Tool for peer-to-peer topologies testing in collaborative applications

Quentin LAPORTE-CHABASSE Loria - Team COAST
quentin.laporte-chabasse@telecomnancy.net

June 29, 2016

Abstract

In the context of OpenPaaS::NG project, COAST Team is designing a peer-to-peer infrastructure for real-time collaborative editing. Thus clients are directly inter-connected each other and can communicate without passing through a centralized server. This approach can partially solve the scalability issue, indeed the server is not anymore the bottleneck of the application. However it is necessary to test the application with a large amount of clients, in order to reach the limits of the peer-to-peer toplogy. The tool provided, simulates a large scale collaborative session and measures the effects of topology on network delays and clients ressources. This document provides informations about our motivations, technical and architectural choices and experimentation itself.

I. Motivations

It is necessary to be able to test the performances and the effects of peer-to-peer topology on user experience. We must ensure that the topology used is reliable and is enough sacalable according to the specifications of the collaborative application. In addition, we must collect enough data in order to perform a comprehensive state of the art.

The tool provided is inspired by an experimentation designed to compare the performance provided by our collaborative text editor[1] and Google Doc. The first experimentation was limited by the number of collaborators deployed and could only be run on few computers. Therefore, we decided to focus our work on the capacity to deploy a large amount of clients.

The second main topic that we addressed is the reproducibility of the tests. It is crucial to be able to relaunch the same tests and to ensure that the measurement collected are correct and are not due to side effects.

To summarize, all requirements are listed below.

Requierements :

- The experimentation should be reproductible
- The experimentation should deploy a large amount of collaborators
- The experimentation should allow to test a large variety of collaborative applications
- The experimentation should measure the network delay
- The experimentation should measure the ressources taken by each client (ie collaborator)

II. Architecture and technical choices

i. Environment

As mentioned above, the tool must deploy a large amount of collaborators, we need consequently an high computing capacity al-

[1]MUTE for Multi User Text Editor : `http://mute-collabedition.rhcloud.com/`

lowing to support all the virtual collaborators. At the first sight, flexible solutions provided by cloud servercices [2] should be the easiest way to delpoy this experience. But in order to be close to real conditions, we should be able to simulate workstation and network capacities. Cloud services are therefore limited and cannot easily offer such features. In the context of computer science research, a large scale and versatile testbed is provided (in Loria). This infrastructure named Grid5000 allows us to deploy easily an experimentation and parametrize the running conditions. Technical choices are made in order to be compliant with this tool.

ii. Global architecture

The Figure 1 below shows the global architecture of the testing tool. The architecture includes :

- A server which is in charge of managing the experimentation
- A RabbitMQ server which supports the communication between the server and clients
- Clients which simulate collaborators
- Collaborative application which is currently tested[3]

The following paragraphs describe the behavior of each component of the architecture, the RabbitMQ server is just a communication medium and is therefore ignored.

ii.1 The server

The server manages the experimentation, it keeps in memory all the caracteristics that will be played[4]. It is in charge of synchronize all clients and collects clients measurements at the end of the experimentation. The server communicates with clients by RabbitMQ server. It also provides an HTTP API which allows clients to communicate with it.

[2]Like AWS, Google AppEngin, etc.

[3]In our case, the targeted application is MUTE

[4]In our case : *number of collaborators, typing speed, duration ...*

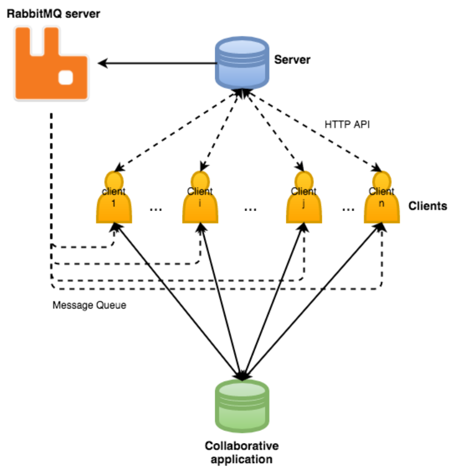

Figure 1: *Architecture of the application*

To summarize, the server should :

- accept subscription queries from clients
- manage lis of clients
- communicate with clients
- accept measurements sent by clients

The behavior of server is described below by Figure 2

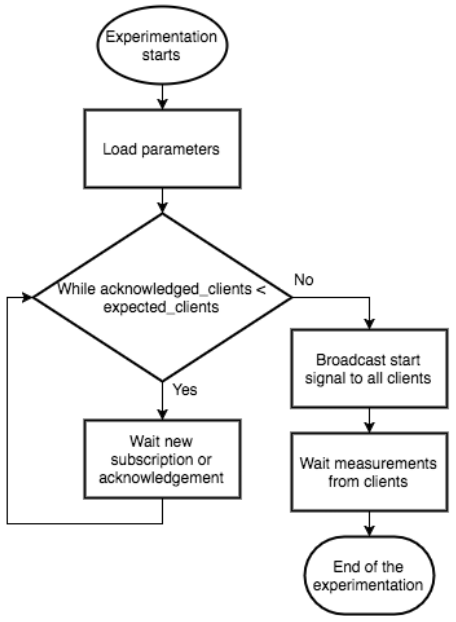
Experimentation starts
Load parameters
While acknowledged_clients < expected_clients
No
Yes
Wait new subscription or acknowledgement
Broadcast start signal to all clients
Wait measurements from clients
End of the experimentation

Figure 2: *Server behavior*

ii.2 The clients

The clients simulate collaborators, in the context of collaborative edition they can play either the role of writer or reader. The roles are given by the server during the subscription step. Clients behave like real collaborators, they use chrome browser and interact with application like a real user. A reader intercepts all modifications applied on shared document and save them in a log object with associated timestamp. At a regular time intervals, the writer write a unique word given by the server. Each writer has a unique word among all the others writers and keeps it during the whole session. For each word written, writer also saves the word written and the timestamp associated in a log object. When the time is up, collaborators send log objects to the the sever. In this way, log ojects ban be combined and we can easily know the delay between message sending and the reception.

In spite of communications with the server, clients communicate directly each other by peer-to-peer connections. All the messages go through a peer-to-peer network as shown in Figure 3. The topic of our research focuses on the peer-to-peer topology. This topology is described by the collaborative application. Thus, it is important to emphasise that the testing tool doesn't affect the application functionning. The topology below is the easiest to implement but the one of least scalable. It may change in a close future according to the progression of our work.

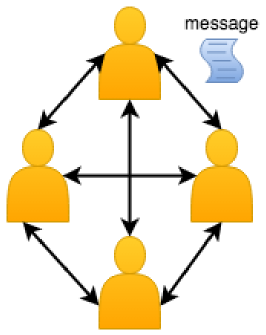

Figure 3: *Topology "fully-connected"*

ii.3 The collaborative application

The collaborative application runs independently from testing tool and it may even hosted anywhere[5]. We are currently working on our prototype MUTE wich is the targeted application of the test. You may change this one, but in this case, you should change the behavior of collaborators.

iii. Technical choices

Apart from the collaborative text editor, all the components of testing tool are deployed over Grid5000. This technical infrastructure provides a wide variety of softwares allowing us to simulate environment and made the tests reproductible.

We use Distem[6], a ruby library provided by Grid5000.

III. Exeperimentation

i. Scenario

References

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.

[5]It must be accessible from internet

[6]Distem : `http://distem.gforge.inria.fr/`