Sliding Window Top-K Monitoring over Distributed Data Streams

Zhijin Lv[1], Ben Chen[1], and Xiaohui Yu[1,2] ⋆

[1] School of Computer Science and Technology Shandong University, Jinan, Shandong, China, 250101
[2] School of Information Technology, York University, Toronto, ON, Canada, M3J 1P3
allen3jin@163.com, CBStubborn@163.com, xyu@sdu.edu.cn

Abstract. The problem of distributed monitoring has been intensively investigated recently. This paper studies monitoring the top k data objects with the largest aggregate numeric values from distributed data streams within a fixed-size monitoring window W, while minimizing communication cost across the network. We propose a novel algorithm, which reallocates numeric values of data objects among distributed monitoring nodes by assigning revision factors when local constraints are violated, and keeps the local top-k result at distributed nodes in line with the global top-k result. Extensive experiments are conducted on top of Apache Storm to demonstrate the efficiency and scalability of our algorithm.

Keywords: Data Stream, Distributed Monitoring, Top-K Query, Stream Processing

1 Introduction

The prior studies for distributed top-k query [5, 7] focus on providing results to one-time top-k queries in distributed settings. These studies are not suitable to continuously query top-k result over distributed data streams. In this paper, we study a new problem of distributed top-k monitoring, which is continuously querying the top k data objects with the largest aggregate numeric values over distributed data streams within a fixed-size monitoring window. Each data stream contains a sequence of data objects associated with numeric values, and the aggregate numeric value of each data object is calculated from distributed data streams. The continuous top-k query we studied is restricted to the most recent portion of the data stream, and the numeric values of data objects are changed correspondingly as the monitoring window slides.

The study of distributed top-k monitoring is significant in a variety of application scenarios, such as network monitoring, sensor data analysis, web usage logs, and market surveillance. The purpose of many applications tends to track the exceptionally large (or small) numeric values relative to the major numeric

⋆ Corresponding author.

values of data objects. For example, in the field of traffic flow monitoring, it is necessary to continuously monitor the top k largest number of road traffic within the last 15 minutes in order to monitor the traffic jams in time. Another example, consider a system that monitors a large network for distributed denial of service (DDoS) attacks. The DDoS attacks may issue an unusually large number of Domain Name Service (DNS) lookup requests to distributed DNS servers from a single IP address. Hence, it is necessary to monitor the DNS lookup requests with potentially suspicious behavior. In this case, the monitoring infrastructure continuously reports the top k IP addresses with the largest number of requests at distributed servers in recent time. Since requests are frequent and rapid at distributed DNS servers, the solution of forwarding all requests to a central location and processing them is infeasible, incurring a huge communication overhead.

The major challenge of our top-k monitoring problem is numeric values of data objects varying independently at distributed nodes. Tracking the top k data objects with the largest aggregate numeric values from distributed nodes results in huge communication overhead, because the global top-k result is affected by local changes of data objects at distributed nodes. It is imperative to find solutions that can effectively monitor the global top-k result, while minimizing communication cost across the network.

Existing algorithms for distributed top-k query such as the Threshold Algorithm [7] focus on efficiently providing results to one-time top-k queries. Though distributed top-k monitoring could be implemented by repeatedly executing one-time query alogrithms, it is useless to execute query if the top-k result remains unchanged. These algorithms do not include mechanisms for detecting the changes of top-k result, incurring unnecessary communication overhead. Most of solutions proposed for sliding window top-k monitoring [14, 17] are inappropriate to our monitoring problem, because their ranking function is based on the dominance relationship of data objects, rather than the aggregate numeric values from distributed data streams. Babcock and Olston present an original algorithm for distributed top-k monitoring [3], which maintains arithmetic constraints at distributed data sites to ensure that the provided top-k answer remains valid. Their algorithms assume that a single node may violate constraints each time, which is unrealistic. Moreover, it is not suitable to the case of sliding window, which focuses on the impact of recent data objects.

For continuous data monitoring, we adopt a time-based sliding window model [14], where the data objects generated within W time-stamps from the current time-stamp are target for monitoring. In this paper, we consider a model in which there is one coordinator node $\mathcal{C}$ and a set of m distributed nodes N connected to the coordinator node as shown in Fig. 1.

