Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 7451582, 11 pages
https://doi.org/10.1155/2022/7451582

Research Article

Novel Big Data Networking Framework Using Multihoming Optimization for Distributed Stream Computing

G. Sanjiv Rao[1], **J. Armstrong Joseph**[2], **Gaurav Dhiman**[3], **Hussien Sobahi Mohammed**[4], **Sheshang Degadwala**[5] and **R. Bhavani**[6]

[1]*Department of Computer Science and Engineering, Aditya College of Engineering and Technology, Surampalem, Andhra Pradesh, India*
[2]*Department of Computer Science and Engineering, Sri Venkateswara College of Engineering and Technology (Autonomous), Chittoor, 517127 Andhra Pradesh, India*
[3]*Department of Computer Science and Engineering, Graphic Era Deemed to Be University, Dehradun, India*
[4]*University of Gezira, Wad Medani, Sudan*
[5]*Department of Computer Engineering, Sigma Institute of Engineering, Vadodara, India*
[6]*Department of CSE, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai 600124, India*

Correspondence should be addressed to Hussien Sobahi Mohammed; hussiensobahi@uofg.edu.sd

Received 19 April 2022; Revised 4 June 2022; Accepted 26 June 2022; Published 9 July 2022

Academic Editor: Amrit Mukherjee

One of the main technologies for big data networking framework is online multihoming optimization that is large-scale dimension table association technology in a distributed environment. It is often used in applications like real-time suggestion and research. Big data is concerned with the quality of large datasets that are distributed. These datasets demand sophisticated network technologies to adequately transmit massive share files. Dimension table association is the process of integrating multihoming stream data with offline stored dimension table data and executing data processing using novel big data frameworks, as described in this study. The current technological options for dimension table connection are assessed first, followed by accompanying optimization technologies and the design route of mainstream distributed engines. The dimension table data query is the one that has been optimized with the greatest performance. Nonetheless, the typical optimization approach is influenced by the dimension, table size, and the design route of the mainstream distributed engine—limits on data flow rate. Second, due to the limitations of existing optimization technologies for the overall consideration of the cluster in a distributed environment, a computing model suited for hybrid computing of offline batch data and real-time streaming data is provided, followed by a single-point reading. Dimension table data, the dimension table associated data technique for distribution and calculation after segmentation, and optimization of the dimension table associated calculation logic adapt to a larger dimension table scale and are no longer restricted to data connections. Since optimizing the query of dimension table, data is employed to reduce the I/O overhead and delay caused by querying dimension table in big data. Finally, both the suggested and standard dimension table association technologies are implemented on the Apache Flink stream computing engine. Through trials, the throughput and latency on data created by Alibaba's "Double Eleven" are compared, demonstrating the usefulness of dimension table association techniques for Distributed Stream Computing optimization by utilizing multihoming networks.

1. Introduction

The amount of data generated in the network is continuously increasing as the Internet develops and becomes more popular. One of the main technologies for big data networking framework is online multihoming optimization that is large-scale dimension table association technology in a distributed environment. It is often used in applications like real-time suggestion and research. Large companies and network infrastructure are gradually shifting to multihoming as a way to get the most out of their supplier connections. To enhance quality over many ISP networks, multihomed

terminal can now use a spectrum of products route control tools. The present data scale has rendered traditional data processing technologies ineffective. Developers have proposed big data processing technology to extract the potential value of these vast data [1]. Big data processing technology has gone through three generations of computing engine changes due to the continuous growth of big data processing technology and the constantly changing of computing needs. Apache Hadoop, which employs MapReduce to process massive data, is the first significant data computing engine generation [2, 3]. The computing model based on physical storage is the distinguishing feature of this generation of computing. The throughput of this type of computation is really high [4]. Despite this, because each step of the computation is copied to physical storage, the discrepancy between memory-based computing speed and disc I/O overhead results in extremely high processing latency. This family of computer engines is best suited for batch processing jobs that do not necessitate high real-time performance [5]. During this time, the majority of big data analysis strategies are focused on offline analysis, which necessitates the collection of statistics and all data prior to inspection. A second-generation big data computing engine, Apache Spark, is an implementation of a second-generation big data computing engine that uses memory for batch computing. The significance of big data technologies can be defined as a piece of software that evaluates, organizes, and recovers information from exceedingly complicated and huge data sets that conventional data processing software could not cope. Compared with the first generation of technology, the distinguishing feature of this generation is that the calculation data is moved into the memory, and the calculation is performed based on the data in the memory [6]. This memory-based computing method dramatically reduces the latency caused by the I/O overhead of writing the results to disk at each step of the first-generation technology. However, since the technology is still based on the batch computing mode, the data processing of each batch has a specific time interval; so, it is still impossible to guarantee extremely low latency in the face of some real-time computing tasks.

Big data analysis techniques are rapidly coming closer to online analysis in this era, and real-time data processing is already in demand. Apache Flink, the third-generation big data computing engine, is a data processing engine that is fully based on stream computing [1]. Apache Flink's computing infrastructure can process hundreds of millions of messages or events per second with millisecond latency. Stream computing technology, represented by Apache Flink, is the preferred choice for real-time massive data analysis due to its high data processing capability and low latency. Stream computing technology is being used by an increasing number of firms to construct their own real-time data analysis engines to replace traditional data warehouse research [7], in processing and interchange databases, clustering leverage datasets with comparable interconnections, initial setup, and procedures. As a corollary, when heterogeneous connections handshake together again to evaluate big data, it restricts big data capabilities and needlessly complicates. It also causes a range of latency challenges. In such data processing and analysis, the real-time stream data is often insufficient in information, and it is necessary to correlate with the data stored offline to expand the data attributes. Dimension table data is data that is stored offline, and dimension table association is the act of associating stream data with dimension table data. One of the most important technologies in today's online extensive data analysis is dimension table association. In a distributed setting, dimension table association technology optimization focuses on querying dimension table data to reduce the I/O overhead and delay generated by querying dimension table data. Through asynchronous I/O technology, the data query optimization associated with dimension tables primarily increases the number of queries per unit time. Furthermore, it accelerates the question by using data caching technology to cache the queried dimension tables in the memory of the computing nodes of the distributed engine. There are usually two caching modes: caching query results and caching the entire dimension table. The principle is shown in Figure 1. These two optimization methods based on computing nodes have their shortcomings. All data being cached cannot support large-scale dimension tables due to memory problems. In the method of caching query results, the processing time of each piece of uncached data is limited by the I/O capability of the database. It will increase linearly with the increase in the size of the dimension table data. When the data flow rate is low, the cache timeout mechanism to ensure data validity will cause each data query to point to the database, invalidating the local cache and reducing data processing efficiency.

