Hindawi
Complexity
Volume 2024, Article ID 9893259, 1 page
https://doi.org/10.1155/2024/9893259

Retraction

Retracted: Optimization of an Intelligent Music-Playing System Based on Network Communication

Complexity

Received 23 January 2024; Accepted 23 January 2024; Published 24 January 2024

This article has been retracted by Hindawi following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of one or more of the following indicators of systematic manipulation of the publication process:

(1) Discrepancies in scope
(2) Discrepancies in the description of the research reported
(3) Discrepancies between the availability of data and the research described
(4) Inappropriate citations
(5) Incoherent, meaningless and/or irrelevant content included in the article
(6) Manipulated or compromised peer review

The presence of these indicators undermines our confidence in the integrity of the article's content and we cannot, therefore, vouch for its reliability. Please note that this notice is intended solely to alert readers that the content of this article is unreliable. We have not investigated whether authors were aware of or involved in the systematic manipulation of the publication process.

Wiley and Hindawi regrets that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our own Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] L. Zhang, "Optimization of an Intelligent Music-Playing System Based on Network Communication," *Complexity*, vol. 2021, Article ID 9943795, 11 pages, 2021.

Hindawi
Complexity
Volume 2021, Article ID 9943795, 11 pages
https://doi.org/10.1155/2021/9943795

Research Article

Optimization of an Intelligent Music-Playing System Based on Network Communication

Liaoyan Zhang

Weinan Normal University, Weinan, Shaanxi 714099, China

Correspondence should be addressed to Liaoyan Zhang; zhangliaoyan@wnu.edu.cn

Received 26 March 2021; Revised 19 April 2021; Accepted 29 April 2021; Published 7 May 2021

Academic Editor: Zhihan Lv

Streaming media server is the core system of audio and video application in the Internet; it has a wide range of applications in music recommendation. As song libraries and users of music websites and APPs continue to increase, user interaction data are generated at an increasingly fast rate, making the shortcomings of the original offline recommendation system and the advantages of the real-time streaming recommendation system more and more obvious. This paper describes in detail the working methods and contents of each stage of the real-time streaming music recommendation system, including requirement analysis, overall design, implementation of each module of the system, and system testing and analysis, from a practical scenario. Moreover, this paper analyzes the current research status and deficiencies in the field of music recommendation by analyzing the user interaction data of real music websites. From the actual requirements of the system, the functional and performance goals of the system are proposed to address these deficiencies, and then the functional structure, general architecture, and database model of the system are designed, and how to interact with the server side and the client side is investigated. For the implementation of data collection and statistics module, this paper adopts Flume and Kafka to collect user behavior data and uses Spark Streaming and Redis to count music popularity trends and support efficient query. The recommendation engine module in this paper is designed and optimized using Spark to implement incremental matrix decomposition on data streams, online collaborative topic model, and improved item-based collaborative filtering algorithm. In the system testing section, the functionality and performance of the system are tested, and the recommendation engine is tested with real datasets to show the discovered music themes and analyze the test results in detail.

1. Introduction

In today's Internet and Big Data era, cell phone APPs are exploding, the data people generate are growing by leaps and bounds, and the information on the Internet has long exceeded the ability of individuals to receive it in their lifetime. With the overload of information, how to get the required information efficiently has become an urgent problem to be solved. On the one hand, the cost for information producers to get their information discovered by users in the huge information flow is increasing day by day; on the other hand, the information that users can access is only a very small part of the huge resources on the network, and there is still a lot of valuable information that cannot be accessed by users. In order to solve this problem, many solutions have been proposed, including portals, search engines, and recommendation systems, which have become popular in the last decade. Portals allow users to browse information by categories, which are usually determined by experts. The drawbacks are that the classification of items or information as perceived by users may not be the same as what experts think and that items can only belong to one category, while in reality, the categories of items are often vague and similar to multiple categories. Compared with the recommendation system, the shortcomings of the search engine are also more obvious. In the above case, the user's keywords must accurately describe their intentions to get the desired results and not use their own and other people's historical behavior information.

The domains where recommendation systems are widely used are similar in that most of them have a large number of items that are difficult for users to browse and filter. Also, most users have no clear needs for the items and it is difficult for users to accurately describe their preferences in specific words. Most of the fields that meet these characteristics belong to the category of pan-entertainment, such as music, books, pictures, short videos, movies, news, and e-commerce. Online music platforms are suitable for recommendation systems. First of all, the mainstream music platforms on the Internet currently have millions or even tens of millions of distinctive music, which are constantly updated every day, making it difficult for users to find their favorite songs; "QQ Music 2018 Listening Data" shows that 88% of digital music users are willing to discover new good songs, but only 20% of users can easily find good songs [1]. Secondly, users sometimes only need some favorite music to regulate the atmosphere at work or in life, so users' needs are very vague. Compared with movies and books, online music charges less, takes less time to listen, and has a high rate of repeat play. Nowadays, music recommendation systems have become an important tool for many online music platforms such as NetEase Cloud Music, QQ Music, and Spotify to improve user experience and fidelity [2], and a good music recommendation system can also bring more attention to promising music. Many solutions are available in this area, but many challenges remain. Due to the large number of users on mainstream music platforms, users can listen to more than a dozen songs in an hour, and these platforms generate hundreds of thousands of listening records per second at their peak [3]; and the field of music recommendation has shifted from being based on explicit feedback (rating data) to being dominated by implicit feedback (play records); keeping the recommendation model up-to-date becomes a big challenge in the face of the constantly updated large amount of play record data [4]. Most of the traditional music recommendation systems are offline, and they have to train their models on static data and retrain them on all data at intervals as new data arrive. Usually, these offline recommendation systems update their models every day or even longer, retraining the models often takes several hours, their models cannot capture the recent changes in users' preferences, and training the models is computationally intensive and takes up a lot of memory and time [5]. Due to the rapid update of online music libraries, which leads to increasingly rapid changes in user interests, the problem of offline recommendation algorithms is becoming more and more prominent as music websites generate interactive data faster and faster, and they are no longer as effective as simple online algorithms, and it is foreseeable that streaming recommendation systems will become mainstream [6–10]. Most of the existing systems are not efficient enough, consume a lot of memory, or have difficulties in giving reasons for recommendations and poor user experience. The booming development of smartphones has led to the emergence of various mobile applications. Listening to music is an effective means to relax and relieve stress, so smartphone music players are gradually becoming popular [11].

