MovieFans: A Movie-enthusiasts Oriented Social Network

Haoyun Qiu, Siqi Huang, Weijian Zhou, Chen Zhu
Department of Computer and Information Science
Univeristy of Pennsylvania

Abstract

IMDB is a great website for retrieving movie information, but it offer little help for you on connecting other people who share common cinematic interests. Our product, **MovieFans,** is a movie social network which exactly meet this need. It allows users to create their own accounts, config settings such as avatar, profile and favourite movie genres, follow/unfollow other users and create/join/leave interest groups. Meanwhile, users are able to search movie information, share/review/like movies on movie pages and post status update with friends. Also, users can tweet and tag people(i.e. use@ you know) and others can get notifications. You can also use hashtag(i.e #helloworld) to submit a topic and users can click it to locate similar hashtags. What's more, MovieFans can recommend potential interesting movies to you in addition to serving as a social network websites and record keeper.

Product Features

- Sign up/Login/Log out and Login via **Facebook** oauth.
- User settings: set up you own avatar(head url), profile(background image url), personal descriptions, favorite genres.
- **Movie search**: via movie name (**fuzzy search** supported)
- **Advanced movie search**: user can specify genres, movie duration and sort the results by rating or votes.
- In **search result page**, we will show all the movies that meet the requirements, each with its name, rating, length, release date and overview. In this page, we also trigger the **google search** and include top results in this page.
- User can click each of these results and jump to its **movie page**, which diaplays much more details here: director, casts, youtube trailer(s) and official link if it has one.
- On movie page, user can also **like, share and review** the movie. These will be posted on his/her personal page and shared as **new activities**.
- User can **follow/unfollow** other users. By following a user, you can view his/her **personal page** and see his/her various activities/updates.
- User can **create, join or leave** a **discussion group**.
- Once joined a group, members can view and post status in group discussions. These will be shared to other members at their homepages.
- Not only movies, you can also **search users, groups, hashtag** status with keywords (also fuzzy supported).

Special Features

Facebook login

Though facebook does not offer sdk for Java, we can still utilize the authentication of Facebook OAuth. Bascically we have this flow: On click of the facebook login button then a Facebook URL will be invoked. Facebook will validate the application ID and then will redirect to its login page so user can enter the FB login credentials. Facebook will validate the credentials and then redirect back to the browser with a request to forward to the redirect_url. Redirect_url is the URL in our application which will take care of further processing. Then, browser will call the redirect url and request for access_token from Facebook. Facebook on validation success will respond back with access_token. Redirect URL page will again call the Facebook to request for user data by sending the access_token. Facebook on validating the access_token will respond back with user data requested. Finally, our application can use this data (i.e. the username and facebook id) to create a user entity in our system and help him/her login.

Google Search Trigger
We include a google search when users trigger the movie search. Users can see several google search results displayed in a column. Simply we send a http GET request to "http://www.google.com/search?q=" with a user agent of "cis550". After we get the response, we parse it into DOM and get the link nodes which are the results url we want.

Group
Users can create groups with interest. And other users can join or leave your group. If you are the member of this group, you can send message in this group and all other members can see it. And these messages will also appear in your homepage's timeline.

Various Searching
We support lots of searching options for users: movie search(by name or keywords), user search, group search, hashtag search and advanced movie search(by genres, rating and year). All of them support fuzzy search, which means substring and case insensitive will be supported.

NoSQL component
We use Berkeley DB as our NoSQL component for storing and processing most of the social network data includes user, group, news, id generator (for news id, group id and so on), movie page, hash tag and so on. NoSQL is more scalable for frequently change data structure so we decide it to use it for the social part. We can config the BDB momory cache, which make data accessible in-memory and responsive. Also we can also set the environment config to transactional-true to make the data updates persistent and reliable.

Hash tag and tag
These two concepts become necessities in social network application and they are really convenient and interesting, so we decide to make it. In backend, we check all the status users sent and use regex pattern to match tokens with tag(@) and hashtag(#). If this status contains tag, we will check the tag and locate the user entity in BDB. If this user exists, we send a new message to his/her messagebox; when this status contains "#", we will extract it and store the status id within the respected hahstag entity (in BDB) for future searching. Each hashtag entity has a list of status id to map all the status contain this hashtag. For frontend, we capture the tag and hashtag by another two pattern regex match iterations and translate them to hyperlinks, so that users can click it on browser and locate the information easily.

