D-JB: An Online Join Method for Skewed and Varied Data Streams

Chunkai Wang, Jian Feng, Zhongzhi Shi

D-JB: An Online Join Method for Skewed and Varied Data Streams

Chunkai Wang[1,2], Jian Feng[1] and Zhongzhi Shi[2]

[1]Post-Doctoral Research Center, China Reinsurance (Group) Corporation, Beijing, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[1]{wangck,fengj}@chinare.com.cn
[2]shizz@ict.ac.cn

Abstract. Scalable distributed join processing in a parallel environment requires a partitioning policy to transfer data. Online theta-joins over data streams are more computationally expensive and impose higher memory requirement in distributed data stream management systems (DDSMS) than database management systems (DBMS). The complete bipartite graph-based model can support distributed stream joins, and has the characteristics of memory-efficiency, elasticity and scalability. However, due to the instability of data stream rate and the imbalance of attribute value distribution, the online theta-joins over skewed and varied streams lead to the load imbalance of cluster. In this paper, we present a framework D-JB (Dynamic Join Biclique) for handling skewed and varied streams, enhancing the adaptability of the join model and minimizing the system cost based on the varying workloads. Our proposal includes a mixed key-based and tuple-based partitioning scheme to handle skewed data in each side of the bipartite graph-based model, a strategy for redistribution of query nodes in two sides of this model, and a migration algorithm about state consistency to support full-history joins. Experiments show that our method can effectively handle skewed and varied data streams and improve the throughput of DDSMS.

Keywords: Distributed Data Stream Management System, Online Join, State Migration, Bipartite Graph-based Model

1 Introduction

Nowadays, with the increasing number of data types and the emergence of data intensive applications, the number and speed of data increase rapidly. It makes that data stream real-time analysis and processing has become one of the hottest research areas. So, distributed data stream management systems (DDSMS) are widely used in real-time processing and query analysis of large-scale data streams.

In applications such as analytic over microblogs, monitoring of high frequency trading and real-time recommendation, it often involves joins on multiple data streams to get the query result and to maintain large state for full-history query requests based on the full-history data [1, 2]. In these applications, data rate tends to fluctuate and the distribution of attribute values is also imbalance. It makes the join operation over skewed and varied streams prone to cluster load imbalance. The phenomenon leads to the decrease of query efficiency and the increase of computation cost in the cloud environment. Due to the imbalance of data rate and distribution, it causes attribute value skew (AVS) [3]. Due to data partition, it causes tuple placement skew (TPS) [3]. So, how to deal with the efficient joins over skewed and varied streams and the load balance of clusters is the focus of our attention.

In order to design an efficient distributed stream theta-join processing system, there are several models designed for join operator over data streams. The join-matrix model [4] and the join-biclique model [5] are two representative approaches to deal with the scalable join processing. For supporting arbitrary join-predicates and coping with data skew, the join-matrix model uses a partitioning scheme to randomly split the incoming data stream into a non-overlap substreams. As a representative of alternative model, the join-biclique model is to organize the processing units as a complete bipartite graph (a.k.a biclique), where each side corresponds to a relation. Two streams are divided into the different side. And, according to the key-based partitioning method (such as, hash function), tuples are distributed to the different nodes for storing in the same side. At the same time, tuples also are sent to the opposite side to do the join operation using the same hashing strategy. After obtaining the results, these tuples are discarded.

Join-matrix and join-biclique models can effectively deal with the online join operation of distributed data streams, but are faced with the following problems and challenges.

1. The join-matrix model needs high memory usage to replicate and store in an entire row or column. Although join-biclique model has the strength of memory-efficiency, it cannot dynamically adjust the distribution of processing units based on skewed streams.
2. Due to the inconsistency of streams distribution, the balance of DDSMS is lost. It leads to performance degradation of DDSMS.
3. It is necessary to have a good salability of DDSMS for join operation. When the pressure of a node is too large/small, the cluster size can dynamically scale out/down according to its application workloads.

So, in this paper, we propose the mixed workload partitioning strategy D-JB for handling the skewed online join based on the join-biclique model, so as to achieve load balancing and high throughput of DDSMS. The contributions of our work are summarized as follows:

1. We propose a mixed key-based and tuple-based partitioning scheme to handle skewness in each side of the join-biclique model, and a normalized optimization objective by combined with the different cost types involved in the dynamic migration strategy.