Given the shortcomings of single-node optimization technology such as reduced I/O overheads, delays in query generation, and increased number of queries per time in dimension table, this paper first proposes a calculation model of novel big data framework that can be used for hybrid computing of offline stored data and real-time generated data and then designs a new optimization scheme to address numerous latency issues, and we employ a multihoming network strategy, which entails numerous networks interacting with a single network activity at the same time and exchanging database effectiveness to provide guidance of dimension table association technology based on this model. Its principle is shown in Figure 2.

This approach reads data from a single dimension database, separates it, distributes it among processing nodes, and associates it with stream data. Hence, the model of novel big data computing diverse networks concurrently interact with an unique network activity and assist in transferring database efficacy to provide dimension table connection technology which will further do all the processing like separation, distribution, processing, and association itself using the proposed approach. Each computer node only needs to cache a portion of the dimension table data in this optimization strategy, which enhances the dimension table data cache capacity and greatly reduces the consumption of the dimension table data query. This system integrates batch processing and stream computing technologies, as well as the study of offline batch data and real-time stream data in hybrid computing.

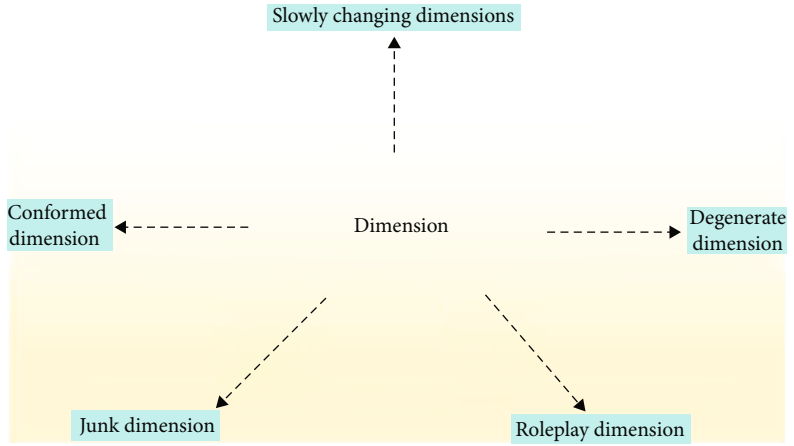

Figure 1: Traditional dimension table connection.

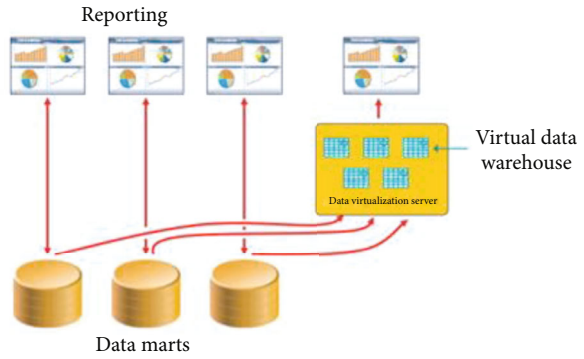

Figure 2: Optimized dimension table connection.

The main contributions of this paper are as follows:

(a) A computing model suitable for mixed computing of offline batch data and real-time streaming data is proposed. The computing model can process both streaming and batch data in a set of API (application programming interface) or process streaming data alone or batch data alone

(b) A dimension table associative data caching method is proposed, which reads dimension table data at a single point and distributes and calculates it after splitting, reducing the pressure on the database caused by dimension table data queries and improving the accuracy of the computing system in the cluster environment—the upper limit of dimension table size support by using a multihoming strategy for data distribution. Simultaneously, the calculation logic of dimension table association is improved, allowing the dimension table association technology to go beyond data connection

(c) The optimized dimensions in the stream computing engine Apache Flink, table association technology, and the traditional dimension table association technology are implemented. The dimension table association technology is verified through experiments. The experimental results show that this method can improve the throughput of computing tasks by 8-9 times compared with the traditional dimension table association method under the same conditions. Reduce the computing delay by more than 40% when the computing power is sufficient

Organization is as follows: the paper is outlined into several sections where Section 1 states about the Introduction of the research followed by Section 2 which discusses about the related work. Section 3 states about experimental analysis, and the ultimate sections discuss the conclusion of the work.

2. Related Work

In the past few years, there has been a lot of progress in the optimization research of dimension table association operations in distributed environments and the typical interaction between streaming and static data.

