Distributed System and Middleware

Patterns and Frameworks

Final Project

KlavoGonki

Shamil Khastiev
Bekzhan Talgat

Content:

Abstract

Have you seen funny videos about senior and junior coders? Where a senior writes code very fast and confidently without even looking at the keyboard and after everything in his/her code works just fine, meanwhile junior is as slow as a turtle writes with his/her two pointing fingers and in the end the computer sets on fire because of the error produced by the junior's code. So to improve your typing skill to a professional level we have implemented the KlavoGonki (Type Racing). This is a game where you can take a part in a competition with other players online to challenge who is the fastest typer. If not fast enough, we give you an ability for training offline. If you are ready then you can create your own room or connect to the existing room.

Introduction

To implement our project we need to implement peer-to-peer communication.

Problems that were awaiting us:

1) Data consistency
2) Latency among all peers
3) Large peer amount
4) Synchronous game start

Data consistency - as our project has peer to peer architecture it is important that all the peers have the same necessary and important information

Latency among all peers - if amount of peers increases, for one peer to share the data to all other peers might take a longer time

Large peer amount - as we are using free server using message brokers, free tire server does not allow us to send as many messages as we want

Synchronous game start - again if amount of players increases, it is harder to keep synchronous game start

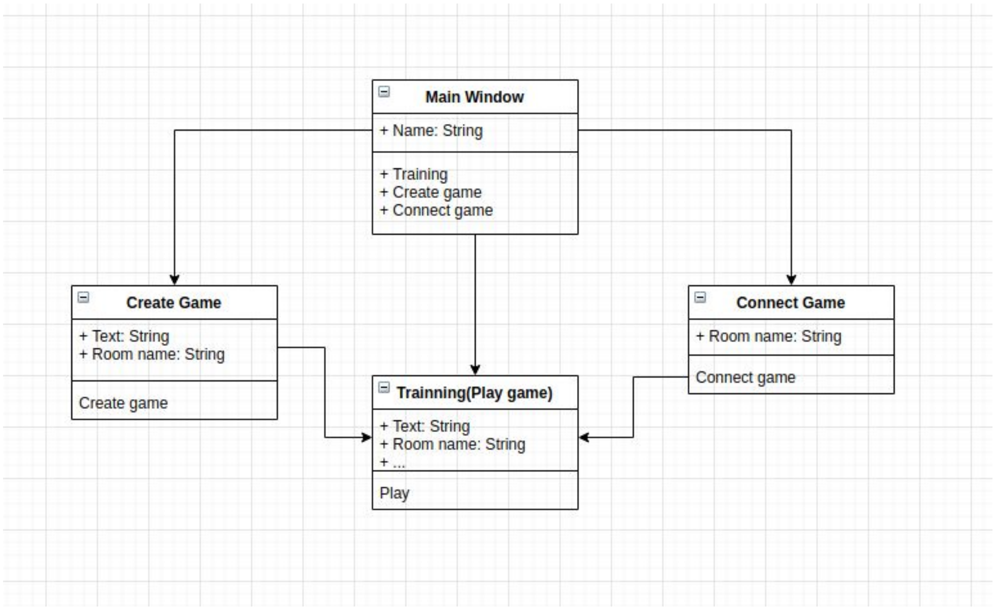
Our goals for the project:

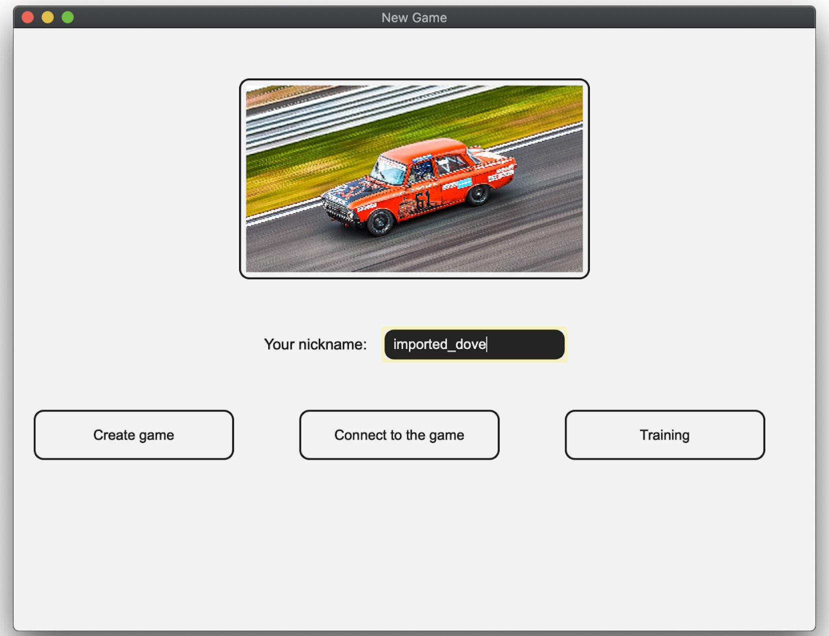
1) Comfortable GUI
2) Training session
3) Ability to create game
4) Ability to join the game
5) Synchronous game
6) Stable peer communication