Modules and architecture

From the macro view, the project include two major components: a Tomcat webserver running on AWS EC2 instance and a Orcale SE database running on AWS RDS. And our application which is contained in Tomcat, was built on a classic MVC structure.

Model includes a SQL database mentioned before, which is holding all the movie data extracted from TMDB and MovieLens. Also there is a NoSQL component Berkeley DB (BDB) embedded in our webserver which holding all data related to the social interaction (i.e. user, group and news).

Controller includes various of Java servlets, function models and database controllers. Java servlets are responsible for dealing with http requests and responses from clients. Function models are within the servlets and responsible for all kinds of backend functions such as processing a user's new status and boradcast to his/her fans's timeline. Database controller include a SQL query model for Oracle db and several BDB accessors for each NoSQL store entities. Controller will read/write the database and send updates to frontend view and display them.

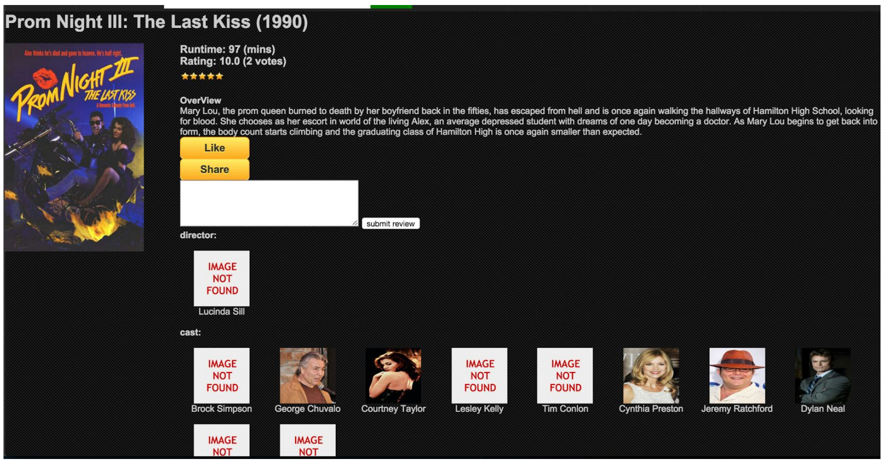
View inlcudes an intermediate layer written in Java and frontend scripts include JSP, htmls and css. The intermediate layer includes various of view classes contain data from the controller, so frontend can use view objects to directly display the data without knowing the complicated logic controller which make the development isolated and easy.

Data instance use

Movie Data is mostly used for movie searching and information display.

Each search result will include this movie's overview, rating, poster, release date and runtime.

When you click into this movie's page, you will see more movie data: director and cast's pics and name; a youtube link or even its official homepage if exists.



Prom Night III: The Last Kiss (1990)

Runtime: 97 (mins)
Rating: 10.0 (2 votes)

OverView
Mary Lou, the prom queen burned to death by her boyfriend back in the fifties, has escaped from hell and is once again walking the hallways of Hamilton High School, looking for blood. She chooses as her escort in world of the living Alex, an average depressed student with dreams of one day becoming a doctor. As Mary Lou begins to get back into form, the body count starts climbing and the graduating class of Hamilton High is once again smaller than expected.

director:
Lucinda Sill

cast:
Brock Simpson, George Chuvalo, Courtney Taylor, Lesley Kelly, Tim Conlon, Cynthia Preston, Jeremy Ratchford, Dylan Neal

Data cleaning and import mechanism

Data Source:
The data are mainly from TMDB and MovieLens. The connection between them are the links csv table in MovieLens. The data in MovieLens is used for selecting movie genres. And also one part of the recommendation system, the recommendation of a movie by a movie you have already watched, is based on the user comment in MovieLens. The TMDB offers the rest of the information of movies and cast.

Data Transfrom:
Most data is stored in json files. Some tuples have attributes with multiple values, making it impossible to be transformed into csv directly. So we use the apache poi API in Java to write data into Excel table and convert them into csv.