To solve the problem of massive data association computing, the literature proposed a MESHJOIN algorithm, which optimizes the connection process of continuous data flow and dimensional data in single-point computing using multihoming basis. Still, the algorithm is not efficient enough for memory allocation [8]. To this end, we proposed an improved algorithm to improve the memory allocation problem. Authors proposed an algorithm based on the idea of the block to improve the connection performance of the MESHJOIN algorithm [9, 10]. The MESHJOIN technique, which employs multithreaded concurrent connection technology, is illustrated by the author to provide a concept of its working. It achieves the optimal scheduling of join operations and relational R read operations, ensuring that the join algorithm's efficiency is maximized and the connection efficiency is further improved, according to engineering principles. Based on the MESHJOIN algorithm, the literature proposed the EHJOIN algorithm, which improved the traditional hash join method. It used the index to store some frequently used primary data in the memory, solving the problem of frequent disk access under high-speed data flow [11, 12]. The above algorithms are based on dimensional

data to optimize the connection process of dimension tables without considering the skew of streaming data. Based on this, the literature proposed the CSJR (cached-based stream relation join) algorithm, which optimized the skewed environment in streaming data connection efficiency [13].

In the field of distributed computing engines, many development contributions come from the community. For example, the mainstream distributed systems Apache Spark and Apache Flink have adapted for the dimension table association scenario. Apache Spark proposed Spark Streaming to improve computing latency and provide stream computing support [14]. This computing model minimizes each elastic distributed dataset by splitting the unlimited streaming data into discrete (discredited) streams (Resilient Distributed Dataset, RDD) size, built microbatch data set, to achieve the effect similar to stream computing, and the delay of stream computing at this stage is 100 ms. In the technical background of Spark Streaming, Apache Spark can define dimension table data as an independent RDD and cache it on each associated node. After each batch of microbatch data sets arrives, they are processed in a local batch process association. At the same time, the RDD content can be updated to achieve the same state as the offline dimension table data. The above update operations require users to define their update logic. Since the data transmission of Spark Streaming is based on the method of the microbatch data set, it is not the real meaning of the data transmission. For continuous transmission, Spark proposed structured streaming [15].

This research was done to arrive at a solution faced by company network in the incident of node failures as well as what routing protocols they must use to mitigate network delay, queuing delay, convergence rate, and backup and recovery time complexity for improved network quality [16]. The paper suggests that achieving a balance among innovation process and the multihoming of a multitransaction medium is the right strategy to mitigate the impacts of the service sector and accomplish long-term development [17]. In this computing model, the concept of constant processing is introduced, which reduces the delay of stream computing to the level of 1 ms. At the same time, Structured Streaming natively supports Stream-Static Join, but its underlying implementation logic is still to perform independent caching operations for each computing node.

Apache Flink itself is a distributed computing engine based on stream computing. However, to be compatible with batch processing, it still maintains a set of dataset APIS for batch processing. In addition, Apache Flink does not provide official support for dimension table association calculation in version 1.8. Therefore, users must manually implement dimension table association calculation support in stream computing through some operators.

Blink, an open-source engine built by Alibaba based on Apache Flink, proposes the operation mode of dimension table association at the level of table API. Users need to implement the query logic for dimension table data manually. Relying on the asynchronous query of dimension table data and caching of query results reduces database I/O overhead and improves query efficiency. The cache mode of this scheme is that all data is cached and updated regularly and LRU Cache, both of which are based on the cache mode of computing nodes [18]. The above dimension table associative computing designs essentially initiate the queries directly to the database during the stream computing process, binding the query logic to each independent computing node. All computing nodes must simultaneously access the data source or cache—complete dimension table information. In the face of high-speed streaming big data input, querying the database asynchronously will cause significant pressure on the database; in front of massive dimension table data, due to the memory limitation of each computing node, the dimension table data cannot be written entirely. In memory and cause, the execution engine generates a memory exception. As mentioned above, the existing big data computing platforms do not have a dimension table association mechanism that supports the horizontal expansion of the cache. Therefore, in the face of large-scale dimension table data, the dimension table cannot be utterly cached in each computing node in the full cache mode. However, using the asynchronous connection method will generate a large amount of data I/O during high-speed computing, which will cause excessive pressure on the database, and even cause the database connection to time out and lose its response. Aiming at this problem, this chapter designs and proposes a dimension table association mechanism for large-scale distributed computing using the multihoming model as a novel big data framework. This paper introduces several specific designs of dimension table association technology optimization. Firstly, a hybrid computing model is presented, achieving unified batch data processing and stream data processing in the same computing task. Then, the cache optimization of dimension table association technology is introduced, including the cache design of dimension table data source and the cache design of dimension table data by each computing node. Finally, the calculation design of dimension table association technology is introduced, including the processing of big data distribution under parallel computing by employing multihome system base and the processing of association calculation by each computing node.

2.1. Definitions and Concepts. This section gives some basic definitions and concepts involved in the text.

Definition 1. (dimension table). A dimension table is a type of data table stored in an external database with the characteristics of a large data scale and slow update time.

Definition 2. (operator). An operator refers to the minimum calculation unit for data processing in the calculation process. Different operators have different calculation logic. For example, batch and stream computing have operators that functions the same but operate differently.

Definition 3. (hybrid computing). Hybrid computing refers to stream computing operators and batch operators in the same computing task, also known as batch-stream fusion computing.

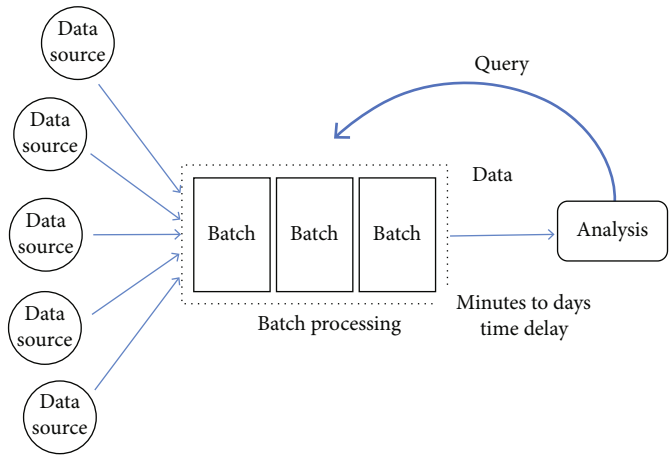

Figure 3: Batch computing architecture.