The coordinator node tracks the global top-k result and assigns constraints to each monitoring node, at which local top-k result should be in alignment with the global top-k result. Each monitoring node receives data objects from an input stream and detects the potential violations of local constraints whenever the window slides. When local constraints are violated at some monitoring nodes, it is necessary to send the violated data objects and their numeric values to

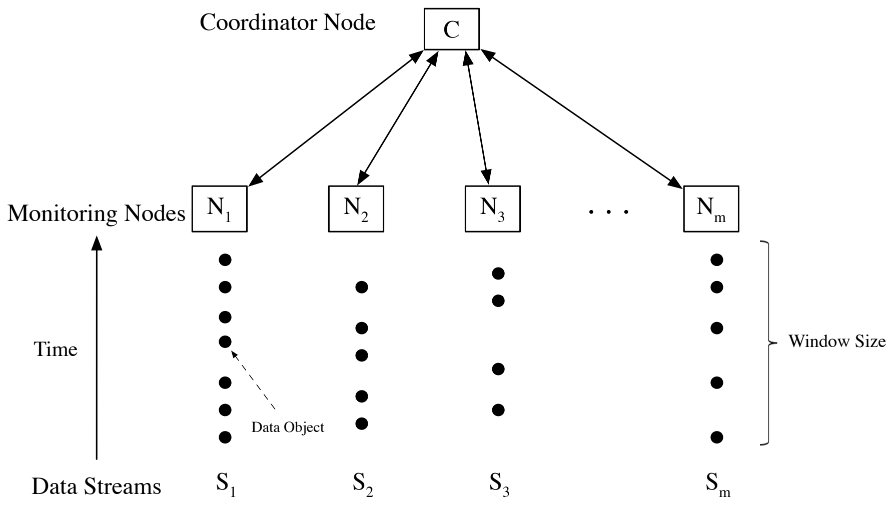


Fig. 1. Sliding Window Distributed Monitoring Architecture

the coordinator node. Then, the coordinator node tries to resolve the violations, called *partial resolution*. If the global constraint is satisfied by assigning new local constraints to the violated nodes, then the global top-k result remains valid. Otherwise, the coordinator node requests all distributed nodes for current numeric values of violated objects to determine whether the global constraint is still satisfied. We refer to this process as *global resolution*, which does not always occur.

We implement our distributed top-k algorithm on top of Apache Storm [1], an open-source distributed stream processing platform, on which we conduct extensive experiments to evaluate the performance of our solutions.

Our main contributions can be summarized as follows.

– We investigate the problem of sliding window top-k monitoring over distributed data streams. To the best of our knowledge, there is no prior work regarding this.
– We propose a novel algorithm for top-k monitoring over distributed data streams, which achieves a significant reduction in communication cost.
– We implement our algorithms on top of Apache Storm, and conduct extensive experiments to evaluate the performance of our algorithms with real world data, which have demonstrated the efficiency and scalability of our algorithms.

The rest of the paper is organized as follows. Section 2 reviews previous work on monitoring over distributed data streams. Section 3 formally defines the top-k monitoring problem studied in this paper. We describe our top-k monitoring algorithm in detail in Section 4. Section 5 experimentally evaluates the performance of our algorithms. Finally, we conclude the paper in Section 6.

2 Related Work

Prior work on monitoring distributed streams can be classified into two categories. One category is monitoring functions over the union of distributed data streams, and another is monitoring a ranking function, which is based on the dominance relationship of data objects over distributed data streams.

In the first category, algorithms have been proposed for continuous monitoring of sums and counts [10], heavy hitters and quantiles [18], and ratio queries [9]. Sharfman et al. [16] present a geometric monitoring (GM) approach for efficiently tracking the value of a general function over distributed data relative to a given threshold. Followup work [8, 11, 13] proposed various extensions to the basic method. Recently, Lazerson et al. [12] presented a CB (for Convex/Concave Bounds) approach, which is superior to GM in reducing computational complexity, by several orders of magnitude in some cases. Cormode et al. [6] introduced the Continuous Monitoring Model focusing on systems comprised of a coordinator and n nodes generating or observing distributed data streams. The goal shifts to continuously compute a function depending on the information available across all n data streams and a dedicated coordinator.