Main features

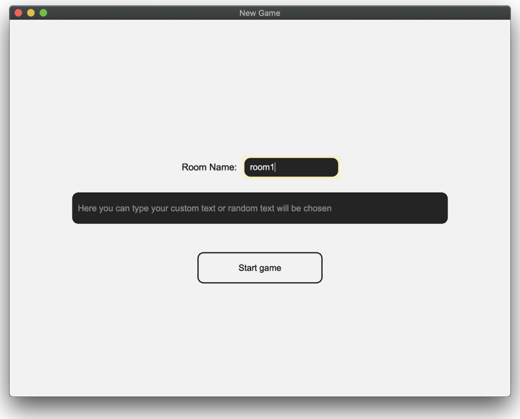
Overview:

1) Main window
    a) Create game
    b) Connect to the game
    c) Training



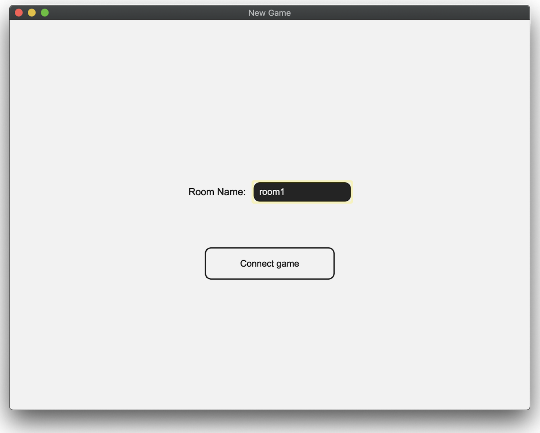

New Game

Your nickname: imported_dove

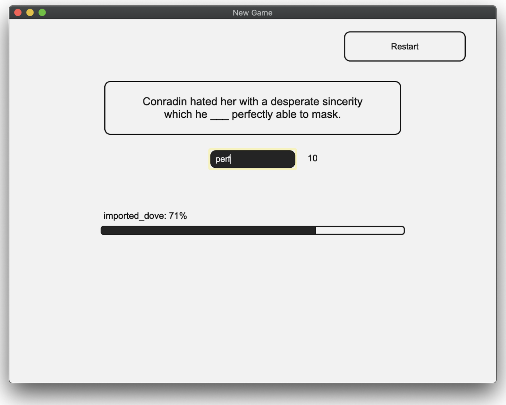

New Game

Room Name: room1

Here you can type your custom text or random text will be chosen


New Game

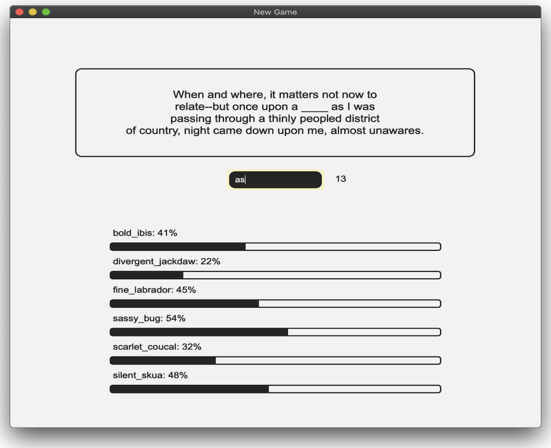
Room Name: room1


New Game

Conradin hated her with a desperate sincerity
which he ___ perfectly able to mask.

perf 10

imported_dove: 71%


New Game

When and where, it matters not now to
relate--but once upon a ____ as I was
passing through a thinly peopled district
of country, night came down upon me, almost unawares.

as 13

bold_ibis: 41%
divergent_jackdaw: 22%
fine_labrador: 45%
sassy_bug: 54%
scarlet_coucal: 32%
silent_skua: 48%

In the game there are several texts. If you create a game, there is the ability to write your own game text and that text will be shared with other players. If you do not write your custom text, then one of the in game text will be chosen randomly. Every player will have the same text. Also, when you create it is necessary to write game room name, in order to other people could join you by room name

If you want to connect to the game, you need to write a game room name. Then, the room creator shares the text and starts the game.