Definition 4. (dimension table association). Dimension table association means a specific relationship between data flow and dimension table, and data processing is performed according to this relationship. The process of determining connection information and combining data tables according to the relationship is called connection. In a broad sense, dimension table association includes but is not limited to join operations. Operations such as calculation and analysis of dimension table data also belong to the dimension table association.

Definition 5. (back pressure). Backpressure refers to the backlog of received data when the data processing speed of a computing node is lower than the data transmission speed. In turn, the input cache of a single computing node exceeds the limit and refuses to receive new data. Finally, the entire stream computing system starts from a series of upstream nodes from the overloaded node until the chain buffer overflow of the data source node stops receiving data and waits for the congested node to process the data.

Definition 6. (multihoming strategy). We employ a multihoming networking strategy, which entails numerous networks interacting with a single network activity at the same time and exchanging dataset efficiency to reduce uncertainty.

2.2. Hybrid Computing Model. This section introduces the hybrid computing model proposed in this paper. In the traditional batch computing architecture, the operation logic of the calculation is shown in Figure 3.

The computing nodes are started in stages. After each group of measures is completed, the synchronization wait will be released. The next group of computing tasks will be created, and the computing data will be redistributed to the next round of computing nodes. In the traditional streaming computing architecture, the operation logic of computing is shown in Figure 4. All computing nodes are created at the beginning of the calculation, and then the data flows continuously between the computing nodes. Each node achieves continuous stream processing by constantly pulling and consuming the data of the upstream computing nodes. Stream computing does not redistribute data as much as possible, allowing data to be processed continuously.

This paper redesigns the computing operator to be compatible with batch data processing in stream computing. In-stream computing, an operator marked as batch processing is constructed, and a hybrid computing model is proposed. Streaming analysis, often referred to as real predictive modeling, is a technique of big data that exhibits real-time information and uses basic computations to be executed on it. When contrasted to functioning with past information, interacting with real world data necessitates entirely different practices. It does so by employing a technique known as stream computing, which involves extracting massive amounts of constantly evolving data.

This style of intelligence specifically for data transfers rather than more sophisticated analytical activities. Its prime objective is to provide consumers with speed knowledge and to keeping information in a current state. This type of model operator is divided into three categories: stream computing operators, batch operators, and hybrid computing operators. The flow operators in this model are still consistent with the calculation logic of traditional flow computing and are calculated in real-time as the data flows in. Batch operators can achieve the same effect as conventional batch computing by modifying the data distribution and computing logic in the stream computing environment. The upstream operators are stream computing operators and batch operators, respectively, while the hybrid computing operator is utilized for hybrid computing of stream and batch data. Before the batch data arrives, the hybrid computing operator will stop the flow computation and begin the analysis after the batch data distribution is complete.

2.2.1. Cache Optimization of Dimension Table Association Technology. This section introduces the design and optimization of the data source node cache and the compute node cache in the large-scale dimension table associative computing environment.

2.2.2. Cache Design and Optimization of Data Source Nodes. Since the dimension table data also changes, if all read data is sent to the computing node every time, it will still cause

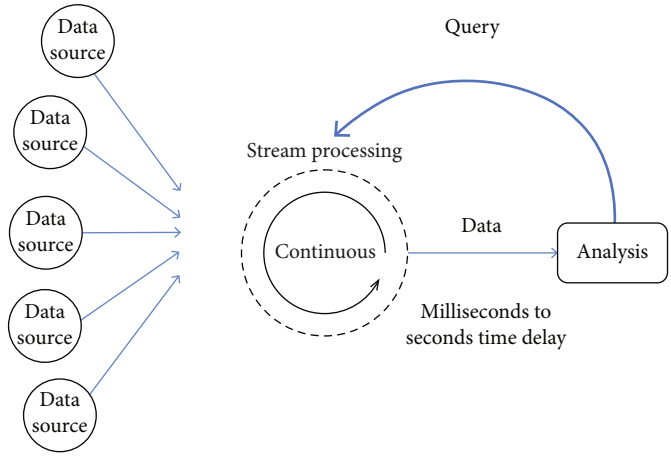

Figure 4: Stream computing architecture.

a significant network transmission overhead and block streaming data processing during the update. This article refers to the idea of the database operation record log. It only distributes the data that needs to be modified when the information is updated to reduce the network transmission overhead. To determine the specific content of the data update, this paper establishes a caching mechanism at the data source to cache the distributed data records according to the slow change of the dimension table. In the key-value pair of the caching tool, the key value is obtained by merging the data of the primary key columns required for the association. The value is changed to an array that stores multiversion dimension table data with the same primary key and other data. However, caching the complete dimension table data will occupy too much memory of the data source operator and cannot support too large dimension tables. Since the data source does not need to operate the data, the hash value is calculated after splicing the dimension table data into a string and stored in the corresponding array, without saving the original data. When reading the dimension table data in each round, the primary key and hash value are cached, and the data distribution behavior is judged according to the last cached result [19]. The specific algorithm first builds the critical value by calling the associated information selection function and constructing the entire big data's hash value and then writes the essential value and the hash value into the new cache. Finally, compare the vital importance with the old store to determine whether the data exists. Then, the data is skipped, and the corresponding old cache is cleaned up. If it does not exist, a new behavior request is issued. The initial value of the old cache is empty in the first data distribution so that all data will be distributed the first time. After the data read and distribution is completed, since the read data and the old cache have been matched and deleted during the data read process, all data still existing in the old stock is the data that needs to be deleted. The distribution of data deletion behavior for expired information is performed and is specific. Traverse the old cache after data reading is completed, generate a data deletion request for each piece of data in the old store, and send it to the downstream computing node to clean the expired data.