There are also plenty of works in the second category. These works study the monitoring problem with essentially different semantics compared to the first category. Mouratidis et al. [14] proposed an efficient technique to compute top-k queries over sliding windows. They make an interesting observation that a top-k query can be answered from a small subset of the objects called k-skyband [15]. Existing top-k processing solutions are mainly based on the dominance property between data stream. The dominance property states that object O_a dominates object O_b iff O_a has a higher score than O_b. Amagata et al. [2] presented algorithms for distributed continuous top-k dominating query processing, which reduces both communication and computation costs. Unfortunately, their algorithms are inappropriate for the top-k monitoring problem we studied.

Further problems related to our distributed top-k monitoring are distributed one-time top-k queries [4, 5, 7]. Fagin et al. [7] examined the Threshold Algorithm (TA) and considered both exact answers and approximate answers with relative error tolerance. TA goes down the sorted lists in parallel, one position at a time, and calculates the sum of the values at that position across all the lists. the sum of the values is called “threshold”. TA stops when it finds k objects whose values are higher than the “threshold” value. Cao et al. [5] proposed an efficient algorithm called “Three-Phase Uniform Threshold” (TPUT), which reduces network bandwidth consumption by pruning away ineligible objects, and terminates in three round-trips regardless of data input. However, these studies are interested in algorithms that can obtain the initial top-k result efficiently and provide top-k result to one-time queries. Our study focuses on monitoring whether the top-k result have changed after an initial answer has been obtained.

3 Problem Definitions

Now we more formally define the problem studied in this paper. As described above, there is one coordinator node $\mathcal{C}$ and m distributed monitoring nodes $N_1, N_2, ..., N_m$. Each monitoring node N_j continuously receives data records from an input stream. Collectively, the monitoring nodes track a set $\mathcal{O}$ of n logical data objects $\mathcal{O} = \{O_1, O_2, ..., O_n\}$. Each data object is associated a numeric value within the current monitoring window. The numeric value of each data object is updated at distributed nodes as the monitoring window slides. For each monitoring node N_j, we define partial numeric values $C_{1,j}(t), C_{2,j}(t), ..., C_{n,j}(t)$ representing node N_j's view of the data stream S_j within monitoring window W at time t, where

$$C_{i,j}(t) = |\{O_i^{t'} \in S_j \mid t - t' \leq W\}|. \tag{1}$$

The aggregate numeric value of each object O_i from distributed monitoring nodes is defined to be $C_i(t) = \sum_{1 \leq j \leq m} C_{i,j}(t)$. Tracking $C_i(t)$ exactly requires alerting the coordinator node every time data object O_i arrives or expires, so the goal is to track $C_i(t)$ approximately within an ϵ-error. The coordinator node is responsible for tracking the top k data objects within a bounded error tolerance. We define the approximate top-k set maintained by the coordinator node as $\mathcal{T}$, which is considered valid if and only if:

$$\forall O_a \in \mathcal{T}, \forall O_b \in \mathcal{O} - \mathcal{T} : C_a(t) + \epsilon \geq C_b(t) \tag{2}$$

where $\epsilon \geq 0$ is an user-specified approximation parameter. If $\epsilon = 0$, then the top-k set is exact, otherwise a corresponding degree of error is permitted in the top-k set. The goal of our approach is to provide an approximate top-k set that is valid within an ϵ-error in the case of sliding window, while minimizing the overall communication cost to the monitoring infrastructure.

3.1 Revision Factors and Slack

We realize that the global top-k set is valid, if distributed monitoring nodes have the same top-k set locally. Since the actual local top-k set at distributed monitoring nodes may be differ from the global top-k set, we use $revision\ factors$, labeled $\delta_{i,j}$, to reallocate the numeric values of data object O_i to monitoring node N_j to satisfy the following local constraint:

$$\forall O_a \in \mathcal{T}, \forall O_b \in \mathcal{O} - \mathcal{T} : C_{a,j}(t) + \delta_{a,j} \geq C_{b,j}(t) + \delta_{b,j} \tag{3}$$

In addition, the coordinator node maintains partial revision factors of data objects as global slack, labeled $\delta_{i,0}$. To ensure correctness, the sum of revision factors for each data object O_i should satisfy: $\sum_{0 \leq j \leq m} \delta_{i,j} = 0$.

