Partitioning Road Network Streams Based on Runtime Correlation Discovery

Chunkai Wang*, Xiaofeng Meng†
School of Information, Renmin University of China, Beijing, China
Email: *chunkai_wang@163.com, †xfmeng@ruc.edu.cn

***Abstract*—Distributed data stream management systems (DDSMS) are often used to analyze and process road network data streams. DDSMS are composed of upper layer relational query systems (RQS) and lower layer stream processing systems (SPS). DDSMS usually need to meet multiple query requests. This often converts RQS submitted by users into different quest tasks on SPS, and executes query plans on different nodes in parallel by partitioning stream according to the values of specific attributes or partitioning keys. However, executing multiple query plans can cause redundant and repetitive partitioning. This article presents the framework of data stream partitioning based on runtime correlation discovery. It combines the runtime positive-correlation partitioning (RPC-partitioning) and the clustering partitioning (Clu-partitioning). In the process of RPC-partitioning, we first use batching schemes to reduce the number of output buffers and then partition data streams using the correlation between different partitioning keys. In the process of Clu-partitioning, we re-partition data streams by clustering for skewed data. Experiments show that our method can reduce the network communication cost from 16% to 20% with two workloads of road network data streams and improve the throughput in DDSMS. It proves the effectiveness of our method, especially on reducing the operational cost in the cloud environment.**

***Keywords*—Stream Processing System, Relational Query System, Partitioning, Runtime Positive Correlation, Clustering**

I. Introduction

DDSMS can be used for real-time processing and analysis of large-scale data streams. There have been many open source SPS, such as Storm [1]. For enhancing the ease of use and processing capability of them, some RQS that contain query languages have been developed, such as Squall [2]. When users submit a query by RQS, the query plan can be converted to topology tasks of Directed Acyclic Graph (DAG) running on SPS. In the process of distributed data stream query processing, a query application for the data stream often make up of multiple queries. It can be parallelized by partitioning data stream into smaller subsets by the value of specific attributes or partitioning keys, and each computing node can execute queries against a reduced amount of data [3]. However, according to the multiple queries, how to avoid re-partitionings and reduce the extra network communication cost is a very challenging problem.

Motivating Example. Consider an application which analyzes the traffic GPS data collected from taxes and buses in real time. We take traffic-monitoring (TM) system as a query example.

Traffic-monitoring needs to process and analyze the traffic GPS data collected from taxes and buses in real-time. The scenario is a typical application of streaming execution. We use GeoLife GPS Trajectories [4] for this workload that contains GPS trajectory data collected by 182 users in a period of over five years, and simulate a traffic-monitoring system. Let us consider a application about calculating the average speed of different dimensions in ten-second time window, and printing the result in one-second window step. The dimensions include latitude, longitude and road ID that can determine the location of this object by using Geocoding API [5]. The SQL description of the application is shown as below.

```
Q1: SELECT altitude, AVG(speed)
    FROM GEOLIFE
    GROUP BY latitude
    WINDOW(SLIDING, 10, 1)
Q2: SELECT latitude, AVG(speed)
    FROM GEOLIFE
    GROUP BY longitude
    WINDOW(SLIDING, 10, 1)
Q3: SELECT roadID, AVG(speed)
    FROM GEOLIFE
    GROUP BY roadID
    WINDOW(SLIDING, 10, 1)
```

