Universidade de Aveiro

Distributed Photo Organizer

Final Report

Inês Baptista 98384, Diogo Silva 103925

Distributed Systems

Computer Science

2022

1 Introduction

A group of friends is collecting photos of Aveiro. They are storing these photos on their personal computer and now they want to share them with each other. But there's a catch! Joining every photo in a computer is wasteful in terms of memory and doesn't avoid data loss. To solve this, we need to develop a Distributed Photo Organizer.

A peer-to-peer (P2P) network is a group of computers that are linked together with equal permissions and responsibilities for processing data. Unlike traditional client-server networking, no devices in a P2P network are designated solely to serve or to receive data. In this report, we are going to report our P2P design for the Distributed Photo Organizer.

2 Methodoly

In this section of the report, we will detail the methodoly used to complete the goals of the project.

2.1 Peer-To-Peer Architecture

Our peer to peer network is unstructured meaning that we do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other. Our peer network has an anchor node where the nodes connect when they first enter the network. This anchor node also attributes a backup node for every peer that is in the network.

2.1.1 Protocol

In this project we used the Transmission Control Protocol (TCP/IP) as the standard for our messages to be sent in the network. We used this protocol because it provides error checking which makes fault tolerance easier (we will talk more about it briefly). We also used selectors which provides methods for specifying what events to look for on a socket, and then lets the caller wait for events in a platform-independent way. Registering interest in an event creates a SelectorKey , which holds the socket, information about the events of interest, and optional application data.

2.2 Image Sharing

2.2.1 Identifying the image

In order to identify each image we pass it to a hash function. The hash is guaranteed to be unique since it is also used to search for similar images, and when two are found to be too similar one of them is removed from the directory, eliminating the repeated hash.

2.2.2 Getting an Image

For a client to get an image he should run the command "getimage (hash)" providing the desired image's hash. The deamon he is associated to will then search its directory for the image and if it's not there it will send a request to all its peers asking them to search in their directories. When one of them finds it, it will send back a response with the image, so that the first deamon can give it to the client.

2.2.3 Getting all images list

The client can get the list of images names and respective hashes by running the command "getimageslist" on any node he associates with. The node will get a list of its images and ask the other nodes for their lists which he will then join to his.

2.3 Fault Tolerance

As we talked before, every peer has a backup node. Every node sends a copy of its messages to his backup node. Once a peer is killed, every peer receives an event of his death. If the death node is not the anchor node, the other peers will continue to run smoothly. If the dead node is the anchor node, we need to make an election to elect the new anchor node. At first, we thought of using the Bully Algorithm but because every node receives an event with the dead node, we then modified our idea and in our project the node with the biggest ID is elected as the new node. The first node to receive the event of the death anchor node will calculate the new anchor node and will send a message to everyone.

2.4 Storage Efficiency

2.4.1 Load Balancing

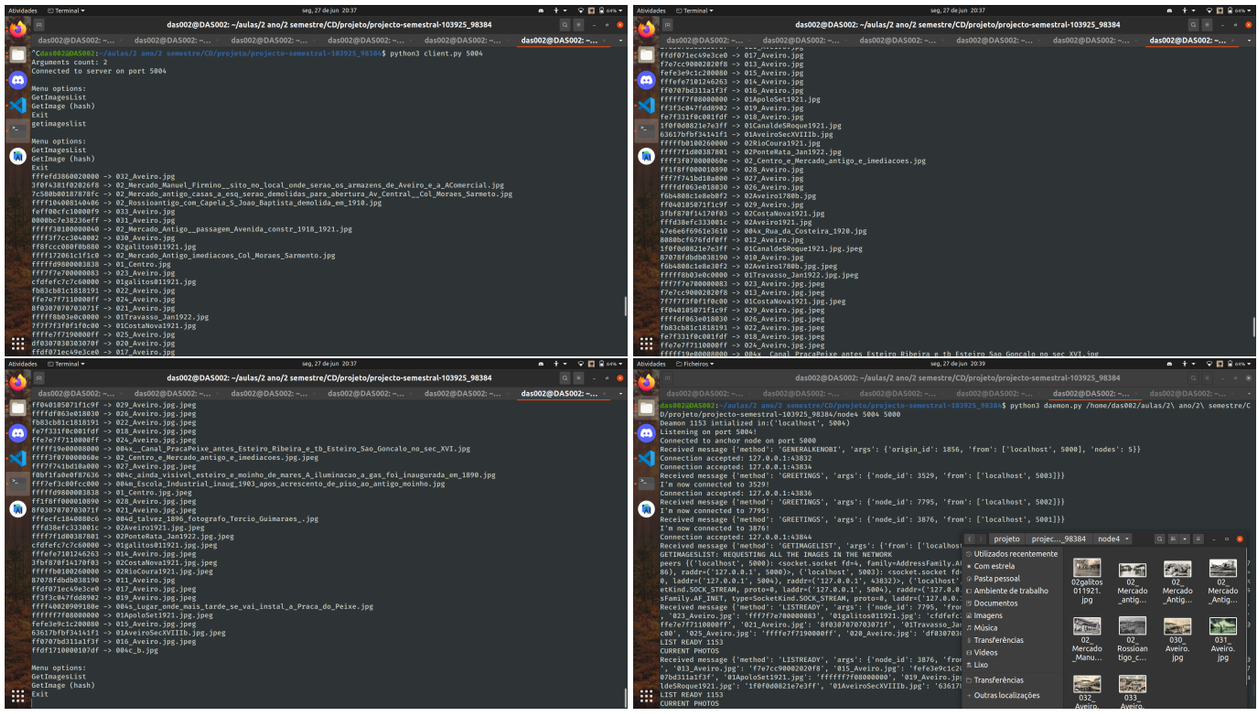
Regarding the backup process, it would be ill advised to rely on a few nodes to backup the majority of the information when others are available storing little information. With that in mind we developed a load balancing system which relies on each node needing a backup to ask the anchor node to choose the node with the smallest current size. To get that information, the nodes send their initial size in the first message. To accommodate for each time its size increases by doing a backup of another node, it sends a new message to the anchor warning him to update the information he has about its size.

