maintained by the whole system. There are multiple different levels of consistency for data as well as the synchronization between nodes. The required level is application-specific. Some applications tolerate lower levels of synchronization and/or consistency than some other applications.

Usually these have to be addressed in all distributed systems, but you might be able to design and implement yours without giving any specific focus on these.

Consensus

Consensus is a joint decision-making mechanism, where all the nodes agree on something. It is a distributed process where all nodes participate. If the system has a single, central decision maker (e.g. a master node) then there is no need for consensus protocols.

Naming and node discovery

Because different parts (processes) of the distributed systems are located on different nodes, there has to be a mechanism to find the nodes. The nodes have to be identified by e.g. name or number and a process on node must be able to communicate with a process on another node. It has to be able to find the other nodes one way or another.

You can do this by hard coding the IP numbers of other nodes in your code, but then you are not using naming and node discovery.

Fault tolerance

Fault tolerance is an attribute of a system and it covers issues related to failure of one or few nodes, not the whole system. It defines how the remaining nodes handle such a situation and how the system might recover from such an incident.

Additional features: scalability and performance

Scalability and performance of your system has to be addressed in the final report. In case you did not focus on these in the planning, then you still need to discuss them in the report. So, you do not need to design these features in the system, but simply to discuss the system also from these viewpoints.

Deliverables and grading

You need to submit four deliverables (design plan, video or demonstration in demo session, program code and a report)

The reports (design plan and final report) must have a cover page with project name, group number and team member names. Final report must also have table of content.

Design plan (5 points)

The goal of the design plan is to start working with your programming project early during the course. Plan gives you maximum of 5 points. You will get one point for each of these elements

- names of team members and one paragraph description of the selected topic
- more detailed description of the topic and/ or selected solution techniques/methods
- description of different nodes, their roles and functionalities
- description of messages send and received by the nodes (syntax and semantics) – this contains the ones you have identified so far
- some comments (one or two paragraphs about which features are you going to implement, and/or the scalability and fault tolerance aspects or goals)

Design plan must contain team member names and short description of the selected topic

Video or demonstration in a demo session (5 points, maximum duration 5 minutes)

We will organize at least one demonstration session, where the groups can present their projects. Alternatively to live presentation groups can provide a video. Video or on-site demonstration must demonstrate the system setup and all key functionalities.

Points 1-5 will be given based on the coverage of features in the demonstration/video, as well as clarity of the demonstration. The demonstration clarity does not contain fancy features or technical video editing features, but the flow of presentation, expectations of the audience knowledge level, etc features of a good oral presentation.

Source code in repository (5 points)

Source code must be submitted for review in a repository (preferably GitHub or version.helsinki.fi).

The code must be readable. This usually requires comments and good naming conventions with clear and meaningful names.

The code is graded with points 1 to 5 depending on the clarity of the code. Also good coding practices play a role in the grading.

Note that the teachers of the course must be able to access the repository content.

Final report (15 points)

Final report must be in line with design plan (all major changes must be documented as appendix), program code and the video. If there are issues in this, it will be notified in the grading of the final report.

In addition to cover page and table of content, the final report must address:

1. The project's goal(s) and core functionality. Identifying the applications / services that can build on your project.
2. The design principles (architecture, process, communication) techniques.
    - This is typically the main part of the report, because it documents the system design and maps it with the source code

3. What functionalities does your system provide? For instance, naming and node discovery, consistency and synchronization, fault tolerance and recovery, etc? For instance, fault tolerance and consensus when a node goes down.
4. How do you show that your system can scale to support the increased number of nodes?
5. How do you quantify the performance of the system and what did you do (can do) to improve the performance of the system (for instance reduce the latency or improve the throughput)?
6. The key enablers and the lessons learned during the development of the project.
7. Notes about the group member and their participation, work task division, etc. Here you also may report, if you feel that the points collected to group should be split unevenly among group members. Use percentages when descripting this balancing view point.

The grading of the final report will be split among the topics so that parts 1&2 together will give you 1-8 points and the parts 3 to 6 each 1-2 points. (Note this totals to 18, but maximum is 15)

Please notice that some elements (like design principles, scalability, performance, etc.) can also be explained in video/demonstration or commented in the source code, in such a case please mention that in the final report.

Other elements that play a key role in grading of video/demonstration, source code and final report are

1. A running system that implements the basic goals and functionality
2. Demonstration of system scaling. For instance, begin with 3 nodes, and then show the system can support 4 nodes, 5 nodes, and more than 5 nodes.
3. Evaluation of the performance in term of relevant metrics such as throughput or latency etc. Identifying what can be done to improve the performance.
4. Functionalities provided by the system. At least three of the following: a) naming and node discovery, b) synchronization and consistency, c) fault tolerance, d) consensus.

Point balancing

By default the grading of project is even, so every member gets the same amount of points for each element.

If the workload or contribution is unbalanced, the group can add a separate section to the final report and inform teacher how the points should be balanced. This balancing must be informed using percentages. In a balanced situation each group member gets 100% of the points. Team can move 10, 20 or 30% from one or more students to one or more another students, if they feel that some students' contribution is less that some others' contribution. You must not leave unused percentages, but you cannot invent extra percentages either. The total has to be x*100, where x is the number of students in the group who have not dropped out

Student, who has dropped out of the group, will get 0 points. However, it is possible to get points for the design plan, if the student drops out after the design plan has been delivered. If the group feels that the dropped out student has had feasible contribution to code or final report, then they can recommend a percentage that this student will get from the project points. This percentage can be 10, 30 or 50.

Suggested Schedule:

Weeks 1 and 2; form the team and agree about the project topic

Week 3: submit the design plan, decide the programming language, create repository (deadline for design plan Monday 21.11 afternoon)

Week4: basic skeleton/outline of the code, identification of the strategy to evaluate and demonstrate the system.

Week 5: A basic running system that can work on 3 nodes

Week 6: Preliminary evaluation, and implementation of the scaling and functionalities (teacher feedback from design plans)

Week 7: Finalize your system and create the final deliverables, live demo sessions, deadline for submission Friday