Compared with traditional music playback devices (recorders, MP3, etc.), smartphone music players have obvious advantages. First of all, traditional music playing devices are not easy to carry. Traditional music playback devices usually only provide the function of playing music, while smartphone music players have the function of making calls and other general cell phones, but also through the use of smartphones to download music playback software to achieve music playback, the user does not have to carry a cell phone at the same time and then carry an additional independent music playback device, which provides convenient conditions for the carrying of equipment. Secondly, the traditional music playback device basically only supports local playback. Recorders can only play songs on the tapes they own, while smartphone music players can achieve music caching and online playback, which allows users to listen to an increased range of music. Most importantly, traditional music playback devices do not have a recommendation function, and as the number of songs increases, it becomes very difficult for users to pick out the songs they may be interested in from a large number of tracks. To enable the above problems to be successfully solved, this article designs a mobile music playback system based on a joint track recommendation algorithm. Therefore, the study of real-time music recommendation systems is of great practical value. With the introduction of social tagging, users can tag songs and artists, making it easier to find the songs they want and also providing useful information for the recommendation system. The lyrics also reflect the theme and emotion of the song to a large extent. It is also a worthwhile research problem to improve the recommendation effect by using music auxiliary information such as tags in the streaming music recommendation system. In this article, we analyze the requirements and design the overall music recommendation system based on the software engineering process, then implement each functional module of the system using the current popular theories and technologies, and test the system in terms of functionality and performance.

2. Related Work

Recommender systems have been a hot topic of research in academia and industry for the past two decades. Its advantage lies in the proactive nature, which can automatically collect user behavior data and build models to find suitable items to recommend to users in the system. Some of the more popular recommendation techniques and recent research trends are as follows.

Content-based recommendation: it is concerned with extracting enough item feature information, calculating item similarity accordingly, and recommending items similar to the user's favorite items. It is limited by items and highly dependent on the domain of recommended items, and the way of calculating item similarity varies greatly from domain to domain, and it is difficult to extract the features for multimedia resources. And it often lacks novelty.

Collaborative filtering recommendation: it is not related to the domain of recommendation system, and there are two

main directions: one is the nearest neighbor-based collaborative filtering, which uses user feedback data to calculate the similarity between items or users, and the main methods to calculate the similarity are Euclidean distance, cosine similarity, Pearson correlation coefficient, etc.; the other is model-based collaborative filtering, which is mainly based on the matrix decomposition model of hidden features, where the user-item interaction matrix is decomposed by Singular Value Decomposition (SVD) or Nonnegative Matrix Factorization (NMF) to obtain the hidden feature matrix of users and items. In the 2006 Netflix competition, the previous SVD-based recommendation algorithm won the championship over many algorithms, which made the collaborative filtering algorithm based on matrix decomposition very popular [12]. Positive Matrix Factorization (PMF) introduces a probabilistic model to further optimize the SVD decomposition process. Generally, the model-based collaborative filtering algorithm outperforms the nearest neighbor-based collaborative filtering algorithm but lacks interpretability. In the literature, the authors designed a collaborative filtering algorithm based on label weighting with time decay and implemented a music recommendation system [13].

Hybrid recommendation systems: hybrid recommendation systems allow different recommendation systems to complement each other and can improve novelty while alleviating the cold start problem to some extent. Most of the previous related studies use auxiliary information of items, mainly textual information such as abstracts of papers [14], item descriptions, and reviews. Many approaches are used by hybrid recommender systems for document modeling, such as Latent Dirichlet Allocation (LDA), Stacked Denoising Autoencoders (SDA), or Convolutional Neural Network (CNN) [15]. There are also algorithms to model the visual features of images [16].

Online and streaming recommender systems [17]: most hybrid recommender systems are designed for batch processing; i.e., the initial model is constructed from static data and the model is periodically reconstructed on all data as new data arrive. However, keeping the recommendation model up-to-date with the arrival of new data is an important requirement to ensure the quality of recommendations [18]. In practical applications, recommendation systems can be abstracted as data flow problems. Online recommendation systems need to update models in real time from a continuous stream of data, and limited computational resources often make it difficult to store all the data and to recompute all the data. It must learn from a continuous stream of data and adapt to changes in the data in real time. Recent studies have pointed out that simple online algorithms can generate better recommendation results than complex offline algorithms with periodic updates. Most online recommender systems are based on incremental learning, such as incremental neighbor-based algorithms and incremental matrix decomposition [19] (based on stochastic gradient descent or alternating least squares [20]). The literature [21] proposed Stream Rec, a streaming recommendation system based on collaborative item filtering. It implemented a video recommendation system using a commercial cloud computing platform by decomposing the similarity calculation and incrementally updating the similarity. The literature solved several problems in terms of practical applications in the industry using a scalable update mechanism. The literature [22] for matrix decomposition proposed incremental updates for the positive feedback.

Recommendation systems on big data platforms: with the popularity of distributed computing platforms such as Hadoop and Spark, these big data processing frameworks have been widely used in recommendation systems [23–26]. Compared with Hadoop, Spark tries to store the intermediate results in internal storage instead of HDFS, which avoids a lot of disk IO operations and greatly improves the computational speed. A recommendation system based on collaborative filtering on the Spark platform is designed and tested on datasets such as Movie Lens. The paper investigates a big data computing and storage platform and implements a real-time recommendation algorithm using Spark. The paper solves the information loss problem in matrix decomposition and implements the recommendation system on the Spark platform.

Topic modeling: topic models treat documents as unordered lists of words, usually considering only the number of word occurrences without considering word order, and documents have a topic vector. The most influential topic model is the Hidden Dirichlet Distribution model, which treats the topic as a Multivariate Distribution with parameters generated by the Dirichlet distribution. LDA is popular because of its simplicity and interpretability. Variants of LDA add a temporal dimension to it and are able to handle distributions that change over time.

3. Detailed System Design and Implementation

3.1. Overall System Design. In order to store and process massive music and review data and user behavior data streams, this system is designed based on the existing big data and recommendation system architecture, and the overall architecture diagram of this system is shown in Figure 1.

(1) The user interaction module uses B/S architecture and is built by Spring Boot.

