Streaming process based on Spark and Kafka and its case application

Jiayi Fang

School of Computer and Information Engineering, Henan University of Economics and Law, Zhengzhou, Henan, 450000, China

631402070315@mails.cqjtu.edu.cn

Abstract. E-commerce platforms gradually integrate into our lives. Online shopping has become a daily habit for people. People who shop online will generate online shopping records in real time. These online shopping records are often massive. E-commerce platforms use relevant big data technology to conduct statistics on real-time user data. In response to the current demand for e-commerce platforms for big data real-time computing technology, this article provides an idea of big data streaming computing, a framework for combining two big data technologies, Spark and Kafka, to process streaming data. This framework uses Kafka to send real-time data to SparkStreaming for calculation. Then the obtained data is visualized. This article applies these two big data technologies to simulate and implement an application case of an e-commerce platform. This case design and implementation enables real-time statistics of the click volume of a mobile phone brand on an e-commerce platform and dynamic display of streaming data on the Web.

Keywords: Spark, Kafka, flow calculation.

1. Introduction

With the development of science and technology, the continuous integration of information technology into society has triggered an explosive growth of information. The background that online shopping has become the current mainstream shopping method. The overall trend of e-commerce platform transaction volume is rapid growth. This process will accumulate more user review data. These data reflect more product defect information and the actual needs of users for improving product functionality [1]. The data level may reach TB, PB, or even FB. Therefore, corresponding big data technologies are needed to store and process a large amount of information. Hadoop, as one of the earliest big data processing frameworks, can store and calculate massive amounts of data. The core of Hadoop design is Hadoop Distributed File System(HDFS) and MapReduce. Hadoop has the advantages of high reliability, high scalability, efficiency, and high fault tolerance. Compared to its advantages, the disadvantages of Hadoop are also very obvious. The MapReduce computing model is a dataset-based computing model. The data input and output method is to load data from physical storage, then operate the data, and finally write it to physical storage devices [2]. This disk-based computing model is suitable for handling most batch-processing tasks. The object corresponding to this batch processing method is generally static data. That is to say, a large amount of data can be processed in batches in sufficient time. However, if a large amount of data is calculated in real-time, such as viewing real-time clicks in the e-commerce field. So the response time obtained usually needs to be at the second level. This is where the above technology is no longer applicable. At this point, flow calculation can be adopted.

Stream computing is an important computing mode for big data. Unlike traditional batch computing based on determining the size of data, stream computing has the characteristics of unlimited data size, continuous, fast, and unordered data arrival, unstable data, and diverse data processing [3]. The process of streaming processing includes collection, calculation, and query. There are various streaming processing frameworks available for streaming processes, such as Storm, Flink, and so on. This article discusses Spark Streaming built on Spark. The data source of Spark Streaming can be obtained from Kafka. This article will introduce Spark and Kafka's stream processing frameworks and use them to simulate real-time click-through traffic of e-commerce platform products.

2. Techniques for building a stream process

This section will briefly introduce the Spark and Kafka technologies involved in the data processing process and the data visualization technology to be used after processing.

2.1. Spark

Spark is a general-purpose parallel computing framework similar to MapReduce that is open source by UC Berkeley AMP Lab, taking into account the characteristics of distributed parallel computing models and memory-based computing [4].

2.1.1. Features and applications of Spark. Spark is fast. Spark uses a combination of disk and memory to store intermediate results in memory without repeatedly dropping the disk. To support cyclic data and memory computing, Spark uses DAG to reduce secondary IO for shuffles and disks. In resource application and scheduling, Spark is based on coarse granularity, while MapReduce is based on fine granularity [2]. Spark will apply for all resources in advance, and there is no need to apply for resources when running tasks. Each task in Spark eliminates the time consumption of starting and destroying processes based on threads.

Spark has many flexible applications. Spark establishes multiple programming language APIs that can provide multiple language work environments for various developers and provides a shell environment for interactive queries. Spark can also provide various technologies, such as SQL queries, streaming computing, and machine learning libraries. It allows developers to implement stronger metafunctions.

Spark has more compatibility. Spark has multiple operating modes, which can be run independently or based on yarn. There is also a more flexible version of Mesos, which implements two forms of allocation: early allocation and on-demand allocation. It also runs based on multiple data sources, such as Hbase, Hive, HDFS, and so on. Spark can be applied to scenarios based on the above characteristics, such as complex batch data processing, the interactive query of historical data, real-time processing of data streams, and so on.

