A Scalable Dynamic Ad-Insertion System

CSC 462: Distributed Systems

Term Project Report

Dr. Yvonne Coady

Griffin Storback

V00849885

August 10th 2019

Repo:

http://github.com/…

Project Overview

The goal of this project is to create a scalable application for dynamic ad insertion. Dynamic ad insertion is a technology which allows advertisers to serve relevant ads to users based on their activity, personal information shared on the website, or other attributes known about the user [1]. It benefits both the advertiser and the consumer, as the advertiser spends less to get their message across to their target audience, and the user gets to see ads for products that may actually be relevant to their life.

The application consists of a single webpage; an example webpage, with some user information (either pulled randomly, or the user is specified in the URL), and an ad for the given user at the top of the screen (Figure 1). The rest of the page just describes the project, as it doesn’t really matter about the content of the website when we are only concerned with relevancy of ads. This page contains one ad at the top, but in many real webpages there are multiple places for ads.

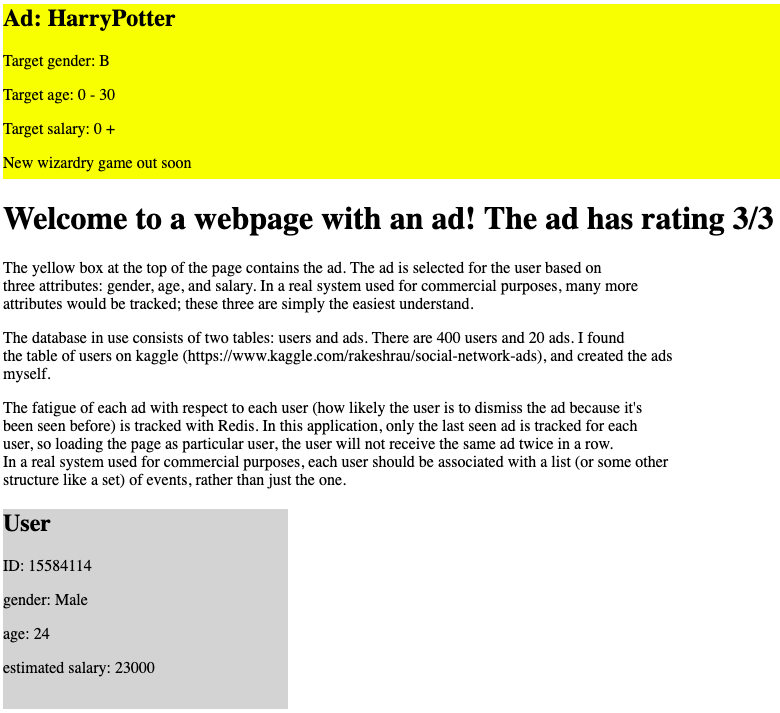


Figure 1: The application’s landing page

The application can be run in two ways: reload ad and reload user, or reload ad and retain user. To reload both the user and the displayed ad, simply load the webpage at path “/”. If the user should be retained and a new ad shown for them, load the webpage at path”/<userid>”, where <userid> is the unique integer associated with the user.

Design

The application consists of the following components: a Python Flask web server, a Postgres database, and Redis (Figure 2). When the Flask web server receives a request, it will first query Redis to find which ads the user has received recently, to prevent ad fatigue. Ad fatigue is the condition that arises when users have grown tired of seeing the same ad(s) over and over, and stop paying attention to them [2]. Using the results of the Redis query within the query to Postgres, a new relevant ad is found and returned.

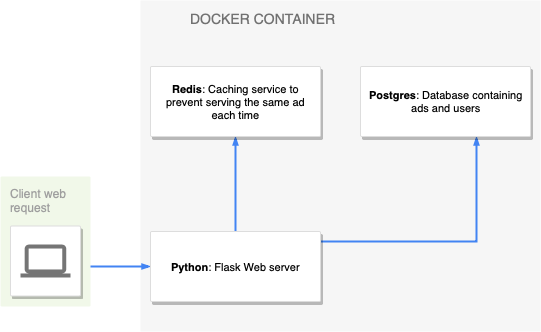


Figure 2: Single system architecture

When querying the database, first the application will try to find ads relevant to each attribute of the user. If no results are found, it tries with each attribute except one, and then keeps querying for less and less attributes as the queries keep coming back empty. This means that the less ads with relevant attributes are available for a user, the longer the query-time will be.

When the application is run in a distributed manner, the only change in system architecture is the advent of a load balancer (Figure 3). The load balancer would assign a request to one of the hosts, which would query its Potgres and Redis replicas. Since the system does not push new ads to the Postgres database, we don’t need to worry about out-of-date replicas. Posting to Redis is done after receiving a new ad, and will only affect the key associated with the user, so Redis data shouldn’t be out-of-date; even if it is out-of-date once in awhile it shouldn’t be concerning, as the worst case scenario involves the user getting an ad they had already seen, which can cause ad fatigue, but only if it were to happen frequently.