(2) The Flume platform is used to collect user-generated logs and send them to Kafka cluster in real time and Kafka cluster load-balances and buffers the received data to generate discrete streams for pushing to Spark Streaming, Flume, and Kafka; both have better fault tolerance mechanisms to ensure reliability.

(3) Spark Streaming calculates statistical metrics, such as the number of music plays in the last 24 hours and daily plays and the number of users playing, on the user click stream received from Kafka in real time and writes them into the Mongo DB database, while training online collaborative theme models, incremental matrix decomposition, and other models on the data stream.

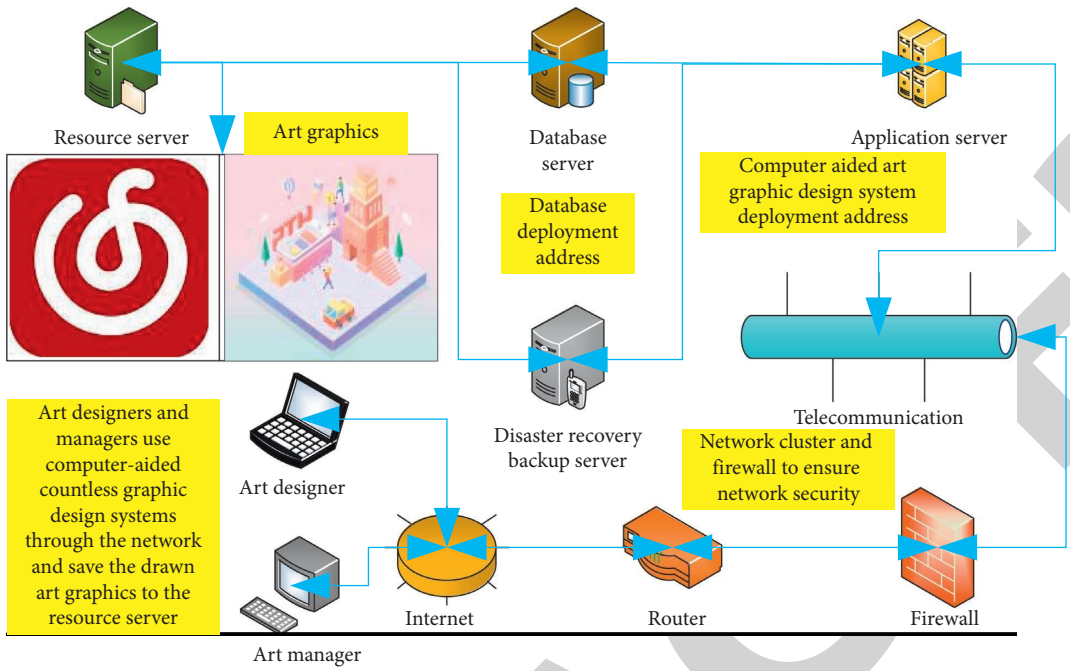

Figure 1: Overall system architecture diagram.

(4) The Redis in-memory database mainly caches the number of times the user recently played music machines and the user's recent recommendation list in order to take advantage of the features of Redis to efficiently process recommendation requests. When the Redis memory space is full, the Least Recently Used (LRU) policy is used to eliminate keys.

(5) This paper adopts Mongo DB as the business database, which is fast in insertion and retrieval, suitable for storing a large amount of data, has good high availability, and can be well integrated with Spark SQL. It works with Elastic Search to support efficient music retrieval by tags and titles.

Derived from the functional requirements, the functional division of this system is shown in Figure 2. The system is divided into music browsing, data collection and statistics, recommendation engine, personal homepage, and other functional modules, and each functional module is divided into one or more submodules and each module is responsible for independent functions and may depend on other functional modules. The music browsing module supports browsing artists and music in various ways, including browsing by artist categories, browsing by artist details and their popular singles, browsing music by category tags, and browsing by daily and real-time music charts. It also writes a log record when users play and switch music, and users can add, view, and like comments under music. The recommendation engine module implements recommendations based on various models. The personal homepage module includes submodules for personal account login, registration, information modification, and viewing recent personal plays.

3.2. Music Browsing Module Implementation. This module realizes the functions of browsing singers and singles by category, querying singers' popular singles, and playing music. There are three tabs of "Recommend," "Music Gallery," and "My" on the home page of the system, displaying different pages, respectively. Under the "Music Gallery" tab, you can see 3 subtabs, that is, "Artist," "Chart," and "Category." Under the "Music Gallery" tab, you can see 3 subtabs, namely, "Artists," "Charts," and "Categories." The main functional submodules of this module are designed as follows.

3.2.1. Browse Music by Artist. In order to let users quickly find their favorite singers, this module supports various combinations of conditions such as region, gender, and style to search for singers. The singer tab under the music library provides three columns of filtering criteria, which can be limited to the region, gender, and style of the singer, and the list of singers that meet the criteria is displayed below, including the singer's avatar and name. The list of singers will be refreshed when the user switches the filter criteria. Click on the singer name in the singer list to enter the singer details page, which shows the singer picture and popular singles and albums they belong to. Click on the music in the singer's worklist to play the music, while the background will record the log. The sequence diagram of browsing and playing music by the singer is shown in Figure 3.

3.2.2. Browse Music by Category. Users select the category subtab in the music library; the system is divided into several areas according to music themes, scenes, moods to display common category tags; by selecting a category, you can view the list of popular music under that category.

