Multi-link query optimization based on heuristic search in data stream

Zhiming Chen, Fengzhen Wang, Boyang Liu
Beijing University of Technology, China
*chzhm25@163.com

Keywords: heuristic search; data stream; undirected weighted graph; connection tree

Abstract. In the stream processing, the connection operation is one of the very important basic operations, and it plays an important role in real-time data query and mining. According to the existing data stream connection technology and combining with the characteristics of coarse granularity processing unit of Sparkstreaming platform, this paper designs a multiple connection query optimization algorithm based on heuristic search. The algorithm first transforms the query into an undirected weighted graph, constructs the connection tree through the heuristic function to optimize the connection order, and combines the sliding window technology to introduce the cache mechanism and reduce the computational amount. Due to the characteristic that during the stream processing the data is coming continuously and constantly, the function evaluation value is periodically updated and the connection tree is rebuilt dynamically, in order to ensure the continuous and efficient connection operation. The experimental part uses the distributed Message Queue Kafka to generate multiple data streams, and uses Sparkstreaming as a consumer to connect operations and verify the feasibility of the algorithm.

1 Introduction

With the constant development of science and technology, information technology, as one of the important parts of it, is also continuously developing. From financial transactions, web browsing to the highway surveillance camera, the changes brought by computers can be seen everywhere in life, and the specific media to achieve these changes is data. The flow of data has impacted the traditional databases and centralized processing systems that have dominated in the past few decades, bringing humans into the big data age. The processing mode is mainly divided into two major categories: flow processing and batch processing [1]. In batch processing, the data is first stored in the database, and then to be processed, which is equivalent to the extension of traditional processing technology, but greatly improves the data volume. While in stream processing, the flow of data is seen as a stream in the river; instead of being stored for very long time, the data is processed directly.

The core of stream processing is data stream [2]. It refers to the continuous generation of data from one or more devices, which is generated over time, and because of time, storage, and other factors, data can only be read one or more times in a particular order, and cannot be read repeatedly from the database as traditional data [3]. In the process of data stream operation, the connection operation is a very important basic operation. Its consumption of resources is also very large. It also plays an important role in real-time data query and mining [4].

2 Related work

2.1 Sliding window

The constant arrival of data stream and its query requirements of high proportionality, combined with the growing size of the data under big data environment, will bring unacceptable huge delay, if the system gives query results based on all the data as a traditional database. At the same time, for the flow query application, it solves more dynamic requirements, so the data produced in a shorter period often has higher value that users more care about. In this context, the concept of sliding window [5] appears. The basic principle of sliding windows is that, during data stream processing, a buffer is set and only the data in the buffer will be processed each time. Over time, the data in the buffer changes,

and the new data goes in and the old data expires. This ensures that the data will not be too large for each processing, while ensuring the timeliness of the results.

2.2 Heuristic Search

The core of heuristic search is to use the heuristic information of the existing problem to determine the direction of search. Compared with traditional depth-first and width-first algorithm, this search method can greatly reduce the search scope and the complexity of the problem. The intensity of the heuristic information determines the effect of the searching. The weak information point will cause the redundancy of expanded node, resulting in a higher cost; while, though the strong information points can reduce the cost, it is easily to fall into the local optimum and miss the optimum solution. In practice, it is often difficult to find the heuristic information that neither cost too much nor cause the missing of the optimum solution. Usually, the node that is to be expanded in the next step can be selected by defining an evaluation function [6].

3 Algorithm Description

3.1 Building an Undirected Graph.

The connection between data streams means to generate a new stream of data in accordance with a given rule for a property that conforms to a connection condition between each data stream. In traditional database, the connection condition includes in-connection and out-connection. The in-connection matches two tables based on the value of the Connection Rule column, and only the rows that conform to the rule appear in the result set; the result set of the out-connection not only has the result matching the rules of the in-connection, but also all the rows of the specified table. The connection between data streams also conforms to the same characteristics, and in this article we will focus on in-connection. By transforming the connection rules into an undirected graph, it will be more intuitive to understand the correlation between data streams and find a better connection order.