2.1.2. Spark ecology. The data processing process of big data technology is divided into several steps: integration, storage, distributed computing, analysis, and visualization. Spark is located in the stage of distributed computing.

The core of the Spark ecosystem is the Spark Core, which reads data from persistence layers such as HDFS, Amazon, S3, and HBASE. The job manager completes application calculations through the MESS, YARN, and Standalone instructions placed in Spark. These Spark applications can come from Spark Submit's batch processing, Spark Streaming's real-time stream count Data processing, interactive queries in Spark SQL, BlinkDB tradeoff queries, machine learning in Spark MLlib/MLbase, graph processing in Graphx, and mathematical calculations in SparkR [5].

2.1.3. Spark architecture. First, enter the program from the SparkContext created on the Driver side of Spark. During its initialization, a DAGScheduler and a TaskScheduler are created, respectively. DAG Scheduler divides job tasks into multiple stages. Then, each stage is divided into specific tasks. These tasks will be submitted to the task scheduler for task scheduling. The resources required during this process are requested from the cluster manager.

The values passed to different environment variables in SparkContext can lead to different operation modes. This allows Spark to run in local mode or pseudo-distributed mode, as well as the standalone mode that Spark comes with. Mesos or Yarn can also perform resource scheduling.

2.1.4. Resilient Distributed Datasets(RDD). As the core of the ecosystem, the core abstraction of Spark Core is RDD. RDD is a collection of read-only objects distributed in a cluster. Spark records a series of transformations that create RDDs. If a partition or part of the data of an RDD is lost, fault tolerance can be performed based on the reconstruction of its parent's RDD. This strategy is called lineage [6].

The operation of the RDD operator in Spark is divided into three steps: input, run, and output. Specifically, RDD is generated after reading data from a data source in memory. The generated RDD is then converted into a new RDD through various conversion operators. This can also cache intermediate results into memory. The job is then submitted using the Action operator. After data processing, it is stored in an external data space.

2.2. Spark streaming

Streaming computing processing also includes data collection, computation, and result presentation. But the entire process is real-time, which means improving the response requirements to the second level or even faster. The processed data objects also become data streams abstracted from unbounded data sets.

There are various frameworks for processing massive data streams. This article introduces the Sparkstreaming framework based on Spark. Spark Streaming is a streaming batch processing engine based on Spark, which can achieve high throughput and fault-tolerant real-time streaming data processing. It can seamlessly connect with RDD operators, machine learning, SparkSQL, and graphics and image processing frameworks [7].

2.2.1. Screaming structure. Spark Streaming is based on the API and memory computing provided by Spark, which splits the continuously input data stream into small batches for calculation and then generates new micro-batches. This is called incremental computation of data streams.

2.2.2. Discretized Stream(DStream). Spark Streaming uses an RDD sequence discretization data stream DStream as its data format.RDD is an abstract class, with DStream as an abstract subclass that implements functions. DStream separates data streams into RDDs and converts these RDDs into new RDDs.

2.2.3. Window operation. The concept of the time interval is not reflected in the above figure. Real-time refers to processing data in relatively short time intervals. However, from a microscopic perspective, these shorter time slices also include some RDDs processed in a shorter time. Before generating a new RDD, the processed RDD is aggregated within a time slice. This aggregation process is called a sliding window operation, and the window will slide according to the time slice.

2.3. Kafka

Apache Kafka is a distributed messaging middleware. It is a distributed publish-subscribe message system with high throughput, storing messages in a fault-tolerant manner. It has high performance, persistence, multi-copy backup, and horizontal scalability [8].

2.3.1. Kafka architecture. Before proceeding with the Sparkstreaming calculation process, data must first be collected. In e-commerce platforms, a large number of logs are generated in real-time every day. Kafka, similar to intermediaries, can be used to buffer data and provide it to the target system. Producers' messages are published to the primary agent of the partition, saved to the broker in Topics, and subscribed to by Consumers.

Producers serialize keys and values into byte arrays during the process of sending data. Kafka provides a flexible serialization process for transmitting different data types. The specific partition in the Topics to which the Producer sends messages can be specified. If not specified, the partition will be randomly selected. The agent will respond to the corresponding information after receiving the message. Kafka can also equip each partition with replicas to avoid proxy failures.

Consumers can read data from it. Consumers can read data from multiple agents simultaneously. The number of Consumers can also be multiple. The same group of Consumers forms a Consumer group. For feedback on Consumers' consumption of data, the submission offset can be used.