Data Cleaning:
For null value, if type is string we set it to “”; if the type is numerical we set it to 0. For other dirty data, we will check it and may remove it.

Tables:
In general, we have 12 tables:
AlternateTitle: Stores the alternate title of a movie, the id of it is an external key, which is the key of BasicTMDBInfo.
BasicTMDBInfo: Stores the basic information of a movie in TMDB database like names, id, votes, date, runtime, poster, overview and so on. The key is id.
Genre: Stores the genres of all kind. Genres itself are stored as string with an genre id, which is the key.
Keywords: Stores the keyword of movies. Keyword ifself are stored as string with an keyword id, which is the key.
MovieConnection:Stores the connection of movie1 and movie2. The connection is the likeness that if a person likes movie1 he will also like movie2.
MovieGenre: Stores the movie id and its genre id. It is a relation between movie and genre.
MovieKeywords:Stores the movie id and its keywords id. It is a relation between movie and keywords.
MoviePerson: Stores the movie id and person id. It is a realtion between movie and person.
MovieStudio: Stores the movie id and studio id. It is a relation between movie and studio.
Person: Stores the person information in a movie, including cast and director.
Studio: Stores the studio information in a movie.
YoutubeTrailer: Stores the youtube trailers of a movie. The key is both the movie id and the trailer url. The movie id is the external key from the BasicTMDBInfo.

Data Upload:
After we get the excel data, we convert them to csv files and upload to the Oracle Database via SQL developer.

Database Schema

Movie:
MovieID, Name, Year, Rating, Overview, Genre, Length, Trailer, Review, ReleaseDate

User:
UserID, Facebookd, Name, Password, NewsList, AvatarURL, BackgroundImageURL, Description, LoginTime, FollowList, FansList, CreateGroupList, JoinGroupList, MailBox, FavorGenreList, LikeMoviesList

News:
NewsID, CreatorName, MessageBody, ReleaseTime, Title, NewsType, MovieId, MovieName, MoviePoster, ReceiverList

Group:
GroupID, Name, Creator, MemberList, NewsList, AvatarURL, BackgroundImageURL, Description

MovieRecommendation:
UserID, MovieList, genre

Algorithms, communication protocols

Movie Mapping

This is used to calculate the connection between different movies for the movie recommendation. We first get the original data from the movieLens. The format is a userId(in MovieLens) and the movie rating of different movie id. The rating system in MovieLens is between 0 to 5. If a movie is rated no less than 4, we think the user like the movie. And we get all liked movie of a user and do a pairwise join of them, with the smaller id to be the first value of the pair. Then we keep this value. For all the users, we run this algorithm. If the pair exist, we add one to the connection of the pair. After all users pair are calculated, we sort the pairs with the connection and extract the top 50,000 pairs and store them in the SQL database.

Communication Protocols

We use Tomcat as our web application container. So all communications between server and clients will via http protocol. Users can operate the user interface and trigger a http GET (i.e. fetch a movie page) or http POST (i.e. type in a search keyword and submit it) to the webserver. The webserver will route the request urls to a specific servlet. And the servlet will deal with the request and return a response to the client.

Use Cases

Use Case Name: Searching movie
Actors: User; System
Triggers: The users indicates that they want to search for a movie.
Preconditions: User knows the keyword of movie such as movie name, genres and length.
Post-conditions: All search results (each movie) related to the seach keywords will be displayed.
Normal Flow:

1. Users indicate that they want to search for a movie.
2. If users know some keyword of the movie, they click “options” in navigation bar and choose “Search Movie”
3. Users input the keyword.
4. The result page will shows all movies corresponding with the input.

Alternate Flows:
2A1: The users do not know the keyword of the movie.

1. They click “options” in navigation bar and choose “Advanced Search”.
2. User select a genre, length, and the result sort method.
3. The result page will shows all movies corresponding with the search settings.

Use Case Name: Movie Recommendation
Actors: User; System
Triggers: The system can recommend potential interesting movies to users based on their preferences.
Preconditions: User has selected their favourite movie genres or they have shared/liked a specific movie.
Post-conditions: The recommended movies will be shown on user’s homepage.
Normal Flow:
1. The system get the information of user’s favourite movie genres and movies he/she liked.
2. The system will select top movies of these genres in database or high mapping score movies correspond to movies liked.
3. The selected movies will be shown on homepage.

