Recommendation System Middleware

Zhenyuan Zhang, Pansheng Fang, Dehao Li

1. Introduction

Nowadays, recommendation services are becoming increasingly important in people's life. There are multiple types of recommendation applications based on different content. For example, people might turn to restaurant recommendation applications when they are planning for a family dinner. They might also need to take a look at the top (recommended) choices when they want to buy a specific kind of product. There are also destination recommendation applications[9] and E-commerce recommendation applications[10]. A recommendation system middleware application can perfectly meet the requirements for providing recommendation information of all kinds of these scenarios.

As a recommendation system middleware can contain many modules providing different recommendation services for different items, we intend to bu ild a workable sub-module, a movie recommendation system(or API) as a showcase. Watching movies, however, is now a significant off-work activity for people. The movie/video recommendation system provided by YouTube is a good example[7]. But there are not many services that are recommending movies based on movies' information and viewing status, especially those adding in local and global features. [8] proposed a thought of using local and global features for recommending items so that providers can provide more customized services. Thus, we want to design a movie recommendation system as our project in an order to provide most viewed movies within a specific range of locations, so that people can get to know the most popular and heated movies that are around. Also, we implement the function of giving recommendations based on different labels, which can provide more customized services for different clients. In our project, we use Java to do the movie data statistics and recommending, Python for building a Web server, and Spark as middleware to calculate the recommendation results. More information will be introduced in the coming parts.

2. Related work

Nowadays when software engineers want to develop specific products, it is nearly a must to consider dealing with data within large scale. When executing tasks with such huge datasets, it is a trend to make use of distributed calculating platforms and tools such as MapReduce[1]. Basically, it consists of master nodes and worker nodes. Master nodes get the input data(called the Map operation) and the worker nodes are responsible for computing their assigned tasks. The master results will then combine the computing results, with data stored in distributed ways using Hadoop Distributed File System(HDFS)[2]. In recent years, Spark is widely used in distributed computing for its better performance in computing speed, high compatibility with distributed computing modules such as HDFS, and rich API for users to develop well-structured architectures[3]. Spark is a keyword search engine on relational databases, which perfectly solves insufficient flexibilities and capabilities problems.The technical challenge lies in how to find a general-purpose and effective ranking method for the search results while optimizing the search within a potentially huge search space. In [1], the problem is handled by proposing a novel ranking method that takes into several important ranking factors.

With their algorithms, the query processing speed could be up to two orders of magnitude faster than alternative methods. As a result, the Spark system consists of these kinds of speed-up systems.

Like other well-designed projects and architectures, our project, which aims to design a recommendation system middleware, utilizes Spark's good performance in big data analysis. We intend to build a movie recommendation system as a showcase, which needs to deal with a great deal of data to compute(might exceed the ability of a single machine). In the area of big data, Apache Spark is a fastest and general-purpose engine for large-scale data processing, which is achieved by In-memory computing. With this feature, Spark can query data much faster compared to Hadoop and can offer a general execution model that can optimize an arbitrary operator graph, which is supported by Spark[4]. The article also concludes that within the APIs provided by Spark, it can well support the following services: 1. Top ten words tweeted during the last specific period of time. 2. Top ten languages used to tweet during a specific period of time. 3. A list of tweeted items matching a given search keyword. Our project includes the modules that give ranking data of movies. Also, we need to calculate the labeled data periodically. Last but not least, we are using Java to construct each module and use Maven to manage the whole project, which is also perfectly supported by Spark and its cluster computing modules. So, we believe that Spark is a powerful tool that can enhance the system performance in our project.