Moreover, if you want to make a training, you can go to the training and train offline with one of the in game text. Also, you can make a restart.

Main problems:
- Set up communication between players, bypass NAT.
- Sync the start of the game

Solutions:
- Using single RabbitMQ message broker to connect all peers
- Right celery tasks helps to sync the game start with latency less than 1 second

Abstract technologies
Each peer has a database. Only fundamental datas are shared among peers. Middleware is used to share data among peers. Finally each peer is processing data locally to display it. Diagram will be shown in the following section

Middleware:
Python Celery is used as a middleware. It is an open source asynchronous task queue or job queue which is based on distributed message passing. While it supports scheduling, its focus is on operations in real time. The execution units, called *tasks*, are executed concurrently on one or more worker nodes using multiprocessing, eventlet or gevent. Tasks can execute asynchronously (in the

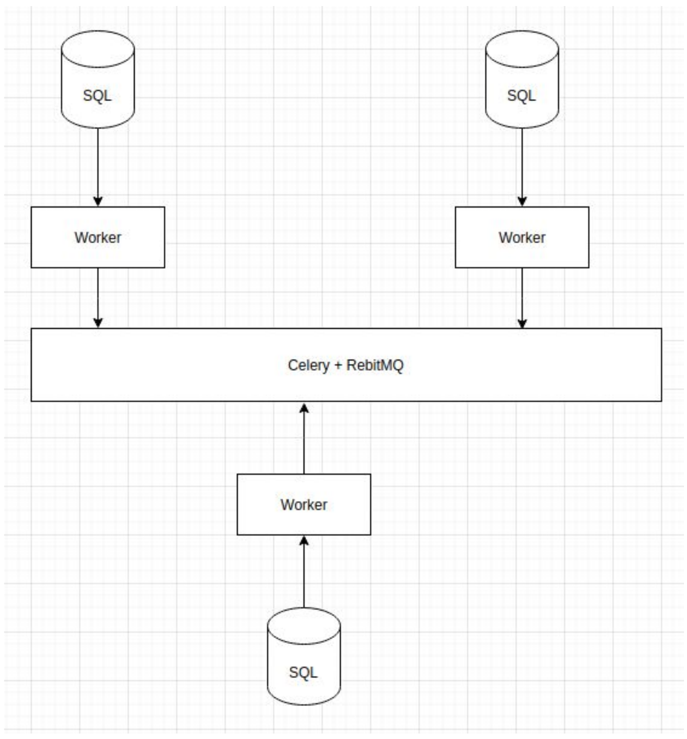
background) or synchronously (wait until ready). Celery is used in production systems, for instance Instagram, to process millions of tasks every day

Implementation

In this section we will discuss the technologies used in implementation and architecture decisions.

a) Used technologies:
   - Celery - is an open source asynchronous task queue or job queue which is based on distributed message passing. We used celery for our peer(workers communication)
   - RabbitMQ - message queue for celery
   - SQLite - database for each peer. We have chosen SQLite not some more powerful SQL like PostgreSQL because SQLite is not needed to be configured before using
   - QT5 - our GUI
   - Python - language that we used to make our app

