Algoritmos e Sistemas Distribuídos- Project Phase 1

João Carlos Antunes Leitão

NOVA Laboratory for Computer Science and Informatics (NOVA LINCS)

and

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade NOVA de Lisboa

V 0.5

16^{th} October 2020

1 Overview

This document discusses the first phase of the ASD project for 2020/21. The project has two phases which are fully independent between them. The primary goal of the project is to build a robust and efficient peer-to-peer broadcast system and experiment it in practice. The secondary goal of the project is actually to build different variants of a peer-to-peer broadcast system by combining two diferent epidemic dissemination protocols and two different unstructured overlay networks, such that you get a total of four different solutions, and experiment these solutions in a computational cluster such that you can observe (and discuss) the differences in performance and costs that you might observe. The project is designed as a very small introduction to research in distributed systems, in particular for peer-to-peer systems. In the following the target protocol architecture (and their interfaces) for solving this phase of the project are presented (Section 2). The programming environment for developing the project is briefly discussed next (Section 3). Finally, the document concludes by providing some information on operation and delivery rules for this phase of the project (Section 4).

2 Solution Architecture

Figure 1 illustrates the layering of protocols that you will have to have to for phase 1 of the project[1]. The figure also describes the interactions between the protocols, from which the **interfaces** of the protocols can be derived.
The project will have a strong emphasis on the experimental component, where you will assert the performance of cost of a peer-to-peer broadcast solution in a system with at least 100 different processes. To do this you will be responsible for implementing (up to two) epidemic broadcast protocols and (up to two) unstructured overlay network protocols.
Every protocol that you will implement should have an equivalent interface to communicate with the other local protocols (this is materialized by Requests and Indications). This is essential to ensure that you can swap one protocol

[1] for evaluation purposes this layering is *mandatory*.

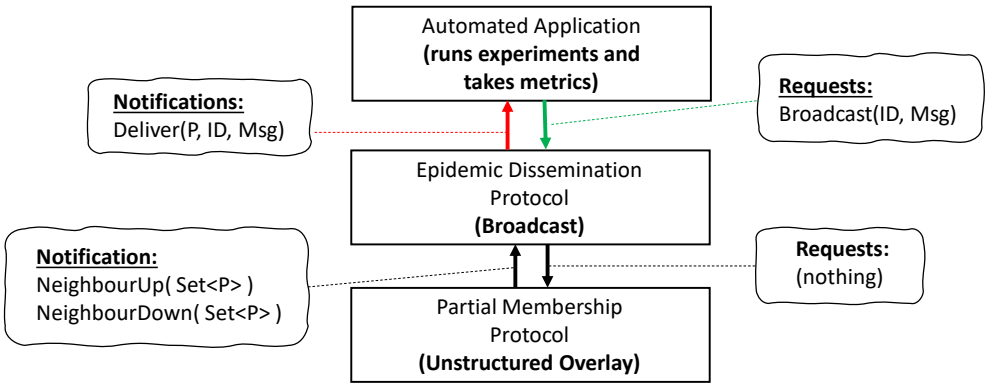

Figure 1: Architecture of a Process.

implementation for another and that your process can still execute correctly. Notice that the messages exchanged by each protocol that you develop with the (equivalent) protocol on other processes is not fixed. You should model these messages following the specification of the protocols that you are implementing.
Below the options for each of the relevant protocols is provided (with the references to papers when applicable).

2.1 Epidemic Broadcast Protocol

You can pick up to any two of the following protocols when executing this phase of the project:

Eager Push Gossip: You can implement a simple eager push gossip protocol akin to the one described in the HyParView paper [4]. Notice that you should not use a flood protocol. Instead you should consider what you have learned in lectures to have the protocol to operate adequately.

Adaptive Gossip Strategy: You can implement an adaptive gossip protocol in line with the ideas describe in [1]. This includes commuting the gossip strategy considering aspects such as the number of hops that the message has already taken in the overlay.

Plumtree: You can implement Plumtree [3] a gossip solution that combines eager push gossip and lazy push gossip to embed spanning trees on top of an unstructured overlay network. This tree is then used to propagate messages fast by using eager push, while the remaining links are used for lazy push gossip as a mechanism to allow processes to detect (and recover) from tree partitions.

Your own solution: You can also propose your own design (or adapt the protocols above[2]) for an epidemic broadcast protocol. You evidently should strive to come up with designs that are better than existing solutions and be careful in exploring the inherent trade-offs between the reliablity and latency of the broadcast process and the total cost in terms of messages. You cannot have two epidemic broadcast protocols of your own design.

2.2 Unstructured Overlay Network Protocol

You can pick up to any two of the following protocols when executing this phase of the project:

Cyclon: Cyclon [5] in a cyclic protocol where each process continually performs shuflles with one neighbor which results in both processes updating their own neighbor set. The protocol is quite simple and has been studied in the lectures.

[2]If you adapt one of the protocols above, the second protocol cannot be the same that you used as base for your implementation.