2. We present a strategy for redistribution of processing units in two sides of this model. The load balance of D-JB is implemented by repartitioning query nodes logically. And, we prove the efficiency and feasibility by using the different query tasks.
3. We use the operator states manager to migrate processing units between different nodes to ensure the consistency and scalability of the full-history join operation.

The rest of the paper is organized as follows. Section 2 surveys the related work. Then, there are the preliminaries in section 3. And, in Section 4, we give the architecture of D-JB and describe details of data migration. Section 5 gives the results of our experiment evaluation. Finally, Section 6 concludes this paper.

2 Related Work

In recent years, there has been a lot of research work on the online join operation for skew resilience.

Online joins often require un-blocking tuple processing for getting query results in real-time. The join-matrix and join-biclique models are the most popular research models in parallel and distributed environment. Intuitively, the join-matrix model design a join between two data stream R and S as a matrix, where each side corresponds to one relation. In data stream processing, DYNAMIC [4] supports adaptive repartitioning according to the change of data streams. To ensure the load balancing and skew resilience, Aleksandar el.al [6] proposed a multi-stage load-balancing algorithm by using a novel category of *equi-weight* histograms. However, [4, 6] assumes that the number of partitions must be 2^n. So, the matrix structure suffers from bad flexibility. For reducing the operational cost, Junhua et.al [7] proposed the cost-effective stream join algorithm by building irregular matrix scheme. However, The basic model is also matrix-based, data redundancy is still more. JB [5] can save resource utilization significantly. And in order to solve the problem of load imbalance by key-based partitioning, it adopt a hybrid routing strategy, called *ContRand* [5], to make use of both the key-based and random routing strategies. *ContRand* logically divide processing units into disjoint subgroups in each side, and each subgroup contains one or more units. The key-based (content-sensitive) routing strategy is used between the subgroups, and the tuple-based (content-insensitive) routing strategy is used in each subgroup. However, this strategy requires the user to define parameters of subgroupings, and cannot be adjusted dynamically according to the dataflow. Moreover, if the data stream is too skew, it will cause a key to be overloaded in a processing unit and exceed the upper bound of imbalance tolerance. So, we need to partition the tuples with the same key into several processing units. In this case, the problem cannot be reduced to Bin-packing problem [8] in paper [9], which is not considered that a processing unit exceeds the threshold storing tuples with the same key. So, In this paper, we propose the mixed workload partitioning strategy for handling the skewed and varied online join based on the join-biclique model.

3 Preliminaries

We gives the relevant preliminaries for the full-history online join operation in this section.

3.1 Concept Description

In order to make clear the optimization target, the notations involved in our proposed model are listed in Table 1.

Notations	Description
K_{tup}	the key of a tuple
pu	the processing unit (called *task instance* in Storm [10])
m, n	the number of pu in two sides of B
$L(pu)$	the total workload in pu
$\theta(pu)$	load balance factor of pu
θ_{max}	the maximum bound of imbalance factor

Table 1: Table of Notations

Definition 1. At time t, the ***load balance factor*** of a pu is defined as:

$$\theta_t(pu) = \left|L_t(pu) - \bar{L_t}\right| / \bar{L_t} \tag{1}$$

where $\bar{L_t}$ represents the mean of total workloads in PU. $\bar{L_t}$ is defined as:

$$\bar{L_t} = \sum_{pu=1}^{N_{PU}} (L_t(pu))/N_{PU} \tag{2}$$

So, the pu is relatively balanced at time t, if $\theta_t(pu) \leqslant \theta_{max}$.

Definition 2. There are three types of migration at time t, when $\exists$ pu ($pu \in PU$) $\theta_t(pu) > \theta_{max}$. (1) *data immigration*. Tuples with the same K_{tup} at different pus merge to the starting pu. (2) *data emigration*. All tuples with the same k are migrated to other pu. (3) *data splitting*. Some tuples with the same k are migrated to other pu, the other tuples are not moved.