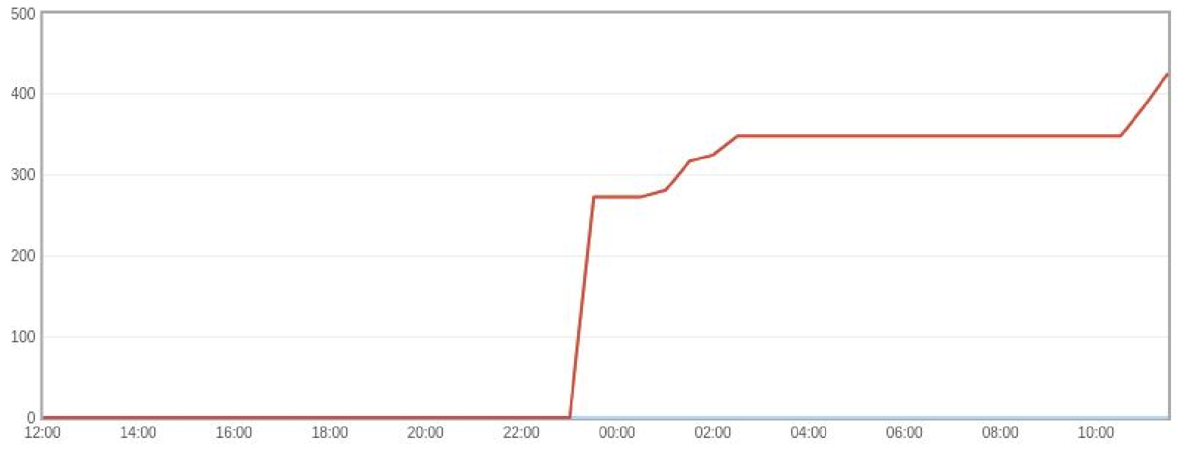
b) There is our UML diagram:

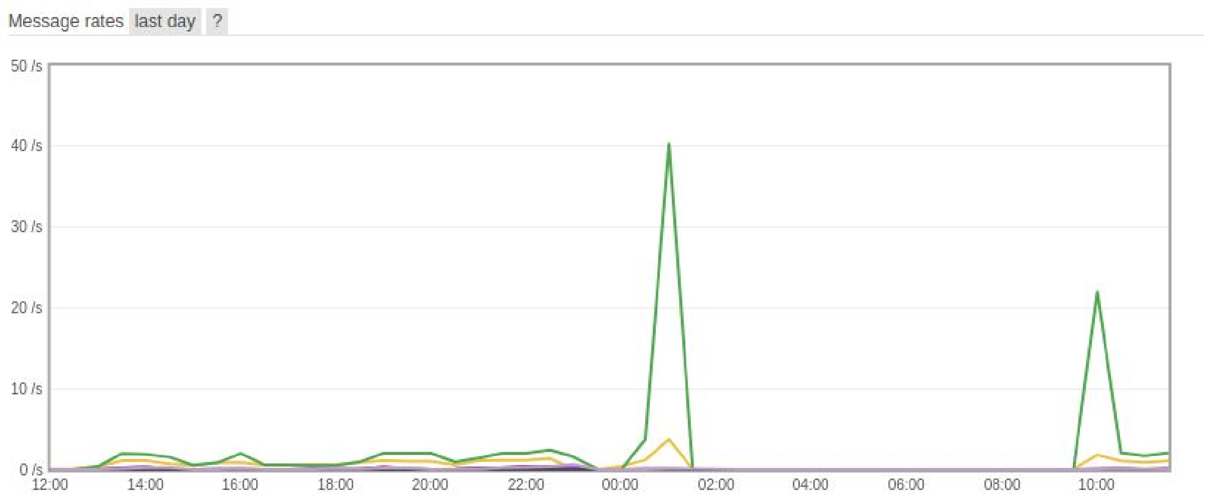


Each peer uses celery to communicate with each other. The messaging goes from RabbitMQ server. Worker calculates his progress in the race, and average speed and using celery broadcast sends this information to the other peers(workers). Workers write this information to the local SQLite database.

Evaluation

We hosted our messaging broker - RebitMQ, on the free server cloudamqp.com which provides 1000000(one million) massages per month and 20 connections at one time. Connecting the celery worker to the massage broker takes 4 connections so at one time only 5 players can play. The number can be increased but to do that we need to have a message broker on the better host. We had tested the performance of our app on the highest possible number of players. Results of the throughput are the following:





Massage rate growed up to 40 m/s average massaging is near 500-600 massages per minute. Our server can hold these values without any problems. Also the performance of data transmission does not really affect the performance of the app. Any delay(our average is 0.006 sec) will affect only delay in displaying the position of the players in the progress bar, that is not the main purpose of the game.

Conclusion

We have developed a distributed system keyboard racing game that has quite a small latency and uses celery + RabbitMQ as middleware. This game is called to help people improve their keyboard typing speed and compete with friends.