Scamp: Scamp [2] is a reactive protocol where processes keep their neighbors except when an external event leads them to change their neighbors (such as a new process joining the system or an existing process failing). The protocol ensures that processes converge to have a number of neighbors that is logarithmic with the total number of processes in the system without any process being aware of this number.

HyParView: HyParView [4] combines ideas from both protocols above to improve the fault tolerance of the unstructured overlay network. It has two partial view, the first and smaller one is managed using a reactive strategy, and it is this one that is used to disseminate messages. The second and larger partial view is used to store candidates to become replacements in the first view when failures happen.

Your own solution: You can also propose your own design for an unstructured overlay network. You evidently should strive to come up with designs that are better than existing solutions and be careful in exploring the inherent trade-offs between the management cost of the overlay and the guarantees that are provided to you. You cannot have two overlay network protocols of your own design.

2.3 Application

The application is provided to you (source code and maven project is availble in clip). The application is relatively simple but it contains a set of features that will be quite useful for you.
The application will generate messages that are broadcasted (by sending a request to the epidemic broadcast protocol) throughout the system. The application also receives messages sent (by the local process or other process). It registers all events (message being sent or message being received) in a log, that includes the timestamp of the event and a unique identifier of the message. Furthermore, the application also extracts some metrics from the use of the network, in particular the number of messages sent/received as well as the bytes sent/received per TCP connection used by the process. This will greatly simplify the collection of performance metrics during the evaluation step of (different) broadcast solutions.
Furthermore, the application allows you to configure several operational aspects for your experiments. The two key parameters that we will be interested in manipulating for experiments are:

Rate of transmission: The time interval at which the application on a process generates and broadcasts messages.

Payload size: The size of the message in bytes (excluding mandatory headers and control information.

2.4 Evaluation

You will conduct an experimental evaluation where you compare different solutions (i.e., different combinations of epidemic broadcast protocol and unstructured overlay protocol) in different conditions. You will use the computational cluster of DI/NOVA LINCS (see below) to conduct these experiments.
You should compare several aspects of the operation of each solution. In particular you must evaluate the following aspects (you can evaluate others that you consider relevant):

Average Broadcast reliability: The reliability of a single broadcast is computed as the fraction of (correct) processes in the system that effectively deliver the message. This metric considers the average of the reliability of all messages broadcasted during the experiment.

Average Broadcast latency: The latency of a single broadcast is computed as the time since the sender application issues the request to the epidemic broadcast until the time at which the last process that delivers the message receives the message at the application layer. The metric considers the average of this value for all messages broadcasted during the experiment.

Total Messages/Bytes Transmitted: The amount of messages/bytes sent to the network collectively by all processes during the experiment.

Total Messages/Bytes Received: The amount of messages/bytes received from the network collectively by all processes during the experiment. This number should be the same as the above in the case where no messages were lost of processes failed.

In your experiments you can use two different values for the payload size (one small another bigger) and two different values for the rate of transmission (again, one small and another bigger). As defined in the evaluation rules (below) you can conduct experiments with a single pair of values, or you can optionally use the four combinations of parameters that are possible using the two distinct values for each parameter. You can conduct experiments only in steady state (i.e., in scenarios where no process crashes during the experiment).
You evidently can consider other experimental conditions **in addition** to the ones described above. These can include varying parameters that govern the behavior of the epidemic broadcast protocol or the unstructured overlay network. You can also consider to perform experiments where processes fail (in a way controlled by you). Every extra aspect will be taken into consideration in the evaluation process.

3 Programming Environment

The students will develop their project using the Java language (1.8 minimum). Development will be conducted using a framework developed in the context of the NOVA LINCS laboratory[3] written by Pedro Fouto, Pedro Ákos Cost, João Leitãoa, whose internal code-name is (still) **Babel**[4].
The framework resorts to the Netty framework[5] to support inter-process communication through sockets (although it was designed to hide this from the programmer). The framework will be discussed in the labs, and example protocols will be made available to students. Moreover the top layer that is responsible for injecting load in the system (i.e., propagate messages and receive them) is offered.
The javadoc of the framework can be (temporarily) found here: `https://asc.di.fct.unl.pt/~jleitao/babel/`.
The framework was specifically designed thinking about two complementary goals: i) quick design and implementation of efficient distributed protocols; and ii) teaching distributed algorithms in advanced courses. A significant effort was made to make it such that the code maps closely to protocols descriptions using (modern) pseudo-code. The goal is that you can focus on the key aspects of the protocols, their operation, and their correctness, and that you can easily implement and execute such protocols.
The course discussion board in Moodle can (and should be) used to ask for support in using or clarify any aspect on the operation of Babel.
While this is the second year that Babel is being used in this course, and the current version has also been used to develop several research prototypes, the framework itself is still considered a prototype, and naturally some bugs (or idiotic interfaces) can be found. We will address these as soon as possible. The development team is open to suggestions and comments. You can make such comments or improvement suggestions either through the Moodle discussion board or by e-mail to the course professor.