In order to reallocate numeric values of data objects among nodes, it is necessary to compute additional slacks of data objects at each node. We define resolution set which contains data objects from global top-k set $\mathcal{T}$ and all violated objects as $\mathcal{R}$. Our algorithm selects the maximum values $\mathcal{P}_j$ of data object not in the resolution set $\mathcal{R}$ as a baseline for computing additional slacks of data objects at each node N_j:

$$\mathcal{P}_j = \max_{O_i \in \mathcal{O}-\mathcal{R}} (C_{i,j}(t) + \delta_{i,j}) \tag{4}$$

Thus, the overall slack $\mathcal{S}_i$ for each data object O_i from the resolution set $\mathcal{R}$ is given by:

$$\forall O_i \in \mathcal{R} : \mathcal{S}_i = \sum_{1 \le j \le m} (C_{i,j}(t) - \mathcal{P}_j) = C_i(t) - \sum_{1 \le j \le m} \mathcal{P}_j \tag{5}$$

It is important to reallocate the overall slack of data objects among the coordinator node and distributed monitoring nodes. If the slack is tight at monitoring node, the violation of local constraints would be frequent. However, smaller slack at coordinator node results in higher probability of violation of global constraints. The optimum slack allocation polices balances these two costs.

3.2 Sliding Window Unit

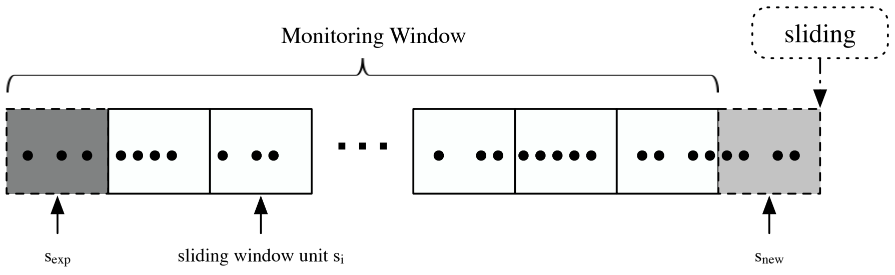


Fig. 2. Sliding Window Unit Structure

In the sliding window scenario, distributed monitoring nodes track numeric values of data objects within the monitoring window W. Based on the arrival order of each object in W, the data objects in window W could be partitioned into several small window unit $s_0, s_1, ..., s_{l-1}$ ($l = \frac{W}{w}$). The size of sliding window unit w is specified according to the actual application scenario. Small window unit is more suitable for near real-time applications, but incurring more communication and computation costs.

As shown in Fig 2, the monitoring window W slides, whenever a new sliding window unit s_{new} has been created and the expired window unit s_{exp} is removed. Thus, the partial numeric values $C_{i,j}(t)$ of each data object O_i at monitoring node N_j updates within the new monitoring window W' ($W' = W + s_{new} - s_{exp}$). Obviously, the changes of data objects may violate the current local constraints.

4 Top-K Monitoring Algorithm

We now describe our algorithm in detail for sliding window top-k monitoring over distributed data streams. At the outset, the coordinator node initializes the global top-k set by running an efficient algorithm for one-time top-k queries, e.g. TPUT algorithm [5]. Once the global top-k set $\mathcal{T}$ has been initialized, the coordinator node $\mathcal{C}$ sends a message containing $\mathcal{T}$ and initial revision factors $\delta_{i,j} = 0$ corresponding to each monitoring node N_j. Upon receiving this message, the monitoring node N_j creates local constraint (3) from $\mathcal{T}$ and revision factors to detect potential violations due to local changes of data objects.

If one or more local constraints are violated, the global top-k set $\mathcal{T}$ may have become invalid. We use a distributed process called *resolution algorithm* to determine whether current top-k set is still valid and resolve the violations if not.

4.1 Resolution Algorithm

Resolution algorithm is initiated if one or more local constraints are violated at some monitoring nodes $N_{\mathcal{R}}$. Our resolution algorithm consists of three phases, and the third phase does not always occur.