Definition 3. According to the distribution and skewness of data streams, we need to design the migration strategy dynamically. It involves three types of costs: (1) routing cost $C_{routing}$ is the cost of maintaining the routing table for mapping relationships between K_{tup} and pus. (2) duplication cost $C_{duplication}$ is the cost of replicating tuples with the same K_{tup} after data splitting. (3) migration cost $C_{migration}$ is the cost of migrating tuples from a pu to the other.

From definition 2 and 3, it is known that *data immigration* involves $C_{migration}$; *data emigration* involves $C_{routing}$ and $C_{migration}$; *data splitting* involves $C_{routing}$, $C_{duplication}$ and $C_{migration}$.

3.2 Optimization Objective

Let $F = (f_1, f_2, f_3, ...)$ be the set of all migration functions at time t. The migration cost by using each function f_i can be defined as follows.

$$C_t(f_i) = \alpha * C_{routing}(f_i) + \beta * C_{duplication}(f_i) + \gamma * C_{migration}(f_i) \qquad (\alpha + \beta + \gamma = 1) \tag{3}$$

where, α, β and γ are the weight of three costs respectively. In order to determine the specific weights, we use the consumed time to calculate. For detail, the hash routing time T_{hash} affects $C_{routing}$; one tuple transferring time T_{trans} affects $C_{duplication}$ and $C_{migration}$. So, the total time $T_{total} = T_{hash}$ + m*T_{trans} + n*T_{trans}, where, m is the number of duplication tuples, n is the number of migration tuples. Finally, $\alpha = T_{hash}$ / T_{total}; β = m*T_{trans} / T_{total}; γ = n*T_{trans} / T_{total}.

The optimization objective is as below:

$$\begin{aligned} &\min_{f_i \in F} C_t(f_i) \\ s.t. \quad &\theta_t(pu) \leq \theta_{max}, \forall pu \in PU. \end{aligned} \tag{4}$$

The target is to minimize the total costs, while meeting the constraint on load balance factor. It involves the range of K_{tup}, the number of PU and the maximum bound of imbalance factor θ_{max}, which is a combinatorial NP-hard problem. And, this problem is more complex than Bin-packing problem, due to the data inside a *pu* may be split. Therefore, in the next section, we set up a number of heuristics to optimize it.

4 D-JB Model Design

This section gives the architecture of D-JB and the algorithms of data migration.

4.1 D-JB Architecture

The architecture of D-JB is shown in Fig.1, we design the *controller* on Storm by using the join-biclique model. The basic process of the workflow is as follows.

(1) Firstly, data stream R (resp. S) are partitioned by the key-based hash function, stored to n (resp. m) *pu*s, and sent to the opposite side for online join operation.

(2) At each time interval (the setting is 5 seconds in our experiment), we periodically monitor the statistical information of each *pu* load on both sides of B, and collect the information to *controller*. And, we develop migration strategies based on the heuristics (see in section 5.1).

(3) Then, new data streams are temporarily stored in Kafka [11] and postponed online joins before data migration. Meanwhile, we migrate data streams and the state based on migration strategies, and update the routing table synchronously.

(4) Finally, we continue to send data streams and do the online join operation.

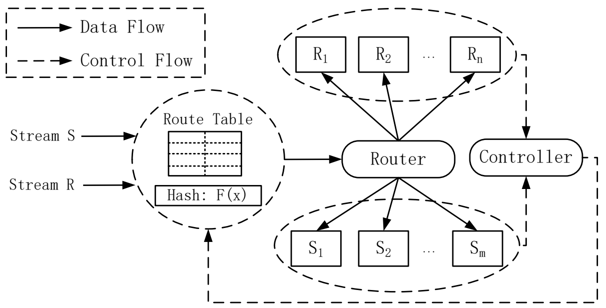


Fig. 1: architecture of D-JB

4.2 Algorithms of Data Migration

The problem of minimizing migration cost is the NP-hard problem. So, we need to set some heuristics to optimize the objective function. There are two migration strategies: the data migration strategy in one side of B, called *internal-side migration* (ISM); and the logical migration strategy in two sides of B, called *side-to-side migration* (S2SM).