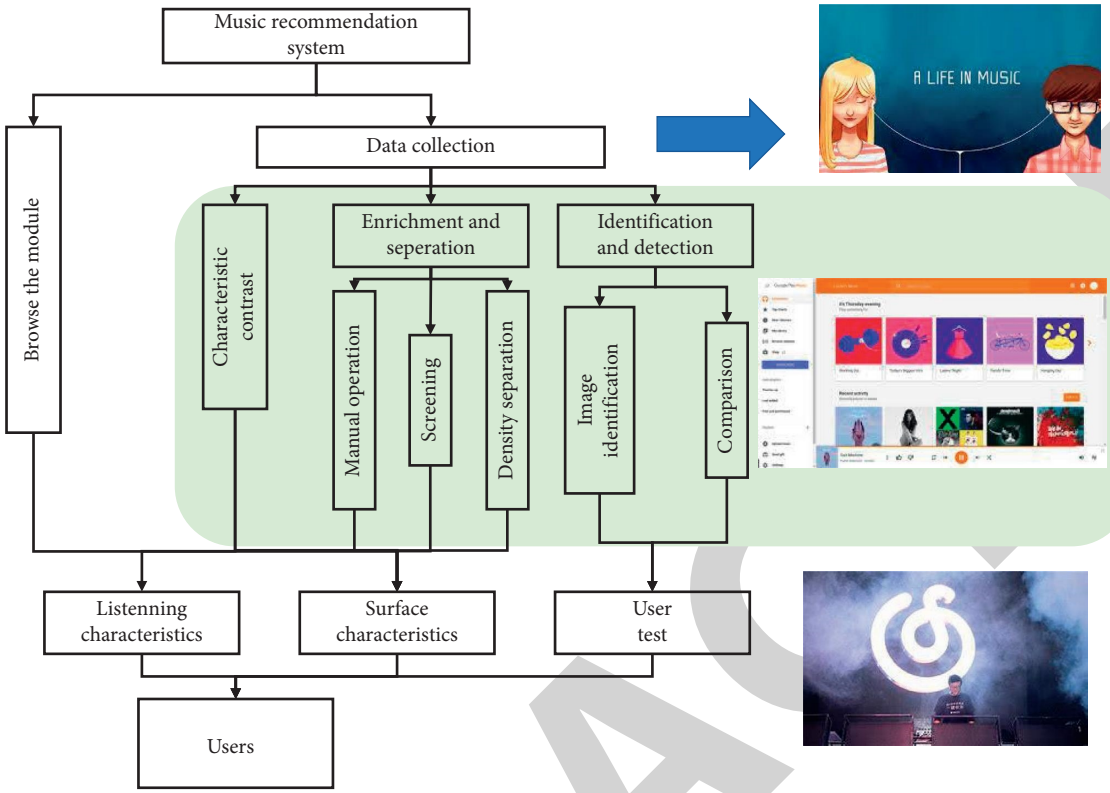

Figure 2: System functional structure diagram.

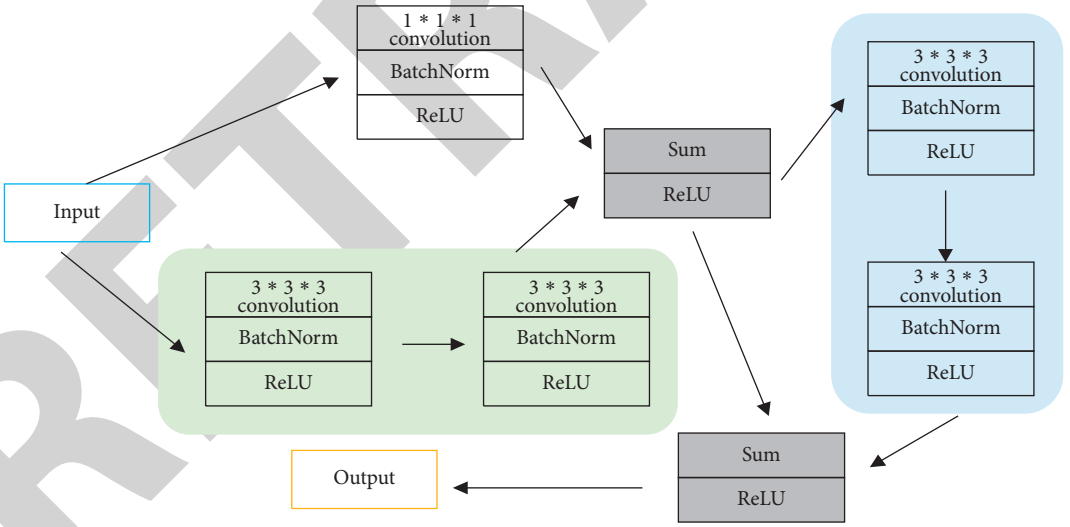

Figure 3: Browse music timeline by artist.

3.2.3. Music Charts. Users select the Charts subtab in the Music Gallery and the system displays a real-time chart of the last 24 hours of music plays. Users can switch to the current day's chart and the current week's chart, and in the daily and weekly charts, they can choose to sort by the number of plays or by the number of users playing.

3.2.4. Personal Recently Played Music. Users can select the My tab on the system home page to enter their personal home page, where they can see their account avatar and nickname, and the recently played music is displayed under the account, and users can switch between sorting by recent playing time and sorting by playing times.

The class diagram of the music browsing module is shown in Figure 4. This module is divided into three layers: the controller layer is responsible for receiving and returning requests, the service layer is responsible for handling specific business logic, and the data access layer is responsible for accessing database operations. In the controller layer, Artist

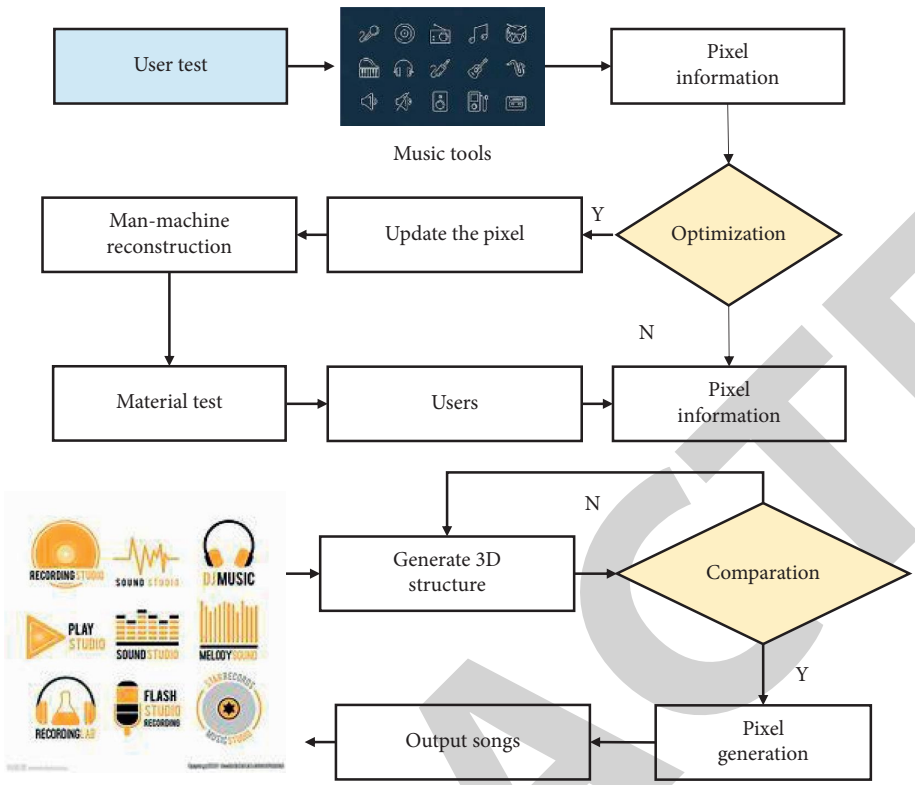

Figure 4: Music browsing class diagram.