Algorithm 1 Partial Resolution Algorithm

```
Input: $\mathcal{T}, \delta_{i,j}, \{\mathcal{R}_j\}, \{C_{i,j}(t)\}$
Output: succeed or failed
1: for $\forall N_j \in N_{\mathcal{R}}$ do
2:     $bound_j \leftarrow \max(C_{i,j}(t) + \delta_{i,j}), O_i \in \mathcal{R}_j - \mathcal{T}$ //find the max value of violated
       data object at each violated node.
3: /* try to reallocate the numeric value of $O_i \in \mathcal{T}$ among violated nodes to resolve
   violation of constraints */
4: for $\forall O_i \in \mathcal{T}$ do
5:     $slack_i \leftarrow \delta_{i,0}$
6:     for $\forall N_j \in N_{\mathcal{R}}$ do
7:         $used_i \leftarrow bound_j - (C_{i,j}(t) + \delta_{i,j})$
8:         $slack_i \leftarrow slack_i - used_i$
9: $bound_0 \leftarrow \max(\delta_{i,0}), O_i \notin \mathcal{T}$
10: /* determine whether the coordinator node satisfies the Constraint (3) */
11: for $\forall O_i \in \mathcal{T}$ do
12:     if $slack_i + \epsilon < bound_0$ then
13:         return failed
14: return succeed
```

– **Local Alert Phase.** The monitoring node N_j at which violated constraints have been detected sends a message containing a local resolution set $\mathcal{R}_j$ (containing data objects from global top-k set $\mathcal{T}$ and current local top-k set) and a set of partial numeric values $C_{i,j}(t)$ of data object O_i in the local resolution set to coordinator node.

– **Partial Resolution Phase.** The coordinator node determines whether all violations can be solved based on the messages from violated nodes $N_{\mathcal{R}}$ and itself alone according to the Algorithm 1. If the coordinator node resolves all violations successfully by assigning updated revision factors to the violated nodes, the global top-k set remains unchanged and resolution process terminates. Otherwise, the coordinator node is unable to rule out all violations during this phase, the third phase is required.

– **Global Resolution Phase.** The coordinator node requests the current partial values $C_{i,j}(t)$ of data objects O_i in overall resolution set $\mathcal{R} = \cup_{N_j \in N_{\mathcal{R}}} \mathcal{R}_j$ as well as the baseline value $\mathcal{P}_j$ from all monitoring nodes. Once the coordinator node receives responses from all monitoring nodes, it computes a new top-k set and new revision factors of data objects in the resolution set $\mathcal{R}$, and notifies all monitoring nodes of a new top-k set $\mathcal{T}'$ and their new revision factors. Our algorithm adopts even policy to divide the overall slack $\mathcal{S}_i$ of data object O_i among monitoring nodes and coordinator node. The revision factors allocation algorithm is described in Algorithm 2.

For notational convenience, we extend our notation for partial numeric values and baseline value to the coordinator node by defining $C_{i,0}(t) = 0$ for all data object O_i and $\mathcal{P}_0 = \max_{O_i \in \mathcal{O}-\mathcal{R}} \delta_{i,0}$. We also define nodes set $\mathcal{A}$ as all nodes involved in the resolution process. For each object O_i, $C_{i,\mathcal{A}}(t) = \sum_{0 \le j \le m}(C_{i,j}(t) + \delta_{i,j})$. Similarly, we define the sum of the baseline values from the nodes set $\mathcal{A}$, $\mathcal{P}_{\mathcal{A}} = \sum_{0 \le j \le m} \mathcal{P}_j$.

Algorithm 2 Revision Factors Allocation Algorithm

```
Input: $\mathcal{T}', \mathcal{R}, \{\mathcal{P}_j\}, \{C_{i,j}(t)\}$
Output: $\{\delta_{i,j}\}$
1: /* compute the overall slack of $O_i \in \mathcal{R}$ */
2: for $\forall O_i \in \mathcal{R}$ do
3:     if $O_i \in \mathcal{T}'$ then
4:         $\mathcal{S}_i = C_{i,\mathcal{A}}(t) - \mathcal{P}_{\mathcal{A}} + \epsilon$
5:     else
6:         $\mathcal{S}_i = C_{i,\mathcal{A}}(t) - \mathcal{P}_{\mathcal{A}}$
7: /* compute new revision factors $\{\delta_{i,j}\}$ using even policy */
8: for $\forall O_i \in \mathcal{R}$ do
9:     $ps \leftarrow \mathcal{S}_i/(1+m)$
10:    for $j = 0 \rightarrow m$ do
11:        if $O_i \in \mathcal{T}'$ and $j = 0$ then
12:            $\delta_{i,j} = \mathcal{P}_j - C_{i,j}(t) + ps - \epsilon$
13:        else
14:            $\delta_{i,j} = \mathcal{P}_j - C_{i,j}(t) + ps$
```