Use Case Name: Send status
Actors: User; System
Triggers: Users want to express themselves and share them to friends.
Preconditions: User has already logged in.

Post-conditions: User will post a new activities on his/her timeline both in his/her homepage and user page, and his/her fans can also capture this news. If status includes #, system will store it for future search. If status includes @, system will validate the user and send a new message to this user's messagebox.
Normal Flow:

1. User login and go to homepage or his/her own page.
2. User type in some text in status window and click submit.
3. User can see his/her timeline updates and so do his/her fans.

Use Case Name: Follow
Actors: User; User
Triggers: Users want to follow other users and see what happen to him/her.
Preconditions: User already logged in and goto other user's page.
Post-conditions: User can see following user's activities appear in his/her homepage timeline. Followed user can see a notification when other user follow himselve/herselve.
Normal Flow:

4. User login and go to some user's page.
5. User type in some text in status window and click submit.
6. User can see his/her timeline updates and so do his/her fans.

(Sorry, since we have too many use cases we cannot include all here because of the length restriction.)

Optimization techniques employed

Indexing

In addition to the system default indexing on the primary keys of each table, we indexed three additional variables in the movie table for query optimization. The three vairables are **userrating**, **votes** (# of people rated a certain movie) and the **runtime** of each movie.

Our selection of indexed variables is based on the frequency of each variable used in our queries. The performance of indexed query is 2 - 3 times faster than non-indexed query. The detailed performance reports is shown below:

Index	Sample Query	Runtime without index (average time)	Runtime with index (average time)
Runtime_DESC	select title, runtime from BASICTMDBINFO where runtime > 0 and runtime < 20;	328 rows in 0.80s	328 rows in 0.56s
Rating_DESC	select title from BASICTMDBINFO where userrating = 10;	92 rows in 0.46s	92 rows in 0.16s
Vote_DESC	select title from BASICTMDBINFO where VOTE >2000;	105 rows in 0.45s	105 rows in 0.23s

Caching

SQL connection pool

We found that creating a new sql connection via normal JDBC driver is synchronized and it often takes about 2 seconds. It becomes a bottleneck even for light query. So we use c3p0 library for caching and managing the sql connections easily. After optimization, get a sql connection will take fewer than a millisecond.

LRU cache for Movie Page

We support users to locate a specific movie page and this will include a sql query to fetch the data related to this movie. Because movie pages can be accessed very frequently so it is better to cache it. Basically we implement an LRU cache with fixed capacity and it contains the “hottest” movies which were accessed most frequently recently. If user access a movie page, we first look for it in cache, if it exists just return it; otherwise, we do a sql query and insert it to cache. If the cache is full, we remove the least recently used data. A double-linked list with a hashmap to implement the LRU cache can have a low time complexity both for update and retreive. And we can also config the cache capacity to has controllable memory footprint. Before cache, it takes at least 1~2 seconds to fetch the movie data for every request, but cache can make it real time.

Technical specifications

SQL database
Engine: Oracle SE One (deployed on: AWS RDS)

Web application
Webserver: Tomcat 8 (deployed on AWS EC2)
Frontend: HTML(for static web page), CSS (for content layout), JSP (for dynamic web page).
Backend: Java 8

NoSQL database
Engine: Oracle Berkeley DB (embedded db in webserver)

Referenced Libraries
cp03: For JDBC driver and SQL connection pool
jsoup: HTML parser, for parsing the google search results.
prettytime: For translating absolute time (a Long type of system time) to relative time (a readable string, eg: 1 hour ago).
json: For parsing Facebook response.
je: Berkeley DB
jetty-runner: For local webserver testing.

Division of work

Haoyun Qiu: Architecture, AWS, NoSQL, Backend functions and optimization.

Siqi Huang: Data parsing and cleaning, Intermediate Layer for view, SQL Query.

Weijian Zhou: SQL database deployment and Front End Implementation.

Chen Zhu: Front End Imeplementation and Indexing optimization.