2.2.3. Cache Design and Optimization of Compute Nodes. In the standard dimension table association calculation, the data is cached in key value. The Key value is calculated through the associated primary key column, and the value stores the specific data content of the dimension table. Unfortunately, this traditional data caching mode cannot support multiversion dimensions with the same primary key. Table data is cached; so, it cannot support fully caching and querying historical dimension table data. Aiming at this problem, this paper proposes a layer-2 caching strategy in computing nodes. In the original key-value cache mode, the value format is optimized; the dimension table data with the same key value is converted into a JSON string and stored in value in the form of an array. The cache structure supports historical dimension table data. At the same time, compared with keeping the complete dimension table structure, converting to JSON strings can reduce specific memory storage requirements.

In the process of data acquisition by computing nodes, the data acquisition algorithm should be redesigned to support the optimized data distribution of data source nodes. Judging the behavior of data change, if it is adding data (ADD), write the data to the cache. If deleting data (DEL), delete the cache-related data according to the critical value.

2.3. Computational Design of Dimension Table Association Technology. The calculation design of dimension table association technology is divided into two parts: data analysis district mode design and data calculation mode design.

2.3.1. Design of Data Partitioning Method. In the traditional dimension table association, the dimension table data is stored independently among each computing node so that the streaming data can be distributed according to the distribution strategies of different computing systems, and the dimension table data can be obtained by querying the remote database when flowing through the computing nodes. In the globally distributed dimension table association policy, if the flow data distribution rule does not match the dimension table data distribution rule, the association

result will be empty. Therefore, to commonly perform the optimized dimension table association, the big data distribution strategy for streaming data and dimension table data will be redesigned [20]. The associated primary key will be calculated uniformly. The primary key will be hashed to obtain the corresponding data partition.

2.3.2. Data Calculation Mode Design. In the standard dimension table association calculation, the connection information is determined by specifying the input data and the primary key column of the dimension table. The output is the set of data columns after the association. This paper firstly redesigned the data cache to support distributed cache strategy and multiversion data cache. This paper redesigns the big data association calculation mode to adapt to the connection selection problem for multiversion data. It abstracts the association logic and the associated solid value selection logic [21]. It separates the associated calculation function and the strongly associated value selection function that the user can define and implement. Based on the connection calculation, the user can determine the big data association key value of the data flow and dimension table according to the actual calculation requirements and customize the associated calculation result. Since each cell of the association calculation function is to pass on the entire version the dimension table data obtained through the association information selection function, the user can flexibly process the dimension table association to support the requirements of specifying a specific historical version of the association, calculating a particular relationship, etc. A typical not-connected association scenario is the commodity association calculation in actual production and life scenarios [22]. By processing the similarity relationship between the input commodity and the order in which the item exists in the chronological order in the calculation function, other things related to the entity are calculated to obtain the related commodity of the input commodity.

3. Experimental Analyses

In this paper, the abovementioned optimized dimension table association technology is implemented in Apache Flink, and the efficiency difference between the multihoming optimized dimension table using big data frameworks association technology and the traditional dimension table association technology is compared and verified at various stream data and dimension table data scales.

3.1. Coding Implementation. Coding implementation is divided into traditional dimension table association implementation and optimized dimension table association implementation.

3.1.1. Traditional Dimension Table Association Implementation. In Apache Flink 1.8.0, the DataStream API does not natively support dimension table association calculations. Since the traditional dimension table association is essentially stream computing, this paper transforms the asynchronous computing operators according to the computing logic, adding support for database query and data caching mechanisms based on LRU (least recently used) strategy. Then, the operator is encapsulated to support the definition and reading of dimension tables and the setting of connection conditions.

Table 1: Size of the data table.

Table name	Number of records/104	Data capacity/MB
User information form	1 000	11 969.00
Product information sheet	1 000	11 115.00
User clicks on data table	10 000	11 118.06

3.1.2. Optimized Dimension Table Association Implementation. The optimized multihoming dimension table association is implemented based on the mixed API. By designing a new dual-input operator, one input is stream data, and the other is dimension table data to support hybrid computing. Then, the data source method of dimension table data is designed separately. The actual data distribution process in the system is transformed to adapt to the new data distribution logic. Finally, the whole process is encapsulated, and the abstraction and default logic are realized for the data connection information selection function and the dimension table association calculation function. For the cache, relying on the features of the mixed API, the stream data processing is blocked until the batch data calculation and distribution is completed, which ensures the integrity of the dimension table cache.

3.2. Dataset. For the calculation type encountered in the actual production activity of dimension table association, this paper uses the real production scene in Alibaba's "Double Eleven" for performance testing. After data desensitization and business simplification, this scenario involves a total of 3 data tables: user information table, product information table, and user click data table. In this experiment, a corresponding data table is established in the MySQL database according to the table format, and the corresponding data is generated. The data records and data capacity of the data table are shown in Table 1. At the same time, a corresponding data table of 100,000 pieces of user information data and 1,000,000 pieces of user information data is constructed according to the same proportion of this table for comparative testing. The key-value columns required by the query are all indexed to ensure query efficiency during asynchronous access.

