Automated offensive chat detection in multi-player games

Tianyu Deng
tianyud@kth.se

Minhao Ni
minhaon@kth.se

1 Introduction

Multi-player games have been a thriving industry for years, with millions of players participating in the exciting game plays. The gaming environment offers an unparalleled opportunity for social interaction, where players can foster friendship with their teammates and compete with rivalries. However, as the player base expands, so does the challenge of maintaining a positive gaming experience for players. One significant concern is the proliferation of inappropriate chat content inside the game. Offensive languages or even hate speech can sometimes occur. Such situation not only affects gaming experience but also poses a threat to the emotional well-being of players. To combat offensive chats, we designed and implemented an automated offensive detection system for multi-player games.This report will briefly demonstrate the motivation, design and actual implementation of our system.

2 Motivation

As mentioned above, offensive chat contents can severely affect the gaming experience of players. To combat this, most game companies employ a reporting system that users can use to report other players for inappropriate speech. However, this only happens after the game ends. During game play, other players would still be affected. As a result, we decide to develop a system support not only the reporting system but also real-time detection of offensive contents.

We believe this is a good project topic, since user's speech data need to be stored for a long period and the scale of data keeps growing. It's suitable to use HDFS to store and query these chat messages.

3 Design

Due to the difficulty for actually developing a game, the multi-player game is simulated by an online chat room application where users can post their chats to a public console. There are some core features that we want our system to have:

- There should be a distributed database to store user information
- Permanent storage of all user chat logs
- Store user chat logs in the current session efficiently, so that user's reporting can be dealt with quickly
- The data flow should be decoupled from back-end
- The users who post 'zero-tolerance' content should be immediately banned from chatting
- If one user is reported, the user's chat log in the last chat room session will should be inspected to decide whether the user actually sent offensive content
- If the user actually sent offensive chat, he will be reminded by the system

4 Implementation

Here is a graphical overview of our system:

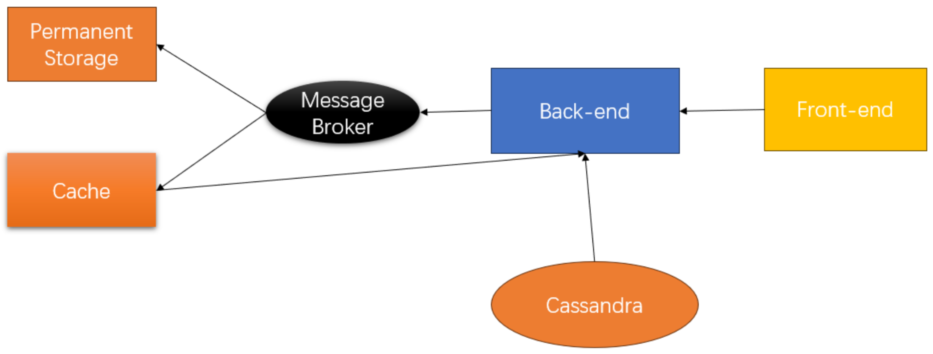

Figure 1: System Overview

The arrows show the data flow. Here are the tools or frameworks we used for our project:

- **Front-end:** HTML, CSS, JavaScript
- **Back-end:** Flask, Flask-socketio
- **Distributed database:** Cassandra
- **Message Broker:** Kafka
- **Permanent storage:** HDFS
- **Cache storage:** Redis

Web socket technique is necessary for implementing real-time online chat, so we utilized flask-socketio which pairs well with flask. Cassandra is used as the database to store user's account information. Kafka is used as the message broker to ingest and output the stream chat logs. By starting a Kafka consumer, the data flow is decoupled from back-end server. The kafka producer is responsible for receiving user's chat message and putting it into message queue. The kafka consumer is responsible sending the messages to both HDFS and Redis. For HDFS, it appends to the chat log file when receiving new chat logs from Kafka. Redis is used as an in-memory database to store user chat logs in the current session. With Redis, it is extremely fast to extract messages back to back-end for the reporting logic. All the chat messages are encapsulated into json format, which consists of three keys: username, timestamp and message.

5 How to Run

1. Install HDFS, Cassandra, Kafka and Redis properly
2. Create the HDFS direcotry "/chatlogs"
3. Create the kafka topic "chatlogs"
4. Install the python libraries using "pip install -r requirements.txt"
5. Run _init_.py
6. Run start_kafka.py in terminal
7. Run app.py in terminal
8. Visit webpage in different browsers, here are five default account:

Username	Password
Jack	123456
Tom	123456
Aaron	123456
Lucas	123456
Anna	123456

6 Conclusion and Future Development

In this project, we utilized Python Flask, Kafka, HDFS and Redis to build a automated offensive language detection system. Although objective is achieved successfully, there are some future works we can do to enhance the performance of our system:

- When handling user reports, an NLP model can be utilized to determine if there exists offensive language
- The 'Time to Live' value of redis caches can be set in a more accurate way, so that it only contains chat logs in the current chat session. In current implementation, it is set to a fixed value.