Controller, Track Controller, and User Controller all inherit from Base Controller, which defines basic methods such as writing log, getting list, and getting single record according to id. Artist Controller provides methods to get artist details, search popular artists list by region and style, etc. Track Controller provides methods to query popular music by tag, query popular singles by artist, query real-time popular music list and daily chart, search songs by lyrics, etc. User Controller provides methods such as login and registration, querying users' recent play history, etc. In the data access layer, Artist DAO, Track DAO, and User DAO handle database operations related to artists, music, and users, respectively, and they all inherit from the Base DAO class, which implements basic operations such as adding and deleting single data and querying the list. The data access layer classes encapsulate the Mongo Template and Redis Template and use them to implement various database operations. For most queries, this module stores the most recent query in the Redis cache, and when processing the query, it first looks in the cache and then queries the Mongo DB database and stores the result in the cache when it cannot be found.

3.3. Data Collection and Statistics Module Implementation. The data collection and statistics module collects user behavior data in real time and carries out popular statistics, while providing training data streams for the recommendation engine. Although this module is not a front-end function module from the user's point of view, it is the basis of music recommendation and music real-time charts, hot charts display, and other functions and is the foundation module of the whole system, so making it independent can greatly improve the code reuse, as shown in Figure 5. This module uses Flume to monitor the updates of user behavior logs, send them to Kafka and Spark Streaming to count the music popularity trends, and store the statistics into the database. It can be divided into the following layers. This section shows examples of testing the similarity of the music recommendation system by querying the recommendation list, artist list query, and daily music catalogue search, and the test results of other interfaces are similar.

(1) Log collection layer: the server collects data on user behavior such as playing music and writes them to the end of the log, and this article uses Tomcat, which is built into Spring Boot.

(2) Producer layer: Flume is deployed on each server node. Flume uses the tail -f command to track the new content at the end of the logs to monitor and collect server logs and send them to the Kafka cluster in the specified file size.

(3) Kafka cluster layer: this layer load-balances and buffers the data coming from the producer layer. It has better reliability and prevents message loss. After Kafka collects log updates, it generates DStream through Kafka Stream program and pushes it to Spark Streaming. DStream consists of RDD ["user ID \t music ID \t operation type \t date and time"].

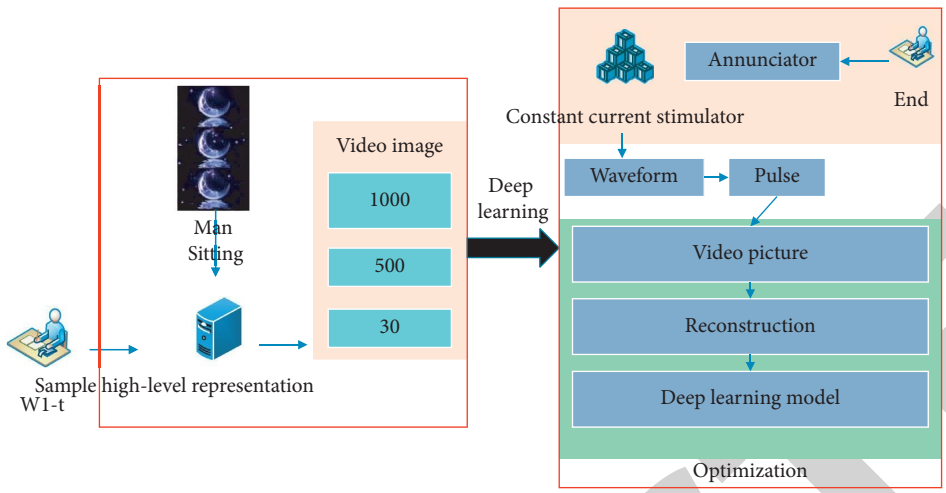

Figure 5: Data collection and statistics module hierarchy.

(4) Stream processing layer: Spark Streaming is used to count Kafka data streams and train online recommendation algorithms and write statistical results such as the number of users playing music daily to Mongo DB.

(5) Storage layer: the log statistics are stored in Mongo DB, including the number of daily music plays and the number of users playing.

The statistical task of this module is to count the number of real-time plays (i.e., plays in a recent time window such as 24 hours) for each music, as well as the number of plays and the number of users playing that day. The number of plays in a time window can be achieved by using Spark Streaming's count By Value And Window, which counts the number of occurrences of each key in a time window. In this paper, we export the results to Redis and use Sorted Set to maintain the number of music plays in the last 24 hours. The elements of an ordered set have a score, and the elements of the set are kept sorted by the score, where the score is the amount of music played. The ZRANGE operation of the Sorted Set can query the elements whose scores are in a certain interval, and it can be used to efficiently query the music with the highest number of plays in real time. For the number of music played on the same day, just count the number of times the music ID appears in the microbatch and add up the number of music played on the same day in Redis. For counting the cumulative number of users who have played certain music on the same day, reduce By Key and group By Key, and other conversion operations are applied to each batch interval RDD and cannot use the previous RDD data, which is obviously not possible; because of the deduplication of users, window operations, such as reduce By Key And Window, are also difficult to handle and have a large overhead.

Therefore, this article uses the Redis set data structure to record the set of users who play each song every day, and the key is a string of the form "date:music ID". In this paper, we can set the expiration time of key to one day so that we can store all the music records of that day, and at most, we only need to store the set of users who played all the songs of today and yesterday. The set operation in Redis is an atomic operation, so we do not need to worry about thread safety in this article.