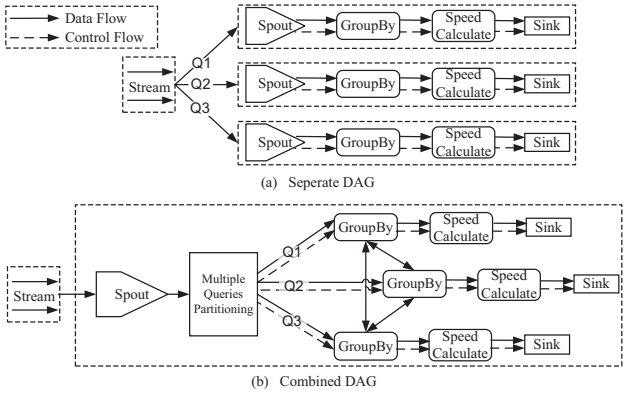


Fig. 1: Topology of multiple queries

For this example, if we build different DAG execution plans for each query, we need to copy multiple data sources (see Fig.1(a)), which will lead to the increase of system overhead. However, if we combine the query set into a compound DAG, we can reduce the duplication of data. It needs to do data partitioning for different sub-queries. As shown in Fig. 1(b), the data stream is partitioned by three partitioning keys, which are latitude, longitude and roadID. And, in order to ensure the correctness of queries, the partitioned data with conflicting need to be copied and re-transmitted. It will cause extra network communication cost and reduce the throughput of

2375-0324/17 $31.00 © 2017 IEEE
DOI 10.1109/MDM.2017.44

DDSMS. Then, the efficient partitioning of input data stream is the key factor to improve the query performance of DDSMS, according to the query tasks and data features.

So, in this paper, we present the design of the partitioning framework based on runtime correlation discovery. The contributions of our work are summarized as follows:

1) We do the compile-time optimization for multiple queries, and find the compatible partitioning keys set based on the different partitioning keys of each query. And according to the window step size, we introduce the batch transmission of data streams.
2) We do the runtime optimization in each window step size and design the RPC-partitioning algorithm, which can perceive the attribute values of data streams, and dynamically partition data streams into segments according to the correlation of attribute values.
3) In order to handle skewed data streams efficiently, we propose a balance optimization approach by using the grid density-based clustering algorithm, which can be a better response to the load balance of DDSMS.
4) We implement our algorithms on Storm and give extensive experimental evaluations to our proposed techniques using two road network workloads, with detailed explanation on observations.

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 the partitioning framework, and describe the details of process. Section 5 gives the results of our experiment evaluation. Finally, Section 6 concludes this paper.

II. Related Work

Parallelization techniques for DDSMS come from earlier research conducted on parallel databases. Database parallelization techniques are summarized in [6]. However, Particularly relevant techniques only support static partitioning and can not be modified for the change of data. Moreover, they are not sufficient to find the optimal partitioning strategy of multiple queries.

For processing of large-scale data streams in real-time, and improving the processing capacity and throughput of DDSMS, we need to partition query tasks and set the query processing unit size. In terms of concurrency of processing data streams, there are three kinds of partitioning mechanism, which are the partitioning object, the partitioning opportunity and the partitioning request.

Partitioning Object. Distributed data stream processing can be implemented in two ways of partitioning object: query plan partitioning [7]–[10] or data partitioning [11]–[13]. Cherniack et al. [7] proposed Aurora* for partitioning query plan using distributed partitioning strategy. DYNAMIC [8] can adaptively partition intra-operator for ensuring minimum state migration overhead. StreamCloud [9] splits queries into sub-queries that are allocated to independent sets of nodes to minimize the distribution overhead. Ying et al. [10] proposed a correlation based load distribution algorithm for avoiding overload and minimizing end-to-end latency. When making operator allocation decisions, they tried to maximize the correlation coefficient between the load statistics of different nodes. However, in the distributed cloud environment, data partitioning is a widely used approach. According to the partitioning opportunity, it can be divided into static partitioning and dynamic partitioning.

