# A cloud-based personalized event ticket recommendation system

Xuanyi Gu, Tianshu Bao

***Abstract*—In this paper, we presented a cloud-based event recommendation system for event search based on geo-location. The model is able to predict and recommend events to users based on content category and user behaviors. Typical functions of the models includes search API, POST servlet, GET servlet and authentication APIs. Meanwhile, various recommendation algorithm can be explored in order to provide an ubiquitous suggestion for users. Also, the model is deployed on AWS EC2 virtual machine which can handle 150 queries simultaneously per second tested by Apache JMeter.**

***Index Terms*—Recommendation System, Cloud Computing, Web application, RESTful**

## I. Introduction

More and more online ordering system are popular recently and provide an friendly and courteous environment to user and extremely facilitate their life. As long as tourist are travelling to a new city, they are normally willing to visit local event and definitely need to order ticket. Sometimes, people are not fully informed which event will be hold in nearby area and therefore a recommendation is really necessary to assist the new comer especially they would like to spend the next couple of days in the city.

Popular recommendation systems are almost everywhere. The Youtube user can always find their favourite videos and movies due to the contribution of their system. Each video are tagged and categorized by their properties or customer preference. Also the system will record the type of video the user watched and suggested them the similar types.

In order to fulfill various type of requirements, several points has to be illustrated and solved. 1)Most ticket websites are just ticket listing websites, they does not recommend any interesting events or utilize user behavior based upon information. 2) A personalized recommendation system is needed due to multiple reasons. 3) The necessity of listing unpopular but valuable event for needed people. In conclusion, our mission is to solve these requirement and provide a simple environment and simulation for it. Details of implementation will be covered in the upcoming sessions and future work will also be discussed.

## II. Architecture

The basic architecture is shown in (1). The user browser is implemented using HTML, CSS and Javascript. The middle layer and backend is deployed in AWS EC2 virtual machine. Handler is realized by Java servlets and connected to ticket client. Our system would call TicketMaster APIs to get related events data. We used MySQL database to store the data and JDBC to control data flow and interaction. JSON is used to realized RESTFful API as a standard tool of web service.

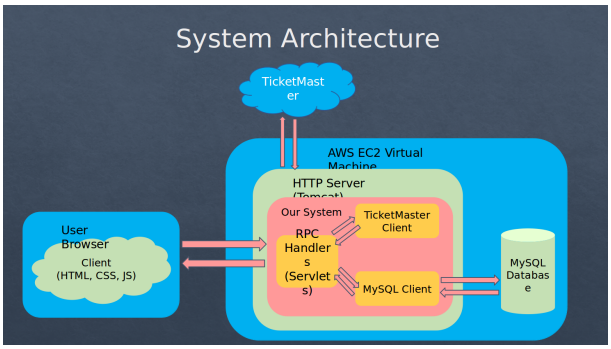


Figure 1: System Architecture

## III. Algorithm

There are three main-steam algorithms for recommendation system to use, which are content-based recommendation algorithm, item-based recommendation algorithm, and user-based recommendation algorithm.

For contented-based algorithm, the key point is people will like people or things of similar characteristics, so if given item profiles (category, price, etc.) of users favorite, system can recommend items that are similar to what user liked before. As the graph is shown below (2), item A and item B are belonging to R&B category, since user A likes item A, so the system recommends item C to user A. Because of limited data, this is the main algorithm we used in our system.

For user-based algorithm, system filters based on the similarity of Items. As the graph is shown below (Figure 3), Item A is liked by User A, User B, User C. Item B is liked by User B. Item C is liked by User A, User B. Thus, Item A and Item C are alike. Since Item C is liked by users who like Item A, so the system recommends it to user C (another user who likes Item A).

For user-based algorithm, system filters based on the similarity of Users. As the graph is shown below (Figure 4), User A likes Item A, Item C. User B likes Item B. User C likes Item A, Item C, Item D. Thus, User A shares similar preference as User C compared to User B. Since User C also likes Item D, so User A may like Item D.