4 Operational Aspects and Delivery Rules

Group Formation

Students can form groups of up to three students to conduct the project. Groups composition cannot change across the different phased of the project. While students can do the project alone or in groups of two students this is **highly discouraged**, as the load of the project was designed for three people.
Since you will be given access to the DI and NOVA LINCS research cluster (see below) to conduct your experiments and extract performance numbers, you must pre-register your group as soon as possible such that you can get credentials to use the cluster. This can be done by sending an e-mail to the course professor with subject *ASD 20/21 GROUP REGISTRATION*. The e-mail should contain for each member of the group: student number, full name, institutional

[3] `http://nova-lincs.di.fct.unl.pt`

[4] At this point no one in this team is happy with the name – for different reasons – so we accept suggestions.

[5] `https://netty.io`

e-mail. Even students that plan to do the project alone must register their group. You will get a reply from the professor (eventually) providing your user name and password to access the cluster.

The DI and NOVA LINCS research cluster

The cluster is used for (mostly) research purposes, although some (advanced) courses also use it. The cluster has a limited amount of machines, and hence people might need to compete over computation time. The cluster features a reservation mechanism where you can reserve a set of machines for exclusive access during a period of time. You should not use the cluster for development purposes and only to extract final numbers. Moreover, each group should only reserve at most two machines at a time. Each machine should be able to execute several tens of independent java processes that will act as different nodes. Docker is available in the cluster but you should not need to use it. We will provide simple scripts to help you in executing your experiments. Finally, each group will be restricted to use only $3h \times 2$ of computational time. This should allow you to execute your experiments for 3 hours on two machines. Notice that automation is key for success, and that should allow you to run experiments during the night period (where typically usage of the cluster is much more reduced).
The technical specification of the cluster as well as the documentation on how to use it (including changing your group password and making reservations) is online at: `https://cluster.di.fct.unl.pt`. You should read the documentation carefully. The cluster is accessible from anywhere in the world through ssh. Be careful with password management. Never use a weak password to avoid attacks and intrusions on our infrastructure. Incorrect or irresponsible use of the department resources made available to students will be persecuted through disciplinary actions predicted in the School regulations.

Evaluation Criteria

The project delivery includes both the code and a written report that should have the format of a short paper. Considering that format, this document referes from this point onward as simply *paper*.
The paper should be at most 10 pages long, including all tables, figures, and references. It should be written with font size no bigger than 10pts and be in two column format. It is highly recommended (for your own sanity) that you use LaTeXto do this. If you do not know LaTeXit is indeed a great time to learn. Videos from an introductory course on LaTeXby João Lourenço shall be provided on-line at a future date. A LaTeXtemplate for writing the paper shall also be provided[6]. The course professor will also dedicate an entire lab class on how to structure and write a paper.
The paper must contain clear and readable pseudo-code for each of the implemented protocols, alongside a description of the intuition of these protocols. A correctness argument for protocol that was devised or adapted by students will be positively considered in grading the project. The written report should also provide information about all experimental work conducted by the students to evaluate their solution in practice (i.e., description of experiments, setup, parameters) as well as the results and a discussion of those results.
The project will be evaluated by the correctness of the implemented solutions, its efficiency, and the quality of the implementations (in terms of code readability).
The quality and clearness of the report of the project will have an impact the final grade. Notice however that students with a poorly written report might (accidentally) penalized on the evaluation of the correctness the solutions employed.
This phase of the project will be graded in a scale from 1 to 20 with the following considerations:

Groups that only implement a single broadcast protocol and a single overlay network protocol, and that experimentally evaluate those protocols with a single set of experimental parameters, will at most have a grade of 14.

Groups implement (and experimentally evaluate) one additional broadcast protocol and one additional overlay network protocol can get up to 4 additional points.

Groups that in additionally conduct experimental evaluation of all implemented protocols using a combination of two different payload sizes for broadcast messages and two rates of broadcast on their test application, can ge 4 additional points

[6] And hopefully, for those that do not mind in using software that is corrupted by pure evil, an equivalent template in Word. However no promises can be provided on this.

Deadline

Delivery of phase 1 of the project is due on 15 November 2020 at 23:59:59.
The delivery shall be made by e-mail to the address jc.leitao@fct.unl.pt with a subject: "ASD Phase 1 Delivery - #student1 .. #studentn".
The e-mail should contain two attachments: $i)$ a zip file with the project files, without libraries or build directories (include src, pom.xml, and any configuration file that you created) and $ii)$ a pdf file with your written report. If you have an issue with sending the zip file you can do one of the following: $a)$ put a password on the zip file that should be 'asd2020' with no quotation marks; or $b)$ put the zip file in a google drive, and send the public access link.
Project deliveries that do not follow these guidelines precisely may not be evaluated (yielding an automatic grade of zero).

References

[1] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. Emergent structure in unstructured epidemic multicast. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 481–490, 2007.

[2] A. J. Ganesh, A. . Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2):139–149, 2003.

[3] J. Leitao, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 301–310, Oct 2007.

[4] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 419–429, June 2007.

[5] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, Jun 2005.