2.4.2 Checking Similarity

3 Results

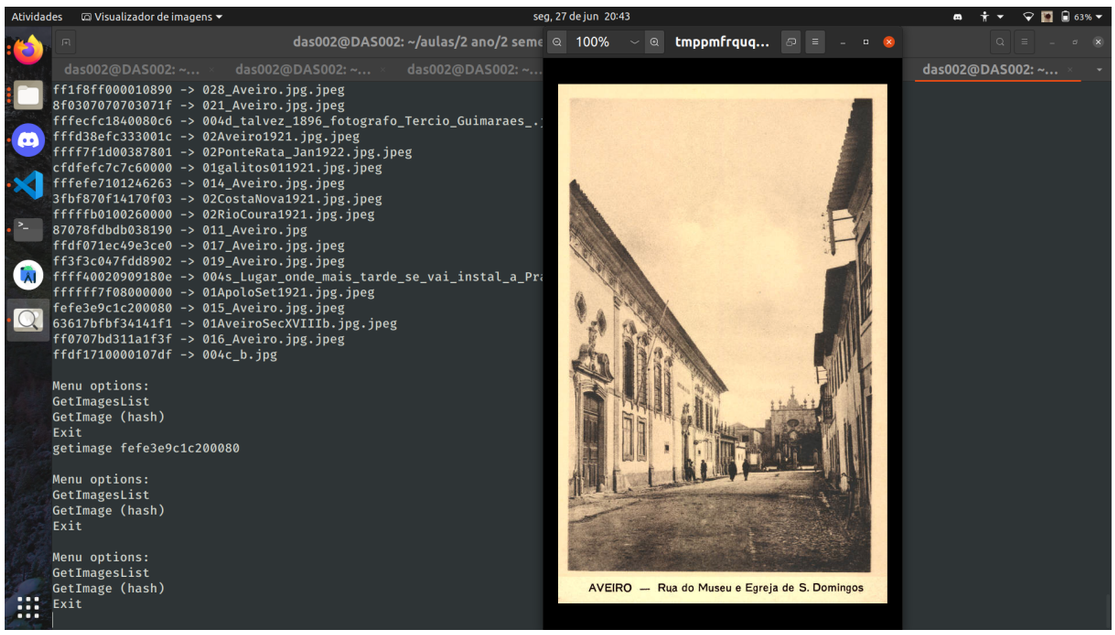
3.1 GetImagesList

The client can get all the images names and hashes, no matter which node he is connected to. In this example, the client is connected to the node in the port 5004, which solely has access to the folder "node4", however when the command "getimageslist" is run, the client is given the whole content of all folders.



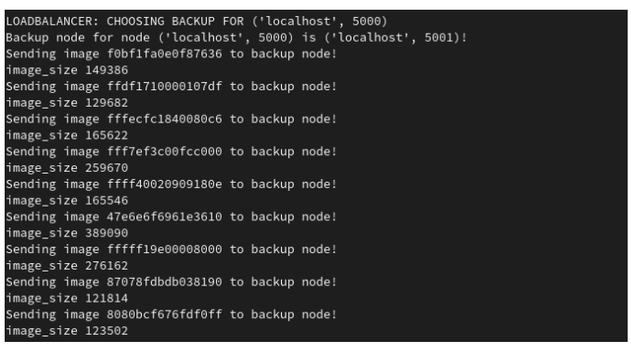
3.2 GetImage (hash)

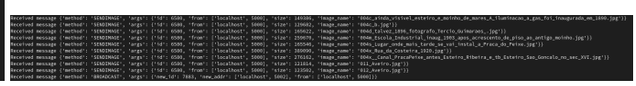
The client get any image, not mattering again which node he is connected to. In this example, the client is also connected to the node in port 5004 and asked for an image which is not present in the folder it has access to (we can check in the last image of the last example).



3.3 Fault Tolerance

In the images bellow, it can be seen the creation of a backup for the anchor node.





In the image bellow, it can be seen the behaviour after the anchor node dies. The node on port 5004 detects the peer death and chooses a new anchor node based on the highest ID which happens to be him.

4 Conclusion

This project helped understand better some of the concepts and technologies taught in the Distributed Systems course. Not everything went well as we realized that maybe the UDP protocol would have been better for these set of particular goals. UDP offers the possibility of multicast sockets and that would have helped the broadcasting of new nodes in the network. We also would like to have developed a more reliable fault tolerance and backup system. Another thing we would have liked to do was to try not having a super peer meaning that new nodes would be able to connect to every peer in the network. However the project was fun to make and we learned a lot!