Partitioning Opportunity. Static partitioning is defined as the specific partitioning keys when the query is complied. AT&T labs proposed the technique of query-aware partitioning [11], which can analyze any given query set and choose the optimal partitioning scheme, and show how to reconcile potentially conflicting requirements that different queries might place on partitioning. Dynamic partitioning is defined as the re-partitioning data streams based on the data distribution at runtime. Viel et al. [12] proposed Temporal Approximate Dependencies (TADs) approach to mine the dependency of partitioning keys in successive queries. In order to avoid extra communication and reduce overhead, TADs rules can merge conflicting partitioning keys to re-optimize candidate strategies. Cao et al. [13] designed correlation-aware multi-route stream query optimizer (CMR), which can exploit both intra- and inter-stream correlations of stream. CMR firstly can partition data streams based on analyzing and judging the skewed uniformity of each stream, and then measure the similarity of two nodes in terms of the distance in this multi-dimensional space and adopt BIRCH [14] as the tightness measure of a cluster to get the final partitioning keys.

Partitioning Request. There are two partitioning types depending on the user's query requests. The one is single query, which is the relatively simple request. We can use one dimension partitioning methods, such as hash, round-robin, range and so on. The other type is multiple queries, which is the relatively complex request. It is divided into query set partitioning [11], [13], which focuses on the sequential insensitive queries. and successive queries partitioning [12], which focuses on the sequential sensitive queries.

Our proposed the partitioning framework is orthogonal to the above works since we focus on partitioning of batching data streams and meeting the various types of partitioning requests under different partitioning opportunities, and responding to changes in data stream skew.

III. Preliminaries

This section presents the related concepts, and gives the problem description.

A. Concept Description

The stream application submitted to DDSMS is often composed of a series of queries. Multiple queries contain partitioning keys and specific attributes, such as aggregation, join, et al. In order to improve the data stream processing performance and throughput, we can partition input data stream to different processing nodes for achieving the concurrent processing. We should try to reduce the extra network communication cost between processing nodes and reduce the processing latency. To deal with the problem of partitioning based on multiple queries, we define the related concepts formally. Note that Tab. I lists the notations commonly used in this paper.

Notations	Description
S	a stream
SA	a stream application
k	the number of sub-queries in SA
Q_i	the i-th query in SA
PK_i	the partitioning key of the i-th query in SA
$C(Q_i, PK_i)$	the extra network communication cost of Q_i using by PK_i
C_j	the cardinality of attribute j

TABLE I: Table of Notations

Definition 1. *A **window model** can divide the infinite data stream into a number of finite sub-streams.* Each query processing is only aimed at the sub-streams in the current window. Generally, we can set the window size based on the time interval or the number of tuples, and use the tumbling window or sliding window semantics in multiple queries.

Li et al. [15] show how tumbling windows can be used for the efficient evaluation of sliding window queries using panes. They divide overlapping windows into disjoint panes computes sub-queries over each pane, and roll up the pane-queries to compute window-queries. And Sax et al. [16] built a transparent batching layer over Storm. Therefore, we use panes and the batching layer techniques to divide the input data stream, then do correlation analysis of each batching stream.

According to attribute values A and B generated from two different attributes, we can use Eq.1 to judge the correlation of attribute values.

$$Kulc(A,B) = \frac{(P(A|B) + P(B|A))}{2} \tag{1}$$

Because $Kulc(A,B)$ is the mean of two conditional probabilities, it is not affected by the null-transaction [17]. So, we use this judgement to get the concept of *positive correlation of attribute values*.

Definition 2. *The **positive correlation of attribute values** A and B* is denoted as

$$Kulc(A,B) > \frac{1}{2} \tag{2}$$

Moreover, we need to consider the balance of A and B with Eq.3 to judge whether they contain the same change direction. So, we give the definition of *imbalance ratio of attribute values*.

$$IR(A,B) = \frac{|P(A) - P(B)|}{P(A) + P(B) - P(A \cup B)} \tag{3}$$

Definition 3. *The **imbalance ratio of attribute values** A and B* is denoted as

$$IR(A,B) < \frac{1}{2} \tag{4}$$

So, assuming that there are two partitioning keys PK_i (contains m attribute values) and PK_j (contains n attribute values) in a pane, we can get the concept of *runtime positive correlation*.