4.2 Correctness and Cost Analysis

The goal of our algorithm is to keep the local top-k set at each node in line with the global top-k set. If the global constraint is satisfied, the global top-k remains valid. When local constraints are violated at distributed nodes, our algorithm reallocates the numeric values of violated data objects by assigning revision factors to distributed nodes.

Example 1. Consider a simple scenario with two monitoring nodes N_1 and N_2 and three data objects O_1, O_2 and O_3, and current revision factors are zero. At time t, the current data values at N_1 are $C_{1,1}(t) = 4, C_{2,1}(t) = 6$ and $C_{3,1}(t) = 10$ and at N_2 are $C_{1,2}(t) = 3, C_{2,2}(t) = 4$ and $C_{3,2}(t) = 3$. Let $k = 1, \epsilon = 0$, the current top-k set $\mathcal{T} = \{O_3\}$. However, the local top-k set at N_2 is $\{O_2\}$, which violates the constraints. Our algorithms find that partial resolution phases are failed to resolve the violations, due to slack at coordinator node are zero. The global resolution phase computes the new revision factors assigned to the monitoring nodes, at coordinator node are $\delta_{2,0} = 1, \delta_{3,0} = 2$ and at N_1 are $\delta_{2,1} = -1, \delta_{3,1} = -4$ and at N_2 are $\delta_{2,2} = 0, \delta_{3,2} = 2$. Then, the local constraints at distributed node are satisfied and the global top-k set $\mathcal{T} = \{O_3\}$ is valid.

Data objects not in the resolution set $\mathcal{R}$ can not be candidates for new top-k set $\mathcal{T}'$, because their numeric values satisfy the current local constraints. Therefore, the sum of all baseline values $\mathcal{P}_{\mathcal{A}}$ should be less than the minimum numeric values $C_l(t)$ of data object O_l in the previous top-k set $\mathcal{T}$. Furthermore, each data object O_i in the new top-k set $\mathcal{T}'$ satisfies: $C_i(t) \geq C_l(t) \geq \mathcal{P}_{\mathcal{A}}$. And, the overall slacks of data objects in resolution set $\mathcal{R}$ satisfy the following inequation:

$$\forall O_a \in \mathcal{T}', \forall O_b \in \mathcal{R} - \mathcal{T}' : \mathcal{S}_a \geq \mathcal{S}_b \textit{ and } \mathcal{S}_a \geq 0 \tag{6}$$

As described in Algorithm 2, we evenly allocate the overall slack $\mathcal{S}_i$ of each object O_i in the resolution set $\mathcal{R}$ to all nodes. As a result, the new local top-k set computed by new revision factors at distributed nodes must be in line with the new global top-k set, and local constraints (3) at distributed nodes are satisfied.

Our resolution algorithm maintains global slack at the coordinator node, which is significant at partial resolution phase. If the partial resolution phase resolves the violations successfully, the third phase does not require. Thus, the communication cost at this phase is just assigning updated revision factors to the violated node, and number of $2 * |N_{\mathcal{R}}|$ messages are exchanged altogether. If all three phases are required, the total of $|N_{\mathcal{R}}| + 3m$ messages are necessary to perform complete resolution.

If the global slack retained at coordinator node is tight, the probability of failure at partial resolution phase becomes high, incurring more communication cost at global resolution phase. However, tight slacks at distributed monitoring nodes result in frequent violations of local constraints. Our even policy for allocating additional slacks balances these two costs well.

5 Experiments

In this section, we provide an experimental evaluation on the communication cost of our resolution algorithm. We implement two different top-k monitoring algorithms as baseline algorithms. One algorithm retains zero slack at coordinator node (LSA), and another algorithm retains zero slack at distributed monitoring nodes (GSA).

