League of Legends Analytic & Recommendation Platform

Lee, Jooseok
University of Colorado Boulder
Jooseok.Lee@Colorado.edu

Lee, Seungwook
University of Colorado Boulder
Seungwook.Lee@Colorado.edu

Park, Chanheum
University of Colorado Boulder
Chanheum.Park@Colorado.edu

1. Motivation

League of Legends is one of the most played games worldwide with exceeding 180 million monthly players in 2022. [4] With our home country winning the recent worlds series and our past experience in enjoying playing League of Legends, we have put our heads together to create a service that can assist players in improving their gameplay. Performance analysis is a well-established discipline in sports science supported by decades of research, comparatively, performance analysis in e-sports is limited. [3] Therefore, there is an opportunity to improve performance outcomes in esports by providing tools for analysis and applying methods grounded in sports science which motivates us to create an extension to the platform OP.GG [2] which is a platform that allows players to view not only their in-game statistics but others as well. However, it does not provide a recommendation system of other players with similar statistics but a higher win rate which motivates us to create one.

2. Project Goal

We aim to build an extension to the platform OP.GG is a platform that allows League of Legends users to view their in-game statistics and view others. On top of its functionality, we build an additional module that recommends other players with similar statistics holding a higher win rate than the requested user. This information will allow the user to learn from similar play-style users with a higher win ratio leading the user to gradually increase their tier. We will be providing a web service where users will input their user ID and Region information. User queries will be stored in databases which we utilize to further enhance the performance of our system.

- Create an analytic web-service when users input their user ID and Region it recommends other players with similar in-game statistics holding a higher win rate than the requested user.
- Achieve multi-region support capable of handling multiple requests with scalability along with safety measures.

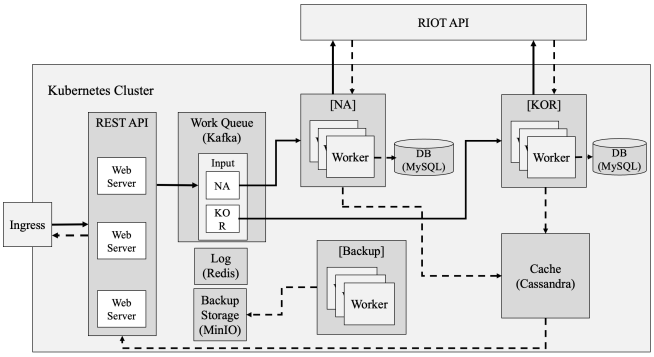

Figure 1. Architectural Diagram

3. Architectural Components

Google Kubernetes Engine will be used forming a cluster along with the following software components:

- Hardware Components
  - Google Kubernetes Engine
- Software Components
  - Kubernetes: Container
  - Flask RESTful API, Riot API
  - MySQL: Player info Database
  - Kafka: Message Queues
  - Cassandra: Caching
  - Redis: Logging & Debugging
  - MinIO: Back-up Storage

For our software components, we will be using Flask RESTful API which users will directly interact with. Flask RESTful is simple and provides a clean interface allowing parsing with ease. We also create a message queue using Kafka and use MySQL to store the retrieved data from the Riot API server due to its simplicity in deploying in different regions. Kafka is an exceptional choice as it provides us with a distributed pub-sub system created as a scalable, fast, and durable solution with low latency. Output results will be cached using Cassandra to quickly handle the same requests made on the same day. Cassandra's integrated caching method works toward scaling our service. For debugging purposes, Redis's exceptional logging method is used for debugging purposes. Minio is also used as a backup storage in case of a point of failure in our MySQL database.