Definition 4. *The **runtime positive correlation** of two partitioning keys PK_i and PK_j in a pane* is denoted as

$$(Kulc(V_{i_m}, V_{j_n}) > \frac{1}{2}) \wedge (IR(V_{i_m}, V_{j_n}) < \frac{1}{2}) \tag{5}$$

where, V_{i_m} is the different attribute values in partitioning key PK_i, $m \in [1, M]$ (M is cardinality of PK_i); V_{j_n} is the attribute value that has the max count corresponding to V_{i_m}, $n \in [1, N]$ (N is cardinality of PK_j).

LEMMA 1. The attribute value pair (V_{i_m}, V_{j_n}) has ***Anti-monotonic Property***. That is, if a given two partitioning keys PK_i and PK_j are *runtime positive correlation*, then each attribute value pair (V_{i_m}, V_{j_n}) is *runtime positive correlation*.

If a attribute value pair (V_{i_x}, V_{j_y}) in two partitioning keys PK_i and PK_j is not *positive correlation*, we can conclude that PK_i and PK_j are not *runtime positive correlation*. So, we do not need to judge other attribute value pairs about PK_i and PK_j.

In addition, we need to select a partitioning key PK_{init} as the initial partitioning key set. According to the principle of minimizing the re-partitioning, we select the partitioning key with least cardinality as the initial partitioning key, and judge the *runtime positive correlation* by using PK_{init} and other partitioning keys to get the final partitioning keys.

We continue to use the motivating example in Section I to explain the *runtime positive correlation*. We assume that a data stream in a pane is as follows (see Tab. II).

latitude	longitude	roadID	speed
39.98	116.32	0	11.5
39.98	117.16	0	5.2
39.98	120.32	1	18.7
42.01	120.32	2	21.5
42.01	118.29	3	30.9

TABLE II: Sample Data Stream in a Pane

Firstly, according to the cardinality of different partitioning keys ($C_{latitude}$ = 2, $C_{longitude}$ = 4, C_{roadID} = 4), we choose the initial partitioning key PK_{init} = $\{latitude\}$. Then, we need to judge the *runtime positive correlation* of $\{latitude\}$ and $\{longitude\}$, $\{latitude\}$ and $\{roadID\}$ respectively.

Judging {latitude} and {longitude}. $Kulc$(latitude=39.98, longitude=120.32) = $\frac{5}{12} < \frac{1}{2} \Rightarrow$ {latitude, longitude} is not *runtime positive correlation*.

Judging {latitude} and {roadID}. $Kulc$(latitude=39.98, roadID=0) = $\frac{5}{6} > \frac{1}{2}$ And, IR(latitude=39.98, roadID=0) = $\frac{1}{3} < \frac{1}{2}$

However, $Kulc$(latitude=42.01, roadID=2) = $\frac{3}{4} > \frac{1}{2}$ IR(latitude=42.01, roadID=2) = $\frac{1}{2} \not> \frac{1}{2}$

$\Rightarrow$ {latitude, roadID} is not *runtime positive correlation*.

So, the final partitioning key in this pane is $\{latitude\}$.

B. Problem Definition

The runtime correlation discovery is a special type of correlations that can be valid within specific time periods, thus allowing the definition of correlations between evolving entities. A data stream S is defined as an unbounded set of elements (t, $\tau \in$ T), where t is a tuple of attributes taken from a scheme A=$\{A_1, A_2, ..., A_n\}$, τ is the timestamp associated with each tuple, and T is a linear time domain. A SA consists of k sub-queries, SA = $\{Q_1, Q_2, ..., Q_k\}$.

Then, for the entire cluster, the optimization problem is to minimize $\sum_{i \in \mathcal{K}} C(Q_i, PK_i)$ for reducing the run time of queries and improving the resource utilization of DDSMS. In order to reduce the run time of a SA, we should partition the data stream based on the partitioning framework to reduce the network communication cost of the SA. To further improve the efficiency of judging *runtime positive correlation*, we should consider the cost of handling special tuples with low frequency. Ideally, this cost should be adaptively reflected in the frequency threshold of the attribute value x correspond to the column X. Based on the experimental results, we set the frequency threshold to 10%, that is, we can directly filter tuples with the frequency threshold less than 10%, and need not to judge the *runtime positive correlation*.