5.1 Setup

The experiments are conducted on a cluster of 16 Dell R210 servers with Gigabit Ethernet interconnect. Each server has a 2.4 GHz Intel processor and 8 GB RAM. As shown in Fig. 1, one server works as the coordinator node and the remaining nodes work as the monitoring nodes. The monitoring node can exchange messages with the coordinator node, but can not communicate with each other. Additionally, the coordinator node can send broadcast messages received by all monitoring nodes.

We implement our algorithm on top of Apache Storm, a free and open source distributed realtime computation system, which makes it easy to reliably process unbounded streams of data. All of nodes are implemented as *Bolt* components within the Storm system, and receive data objects continuously from a *Spout* component, which is a source of data streams. They constitute a $Topology$ run on the Storm system. The version of Apache Storm we used is 1.0.2 in our experiments.

We evaluated the efficiency of our algorithm against sliding window unit size w, number of monitoring nodes m, approximation parameter ϵ respectively. The default values of the parameters are listed in Table 1. Parameters are varied as follows:

– number of monitoring nodes m: 3, 5, 8, 10, 15
– sliding window unit size w: 5s, 10s, 15s, 20s
– approximation parameter ϵ: 0, 25, 50, 75, 100

Table 1. Experimental parameters

Notation	**Definition(Default Value)**
k	number of objects to track in top-k set (**10**)
m	number of monitoring nodes (**10**)
ϵ	approximation parameter (**0**)
W	monitoring window size (**15min**)
w	sliding window unit size (**10s**)

5.2 Data and Queries

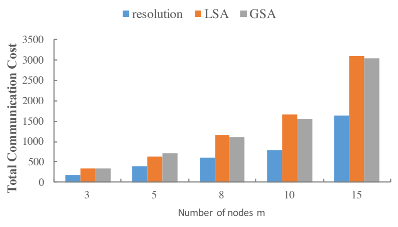
We conducted our experiments on both synthetic dataset and real dataset. The datasets are described in detail as follows:

- **Synthetic Dataset**: The synthetic dataset includes random data records, which follow Zipf distribution [19]. The distribution parameter we used is 2. Each data record contains ID of data object and the time of generation. The goal of experiment is continuous querying the top k data objects with the largest number of occurrences.

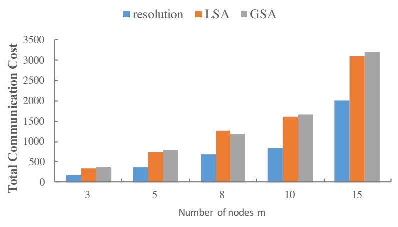
- **Real Dataset**: The real dataset consists of a portion of real vehicle passage records from the traffic surveillance system of a major city. The dataset contains 5,762,391 passage records, which are generated within 6 hours (about 267 passage records per second), and involves about 1000 detecting locations on the main roads. Our experiments continuously monitor the top k detecting locations with the largest number of vehicle passage records.

Our experiments continuously monitor the top k data objects over distributed data streams within last 15 minutes, and the total communication cost is the number of messages exchanged for processing 100 sliding windows.