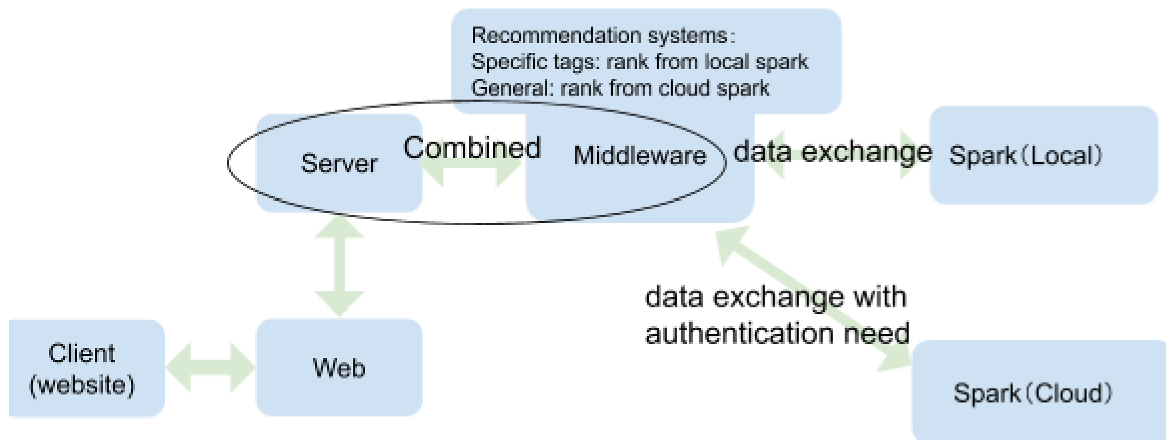
As we plan to deploy our project environment based on a cloud platform like AWS, we need to consider the management of Hadoop in the cloud platform. In paper[5], the author presents and develops an android-based mobile for Hadoop environment management. The structure described in the paper greatly inspires our project design. Compared to the paper, the middleware that we used in our recommendation system would not involve in Android related interactions(of course it could be designed and implemented but due to limited time we would not consider to make it, maybe in future work to complete). Also, we simply use the windows' website as the frontend things rather than the Android they use. In this case, we need to realize the interactions between website and server which makes us consider to use python as the core language to code the middleware. Besides, due to the special structure we designed which has two kinds of HDFS(local and cloud) that cannot directly interact with each other, the middleware part also needs to take responsibility to help the local HDFS and the cloud HDFS doing the interaction. Equally importantly, according to this paper, two proper SSH connections seems to be an effective way to handle the requirement.

The last but not least part in our project is the Hadoop Distributed File System(HDFS). It provides a self-implemented component for distributed data storage for developers[6]. [6] introduces the architecture and mechanism of HDFS in detail. Basically, it stores file system metadata and application data separately. HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols. In HDFS, the namespace is a hierarchy of files and directories and is kept in RAM. The NameNode maintains the namespace tree and the mapping of file blocks to DataNodes. Clients who want to access the files in HDFS can access the NameNode through a TCP-based connection, then get the data from DataNode

assigned by the system. Also, for clients(such as applications) who want to access the file system using HDFS, they can operate through a code library that exports the HDFS interface called HDFS client. In the paper, the high throughput of HDFS when dealing with large scale of data.

3. Approach, methods and technique

Basic architecture is shown below:



3.1 Using tags and distributed system to realize simplified version of personalized recommendation

We divided the file system into two parts, central system and edge systems. The central system was in charge of collecting the amount of related visits of things that were stored in the system(in our system it should be movies). And edge systems recorded the related visits of things that were divided by specific tags(in our system it was simplified to be the first char of the movies' title). When a pure new visitor(or new user if we expanded to have a register and login system) and request to see the list of the most popular things(in our system it was the played movie), the rank stored in the central system would be provided. But if the visitor has requested to see the thing with specific tags, this operation would be recorded and used to decide which group the visitor belongs to(similar to giving a mark or a tag to this visitor). At this time, when visitors request to see the list of the most popular things, the edge system(s) that are related to that group or tags would provide the rank information. The frames that show how the watch movie function(data record) work and how the rank system work are shown below.