IV. Implementation

In this section, we design the overall partitioning framework and give the algorithms to solve the optimization problem raised in previous section. As shown in Fig.2, we give the architecture of the partitioning framework based on the query set. It contains the compile partitioning and runtime partitioning.

A. Compile Partitioning

When users submit queries to DDSMS, the compiler generates the logical plan and physical plan firstly. Then, we use the reconciling partitioning method [11] to get the compatible window pane size. Finally, we use the principal of disjunctive normal form to get the candidate partitioning keys. For example, assuming that there are two queries Q_1 and Q_2, Q_1 has the partitioning keys PK_a, PK_b, PK_c, Q_2 has the partitioning keys PK_a, PK_b, PK_d. And the pane size of Q_1 is one-second, the pane size of Q_2 is three-second. So, the reconciling partitioning key of the time interval is three-second. And, the candidate partitioning keys are PK_a, PK_b, PK_c, PK_d.

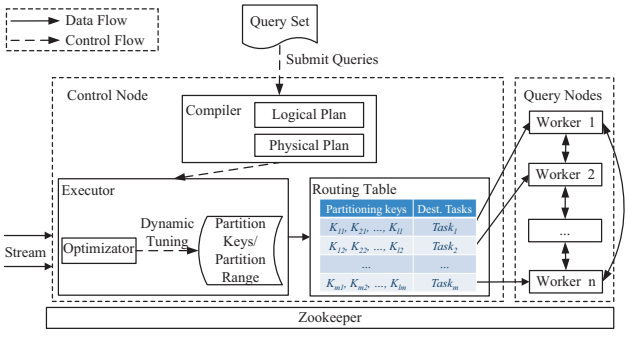


Fig. 2: Architecture of Partitioning Framework

B. Runtime Partitioning

In the runtime phase, we design the runtime dynamic tuning strategy to further optimize the partitioning. It includes the RPC-partitioning of different attribute values and the clu-partitioning based on the grid density clustering algorithm for skewed data streams. Then, we can get the routing table dynamically. As shown in Fig.2, the routing table includes l partitioning keys and m segments in a window pane. The m segments partitioned by l partitioning keys map to m different processing units (called $task$ in Storm). The m tasks belongs to n processors (called $worker$ in Storm). And, the n workers belongs to different processing nodes (called $supervisor$ in Storm). The entire Storm cluster uses Zookeeper [18] to provide the coordinated management of the distributed application.

RPC-partitioning. We propose the RPC-partitioning strategy according to the *runtime positive correlation* of different partitioning keys. Algorithm 1 gives the detailed description. For a stream application SA, firstly, we obtain the original partitioning keys by sub-queries PK_1, PK_2, ... ,PK_k, and judge window pane size $PSize$ and the candidate partitioning keys (Algorithm 1 line 1-4). Then, we choose the minimum cardinality partitioning key as the initial RPC-partitioning key PK_{init}. Line 5-8 in Algorithm 1 depicts this process. Finally, we judge whether RPK_{init} and CPK_i are *runtime positive correlation*, and return the final RPC-partitioning keys (Algorithm 1 line 9-13).

Algorithm 1 RPC-partitioning algorithm.