The transforming process is as follows:

(1) Set={D_1,D_2,D_3,...,D_n} // Initializing an array to store data stream information
(2) Point={} // Graph point array
(3) Edge={} // Graph edge array
(4) while Set.length>0 do
(5) i=Set.getfirst() // Remove the first stream from the data stream array
(6) for(j←0 to Point.length-1) do
(7) if(related(i,Point(j))) // Determining current data stream correlation
(8) then Edge.put(i,Point(j)) // Associated data stream joins the edge array
(9) end
(10) Point.put(i) // Add current data stream to point array
(11) end

3.2 Build the connection tree.

In traditional database, the execution order of the multiple table connection can be represented by the connection tree. The root node in the tree corresponds to each table, the non-root node is the middle state during the connection, and the process of constructing the connection tree layer by tier is the process of finding the connection order. The connection tree can be divided into three kinds – left deep tree, right deep tree and dense tree according to the relationship between nodes. In the left-deep tree, except that the bottom and left two root nodes are tables, the other root nodes that represent tables are the right node. The left node is the middle state, and it is the opposite how the root nodes represent the table in the right deep tree and the left-right relationship of the nodes during the middle state. Traditional database is a stand-alone environment, so it is more inclined to use the left deep tree to build the connection tree, and in the distributed environment, the use of dense tree can better utilize its high concurrency characteristics. So here we choose to build a dense tree to optimize the connection order between the data streams.

In the previous section, we transformed the connection between data streams into a single graph, thus transforming the process of building the connection tree into a graph traversal. When performing a connection operation, two main elements that impact the operational efficiency and resource

consumption is the level of performing frequency and the size of the intermediate quantity. The level of the operational efficiency is determined by the velocity of the data stream. As for the intermediate quantity, due to the overmuch levels, the cost will be too much if we want to predict the size of all the intermediate quantity during the whole calculation process, so we will use the size of the related intermediate amount between any two related data streams instead. In an undirected graph, point represents a data stream, and edge represents the connection between data streams. So, we can use heuristic search to build the connection tree by referencing the traversal of the graph. The whole process is as follows:

```
(1)  for(i←0 to Point.length-1) do
(2)        Point(i).setvalue() // Point weights according to the stream velocity
(3)  end
(4)  for(i←0 to Edge.length-1) do
(5)        Edge(i).setvalue() // Edge weights according to the average intermediate quantity
(6)  end
(7)  Tree={}//Initializing an array of trees
(8)  f=w*Edge(n).getvalue()+Edge(n).getpionta().getvlaue()+Edge(n).getpiontb().getvlaue()
     // Set the evaluation function, w is the balance factor, control the middle quantity
(9)  Edge.sort(f) // Sorting edge array elements according to the evaluation function
(10) while Edge.length>0 do
(11)       i=Edge.getfirst() // Remove the first element of an edge array
(12)       if(Tree.contains(i.getpointa())) // If the left endpoint is already in the tree
(13)             then if(Tree.contains(i.getpointb()))
(14)                   then j=Tree.gettop(i.getpointa()) // Find left point tree vertex
(15)                   k=Tree.gettop(i.getpointb())
(16)                   if(j==k) // If two vertices are the same, the edge is redundant
(17)                         then continue // Skip action
(18)                   else Tree.merge(j,k) // Merging trees
(19)             else j=Tree.gettop(i.getpointa())
(20)                   Tree.add(j,i.getpointb()) // Add right endpoint to tree
(21)       else if(Tree.contains(i.getpointb()))
(22)             then j=Tree.gettop(i.getpointb())
(23)             Tree.add(j,i.getpointa())
(24)       else Tree.create(i.getpointa(),i.getpointb()) // Generate a new tree from both ends
(25) end
```

3.3 Incremental Query.

The connection tree method has an advantage compared with the connect operator, which caches intermediate information in each parent node. Combined with the characteristics of coarse granularity processing unit of Sparkstreaming platform and the query characteristic of sliding window, by using spark cache caching mechanism, an incremental method can be implemented to continuously query the result of data stream connection to reduce the computational amount of query operation.