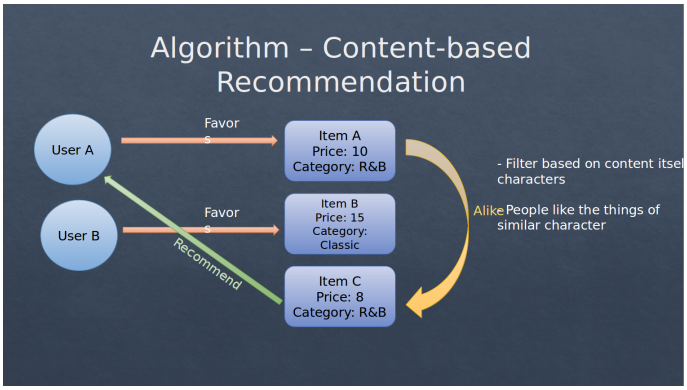


Figure 2: Content-based recommendation

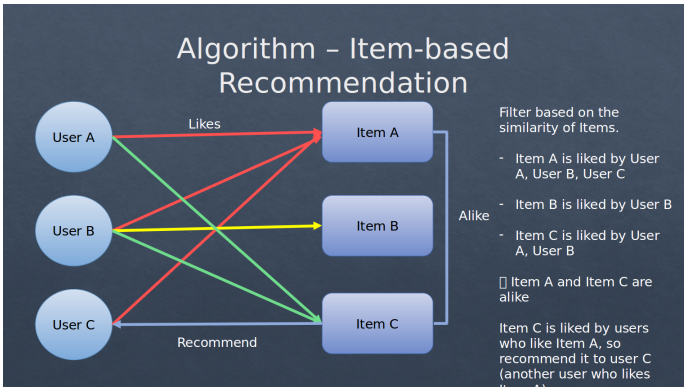


Figure 3: Item-based recommendation

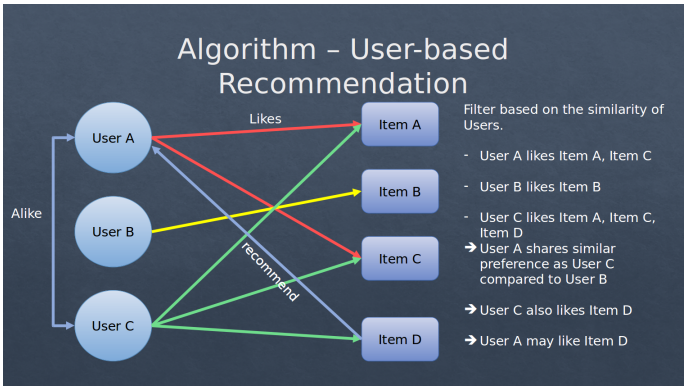


Figure 4: User-based recommendation

## IV. IMPLEMENTATION

### A. HTML

For implementing front-end pages, we first used HTML5 (Hypertext Markup Language) to define the structure and layout of the web pages by applying various tags and attributes. If we imagine the web site as a human, then HTML defines whole bone structure of the human.

For more details of our front-end web pages, as the graph is shown below (5), we divided our home pages into four main areas, there are header area, left menu area, right main div area and footer area.