Require:
The original partitioning keys by sub-queries OPK_1, OPK_2, ... ,OPK_k;
Ensure:
The RPC-partitioning keys $\{RPK_1, RPK_2, ... ,RPK_l\}$;
1: **for** $(i = 1; i \leqslant k; i++)$ **do**
2: Judge the window pane size $PSize$;
3: Get the candidate partitioning keys CPK_1, CPK_2, ... ,CPK_p;
4: **end for**
5: **for** $(i = 1; i \leqslant p; i++)$ **do**
6: Judge the minimum cardinality partitioning key as CPK_{mini}
7: **end for**
8: Set CPK_{mini} as the initial RPC-partitioning key RPK_{init};
9: **for** $(i = 1; i \leqslant p; i++)$ **do**
10: **if** (RPK_{init} and CPK_i are *runtime positive correlation*) **then**
11: Add RPK_i into RPC-partitioning keys;
12: **end if**
13: **end for**

Clu-partitioning. After completing the PRC-partitioning, we need to send the data stream within the pane to different $worker$s for subsequent processing. If the number of data segments partitioned by RPC-partitioning is larger than the number of $worker$s (m>n in Fig.2), we use the default mechanism of load balance in Storm. However, this paper focuses on the communication cost between $worker$s. So, if the data stream partitioned by the RPC-partitioning is too

skewed, it will lead to a large amount of data stream into several $worker$s, while other $worker$s have no data. At this time, although the network transmission between $worker$s is reduced, the delay of the whole query request is raised due to the excessive data stream to be processed by some $worker$s. Therefore, we introduce the method of dynamic clustering to re-partition the data stream. The overall process flow of the cluster partitioning algorithm is shown in Algorithm 2.

Algorithm 2 Clu-partitioning algorithm.

Require:
The m segments partitioned by RPC-partitioning;
The threshold of the load imbalance θ_T;
Ensure:
The k segments partitioned by Clu-partitioning;
1: **if** $(m < n)$ **then**
2: **for** $(i = 1; i \leqslant m; i++)$ **do**
3: **if** $(\theta_i > \theta_T \cdot \frac{PSize}{n})$ **then**
4: $Tgreater$++;
5: **end if**
6: **if** $(\theta_i < \frac{PSize}{n}/\theta_T)$ **then**
7: $Tless$++;
8: **end if**
9: **end for**
10: Set k = n - m + $Tless$;
11: **for** $(i = 1; i \leqslant Tgreater; i++)$ **do**
12: Clustering into k clusters by using STREAM;
13: **end for**
14: **end if**

While the number of segments m is less than the number of $worker$s n (m<n in Fig.2), firstly, we count the number of imbalanced nodes (Algorithm 2 line 2-9). In our experiments, we set θ_T as 3. And then, we can get the number of cluster centers (Algorithm 2 line 10). Finally, we use STREAM [19] to get the final k clusters (Algorithm 2 line 11-13). So, we can re-partition the skewed data stream according to the number of cluster centers.

Finally, in order to ensure the correctness of query results, some partitioned data need to be copied and re-transmitted according to the routing table. We use the queries in section III and the sample data stream in section III.B as the example. In this pane, we partition data by $latitude$, which cause the tuples of $longitude$=120.32 to be partitioned into two different tasks. Therefore, we need to copy one tuple of $longitude$=120.32 and re-transmit to another task for ensuring the query completion.

V. Evaluation

This section presents two workloads of road network scenarios and gives the performance analysis of our proposals by comparing against the baseline approach $TADs$ [12]. The testbed is established on a cluster of fourteen nodes connected by a 1Gbit Ethernet switch. Five nodes are used to transmit data 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.

Road Network Scenarios. We use two workloads of road network scenarios in the experiments. The one is the GeoLife, which has been introduced in detail in Section I, and the other is the simulation of the Linear Road [20]. Linear Road Benchmark (LRB) simulates a toll system for the motor vehicle expressways of a large metropolitan area. In this benchmark, a number of highways are divided into road segments, and the toll charge of each vehicle entering a new segment is calculated based on the average velocity. Let us consider a application about calculating the average speed of different dimensions in ten-second time window, and printing the result in three-second window step.