4. Architecture

Figure 1 is an architectural diagram showing the interactions between the system components. Users interact with our REST API web server where the user inputs their ID/Region information that will be queued into our Kafka database. Depending on the user's Region(KOR: Korea / NA: North America), the data will be sent to the corresponding workers where KOR-workers handle the request of users in the Korea server and NA-workers handle the request from the NA users. Workers then retrieve data from the Riot Games API such as gold per minute, damage per minute, death, assist, and the number of kills. Figure 2 shows an example of the data stored in MySQL. To the left, there is another column of the user ID.

goldPerMinute	damagePerMinute	deaths	kills	assists
670.485	1583.892	7.900	14.900	12.150
444.175	725.074	7.800	7.050	10.800
458.505	626.753	4.700	5.800	6.500
744.308	1406.148	9.050	10.800	21.000

Figure 2. Example Data Stored in MySQL

4.1. Interaction Between Components:

- Ingress is set up against the Flask REST service.
- Users interact with the Flask web server by providing their Region/ID information.
- We created a work queue with Kafka to handle requests by region allowing our service to be more efficient and able to scale. Here, we have two workers corresponding to each region (Korea and North America). Before queuing the work into Kafka Rest server checks whether the result of the request is cached in Cassandra first. If there is a cache, the rest server returns the result to the users. More details will be explained in later steps.
- Workers then request the player's data from Riot Games API and store the retrieved data in MySQL. Using this data, our workers output a similarity by calculating the Euclidian distance upon features such as DPS, gold per minute, KDA, etc… as shown above in Figure 2
- The similarity output is cached in Cassandra 3 to account for the same request made on the same day. Similarity results vary by time as more user data is stored in our MySQL database. when the same request is made on the same date, it retrieves information from our cached data in Cassandra and is sent to the requesting user. Otherwise, the request is queued and runs the cycle. So when a request is made, it initially checks our cached data.

user	today	recommend
IgNar	2022-12-11	Doublelift
IgNar	2022-12-12	Doublelift
G1DEON	2022-12-12	G1DEON
Revenge	2022-12-12	IgNar
winstxn	2022-12-12	Doublelift

Figure 3. Cache Stored in Cassandra

4.2. Back-up: MinIO

In case of a single point of failure in our database: MySQL, we use MinIO to keep as backup storage of player data. This process is periodically done by the Backup worker in Figure 1.

5. Debugging and Monitoring

Initial development was done by dividing the work into smaller components and tested through port forwarding. As for Databases, we would look into the data retrieved from Riot Games API in MySQL and MinIO. Once we were sure that each part was working as expected, we then moved on to merge into Google Kubernetes Engine forming a cluster. Redis logging was used to debug this process as many components came together, it became harder for us to traceback and pinpoint the failing components. Using the log, it became clear to see that our caching mechanism using Cassandra worked properly, and the work message queue in Kafka was separating the work by region accordingly.
There are two major test cases that we must check:

- User requests the same information that is cached.
- User makes a request that is not cached.

For the first test case, if we make a request: User = Ignar, Region = NA on the same day as shown in Figure 3, then the response is immediate. We validate this process by checking the log. For the second test case, we make a new request: User = Kaor1, Region = NA The responses took 7.6 seconds and can vary by sometimes taking more than 30 seconds as the majority of this delay comes from the Riot Games API being our major bottleneck. We validate this using our Redis log information and also check to see if this response has been cached in Cassandra. We should also see another entry in our MySQL as well as in our MinIO backup storage as shown in our demo [1]

6. Discussion & Future Work

Our ambition towards this project has led us to a successful implementation. However, there were some points that we were not able to foresee in our proposal phase and had to learn as we go.

- Users must input Region information: Riot API requires this process.
- MySQL database update requires user requests to be updated as same for our MinIO backup.
- Riot API is the major bottleneck to our system.
- Depending on demand, the load on workers can be too much for our current architecture.

Our future work will be to solve the database update problem by deploying a separate worker responsible for this process as well as having a microservice architecture more scaleable in providing multi-region support and handling high demands/multiple requests. Also, as we were focused on the overall system but paid little attention to tackling how to provide the best matching user (similarity), there are many improvements that can be made in this process with many ongoing research in e-sports analysis.

References

[1] Demo Video league of legends analytics & recommendation platform. https://youtu.be/NsW3ivqf46k. Accessed: 2022-12-11. 3

[2] OP.GG league of legends platform. op.gg. Accessed: 2022-11-02. 1

[3] Andrew R Novak, Kyle JM Bennett, Matthew A Pluss, and Job Fransen. Performance analysis in esports: modelling performance at the 2018 league of legends world championship. *International Journal of Sports Science & Coaching*, 15(5-6):809–817, 2020. 1

[4] Rich Stanton. With 180 million players, league of legends games have more active users than steam. 1