The specific business of this production scenario is to place advertisements to users, and the simplified business process is as follows:

(1) Influx user IDs into the recommendation system

(2) The system obtains the user's product click data and user information data within one day according to the user ID and calculates the recommended product ID through the machine learning algorithm

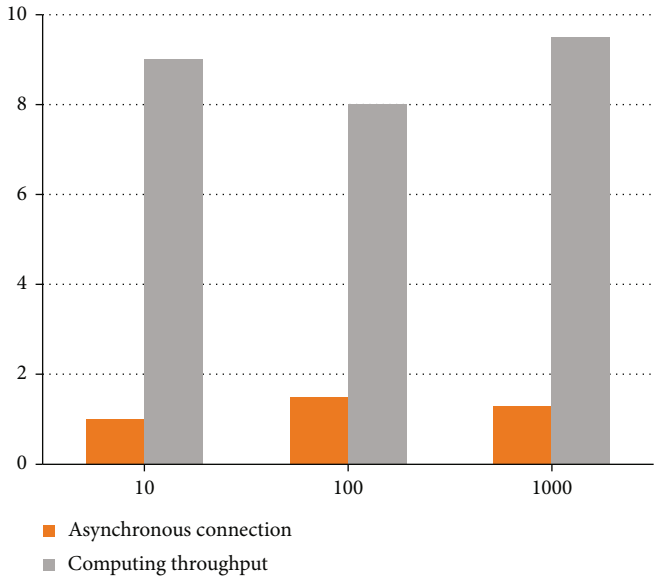

Figure 5: Experiment 1: average throughput statistics.

(3) Obtain the product's specific information through the recommended product ID and return it

To focus on testing the efficiency of dimension table association, this experiment simplifies the recommendation calculation part and only retains the dimension table association part, and the modified business logic edit is as follows:

(1) Influx the user ID into the test system

(2) According to the user ID, the specific information of the user and the ID of the product clicked by the user are expanded

(3) Add a random number to each user clicked product ID as the recommended product ID

(4) Obtain the product's specific information through the recommended product ID and return it

3.3. Experimental Environment. The related technical details described in this article are all implemented in Flink 1.8.0 written in Java. The operating environment and software and hardware environment of the experiment are as follows.

(1) Hardware environment: the distributed environment used in the experiment consists of 4 servers, one controller node, and three agent nodes. The server configuration used by the agent node is as follows: ① CPU Intel Xeon E5-2603 V4 *2, the number of cores is 6; the central frequency is 1.7 GHz. ② The memory is 64 GB, and the central frequency is 2 400 MHz and ③ hard disk 400 GB SSD. The primary node configuration is as follows: ① CPU Intel Xeon E5-2603 V4 ∗2, the number of cores is 6; the central frequency is 1.7 GHz, ② Memory 128 GB, clocked at 2 400 MHz, and ③ hard disk 400 GB SSD

(2) Software environment: ① operating system Centos 7, ② Storage environment MySQL 5.6.45, and ③ Apache Flink version 1.8.0, JDK version 1.8.0

3.4. Experimental Results and Analysis. This article uses the advertising test tool provided by Alibaba to test by modifying the logic of the business calculation part. The test adds timestamps to the data at the beginning and end of the calculation and writes it to Redis and then obtains relevant statistical results by analyzing the data in Redis [21]. The parallelism of all measures in this experiment is fixed at 12, ignoring the cold start time of the dimension table full cache. Generate 1 million pieces of flow data as experiment 1, generate 10 million reports of flow data as experiment 2, and compare the impact of the data volume of the flow data and the data volume of the dimension table on the computational throughput and latency.

3.4.1. Comparative Analysis of Throughput. Throughput refers to the amount of data processed by the computing engine per unit time, which reflects the system's load capacity. The higher the throughput, the greater the extreme load on the system, helping to process more data per unit time. In this experiment, the throughput is calculated by calculating the amount of data written per unit time in Redis. The throughput calculation formula is as follows:

$$\text{through}_{\text{put}} = \frac{\text{curren\_Num} - \text{last\_Num}}{\text{urren\_Time} - \text{last\_Time}}. \quad (1)$$

Among them, last_Num represents the last read data number, and current_Time represents the time corresponding to the existing data table number read. Last time means the time corresponding to the last data number read. In experiment 1, the average throughput of data generated by different scales of dimension table data volume is shown in Figure 5 with Table 2. The experimental results show that

Table 2: Experiment 1: average throughput statistics.

Serial	Asynchronous connection	Computing throughput
10	1	9
100	1.5	8
1000	1.3	9.5

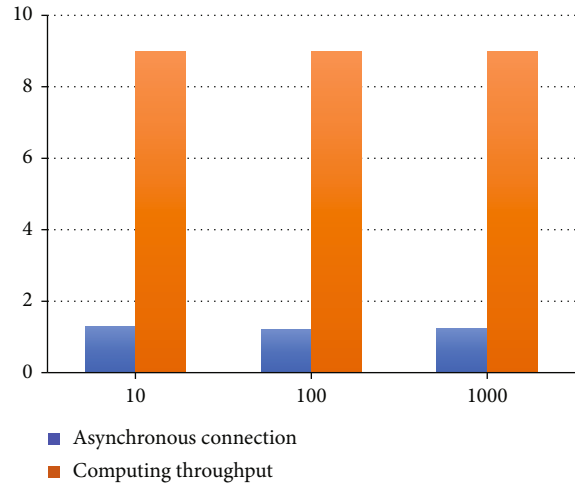

Figure 6: Experiment 2: average throughput statistics.

Table 3: Experiment 2: average throughput statistics.