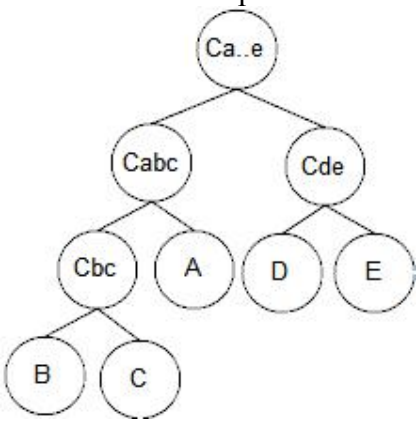

Figure 1. Connection Tree Example

Figure 1 is a generated connection tree example, ABCDE nodes are the data stream nodes in order to receive the data stream information. The $C_{bc}C_{abc}C_{de}C_{abcde}$ node is the intermediate state node, in which the caches of the connection result that are corresponding to subscripted data stream array are left. If the C_{bc} node has the cache of the intermediate results between the connection of the data stream B and the data stream C, the buffer form is (k,v,t), k is the key value, v is the concrete result, t is the time stamp when the result is produced.

If only stream A has new data coming in a unit of time, the new data in A can be jointly operated with the cache data in the C_{abc} node. Update the data in the C_{de} node, and use the part of data to connect with the cache in C_{de}; update the data in C_{abcd} and output the result of the connection. In this process, the connection between the node BC and DE is omitted, and the computational quantity in the query is reduced.

Assume the window size is t_W and the unit time is t_0, and the following is an example of an incremental query process from t_1 to t_1+t_0 time:

(1) Tree // Connect tree array
(2) C={C_{bc},C_{abc},C_{de},C_{abcde}} // Existing cache arrays for each intermediate node
(3) D//New stream of data coming in t_0 time
(4) for(i←0 to C.length-1) do
(5) C(i).clear(t_1-t_w,t_1+t_0-t_w) // Clear the cache from t_1-t_W to t_1+t_0-t_W in each node
(6) end
(7) if(D==null) // If no new data stream coming at t_0
(8) then C_{abcde}.export(t_1+t_0-t_w,t_1+t_0) //Output the cache of C_{abcde} from t_1+t_0-t_W to t_1+t_0
(9) else D.sort() // Group new data streams by hierarchy, the level of B_c node is 1, ADE is 2
(10) for(i←0 to D.length-1) do
(11) for(j←0 to D(i).length-1) do // Concurrent processing of the same-level data stream
(12) k=join(D(i)(j),Tree.getnear(D(i)(j))) // Perform a connection
(13) Update(Tree.getparent(D(i)(j),k)) // Update cache
(14) end
(15) end
(16) C_{abcde}.export(t_1+t_0-t_w,t_1+t_0) // Output cache of C_{abcde} from t_1+t_0-t_W to t_1+t_0 as a result

In the big data environment, the data stream has more obvious uncertainty. So it is necessary to monitor the velocity of the data stream and the change of the size of the intermediate quantity of the connection between different data streams every certain period of time to re-operate the algorithm of building the connection tree in the previous section, update the weights of the midpoint and edge of the graph with the newly obtained data, and calculate the connection order that is more suitable for the current data stream group, in order to achieve dynamic update and ensure continuous and efficient connection operations.