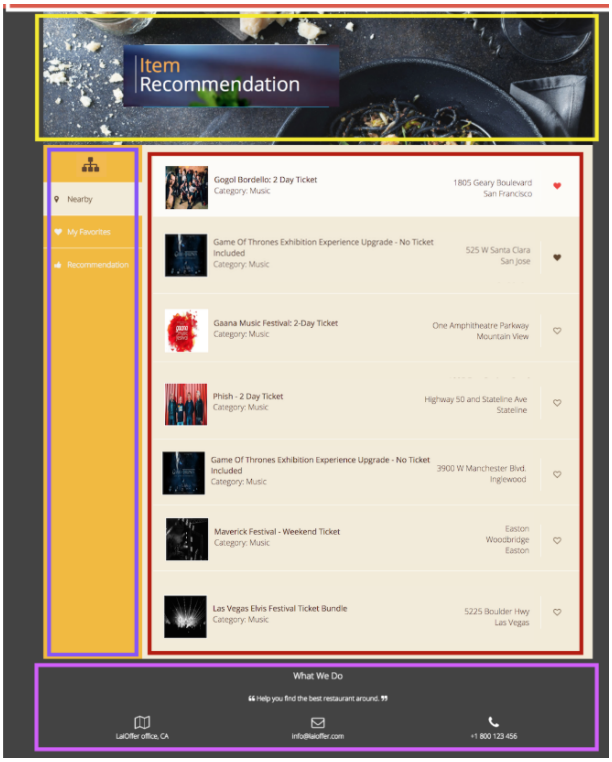


Figure 5: Home page divided areas

### B. CSS

In order to make our front-end web pages be vivid, we then used CSS3 (Cascading Style Sheets) to describe how the HTML elements (titles, images, paragraphs, and etc.) we created are to be displayed on screen. If we still take the human example, CSS delineates which clothes, pants, shoes and what kind of glass he/she wears.

### C. JavaScript

We finally used JavaScript to define the interactive behaviors of web pages. It is the third layer cake of standard web technologies, two of which are HTML and CSS that we have already introduced. If we continue to use the human example, JavaScript decides which actions he/she can do, such as sitting, jumping, running.

### D. Java Servlet

We used Java servlet to make our website to be dynamic. Java servlet is able to handle the request got from the web server, process the request, create the response, then send response back to the web server. In our project, we mainly used doGet method to handle GET request and doPost method to handle POST request. Take an example to explain this process, if our web server receives the POST request because user wants to record their favorite events, then our specific doPost method will receive the request, then process it, write the events into our back-end database, in the end, send the response back to the front-end.

Lets delve into HTTP request and response further, as the two graphs are shown below (6 and 7), HTTP request has a

request line which indicates its type, sources, and version; A request headers which contains its meta data; A request body which contains its content. For HTTP response, it has a similar construction with a status line, response headers, and response message body.

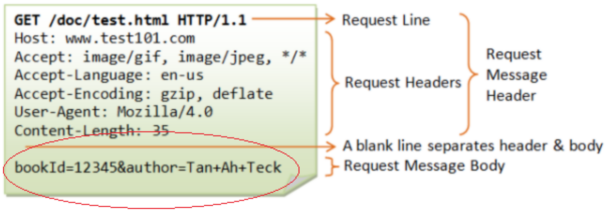


Figure 6: HTTP request

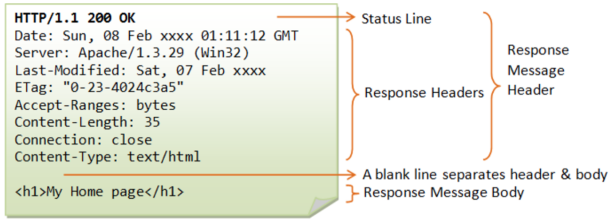


Figure 7: HTTP response

## E. MAMP

We created our database by using MAMP (My Apache, MySQL, and PHP) platform. MAMP is an integrated platform we can install on our machine that allows us to have access to a local MySQL server and its server. As the graph is shown below (Figure 8), our database related operations are saveItem, getFavoriteItem, setFavoriteItem, and unsetFavoriteItem.

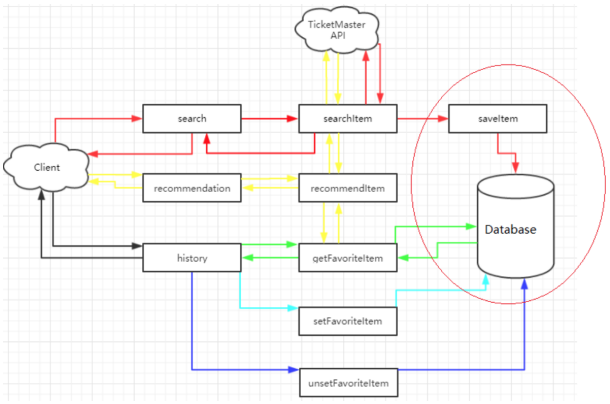


Figure 8: Database related operations

## F. TicketMaster API

TicketMaster API is a web-based API provided by TicketMaster so that clients are able to get real events data from TicketMaster server. We cannot see the source code of it, but we can refer to the documentation to figure out how to use the interface to make connection by sending request to its back-end servers.