4. System Testing

In this paper, we mainly implement an online collaborative topic model (COLDA), incremental matrix decomposition (Inc MF), and improved item/user-based incremental collaborative filtering (Knni++) using Spark. When the data volume is large, the incremental matrix decomposition recommendation engine is preferred; otherwise, the COLDA recommendation engine is preferred. Since the registration information filled by users is likely to have more missing values or inaccuracies due to privacy considerations, this system does not use the registration information to make recommendations for new users but uses the method of letting new users select the tags of interest to make initial recommendations.

In this paper, we first initialize the model parameters and incrementally update the model, when data from microbatch are received, and when a user recommendation list request is received, if the user recommendation list is in the cache and has not expired, the cache is returned; otherwise, its recommendation list is calculated. In the actual implementation, this paper does small-batch gradient descent on each batch interval and uses Dense Vector under org. Apache.spark.ml.linalg package to represent the user-item hidden feature vector. Record class represents a user playing music record. Parameters class encapsulates the hidden feature dimension, learning rate, and user-item regularity. The Rating class represents the calculated recommendation score of the item for the user. In this paper, the RDD [(id: Long, features: Dense Vector)] (i.e., user or item ID and hidden feature vector) key-value pair collection is used to store the hidden feature vectors of users and items, called user features and item features, respectively, and the

parameters of the user features and item features are enclosed in the Streaming MFModel class along with the parameters.

In this paper, we first remove tags with less than 5 songs and users who have listened to the song less than 10 times. This leaves 11,000 tags. When the recommendation list length N is varied, the Recall and F1 variations of several recommendation engines in the system are shown in Figure 6. It can be seen that the COLDA recommendation engine is the best performer among the algorithms, while the incremental matrix decomposition effect is close to the optimized item-based collaborative filtering (Knni++), and they are both significantly better than the original item-based collaborative filtering algorithm (Knni). This indicates that the introduction of auxiliary information such as lyrics and tags can improve the recommendation accuracy. The Recall of all algorithms increases as the recommendation list increases because the number of items preferred by users in the test set remains the same as the recommendation list increases, while items preferred by users in the test set are more likely to appear in the recommendation list. Moreover, the F1 score of most algorithms increases and then decreases, and the optimal value is obtained around $N = 20$. In addition, this paper shows the effect of changing the number of topics in the topic model on the results of the COLDA algorithm. 0(a) (b) shows the accuracy, Recall, and F1 score of the COLDA algorithm when the other parameters are taken as default values and the number of recommended items N and the number of topics K are changed. When N increases, the accuracy of the COLDA algorithm decreases in general, which is the same as expected. The number of topics K has little effect on the metrics in general, and the Recall increases slowly when K reaches 20 and then increases.

In this paper, we tested the response time delay and concurrency on APP side by using the virtual user mechanism of Load Runner to generate virtual users for the test and then execute the prerecorded script to simulate the real users of the system to send requests to the system, and let the number of virtual users increase from 50, 100, to 200, all the way up to more than 500 and, at the same time, send out requests to a certain page. Then, we record the average time for all users to get a response and for the page to be loaded so as to get the concurrency data of the system's response time. This section shows the concurrency test of the music recommendation system by requesting a recommendation list, artist category list query, and music daily list query as examples, and the test results of other interfaces are similar. Figure 7 shows the average response time of requesting recommendation lists by different concurrent users. It can be seen that the response time latency is generally low and all requests are successful when concurrent users do not exceed 300. When the concurrent users reach 500, the average response time remains within 2 seconds and the request failure rate does not exceed 2%.

In this article, the accuracy of the system recommendation algorithm is evaluated using the real-world Last.fm dataset and the Movie Lens dataset, both of which have labeled items and time-stamped interaction records.

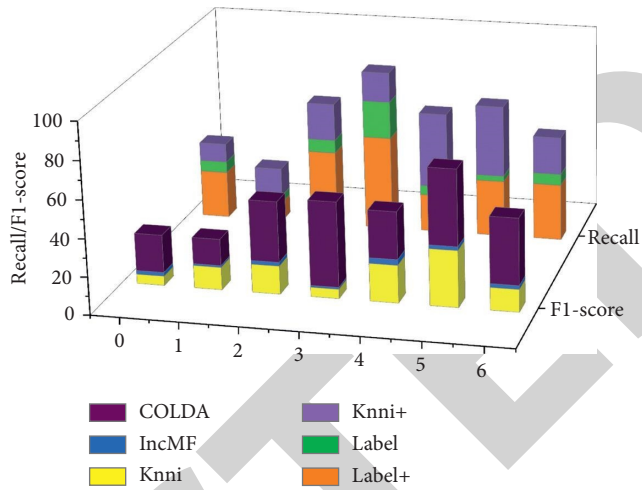

Figure 6: Comparison of datasets, Recall.

The two datasets are first statistically analyzed below. The two datasets are summarized in Figure 8. This dataset is a record of music played by users of the Last.fm website, which has 992 different users and 16.98 million play records involving 180,478 different music tracks. Each play record is time-stamped (to the second) and spans the period from February 2005 to September 2013. In this paper, we crawled the tags of this music using the Last.fm API and ended up with 68,756 different tags. For each tag, the paper calculated the amount of music tagged by it and the total number of times it was used. The analysis shows that there is a clear long-tail distribution of tag usage (Power Law Distribution), with the most tagged songs being "rock," tagged a total of 53,181 songs. In contrast, 13,132 tags were used for only one song, 336 tags were used for more than 1000 songs, and 31 tags were used for more than 10000 songs. The scatter plot (logarithmic coordinates) of the amount of music used by tags versus the number of tags approximates a straight line, which shows that they follow a long-tailed distribution. The distribution of the total number of tags used is similar to the long-tail distribution but is more balanced than the long-tail distribution. The Recall of all algorithms increases as the recommendation list increases because the number of items preferred by users in the test set remains the same as the recommendation list increases, while the items preferred by users in the test set are more likely to be included in the recommendation list, as shown in Table 1.