4 Experimental analysis

First build cluster platform of the Kafka, Zookeeper and Spark. The Kafka cluster contains a leader and two flower, while spark clusters contains a master node and three worker nodes, with 2GB memory respectively. We use the Kafka producer API to generate a four-channel data stream $D_1D_2D_3D_4$,D_1. D_1 is homogeneous data stream, with the appearance of 1 groups per second and 300 data per group; D_2 is the linear data stream, generating 200 sets of data in 1 second, and subsequent generation intervals to increase by 1 second, after 4 seconds the interval changed back to 1 second; D_3 is a homogeneous data stream, generating 1 group per second, 300 data per group; D_4 is the linear data stream, generating 200 sets of data in initial 2 seconds, and subsequent generation intervals increasing by 2 seconds, and after 6 seconds it is changed back to 2 seconds. In the Spark platform, the Direct API is used to receive the data stream generated by Kafka, and then the data stream connection operation is carried out by using the method of Sparkstreaming native window plus join operator and the method of query optimization based on heuristic search. The sliding window size is 5 minutes, and the sliding interval is 2 seconds; the Sparkstreaming query interval is 2 seconds, and the balance

coefficient is set to 0.5. Figure 2 shows the performance of the query operation in the adjacent six job tasks in two ways, from which we can see that the algorithm presented in this paper is slightly better than the query method of Sparkstreaming native operator combination.

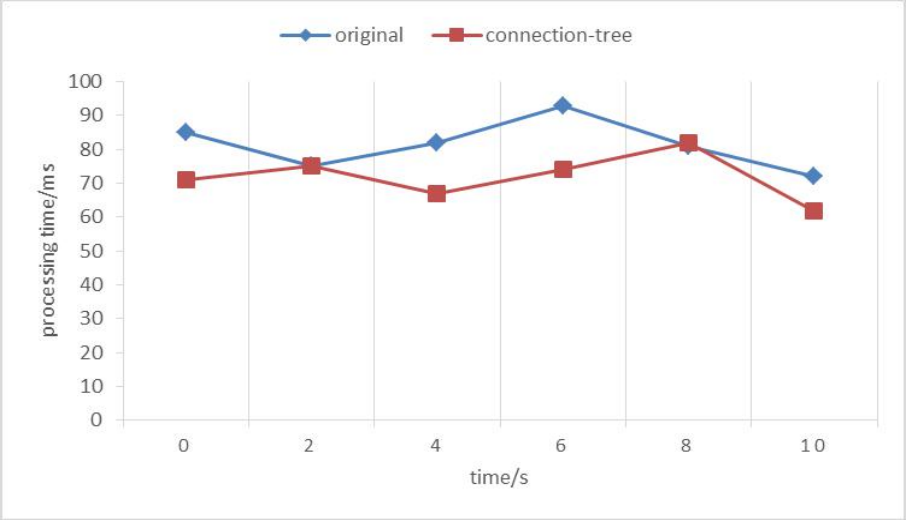

Figure 2. Original connection operator and connection tree optimization algorithm execution time

5 Conclusion

In the big data environment, the data scale of stream processing is expanding, and the connection operation is a very common high consumption operation in stream processing, and its challenge is enhanced. This paper analyzed the existing connection technology in traditional database and stream data, and transformed the connection relationship between data streams into an undirected graph, based on the velocity of the data stream and the intermediate quantity. Then the method of constructing the connection tree is proposed, and a new algorithm based on heuristic search for the query optimization of multiple data stream connection is presented. By generating data stream in Kafka platform and receiving and processing data in Sparkstreaming platform, the experiment is carried out to verify the feasibility of the algorithm.

References

[1] Meng X, Ci X. Big data management: Concepts, techniques and challenges[J]. Journal of Computer Research & Development, 2013.

[2] Margara A, Cugola G. Processing flows of information:from data stream to complex event processing[C]// ACM, 2011:359-360.

[3] Barwick, Hamish. Big data to power insights: Enterprises look to analytics in 2012[J]. 2012(Mar/Apr 2012).

[4] Yang R, Wang K, Mu W, et al. A Windowed Point-Join Processing Algorithm over Data Stream[J]. Journal of Computer Research & Development, 2014.

[5] Chang J H, Lee W S. A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams[J]. Journal of Information Science & Engineering, 2004, 20(4):753-762.

[6] Singh M, Dhillon J S. Multiobjective thermal power dispatch using opposition-based greedy heuristic search[J]. International Journal of Electrical Power & Energy Systems, 2016, 82:339-353.