As the graph is shown below (Figure 9), in our project, we first registered an account on TicketMaster so that we can get our own API key. Then we decided which data type we want to get from TicketMaster servers. According to our hierarchy designing, motivation, and most importantly, the main functions we constructed, we should get following data: item ID, name, rating, address, categories, image URL, and distance. Because TicketMaster may provide lots of data, some of which may not be useful for us, so we also purified the data to make sure we do not get redundant information.

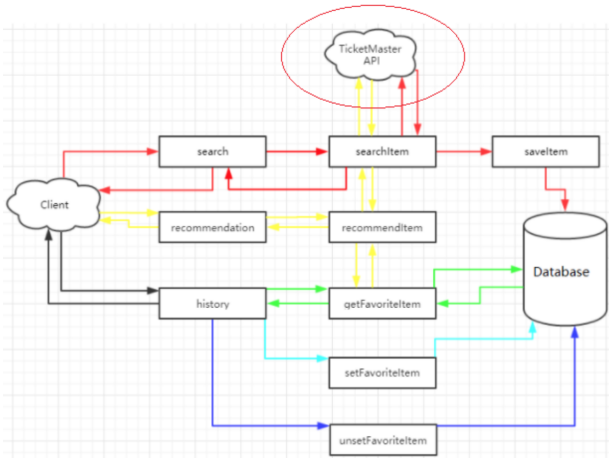


Figure 9: TicketMaster

## G. Authentication

Since we want to make our system to be customizable and safe, which means a user can only access data that are authorized to that user, and also system can record that users specific activity for book keeping, statistics, and etc. Therefore, we implemented Log-in and Log-out functions for our project. As the graph is shown below (Figure 10), once users are authenticated, each user use a session to maintain its status. When user logs out, the session is destroyed. Next time a user comes, he/she has to authenticate again to get a new session.

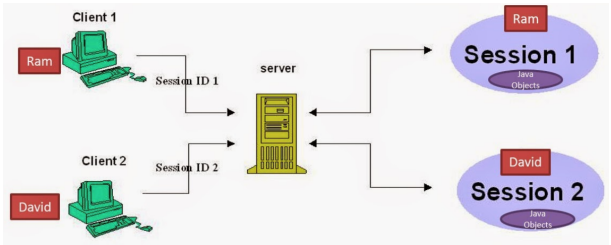


Figure 10: Authentication

## H. Cloud Computing

As we want to make our system to be scalability, flexibility, on-demand, and reduced labor cost and data center cost, we deployed our system on the AWS EC2 Virtual Machine. EC2 is a public cloud, free for students to use, and maintained by Amazon data center, so we choose EC2 as the cloud we deploy our system. For specific details, we chose Ubuntu 14.04

as image, installed Java, MySQL and Tomcat9 on there. Since Tomcat may be down sometimes, we set Tomcat automatically starts when Linux boots and restarts every day.

### I. RESTful API

RESTful API satisfies the following requirements: 1) Using HTTP methods to indicate what kind of operation a client want to take; 2) Using HTTP URL to indicate which service and data a client want to use and what kind of data they request; 3) Every request is separated, there is no support for doing one post request in several post requests, or doing a delete in a pair of get and post requests.

In our project, we used RESTful APIs because of the following reasons: 1) Operations are directly based on HTTP methods, so that server dont need to parse extra thing; 2) URL clearly indicates which resource a client wants, easy for client-side users to understand; 3) Server is running in stateless mode, improve scalability.