Serial	Asynchronous connection	Computing throughput
10	1.3	9
100	1.2	9
1000	1.25	9

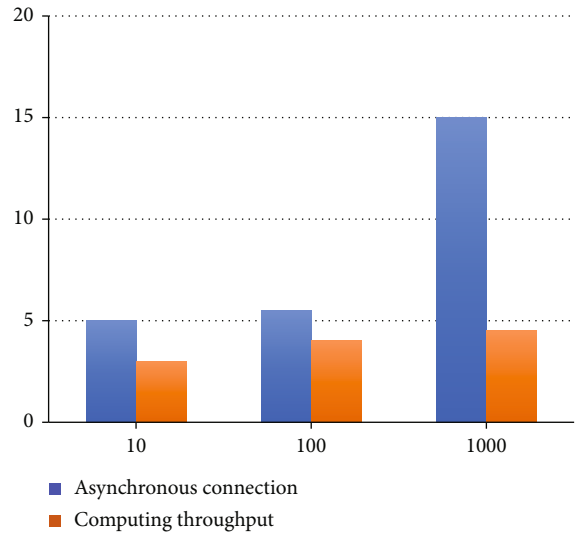

Figure 7: Experiment 1 means delay.

the total amount of data caching can effectively improve the computing throughput, with a 10-fold improvement under the three-dimension table scale. However, the throughput of the asynchronous connection method decreases as the

Table 4: Experiment 1 means delay.

Serial	Asynchronous connection	Computing throughput
10	5	3
100	5.5	4
1000	15	4.5

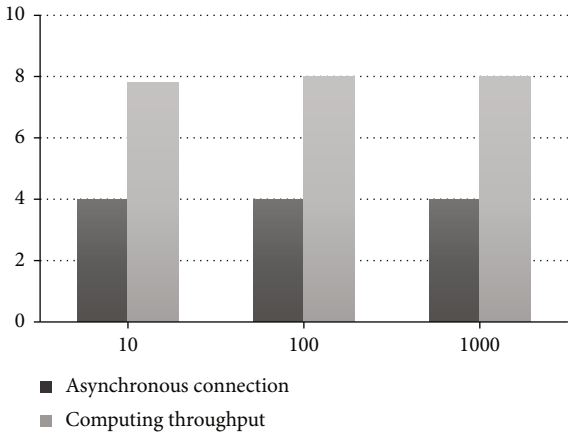

Figure 8: Experiment 2 means delay.

Table 5: Experiment 2 means delay.

Serial	Asynchronous connection	Computing throughput
10	4	7.8
100	4	8
1000	4	8

dimension table data increases. After analysis, it was concluded that this was because the cache built by the asynchronous connection was not hit, and many queries were entered into the database using big data multihoming techniques. As a result, the time used for data queries prolongs the processing time of each piece of data, reducing the amount of data that can be processed per unit time. In experiment 2, the average throughput of data generated by different scales of dimension table data volume is shown in Figure 6 with Table 3. Same as experiment 1, the throughput statistics of the full cache mode are similar, indicating that the data processing efficiency has nothing to do with the incoming data. Increasing the parallelism can further improve the throughput. As the multihoming dimension table data increases, the asynchronous connection throughput stabilizes. This phenomenon occurs because the number of streaming data increases, and the dimension table data queried in a short period have all been loaded into the local cache. Asynchronous computing mode throughput is still at a low level.

It is caused by the low concurrency of the connection mechanism. Still, too high concurrency will directly lead to the collapse of the storage system connection in the early computing stage.

3.4.2. Delay Comparative Analysis. Latency refers to the time taken by data from entering the computing system to be processed and output in milliseconds (ms) in the case of

big data networks. The size of the delay reflects the data processing speed and real-time performance of the system. The advertising system recommended in real-time has a high demand for the delay, and the smaller the delay, the better the user experience. In this experiment, the weight is calculated by the start and end timestamps of each piece of data recorded in Redis. The delay calculation formula is as follows.

$$\text{latency}_{\text{time}} = \text{end}_{\text{time}} - \text{begin}_{\text{time}}. \quad (2)$$

end_{time} indicates when the data is processed, and begin time indicates when the information is marked after it flows into the system.

In experiment 1, the average delay of data generated by dimension table data volume of different scales is shown in Figure 7 with Table 4. The big data has been written to the memory in the full cache mode, and there is no data I/O overhead; so, the average latency does not fluctuate much. However, the asynchronous connection based on the multihoming system calculation mode requires a lot of interaction with the database because there is no cache of dimension table data in the early stage. As the size of the dimension table increases, the processing time of a single query becomes higher and higher, resulting in an increasing average delay.

In experiment 2, the average delay of data generated by dimension table data volume of different scales is shown in Figure 8 with Table 5. In the full cache mode, the wait increases compared to experiment 1 because to amortize the memory pressure, there is no shared calculation slot between multiple dimension table association calculations, and the data distribution needs to go through serialization and deserialization. Excessive throughput and data volume lead to extreme data transmission pressure, resulting in blocking. At the same time, since the caching mechanism is manually implemented, there is no backpressure mechanism for connecting to the Flink computing system, resulting in an excessive backlog of computing node data. The asynchronous connection mode has temporarily cached all the required dimension table data due to a large amount of I/O in the early stage of the calculation. Multiple dimension table associations using the multihoming system share computing slots and data transmission that does not need to go through the serialization and deserialization process and are directly sent to downstream computing nodes. At the same time, due to the backpressure mechanism of Apache Flink, the data actually flagged flow into the system may be later than when the data is produced.

4. Conclusion

This study looks into dimension table association technology in novel big data frameworks that use multihoming optimization for distributed stream computing. In contrast to the existing big data computing platform's association technique, it first presents a range of data processing logic that enables the integration of stream and batch processing, followed by a dimension table associated data full cache solution that enables vertical expansion and data distribution optimization. A massive set of experimental results show that using multihoming optimization technology, this novel big data framework can effectively increase the loading of dimension table data under full cache conditions and can increase throughput by up to 10 times in the high-speed processing link, significantly reducing data query pressure and improving task parallelism. When the dimension table has a large amount of data, the asynchronous query technique has a significant chance of causing the database to freeze. Due to the data I/O connection, high throughput is not possible. If you use the regular full cache mode, the system will throw a memory overflow exception. The dimension table association technique proposed in this paper has a high parallelism capability. The read and write burden on the database does not grow in equality. As per the future scope of this research is considered, we can focus on the asynchronous query technique that has a high potential of suffering the network to malfunction when the dimension table has a giant amount of data, and hence high throughput is not possible due to the data I/O connection. This can be the future motivation to conduct the research to sort this issue.