2.3.2. The combination of Spark and Kafka. On the Producer side, OGG For BigData transfers data to Kafka. On the Consumer side, Spark Streaming is used to continuously extract data from Kafka's agents for data processing and computation [7]. The process of pulling data has two modes:

The first mode is the Receiver mode. Use Kafka to continuously obtain and store data from the receiver's Zookeeper asynchronous thread. The reading time and offset can be configured through relevant parameters. After the Batch task is triggered, transfer the data to the remaining Executors for processing. The offset will be updated after processing is completed.

The second mode is the Director mode. This mode omits the above operation of connecting to the Zookeeper and directly reads data from Kafka. This is a regular query for the latest offsets from Kafka's themes and partitions. The data in each batch is processed within the defined offset range [8]. Let the Executor read the Partition data calculation.

2.4. Other technologies of the project

In this paper, Flask and Echarts technologies are used for data visualization.

2.4.1. Flask. Flask is a microframework developed based on Python and relying on the Jinja2 template rendering engine and Werkzeug WSGI routing service component as the core. It has good scalability and compatibility, which can help users quickly implement a website or Web service [9].

2.4.2. Apache ECharts(ECharts). There are a variety of tools for implementing data visualization. This article uses ECharts, an open-source JavaScript-based visualization chart library. It is very compatible with various browsers. It also has rich interactive functions. This allows it to highly customize data visualization charts [10].

3. Project design process

This article describes the process of building a stream using Spark, Kafka, and their combination. Now, a simple e-commerce case is simulated using the above technology.

In e-commerce, it is often necessary to make statistics on the real-time click volume of products. The statistical results can obtain the real-time popularity of the product and help make subsequent decisions. So a real-time click-counting system was created for certain products. Therefore, this article uses Spark and Kafka to simulate the real-time click volume of a different mobile phone brand on an e-commerce platform. The general process of the project is shown in Figure 1.

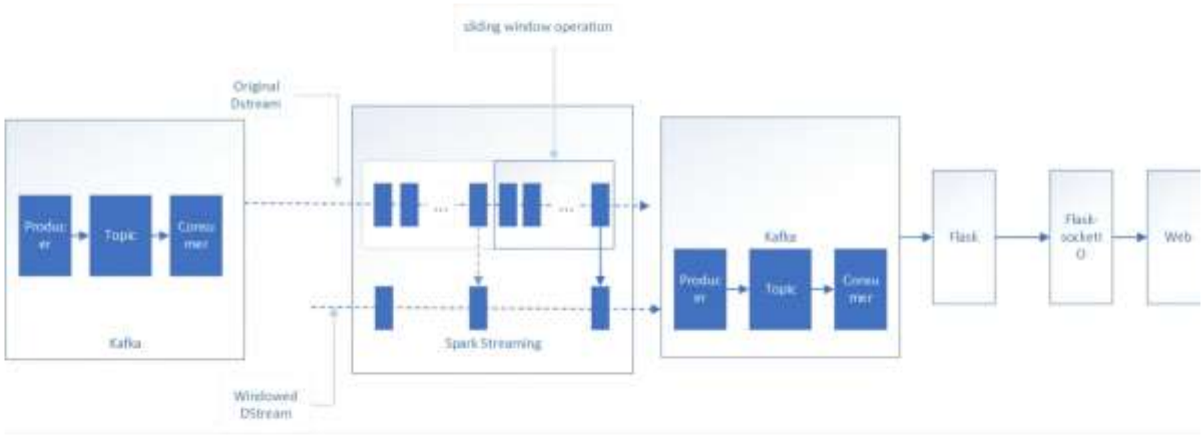


Figure 1. The general process of the project.

In Figure 1, the Kafka Producer sends the read user data to a topic. Spark Streaming then performs sliding window processing in the form of DStream from the mobile click data source formed by subscribing data from the first Kafka. The small blue rectangle above represents an original RDD in a time slice. Then the RDD enclosed by the sliding window is transformed to obtain a new RDD, which is the small blue box below. This is also the word frequency statistics of mobile phone brand clicks during window time. The DStream generated below is sent to a new topic in Kafka. Flask is used to build an application to subscribe to this topic. Flask Socket then pushes real-time phone clicks to the Web client for display.