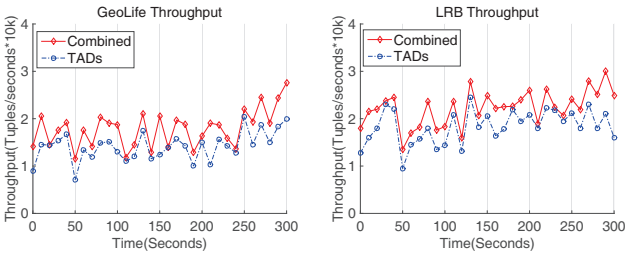
```
Q1: SELECT VID, AVG(Speed)
    FROM LRB
    GROUP BY vid
    WINDOW(SLIDING, 10, 3)
Q2: SELECT Seg, AVG(Speed)
    FROM LRB
    GROUP BY Seg
    WINDOW(SLIDING, 10, 3)
Q3: SELECT XWay, Dir, Seg, AVG(Speed)
    FROM LRB
    GROUP BY XWay, Dir, Seg
    WINDOW(SLIDING, 10, 3)
```

In this two workloads, Geolife has about 24 million records and LRB has 12 million records. For the skewed workload, we generate four streams by using Geolife according to the different $zipf$ distribution. There are 10 thousand unique dimensions, and we generate 24 million records for testing respectively.

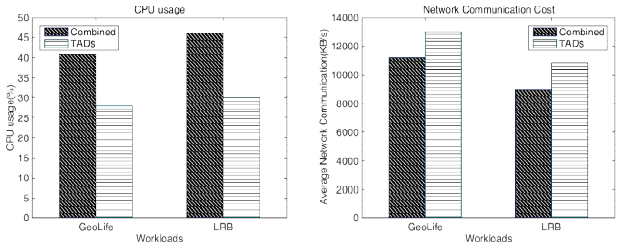
Evaluation Approaches. We use $combined$ to denote our proposed framework mixing two phases, $RPC\text{-}partitioning$ and $Clu\text{-}partitioning$. And, we also include $TADs$ as the baseline approach. $TADs$ as a temporal functional dependency which holds with exceptions over a data window for mining module to re-evaluate partitioning strategies and choose a new compatible partitioning strategy. In the experiment, we set the spout parallelism as 1, the batch size as a pane size $PSize$, set the partitionBolt parallelism as 8 (the number of $supervisors$), and set other bolts parallelism as 32 (the total core number of nodes used by $supervisors$).

Performance Analysis. According to the two workloads, the overall throughput is limited by the network speed in which tuples sent through the network. Fig.3(a) and Fig.3(b) compare the throughput of partitioning done by two evaluation approaches. As seen in the results, $combined$ provide on average of around 26% and 21% higher throughput than $TADs$ in Geolife and LRB, respectively. In addition, Fig.3(c) and Fig.3(d) display two comparisons of the CPU usage and communication cost using each algorithm. We can find that $combined$ is not only the effective way to reduce network communication cost from 16% to 20%, but also can improve the CPU usage from 13% to 16%, respectively. It shows that $combined$ can improve the efficiency and throughput of the cluster and reduce the network communication cost.

Data Skew Phenomenon. In order to verify the skewed data, we compare the network communication cost and run time between $combined$ and $TADs$, respectively. Fig.4 shows



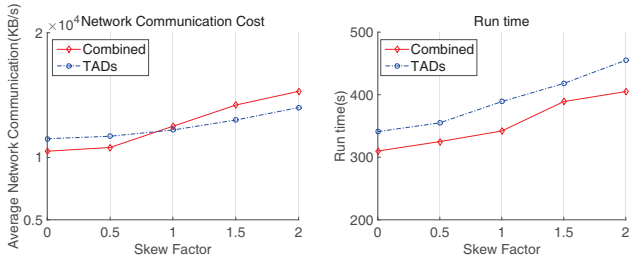
(a) GeoLife Throughput (b) LRB Throughput



(c) CPU Usage (d) Communication Cost

Fig. 3: The Performance of Geolife and LRB

the change of running Geolife workload according to the different skew factor of $zipf$ distribution. By comparing Fig.4(a) and Fig.4(b), we can find that the run time of $combined$ is less than $TADs$ with the increase of the skew factor, although the network communication cost of $combined$ is slightly increased.