Heuristics At time t, it is assumed that the load of pu exceeds the upper limit of the non-equilibrium factor, which is $L_t(pu) > (1+\theta_{max})*\bar{L_t}$, or the load of pu is lower than the lower limit of the non-equilibrium factor, which is $L_t(pu) < (1-\theta_{max})*\bar{L_t}$. In order to satisfy the balance of each pu and minimize data migration, heuristics are as follows.

H1. For data emigration, we can directly migrate keys, if the threshold requirements can be met after migrating these keys, and record them in the routing table.

H2. For data emigration, if the threshold requirements cannot be met after migrating some keys, we need to partition the keys with a larger number of tuples. Then, we migrate some tuples, and record them in the routing table.

H3. For data immigration, if there are records in the routing table, we need to merge data into the key-based hashing pu, and then clear the records in routing table.

According to the above heuristics, we propose the process of moving out tuples, which called $MoveOut(PU_{out}, PU_{in}, RT)$. Firstly, we judge the range of key values of moving out tuples and determine the processing units waiting to move in tuples. Then, for each emigration key, we move out tuples by H1 and H2, and update the routing table.

Next, we propose the process of moving in tuples, which called $MoveIn(PU_{in}, PU_{out}, RT)$. Firstly, we judge the range of key values of moving in tuples and determine pus waiting to move out tuples. Then, for each immigration key, we move in tuples by H3, and update the routing table.

ISM At time t, in order to satisfy the balance of one side of B and minimize the data migration, the ISM algorithm is described in algorithm 1. Firstly, we compute the each $L_t(pu)$ and $\bar{L_t}$ at time t (line 1-4). Then, for the pu that needs to emigrate data, we call the $MoveOut$ algorithm (6-8 lines). Finally, for the pu that needs to immigrate data, we call the $MoveIn$ algorithm (9-11 lines).

Algorithm 1 ISM Algorithm.

```
Require:
    Processing units $PU$ in one side of $B$;
    Routing table $RT$;
    The threshold of imbalance factor $\theta_{max}$;
Ensure:
    The Migration Plan $MP$;
 1: for ($i = 1$; $i < Number\_of\_PU$; $i++$) do
 2:     Computing $L_t(pu_i)$;
 3: end for
 4: Computing $\bar{L_t}$;
 5: for ($i = 1$; $i < Number\_of\_PU$; $i++$) do
 6:     if ($L_t(pu_i) >$ (1+$\theta_{max}$)*$\bar{L_t}$ ) then
 7:         $MoveOut(PU, PU, RT)$;
 8:     end if
 9:     if ($L_{t_{pu_i}} <$ (1-$\theta_{max}$)*$\bar{L_t}$ ) then
10:         $MoveIn(PU, PU, RT)$;
11:     end if
12: end for
```

S2SM Because the stream rate is often dynamic, which results in a large gap between two sides and affects the throughput of DDSMS. In this section, we design the S2SM algorithm for dynamically changing data distribution on both sides. The overall migration is shown in algorithm 2.

Firstly, at time t, we compute the workload of each pu, and the average workload in each side and the whole cluster (line 1-4). Then, we choose the side of data emigration and the side of data immigration (line 5). Finally, for the side of data emigration, we judge pus that need to move out tuples and call the $MoveOut$ algorithm (line 6-10). For the side of data immigration, we judge pus that need to move in tuples and call the $MoveIn$ algorithm (line 11-15).

State Migration BiStream [5] adopts a *requesting* phase and a *scaling* phase to adaptively adjust the resource management. However, it can only be adjusted in the window-based join model. In this section, we introduce the Tachyon [12] as a in-memory file server to store the state information, and use the operator states manager (OSM) [13] to achieve live migration from different nodes. This help us to complete the adaptive resource management in the full-history join model.