3.1. Project process

Firstly, a random CSV file is generated using Python. This file contains click data for 10000 mobile phone brands. Then have Kafka Producers filter and read the relevant phone brand data in the CSV. The data is sent to the theme 'Mobile'. The data in the 'phone' is read and processed by Spark Streaming every two seconds in a sliding window. After processing, the processing results are sent to the new topic 'resultPhone'. Then the Flask Socket IO in Python is used to push real-time to the web client. The JavaScript library on the website is called to receive data and dynamically visualize it.

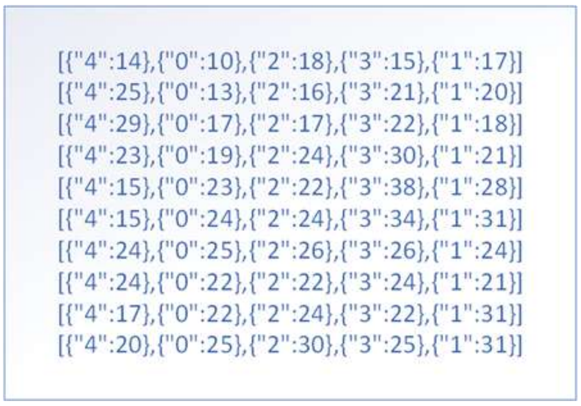
3.2. Project-specificity steps

The specific process of the project is divided into two parts: background data processing and data visualization.

3.2.1. Calculation of stream data. The phone entry in the above-generated CSV file includes natural numbers from 0 to 4. In ascending order, they represent five mobile phone brands: Huawei, Samsung, Apple, Oppo, and Xiaomi. A project was created in Pycharm. A Producer created in the project reads the numbers in the phone column and sends them to the 'Phone' topic.

Then an IDEA project 'KafkaParams' Consumer is created. It subscribed to the data stream in 'phone' in the project. Consumer-related parameters are configured. Then an instance of the SparkConf project, 'SparkConf', is created and set to local run mode. An instance 'StreamingContext' using SparkConf's stream project is created and specified to process data every 1 second.

The newly created 'KafkaParams' and' StreamingContext 'are combined together. A window reading data source lineMap is created. A 2s sliding window is set to read the Dstream data source sent every 1s. A new Producer is created. The result of calling the relevant operator for word frequency statistics is sent to the new Topic 'resultPhone'. The partial running results of the backend data calculation are shown in Figure 2.



```
[{"4":14},{"0":10},{"2":18},{"3":15},{"1":17}]
[{"4":25},{"0":13},{"2":16},{"3":21},{"1":20}]
[{"4":29},{"0":17},{"2":17},{"3":22},{"1":18}]
[{"4":23},{"0":19},{"2":24},{"3":30},{"1":21}]
[{"4":15},{"0":23},{"2":22},{"3":38},{"1":28}]
[{"4":15},{"0":24},{"2":24},{"3":34},{"1":31}]
[{"4":24},{"0":25},{"2":26},{"3":26},{"1":24}]
[{"4":24},{"0":22},{"2":22},{"3":24},{"1":21}]
[{"4":17},{"0":22},{"2":24},{"3":22},{"1":31}]
[{"4":20},{"0":25},{"2":30},{"3":25},{"1":31}]
```

Figure 2. Running results of the background data calculation section.

Figure 2 represents the word frequency statistics results in the new Topic in Josn format. Taking the first row as an example, the data in the first row represents the real-time click-through volume of Xiaomi brand phones in two seconds, which is 14 times, Huawei 10 times, Apple 18 times, Oppo 15 times, and Sunsumg 17 times. The remaining number of lines represents the same meaning. The number of console rows continues to increase in the continuous calculation of data streams.

3.2.2. Data visualization. Then a new Consumer created in Pychar is used to read word frequency statistics results in 'resultPhone'. Flask Socket is created to receive the results just read. The received results are pushed in real-time to the web client. The JavaScript library that is then called is used to write HTML display files. The real-time data pushed by SocketIO is visualized using the line chart. This chart shows the results of streaming processing. Dynamic data transformation is also performed every two seconds. The display effect is shown in Figure 3.