(a) Communication Cost (b) Run time

Fig. 4: The Performance of Skewed Geolife

VI. Conclusion

This paper proposes the partitioning framework of runtime correlation discovery to handle the different requirements of multiple query processing for road network data streams. Through the introduction of RPC-partitioning and Clu-partitioning algorithms, we can achieve the better query efficiency. In the future, we plan to consider the communication cost between different tasks in the same $supervisor$. In this case, we can do the fine-grained monitoring for data streams communication cost.

VII. Acknowledgments

This research was partially supported by the grants from the Natural Science Foundation of China (No. 91646203, 61532016, 61532010, 61379050); the National Key Research and Development Program of China (No. 2016YFB1000602, 2016YFB1000603); Specialized Research Fund for the Doctoral Program of Higher Education (No. 20130004130001), and the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University (No. 11XNL010).

References

[1] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, and J. Donham, "Storm@twitter," 2014.

[2] https://github.com/epfldata/squall/.

[3] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin, "Flux: An adaptive partitioning operator for continuous query systems," pp. 25–36, 2003.

[4] Y. Zheng, L. Zhang, X. Xie, and W. Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *International Conference on World Wide Web, WWW 2009, Madrid, Spain, April*, 2009, pp. 791–800.

[5] http://api.map.baidu.com/lbsapi/cloud/webservice-geocoding.htm.

[6] D. Dewitt, "Parallel database systems: the future of high performance database systems," *Communications of the Acm*, vol. 35, no. 6, pp. 85–98, 1992.

[7] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing." in *CIDR*, 2003.

[8] M. Elseidy, A. Elguindy, A. Vitorovic, and C. Koch, "Scalable and adaptive online joins," *Proceedings of the Vldb Endowment*, vol. 7, no. 6, pp. 441–452, 2014.

[9] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE Transactions on Parallel & Distributed Systems*, vol. 23, no. 12, pp. 2351–2365, 2012.

[10] Y. Xing, S. Zdonik, and J. H. Hwang, "Dynamic load distribution in the borealis stream processor," in *International Conference on Data Engineering, 2005. ICDE 2005. Proceedings*, 2005, pp. 791–802.

[11] T. Johnson, S. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck, "Query-aware partitioning for monitoring massive network data streams," *Proc on Management of Data*, pp. 1135–1146, 2008.

[12] E. Viel and H. Ueda, "Data stream partitioning re-optimization based on runtime dependency mining," in *IEEE International Conference on Data Engineering Workshops*, 2014, pp. 199–206.

[13] L. Cao and E. A. Rundensteiner, "High performance stream query processing with correlation-aware partitioning," *Proceedings of the Vldb Endowment*, vol. 7, no. 4, pp. 265–276, 2014.

[14] T. Zhang, "Birch: an efficient data clustering method for very large databases," *Acm Sigmod Record*, vol. 25, no. 2, pp. 103–114, 1996.

[15] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "No pane, no gain: efficient evaluation of sliding-window aggregates over data streams," *Acm Sigmod Record*, vol. 34, no. 1, pp. 39–44, 2005.

[16] M. J. Sax and M. Castellanos, "Building a transparent batching layer for storm," 2014.

[17] X. Wu, C. Zhang, and S. Zhang, "Efficient mining of both positive and negative association rules," *Acm Transactions on Information Systems*, vol. 22, no. 3, pp. 381–405, 2004.

[18] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," 2010, p. 653C710.

[19] L. O'Callaghan, N. Mishra, A. Meyerson, and S. Guha, "Streaming-data algorithms for high-quality clustering," in *International Conference on Data Engineering*, 2002, pp. 685–694.

[20] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear road: a stream data management benchmark," in *Thirtieth International Conference on Very Large Data Bases*, 2004, pp. 480–491.