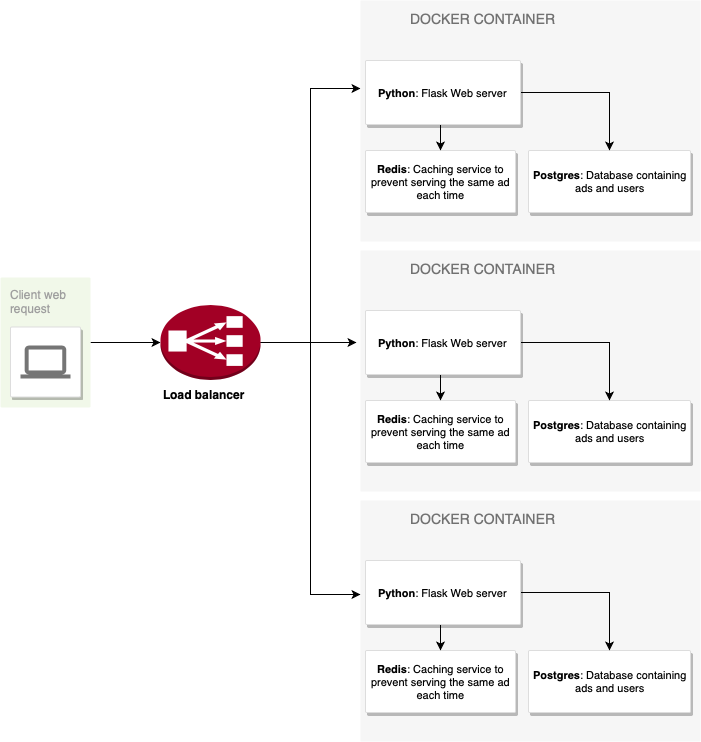


Figure 3: Distributed system architecture (with three nodes)

Implementation and Deployment

The frontend of the application consists of a single Jinja2 template (HTML), with input values for the user and ad, allowing for the server to return the webpage with the user and ad information sections filled in appropriately.

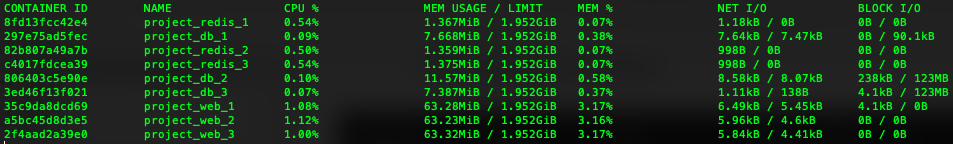
The backend consists of the Flask server, written in Python, which uses SQLAlchemy and Psycopg2 to converse with the Postgres database.

The three components, Flask, Redis and Postgres each run in their own Docker container, specified by the docker-compose file. The application scales up using docker compose, creating multiple instances of the three containers on the same host (my Macbook Air). Hindered by the lack of physical machines, I was unable to run the application in a distributed manner over multiple hosts; to do so would necessitate providing the docker-compose file and a configuration file to create a docker swarm.

Evaluation

Macintosh HD:private:var:folders:2d:t3ydfk7x6gg1_n_hyc6zt2t80000gn:T:TemporaryItems:Screen Shot 2019-08-10 at 12.59.52 PM.png

Figure 4: Docker stats when running a single instance.

Figure 5: Docker stats when running three instances of each service.

Conclusion

With more resources and time, the application could have been properly performance tested in a distributed setting with multiple physical hosts involved. With more physical hosts and more time, I would have tried using docker swarm to run each machine as a node in a network.

As it stands, I think the dynamic ad insertion problem is interesting; as we start to gather more information and build better user ad portfolios, queries will become increasingly complex. Complex queries combined with the fact that some webpages display multiple ads at a time (or, for a video, in a row), we’re left with a problem which can benefit greatly from being implemented in a distributed manner. Ad insertion systems can also be easily isolated as their own service, as usually ads on a webpage don’t have any relationship with the content.

Future Work

For this project, future work could consist of adding more attributes to each user/ad to make them more targeted and restructure the query logic so it serves the best possible ad in the best possible time.

Another feature that could be added is making Redis store a list of ads for each user, instead of just holding the last ad the user saw. By only storing one ad at a time, the user could still easily experience ad fatigue, as it’s possible they see the same couple ads over and over. Keeping a list would make queries more complicated, but overall would likely provide a better user experience.

References

[1] <https://www.spotx.tv/resources/blog/spotxer/what-is-dynamic-ad-insertion/>

[2] <https://www.thedrum.com/opinion/2017/04/04/ad-fatigue-the-silent-killer-ad-performance>