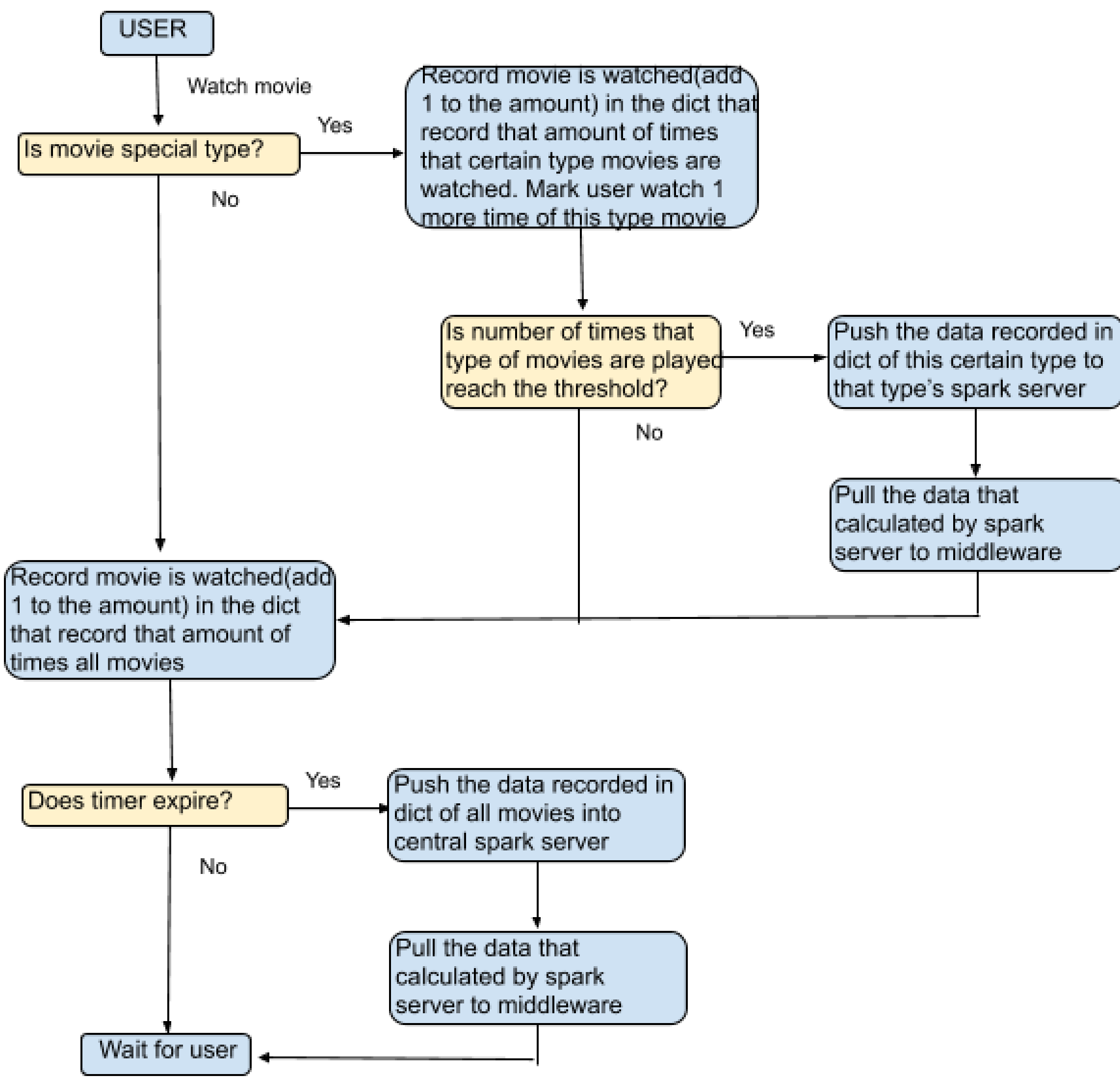


Figure 1 How watching movie function work

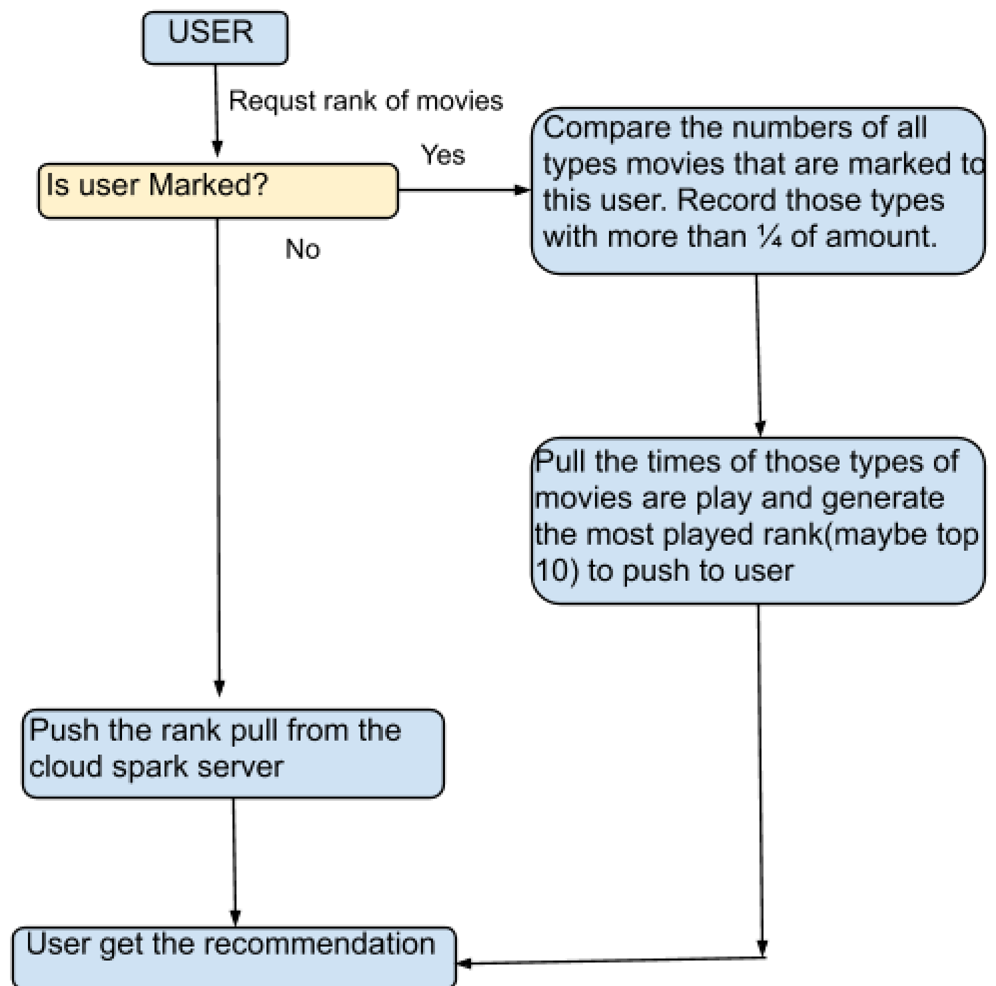


Figure 2 How ranking(recommendation) system work

3.2 Frequency count

Basically, we are doing a frequency count job. In the backend, we are gathering up the appearance of items that appear in the system into multiple files. These files are txt files, which should be located in the “input” directory. Then, we put these files into the Spark file system, and make the Spark do the frequency count job.

3.3 Spark on AWS

Using EMR, we can easily create a cluster with Spark installed and configured. Our cloud cluster consists of one master node and two worker nodes. For the hardware type, we select m5.xlarge option which is the basic hardware provided by AWS. This option makes sure that the hardware is not specifically optimized for storage performance or any specific type of computation.

We deploy our frequency count job on Spark as a jar file. Then, we use Spark on AWS to run this count job. It basically retrieves the result and writes it to Spark HDFS file system.