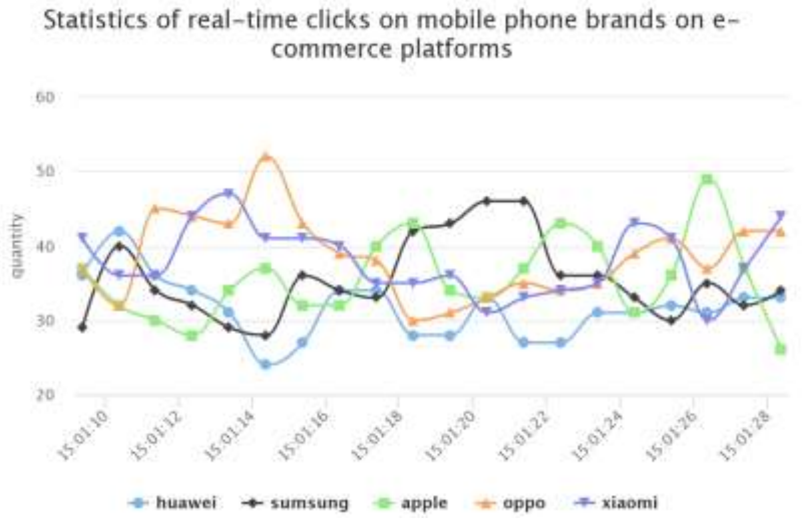


Figure 3. Data visualization of projects.

Figure 3 visualizes the statistical results of Spark Streaming. Display the real-time clicks of mobile phones of all brands in the current period in the form of a line chart, which changes every two seconds.

Figure 3 visualizes the statistical results of Spark Streaming. The results of real-time statistics show the real-time hits of mobile phones of all brands in the current period time in the form of a line chart. Figure 3 will also change every two seconds.

4. Conclusion

Therefore, this article introduces Sparks based on MapReduce optimization. It also introduces the functional framework and data format of Spark. Then it introduces the streaming computing framework in Spark that can meet real-time processing requirements. The data source for the calculation process is Kafka. Then this article simulates an application case where Kafka and Spark are combined on an e-commerce platform. In this case, this article introduces that the CSV file generated by data is produced by Kafka into a topic. The data stream generated by subscribing to this topic is sent to Sparkstreaming for calculation and then sent to a new topic for consumption. The consumption results are sent to the Web using a socket for data visualization.

However, in this experiment, there were flaws in the generation of data. Firstly, the data in this article are randomly generated using Python rather than generated by e-commerce users. The acquisition of real data requires corresponding interfaces on e-commerce platforms. Secondly, real e-commerce data does not directly generate the data form of this article. Before processing production data, complex logs generated by users in real time should first be converted into a standardized and readable format. Therefore, before using Kafka for production and consumption, the collected raw data can be processed first. The collected logs are processed using Flume. The processed results are then sent to Kafka. At this point, the data obtained by Kafka will be processed and displayed using the process mentioned in this article.

References

[1] Suo Hongsheng. Design and Research of Big Data Mining System Based on E-commerce Platform [J]. Internet Weekly, 2023 (06): 29-31
[2] Quan Zhaoheng, Li Jiadi. Innovation from Hadoop to Spark Technology [J]. Computer Knowledge and Technology, 2019, 15 (08): 265-268. DOI: 10.14004/j.cnki.ckt.2019.0823
[3] Meng Yunfei. Research on Key Technologies of Big Data Streaming Computing [J]. Heilongjiang Science, 2022,13 (14): 55-57
[4] Song Lingcheng. Comparative Analysis of Flink and Spark Streaming Streaming Computing Models [J]. Communication Technology, 2020,53 (01): 59-62
[5] Jiang Yongdu, Cheng Desheng, Zhao Zhiwu, Wang Li, Jiang Feng. Big Data Computing Platform Based on Spark Framework [J]. Network Security Technology and Application, 2020 (03): 65-66
[6] Wu Xindong, Ji Shengdong. Comparison of MapReduce and Spark for Big Data Analysis [J]. Journal of Software, 2018,29 (06): 1770-1791. DOI: 10.13328/j.cnki.jobs.005557
[7] Gao Zongbao, Liu Limei, Zhang Jiaming, Song Guoxing. Kafka Offset Reading Management and Design in the Park Platform [J]. Software, 2019,40 (07): 118-122
[8] Ye Huixian. The practice of Building a Data Center Processing Engine Based on Spark Streaming and Kafka [J]. Network Security Technology and Application, 2023 (03): 51-53
[9] Chen Jiafa, Huang Yujing. Application of Flask Framework in Data Visualization [J]. Fujian Computer, 2022,38 (12): 44-48. DOI: 10.16707/j.cnki.fjpc.2022.12.009
[10] Jing Guowei, Huang Dachi. Research on Data Visualization Based on ECharts [J]. Western Radio and Television, 2022,43 (20): 227-230+234