### J. Builder Pattern

Builder pattern has a nested static builder class to build the object step-by-step and provide a method that will actually return the object. Builder pattern is to deal with constructors that require too many parameters. Also, it can make sure the object is immutable. In our project, our Item class has many different attributes and thus we use builder pattern.

### K. Factory Pattern

Factory pattern is a creational pattern for creating objects without having to specify the exact class of the object that will be created. It allows us to easily support multiple implementations from the same interface. We used this pattern to build a DBconnection in our project such that it can build either MongoDB Connection or MySQL Connection on the fly.

## V. Demonstration

For demo, we can access our website by visit the following URL: http://18.188.151.48:8080/Jupiter/

On Authentication page, we can use 1111 as account, and 2222 as password to log in.

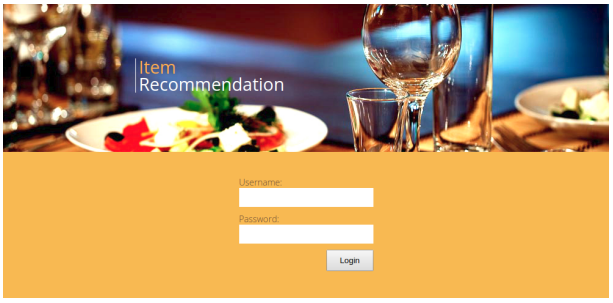


Figure 11: Login interface

## VI. Evaluation

We want to test the performance of our system, for web application, performance can usually be measured by QPS (requests per seconds). There are many tools can test QPS, we chose a widely-used tool Apache JMeter to measure our websites QPS.

As the graphs are shown below (Figure 12 and Figure 13), we first created Thread Group, set number of threads as 100, then we created a HTTP request test. We can see the peak throughput happens between 1000 and 2000 threads, so we can test more data points between 1000 and 2000 threads. The throughput will be around 150qps. When we used 2500 threads, we got an error, which means the memory size is our server's bottleneck.

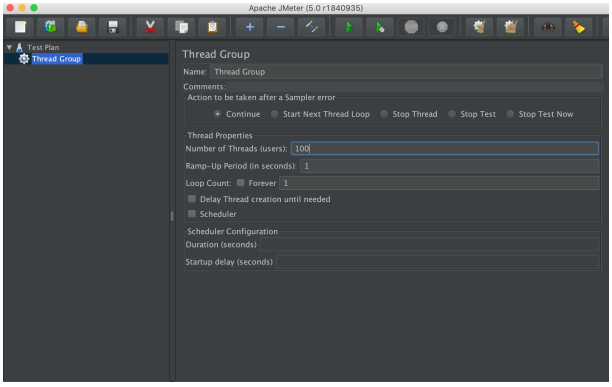


Figure 12: Apache JMeter

| # of Users (threads) | Avg. resp. time | Throughput (qps) |
|---|---|---|
| 100 | 738 | 84.7 |
| 500 | 2420 | 103.4 |
| 1000 | 3267 | 137.4 |
| 1500 | 4497 | 150.6 |
| 2000 | 6587 | 119.9 |

Figure 13: Test result

## VII. Teamwork

How did we cooperate to complete this final project? We first established a group contract to indicate each members working part and groups rules, so we can simply follow these rules to execute. And then we roughly made a schedule for better process control. Moreover, we Meet five times per week, lasts over five weeks. For each meeting time, we were not only executing our own part, we discuss and solve technical problems together.

What is each members working part? For designing part, basically, we designed this project together, includes business logic part and architecture part. For specific implementation section, Xuanyi did Front-end and Authentication section, Tianshu did Back-end and Integration section, and we did test together.

How much effort did we give for the project? Around 130 hours (5 hours for each meeting time, 5 times per week, lasts 5 weeks).