5 Evaluation

5.1 Experimental Setup

Environment. The testbed is established on a cluster of fourteen nodes connected by a 1Gbit Ethernet switch. Five nodes are used to transmit data

Algorithm 2 S2SM Algorithm.

```
Require:
    Processing units $PU_m$ and $PU_n$ in two sides of $B$;
    Routing table $RT$;
    The threshold of imbalance factor $\theta_{max}$;
Ensure:
    The Migration Plan $MP$;
 1: for ($i = 1$; $i < (Number\_of\_PU_m + Number\_of\_PU_n)$; $i++$) do
 2:     Computing $L_t(pu_i)$;
 3: end for
 4: Computing $\bar{L_{t_m}}$, $\bar{L_{t_n}}$ and $\bar{L_t}$;
 5: Choosing the $PU_{out}$ and $PU_{in}$ based on $\theta_{max}$
 6: for ($j = 1$; $j < Number\_of\_PU_{out}$; $j++$) do
 7:     if ($L_t(pu_j) >$ (1+$\theta_{max}$)*$\bar{L_t}$ ) then
 8:         $MoveOut(PU_{out}, PU_{in}, RT)$;
 9:     end if
10: end for
11: for ($k = 1$; $k < Number\_of\_PU_{in}$; $k++$) do
12:     if ($L_t(pu_j) <$ (1-$\theta_{max}$)*$\bar{L_t}$ ) then
13:         $MoveIn(PU_{in}, PU_{out}, RT)$;
14:     end if
15: end for
```

source through Kafka. One node serves as the nimbus of Storm, and the remaining eight nodes act as supervisor nodes. Each data source node and the nimbus node have a Intel E5-2620 2.00GHz four-core CPU and 4GB of DDR3 RAM. Each supervisor node has two Intel E5-2620 2.00GHz four-core CPU and 64G of DDR3 RAM. We implement comprehensive evaluations of our prototype on Storm-0.9.5 and Ubuntu-14.04.3.

Data Sets. We use TPC-H benchmark [14] to test the proposed algorithms, and generate the TPC-H data sets using the *dbgen* tool shipped with TPC-H benchmark. All the input data sets are pre-generated into Kafka before feeding to the stream system. We adjust the data sets with different degrees of skew on the join attributes under $Zipf$ distribution by choosing a value for skew parameter z. By default, we set z=1, and generate $10GB$ data.

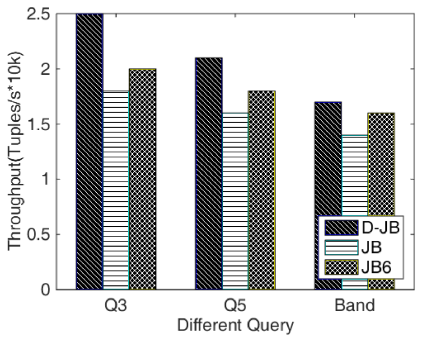
Queries. We employ three join queries, namely two equi-joins from the TPC-H benchmark (Q3 and Q5) and one synthetic band-join (*Band*) is used in [4,5]. The *Band* are expressed as follows:

SELECT *, FORM LINEITEM L1, LINEITEM L2
    WHERE ABS(L1.orderkey-L2.orderkey) <= 1
    AND (L1.shipmode='TRUCK' AND L2.shipinstruct='NONE')
    AND L1.Quantity > 48

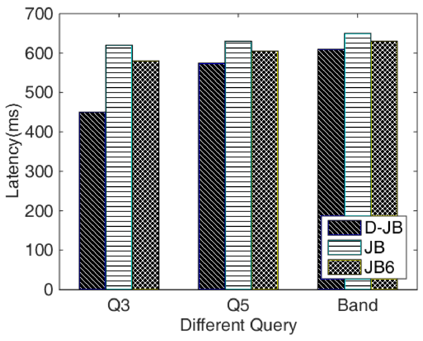
Models. We use three algorithms for evaluating the query performance, namely D-JB, JB [5] and JB6 [5]. D-JB is proposed in this paper. JB divides the *pu*s on average, and each half of the *pu*s corresponds to a data stream. JB6 means there are 6 subgroups in each side for random routing.

5.2 Throughput and Latency

We compare the throughput and latency among the different models by using queries of Q3, Q5 and *Band*. As shown in Fig.2(a), the throughput of D-JB is largest than JB and JB6. However, the throughput of JB is lowest due to it needs to do the whole network broadcast operation, the communication is too large. Fig.2(b) shows that the latency of D-JB is lowest, and the latency of JB is highest.