For both datasets, the interaction record data are sorted by time stamp in this article. The first 50% of the recorded data is used for training, and the second 50% is used as the test set. In this paper, we compare the effectiveness of three recommendation engines, including the system implementation of Collaborative Online Topic Model (COLDA), Incremental Matrix Decomposition (Inc MF), and Improved Incremental Item-Based Collaborative Filtering (Knni++), to find the best parameters using grid search, and the unoptimized item-based collaborative filtering (Knni) was used as a benchmark. For the online collaborative topic

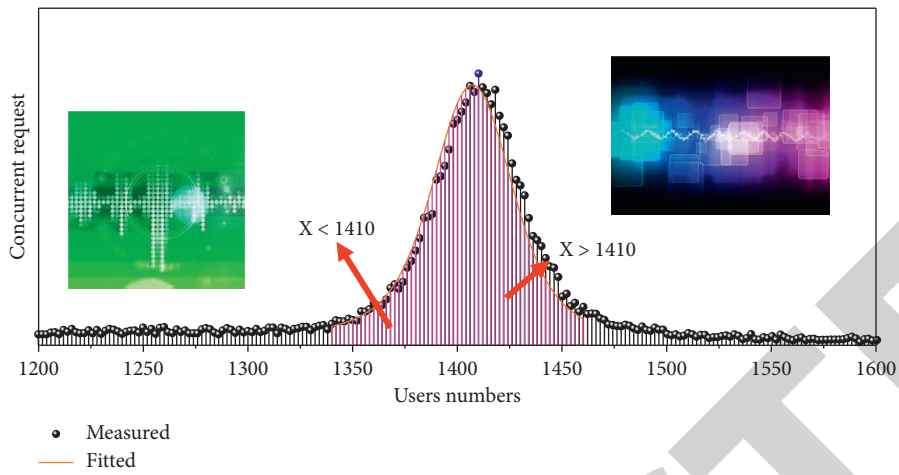

Figure 7: Concurrent request recommendation list test results.

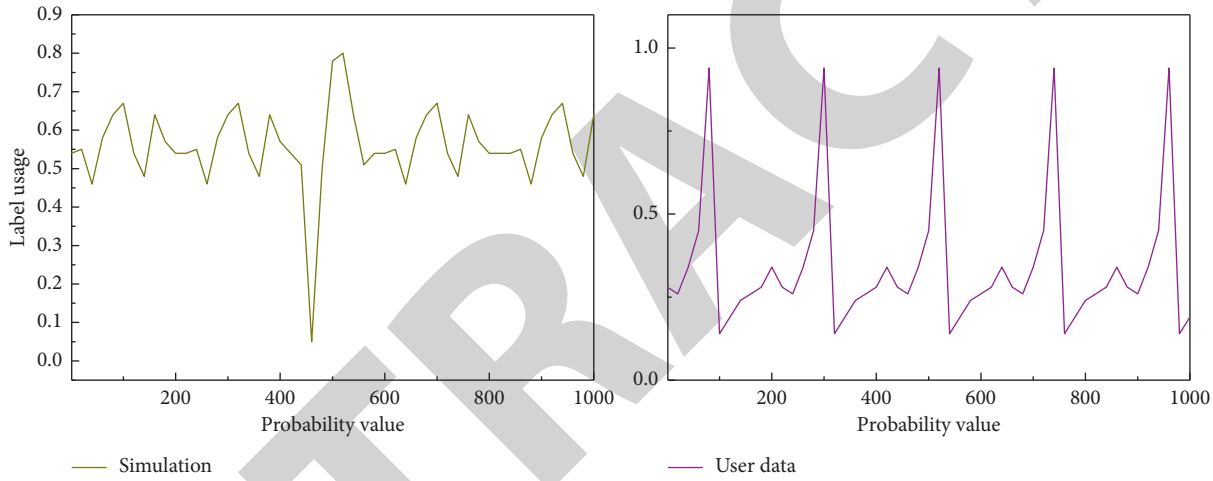

Figure 8: Distribution of dataset label usage.

Table 1: Recall of all algorithms.

Algorithms	List number	Time set/s	Recall time/s
COLDA	2	2	1.7
Inc MF	3	2	1.9
Knni++	2	2	2.4
Knni	1	2	2.5
Blank	4	2	2.7

model (COLDA) algorithm, the default parameters are set as the number of topics $K = 20$, the learning rate $x = 0.05$, the user regularization factor $u = 0.01$, and the item regularization factor $i = 10$. For the Incremental Matrix Decomposition (Inc MF) algorithm, the feature dimension is $K = 50$, the learning rate $x = 0.03$, the user regularization factor $u = 0.02$, and the item regularization factor $i = 0.02$. For the improved item-based collaborative filtering, the similarity weight based on user behavior is $\alpha = 0.4$, the similarity weight based on song tags $\beta = 0.4$, the similarity weight based on song lyrics $\gamma = 0.2$, and the number of item similar neighbors $M = 200$. For all algorithms, the default number of recommended items for each user is $N = 20$.

5. Conclusion

This paper first introduces the configuration of the test environment and then tests the recommendation engine on real datasets such as Last.fm and Movie Lens, analyzes the features of the two datasets, finds that they both obey the long-tail distribution, then compares the online collaborative topic model, incremental matrix decomposition, and improved item-based collaborative filtering algorithms, and finds that the COLDA algorithm is more accurate and better than other algorithms and the system has an obvious effect on the improvement of the item-based collaborative filtering algorithm. It is demonstrated that the online collaborative topic model algorithm outperforms the incremental matrix decomposition and item-based collaborative filtering algorithm, and the labels under the topics found by the collaborative topic model are shown. Then, a large number of test cases are designed for each function of the system, a comprehensive black-box test is performed, the main user interface of the system is demonstrated, and finally, a nonfunctional test is performed for the system. The recommendation engine module of this paper was

designed and optimized using Spark to implement incremental matrix decomposition on data streams, online collaborative topic model, and improved item-based collaborative filtering algorithm. In the system testing section, the functionality and performance of the system are tested, and the recommendation engine is tested with real datasets, the discovered music themes are demonstrated, and the test results are analyzed in detail. This paper solves the problems of the offline recommendation system, is able to update the recommendation model in real time for user interest changes, returns new recommendation results, adds real-time statistics and ranking of music popularity trends, and improves user experience.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The author declares that there are no conflicts of interest in this article.

Acknowledgments

This work was supported by scientific research program funded by Shaanxi Provincial Education Department, Research on the Integration and Innovation of Shaanxi Guanzhong Folk Music and Pop Music-Taking Huayin Laoqiang as an example, under program no. 17Jk0260, and educational science research project funded by Weinan Normal University, Reform and Implementation of Quality Education for Music Majors under the New Curriculum Standard, under program no. 2019JYKX011.