Cluster: My cluster12 Terminated Terminated by user request

Summary | Application user interfaces | Monitoring | Hardware | Configurations | Events | Steps | Bootstrap actions

Summary

ID: j-3NAMPERAMOLQI
Creation date: 2020-06-06 19:36 (UTC-7)
End date: 2020-06-06 22:11 (UTC-7)
Elapsed time: 2 hours, 34 minutes
After last step completes: Cluster waits
Termination protection: Off
Tags: --
Master public DNS: ec2-18-232-113-34.compute-1.amazonaws.com
Connect to the Master Node Using SSH

Configuration details

Release label: emr-5.30.0
Hadoop distribution: Amazon
Applications: Spark 2.4.5, Zeppelin 0.8.2
Log URI: s3://aws-logs-355041912036-us-east-1/elasticmapreduce/
EMRFS consistent view: Disabled
Custom AMI ID: --

Application user interfaces

Persistent user interfaces: Spark history server
On-cluster user interfaces: --

Network and hardware

Availability zone: us-east-1b
Subnet ID: subnet-122dcd4d
Master: Terminated 1 m5.xlarge
Core: Terminated 2 m5.xlarge
Task: --
Cluster scaling: Not enabled

Security and access

Key name: aws-pem-3
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-01dee7ec77d65b26c (ElasticMapReduce-master)
Security groups for Core & Task: sg-0358c68a8c4a97bef (ElasticMapReduce-slave)

3.4 Cloud's Input and output management

Basically we build our application using Java on our laptop. It is really important for us to communicate with Spark on AWS.

There are multiple things we need to do. First, we need to think about how to send input files to Spark on AWS. Second, AWS needs to be manipulated by our Java code once its Spark has received our input files. As a result, our Java code has the responsibility to send commands to AWS so that it can control Spark to do what we want. Third, we need to retrieve the result from AWS.

In general, we use JSCH, Java Secure Channel, to finish this part. JSch is a pure Java implementation of SSH2. JSch allows us to connect to an ssh server and use port forwarding, X11 forwarding, file transfer, etc., and we can integrate its functionality into our own Java programs. JSch is licensed under BSD style license.

The whole process of cloud operation is not difficult. In the beginning, it needs to build a session with certain credentials such as .pem files and the correct ip address. Then, using this session, the application uses SCP command to send the files to the input directory.

It is then our application sends several commands to manipulate the frequency count process in Spark. Firstly, the input directory is copied into Spark's HDFS file system so that all files that are needed to be processed can be seen by Spark. Secondly, by using the jar file we

deployed previously, Spark can do frequency analysis on all the files in the directory. Thirdly, the instance retrieves the output files from Spark's HDFS file system and merges them into a single file name "result". This single result file name "file" is ready to be fetched by our local machines.

Furthermore, we use the SCP command again to download the result file for further use.

3.5 Renew algorithm for central DFS and edge DFS

Since the whole system is not reading only(DFS need to collect the count of times of movies are played), the server(middleware) needs to decide when the data collected by the web server should be transferred to DFS and do the calculation(mapreduce) to get the new rank which then be transferred back to replace the old rank. We simply called this process the renewal of the rank of movies. For central DFS, we assumed it to have a huge amount of data which should not be renewed frequently, so we set it to be renewed periodically with comparatively long period(due to the property of the project and the comparatively small data volume we used, we set it to be five minutes). For edge DFS, we assume it to need to be renewed frequently so we record the total number of times of movies played, when this number reach the threshold we set(due to the property of the project and the comparatively small data volume we used, we set it to be 10) the edge DFS would do the renewal operation.

4. Evaluation plan and evaluation result

4.1 Evaluation plan

Since the main target is not to build a real useful system but help us understand some principles, we do not need to focus on the performance or abilities like to handle the high concurrency of the request or too much fault tolerance. We would just simply simulate two functions for visitors: (1)play specific movies(in a simulated way) (2)show the rank of the most played movies. And the evaluation would just base on these two functions by requesting to play specific tagged movies several times and play other movies several times. Besides, during the play of the movie, request to see the rank of the movies. After getting the renewal from the central DFS and the edge DFS, also request to see the rank of the movies.