## VIII. Conclusion

In our final project, we built a cloud-based event recommendation system with MAMP framework for event search, which is able to predict and recommend events to users based on content category and user behavior. We implemented both front-end (HTML, CSS, and JavaScript) and back-end (MySQL) functionalities that meet the standard requirements of web services by using JSON format and APIs based on RESTful standard. Moreover, we realized search events API to search around, POST servlet that allows users to set their favorite records, GET servlet that recommends similar events to users, and authentication APIs to make the system customizable and accessible. Besides, we Deployed the system on AWS EC2 which can handle 150 queries per second as tested by Apache JMeter.

Due to limited data we got until now, we decide to adopt content-based recommendation algorithm in our system. In the future, we want to explore the other two recommendation approaches (used-based recommendation algorithm and item-based recommendation algorithm) to improve predict precision of our system.

## References


[1] Christian Esposito, Domenico Controneo, Aniruddha Gokhale and Douglas C. Schmidt, *Architectural Evolution of Monitor and Control Systems - Issues and Challenges*, Guest Editorial, Journal of Network Protocols and Algorithms (JNPA), vol. 2, no. 3, Oct 2010, pp. 117.

[2] Aniruddha Gokhale, Mark McDonald, Steven Drager, and William McKeever. *A Cyber Physical Systems Perspective on the Real-time and Reliable Dissemination of Information in Intelligent Transportation Systems* Journal of Network Protocols and Algorithms (JNPA), vol. 2, no. 3, Oct 2010, pp. 116136.

[3] Akshay Dabolkar and Aniruddha Gokhale. *FORMS: Feature-Oriented Reverse Engineering-based Middleware Specialization for Product-Lines* Journal of Software (JSW), vol 6., no. 4, April 2011, Academy Publisher, pp. 519527.

[4] Friedhelm Wolf, Jaiganesh Balasubramanian, Sumant Tambe, Aniruddha Gokhale and Douglas C. Schmidt. *Supporting Component-based Failover Units in Middleware for Distributed Real-time and Embedded Systems* Elsevier Journal of Software Architectures (JSA): Embedded Software Design, Special Issue on Real-time and Embedded Systems, Vol 57, No. 6, June 2011, pp. 597613.

[5] Joe Hoffert, Aniruddha Gokhale, and Douglas C. Schmidt. *Timely Autonomic Adaptation of Publish/Subscribe Middleware in Dynamic Environments*, International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), Vol. 2, No. 4, 2011, pp. 124.

[6] Joe Hoffert, Douglas C. Schmidt, and Aniruddha Gokhale. *Evaluating Timeliness and Accuracy Trade-offs of Supervised Machine Learning for Adapting Enterprise DRE Systems in Dynamic Environments*International Journal of Computational Intelligence Systems, vol. 4-5, Sep-Oct 2011, pp. 806816

[7] William Otte, Aniruddha Gokhale, and Douglas C. Schmidt. *Efficient and Deterministic Application Deployment in Component-based, Enterprise Distributed, Real-time, and Embedded Systems*, Elsevier Journal of Information and Software Technology (IST), Vol. 55, No. 2, pp. 475488, Feb 2013.

[8] S. Strub, Issam, and Alexandre M. Bayen. *Weak formulation of boundary conditions for scalar conservation laws: An application to highway traffic modelling*, International Journal of Robust and Nonlinear Control: IFACAffiliated Journal 16.16 (2006): 733-748.

[9] Subhav Pradhan, Aniruddha Gokhale, William Otte and Gabor Karsai. *Real-time Fault-tolerant Deployment and Configuration Framework for Cyber Physical Systems*, ACM SIGBED Review Special Issue on Work-in-Progress (WiP) session of the 33rd IEEE Real-Time Systems Symposium (RTSS 12), Vol. 10, No. 2, pp. 32, 2013.

[10] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, Douglas Schmidt, and Thierry Gayraud. *Supporting End-to-end Scalability and Real-time Event Dissemination in the OMG Data Distribution Service over Wide Area Networks*, Elsevier Journal of Systems Software (JSS), vol. 86, no. 10, 2013, pp. 25742593.