Data Availability

The data shall be made available on request.

Conflicts of Interest

The authors declare that they have no conflict of interest.

References

[1] T. Hamada, Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Three-dimensional stacking FPGA architecture using face-to-face integration," in *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, Istanbul, Turkey, 2013.

[2] N. Eloe, J. Leopold, C. Sabharwal, and D. McGeehan, "Efficient determination of spatial relations using composition tables and decision trees," in *2013 IEEE Symposium on Computational Intelligence for Multimedia, Signal and Vision Processing (CIMSIVP)*, Singapore, 2013.

[3] S. K. Jayanthi and S. Prema, "Referenced attribute functional dependency database for visualizing web relational tables," in *2011 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India, 2011.

[4] K. J. Schultz and P. G. Gulak, "Physical performance limits for shared buffer ATM switches," *IEEE transactions on communications*, vol. 45, no. 8, pp. 997–1007, 1997.

[5] G. Zhou, X. Gao, X. Lai et al., "FPGA testing points optimization method based on important analysis," in *2016 Prognostics and System Health Management Conference (PHM-Chengdu)*, Chengdu, China, 2016.

[6] I. de Miguel, R. Vallejos, A. Beghelli, and R. J. Durán, "Genetic algorithm for joint routing and dimensioning of dynamic WDM networks," *Journal of optical communications and networking*, vol. 1, no. 7, p. 608, 2009.

[7] P. Randhawa, V. Shanthagiri, and A. Kumar, "A review on applied machine learning in wearable technology and its

applications," in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, Palladam, India, 2017.

[8] V. K. Singh, U. Sharma, B. Singh, and S. Shastri, "Design methodology and considerations to energy efficient switched reluctance motor for ceiling fan application," in *2021 IEEE Energy Conversion Congress and Exposition (ECCE)*, Vancouver, BC, Canada, 2021.

[9] M. M. Abir Bappy, M. R. Sarkar, S. I. Hasan, M. M. Azmir, and D. M. Rashid, "Design process and performance analysis of two stage differential Op-amp by varying the body biasing in fully depleted silicon on insulator technology," in *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Vancouver, BC, Canada, 2021.

[10] H. Ying, K. Hofmann, and T. Hollstein, "Dynamic quadrant partitioning adaptive routing algorithm for irregular reduced vertical link density topology 3-dimensional network-on-chips," in *2014 International Conference on High Performance Computing & amp; Simulation (HPCS)*, Bologna, Italy, 2014.

[11] T. Djerafi, D. Hammou, Ke Wu, and S. O. Tatu, "Ring-shaped substrate integrated waveguide Wilkinson power dividers/combiners," *IEEE transactions on components, packaging and manufacturing technology*, vol. 4, no. 9, pp. 1461–1469, 2014.

[12] L. Xi, W. Hongkai, L. Jinhu, P. Xubin, and Y. Zhanpeng, "Research on multi-dimensional analysis method of power equipment condition monitoring based on OLAP," in *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*, Suzhou, China, 2019.

[13] J. Bhola, M. Shabaz, G. Dhiman, S. Vimal, P. Subbulakshmi, and S. K. Soni, "Performance evaluation of multilayer clustering network using distributed energy efficient clustering with enhanced threshold protocol," in *Wireless Personal Communications*, pp. 1–15, Springer Science and Business Media LLC, 2021.

[14] S. Sanober, I. Alam, S. Pande et al., "An enhanced secure deep learning algorithm for fraud detection in wireless communication," *Wireless Communications and Mobile Computing*, vol. 2021, 14 pages, 2021.

[15] A. Kapoor, A. Gupta, R. Gupta, S. Tanwar, G. Sharma, and I. E. Davidson, "Ransomware detection, avoidance, and mitigation scheme: a review and future directions," *Sustainability*, vol. 14, no. 1, p. 8, 2022.

[16] Z. Basit, M. Tabassum, T. Sharma, M. Furqan, and A. Q. Md, "Performance analysis of OSPF and EIGRP convergence through IPsec tunnel using multi-homing BGP connection," *Materials Today: Proceedings*, vol. 62, pp. 4853–4861, 2022.

[17] J. J. Yun, X. Zhao, L. Ma, Z. Xu, and Z. Liu, "Open innovation and multi-homing of delivery platforms: comparative study of Cardiff, Daegu and Nanjing," in *European planning studies*, pp. 1–22, Informa UK limited, 2022.

[18] K. Chetcuti, C. J. Debono, and S. Bruillot, "The effect of human shadowing on RF signal strengths of IEEE802.11a systems on board business jets," in *2010 IEEE Aerospace Conference*, Big Sky, MT, USA, 2010.

[19] K. Mahajan, U. Garg, and M. Shabaz, "CPIDM: A clustering-based profound iterating deep learning model for HSI segmentation," *Wireless Communications and Mobile Computing*, vol. 2021, 12 pages, 2021.

[20] A. Gupta and P. Prabhat, "Novel approaches in network fault management," *International Journal of Next-Generation Computing*, vol. 8, no. 2, 2017.

[21] Q. Yao, M. Shabaz, T. K. Lohani, M. Wasim Bhatt, G. S. Panesar, and R. K. Singh, "3D modelling and visualization for vision-based vibration signal processing and measurement," *Journal of intelligent systems*, vol. 30, no. 1, pp. 541–553, 2021.

[22] Y.-C. Chen, W. Wang, H. Li, and W. Zhang, "Non-volatile 3D stacking RRAM-based FPGA," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Oslo, Norway, 2012.