(a) Throughput



(b) Latency

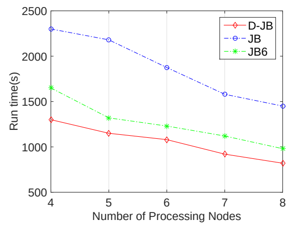
Fig. 2: Throughput and Latency

5.3 Scalability

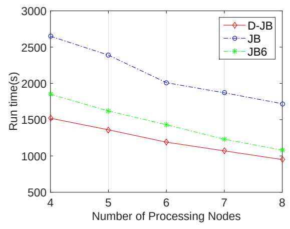
When the cluster scales out, we further study the scalability of D-JB, JB and JB6. Since JB and JB6 do not support scaling out dynamically for full-history online join, we use Taychon to implement these models, which can scale out processing nodes without restarting topologies in Storm. As shown in Fig.3(a), Fig.3(b) and Fig.3(c), the run time of D-JB is the shortest in these models, and the scalability of D-JB is the strongest. Moreover, since Q5 involves the largest number of data streams and the most complex join operation, the run time of Q5 is more than Q3 and *Band*. And, because of *Band* involves only one data stream, it can get the minimum run time.

6 Conclusion

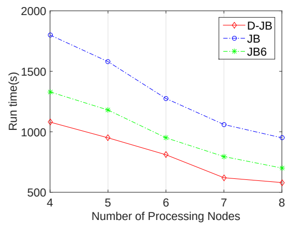
In this paper, we propose online join method for skewed and varied data streams. Based on the join-biclique model, we give a mixed key-based and tuple-based partitioning scheme to handle data skew in one side of the join-biclique model, and present a strategy for redistribution of processing units in two sides of this model. Finally, we design a migration algorithm about state consistency to support full-history joins and adaptive resource management. Our experimental results demonstrate that our proposed framework can be better to deal with data skew, significantly improve the throughput and reduce the latency of DDSMS.



(a) Query for Q3



(b) Query for Q5



(c) Query for Band

Fig. 3: Runtime

References

1. Hwang J, Balazinska M, Rasin A, et al. High-Availability algorithms for distributed stream processing In: Proc. of the 21st International Conference on Data Engineering, pp. 779 - 790. IEEE Press(2005)
2. Fernandez R, Migliavacca M, Kalyvianaki E, et al. Integrating scale out and fault tolerance in stream processing using operator state management In: Proc. of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 725 - 736. ACM Press (2013)
3. Walton C, Dale A, Jenevein R. A taxonomy and performance model of data skew effects in parallel joins In: Proc. of the 17th International Conference on Very Large Data Bases, pp. 537 - 548. ACM Press (1991)
4. Elseidy M, Elguindy A, Vitorovic A, et al. Scalable and Adaptive Online Joins. In: the VLDB Endowmen, 2014, 7(6), pp. 441-452
5. Lin Q, Ooi B C, Wang Z, et al. Scalable Distributed Stream Join Processing In: ACM SIGMOD International Conference on Management of Data, pp. 811 - 825. ACM Press (2015)
6. Vitorovic A, Elseidy M, Koch C Load balancing and skew resilience for parallel joins In: IEEE International Conference on Data Engineering, pp. 313 - 324. IEEE Press (2016)
7. Fang J, Zhang R, Wang X, et al. Cost-Effective Stream Join Algorithm on Cloud System In: the 25th ACM International on Conference on Information and Knowledge Management, pp. 1773 - 1782. ACM Press (2016)
8. Narendra K, Richard K. An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem In: 23rd Annual Symposium on Foundations of Computer Science, pp. 312 - 320. IEEE Press (1982)
9. Fang J, Zhang R, Wang X, et al. Parallel Stream Processing Against Workload Skewness and Variance In: CoRR abs/1610.05121 (2016)
10. Toshniwal A, Taneja S, Shukla A, et al. Storm@twitter In: ACM SIGMOD International Conference on Management of Data, pp. 147 - 156. ACM Press (2014)
11. http://kafka.apache.org/
12. Li H, Ghodsi A, Zaharia M, et al. Tachyon: Memory Throughput I/O for Cluster Computing Frameworks In: LADIS (2013)
13. Ding J, Fu T, Ma R, et al. Optimal Operator State Migration for Elastic Data Stream Processing. HAL - INRIA, 2015, . In: HAL - INRIA, 22(3):1-8 (2013)
14. http://www.tpc.org/tpch