4.2 Evaluation result

Doing the request to see the rank of movies before playing any movie, the system would successfully show the rank from central DFS as long as the first time map/reduce in central DFS is finished. After we played specific tagged movies, the request to see the rank of movies got the rank from edge DFS. After the times of playing the specific tagged movies reaching the threshold, the edge DFS collected the count of times of playing specific movies and successfully completed the calculation and returned the rank to the web server. Then we used the visitor marked as preferring the specific tagged movies to request to see the rank, and the new rank from edge DFS was shown correctly. And since we set the central DFS to collect the count of played movies and do the calculation periodically, after the specific periodical time which we set to be 5 mins, the central DFS successfully collected the data and completed the calculation. Then we used the untagged visitor to request the rank which had nothing wrong.

5 Conclusions and possible extensions

5.1 Conclusion

Using a distributed system makes it easier to realize data distribution and personalized recommendation. Spark is an easy use tool that can realize complicated multi-step map-reduce calculation with only one time read I\O, and this property makes it more practical in recommendation kinds of systems even could allow systems to be real time collecting and do real time recommendation with the collected data. We conduct experiments to simulate possible user behaviors and see whether the server can compute and output the recommendation results. We use a frontend web page to: (1)play specific movies(in a simulated way) (2)show the rank of the most played movies. The result shows that the system can successfully show the movie ranking and update the movie ranking properly.

5.2 Possible extensions

Since we simplified some structure related design and ignored some details to make the system more realizable in limited time. There are truly many extensions we could do to make the system more close to the real-life system.

5.2.1 More complicated multi-tags based system

Currently, we only based our recommended results on single, specific tags. In the future, we wish to extend our system into a multi-tagged one. In other words, we need to construct the system to categorize users with multiple tags, so that the system can provide more accurate services for users.

5.2.2 Real data and have ability to deal with high concurrency

As we are constructing the system in a rather simple way, there are still many aspects in the system that can be better implemented. To be more specific, considering high concurrent scenarios, we should design load balancing functionality to handle concurrent visits to the server. Normally, this can be achieved by setting multiple master nodes, dividing read-write functions in different servers and so on. Also, setting cache using tools like Redis can greatly enhance the ability of handling multiple requests.

Reference

[1]. Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: a flexible data processing tool." *Communications of the ACM* 53.1 (2010): 72-77.

[2]. Karun, A. K., & Chitharanjan, K. (2013, April). A review on hadoop—HDFS infrastructure extensions. In *2013 IEEE conference on information & communication technologies* (pp. 132-137). IEEE.

[3]. Yi Luo, Wei Wang, Xuemin Lin, “SPARK: A Keyword Search Engine on Relational Databases”.

[4]. Abdul Ghaffar Shoro, Tariq Rahim Soomro, “Big Data Analysis: Apache Spark Perspective”.

[5]. Feng-Qi Cheng, I-Ching Hsu*, “Hadoop environment management App based on mobile cloud computing”.
[6] Shvachko, Konstantin, et al. "The hadoop distributed file system." 2010 IEEE 26th symposium on mass storage systems and technologies (MSST) . Ieee, 2010.
[7]. Davidson, James, et al. "The YouTube video recommendation system." *Proceedings of the fourth ACM conference on Recommender systems*. 2010.
[8]. Maruyama, Takuma, Yoshiyuki Kawano, and Keiji Yanai. "Real-time mobile recipe recommendation system using food ingredient recognition." *Proceedings of the 2nd ACM international workshop on Interactive multimedia on mobile and portable devices*. 2012.
[9]. Fesenmaier, D. R., Wöber, K. W., & Werthner, H. (Eds.). (2006). *Destination recommendation systems: Behavioral foundations and applications*. Cabi.
[10]. Schafer, J. Ben, Joseph A. Konstan, and John Riedl. "E-commerce recommendation applications." *Data mining and knowledge discovery* 5, no. 1-2 (2001): 115-153.