References

[1] M. Mohammadi, M. Lakestani, and M. H. Mohamed, "Intelligent parameter optimization of savonius rotor using artificial neural network and genetic algorithm," *Energy*, vol. 143, pp. 56–68, 2018.

[2] M. Qiu, S.-Y. Kung, and K. Gai, "Intelligent security and optimization in edge/fog computing," *Future Generation Computer Systems*, vol. 107, pp. 1140–1142, 2020.

[3] Q. Wu and R. Zhang, "Beamforming optimization for wireless network aided by intelligent reflecting surface with discrete phase shifts," *IEEE Transactions On Communications*, vol. 68, no. 3, pp. 1838–1851, 2020.

[4] J. H. Lee, J.-Y. Song, D.-W. Kim, J.-W. Kim, Y.-J. Kim, and S.-Y. Jung, "Particle swarm optimization algorithm with intelligent particle number control for optimal design of electric machines," *IEEE Transactions On Industrial Electronics*, vol. 65, no. 2, pp. 1791–1798, 2018.

[5] T. Liu, D. Zhang, H. Dai, and T. Wu, "Intelligent modeling and optimization for smart energy hub," *IEEE Transactions On Industrial Electronics*, vol. 66, no. 12, pp. 9898–9908, 2019.

[6] B. Gao, L. Guo, Q. Zheng, B. Huang, and H. Chen, "Acceleration speed optimization of intelligent EVs in consideration of battery aging," *IEEE Transactions On Vehicular Technology*, vol. 67, no. 9, pp. 8009–8018, 2018.

[7] Y. Jia, C. Ye, and Y. Cui, "Analysis and optimization of an intelligent reflecting surface-assisted system with interference," *IEEE Transactions On Wireless Communications*, vol. 19, no. 12, pp. 8068–8082, 2020.

[8] V. D. P. Souto, R. D. Souza, B. F. Uchoa-Filho, A. Li, and Y. Li, "Beamforming optimization for intelligent reflecting surfaces without csi[J]," *IEEE Wireless Communications Letters*, vol. 9, no. 9, pp. 1476–1480, 2020.

[9] R. Xiao, J. Li, and T. Chen, "Modeling and intelligent optimization of social collective behavior with online public opinion synchronization," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 8, pp. 1979–1996, 2019.

[10] X. Jin, "Research on optimization of intelligent warehousing business of state grid based on genetic algorithm," *Journal of Computers*, vol. 13, pp. 1164–1170, 2018.

[11] N. Thakur, Y. K. Awasthi, M. Hooda, and A. S. Siddiqui, "Adaptive whale optimization for intelligent multi-constraints power quality improvement under deregulated environment," *Journal of Engineering, Design and Technology*, vol. 17, no. 3, pp. 490–514, 2019.

[12] L. Chen, M. Ma, and L. Sun, "Heuristic swarm intelligent optimization algorithm for path planning of agricultural product logistics distribution," *Journal of Intelligent & Fuzzy Systems*, vol. 37, no. 4, pp. 4697–4703, 2019.

[13] H. Hu, J. Yang, and C. Tian, "Research on intelligent optimization of parameters of deurring process with fluid-impact to automobile master cylinder cross hole," *Journal of Intelligent & Fuzzy Systems*, vol. 35, no. 1, pp. 315–323, 2018.

[14] K. Anjaria and A. Mishra, "Thread scheduling using ant colony optimization: an intelligent scheduling approach towards minimal information leakage," *Karbala International Journal of Modern Science*, vol. 3, no. 4, pp. 241–258, 2017.

[15] Z. Wang and X. Luo, "Modeling study of nonlinear dynamic soft sensors and robust parameter identification using swarm intelligent optimization CS-NLJ," *Journal of Process Control*, vol. 58, pp. 33–45, 2017.

[16] S. K. Tamang and M. Chandrasekaran, "Integrated optimization methodology for intelligent machining of inconel 825 and its shop-floor application," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 39, no. 3, pp. 865–877, 2017.

[17] W. Chen, Z. Chen, X. Ma, Y. Chi, and Z. Li, "Secrecy rate optimization for intelligent reflecting surface aided multi-input-single-output terahertz communication," *Microwave and Optical Technology Letters*, vol. 62, no. 8, pp. 2760–2765, 2020.

[18] F. Zhao, G. Li, R. Zhang et al., "Swarm-based intelligent optimization approach for layout problem," *Multimedia Tools and Applications*, vol. 76, no. 19, pp. 19445–19461, 2017.

[19] C. Shao, Z.-J. Zhang, X. Ye, Y.-J. Zhao, and H.-C. Sun, "Modular design and optimization for intelligent assembly system," *Procedia Cirp*, vol. 76, pp. 67–72, 2018.

[20] M. Kovalský and B. Mičieta, "Support planning and optimization of intelligent logistics systems," *Procedia Engineering*, vol. 192, pp. 451–456, 2017.

[21] C. Lv, H. Wang, B. Zhao et al., "Cyber-physical system based optimization framework for intelligent powertrain control," *Sae International Journal of Commercial Vehicles*, vol. 10, no. 1, pp. 254–264, 2017.

[22] X. Wang, W. Sun, E. Li, and X. Song, "Energy-minimum optimization of the intelligent excavating process for large cable shovel through trajectory planning," *Structural and Multidisciplinary Optimization*, vol. 58, no. 5, pp. 2219–2237, 2018.

[23] S. Qi, Y. Lu, W. Wei, and X. Chen, "Efficient data access control with fine-grained data protection in cloud-assisted IIoT," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2886–2899, 2021.

[24] A. Zielonka, A. Sikora, M. Wozniak, W. Wei, Q. Ke, and Z. Bai, "Intelligent Internet of things system for smart home optimal convection," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4308–4317, 2021.

[25] W. Wang, Z. Gong, J. Ren et al., "Venue topic model–enhanced joint graph modelling for citation recommendation in scholarly big data," *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, vol. 20, no. 1, pp. 1–15, 2020.

[26] H.-x. Hu, Z.-w. Jiang, Y.-f. Zhao, Y. Zhang, H. Wang, and W. Wang, "Network representation learning-enhanced multi-source information fusion model for POI recommendation in smart city," *IEEE Internet of Things Journal*, vol. 1, 2020.