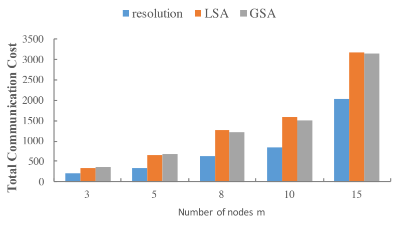
5.3 Evaluation



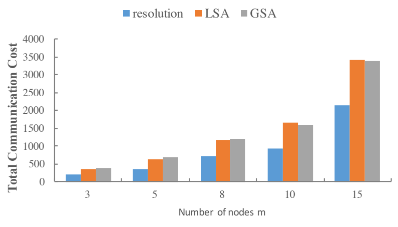
(a) w=5s



(b) w=10s

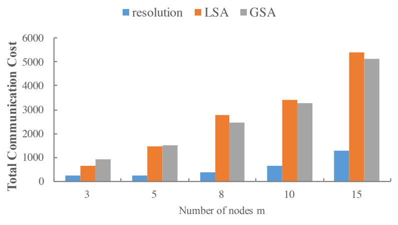


(c) w=15s

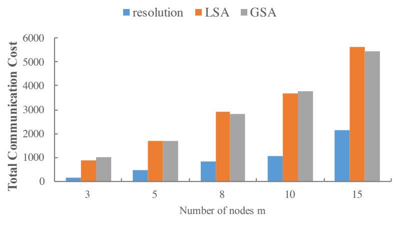


(d) w=20s

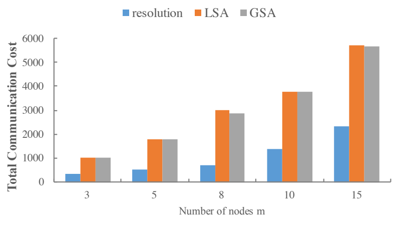
Fig. 3. Varying number of nodes m using synthetic dataset



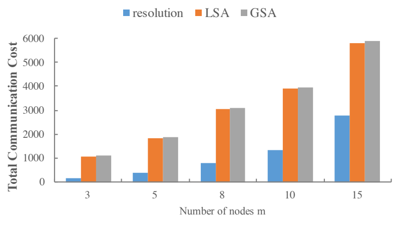
(a) w=5s



(b) w=10s



(c) w=15s



(d) w=20s

Fig. 4. Varying number of nodes m using real dataset

As shown in Fig. 3 and Fig. 4, we vary the number of monitoring nodes m with diverse window unit size w to demonstrate the efficiency and scalability of our resolution algorithm using synthetic dataset and real dataset.

Normally, as the number of monitoring nodes m increases, the overall communication cost of monitoring infrastructure is increased correspondingly. Because the global resolution phase in our resolution algorithm needs to request informations from all distributed nodes to resolve violations of local constraints. Our resolution algorithm outperforms baseline algorithms (LSA algorithm and GSA algorithm) in all cases from the figures. This is because our resolution algorithm retains additional slack at both coordinator node and distributed monitoring nodes and reduces vast communication cost by solving the violated constraints detected at monitoring nodes successfully.

Fig. 5 shows that the total communication cost of all algorithms decreases when the user-specified approximation parameter ϵ grows. With larger ϵ-error, there are less violations of local constraints at distributed nodes, resulting in lower communication overhead. However, the global top-k result is not accurate, and the error tolerance lies on the various application scenarios.

In all cases, our algorithm achieves a significant reduction in communication cost compared to baseline algorithms. Moreover, with the increase of monitoring nodes, the gap between our resolution algorithm and baseline algorithms becomes wider. These experiments results demonstrate the efficiency and scalability of our algorithm.

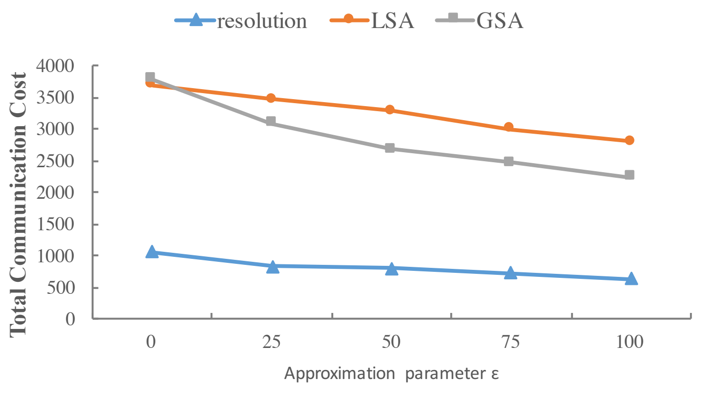


Fig. 5. Varying approximation parameter ϵ using real dataset, w=10s, m=10

6 Conclusions

In this paper, we studied the problem of top-k monitoring over distributed data streams in sliding window case. We propose a novel algorithm, which reallocates numeric values of data objects among distributed monitoring nodes by assigning revision factors to deal with distributed top-k monitoring problem and implement our algorithm on top of Apache Storm, on which extensive experiments are conducted to demonstrate the efficiency and scalability of our algorithm. Future work will concentrate on monitoring other functions over distributed data streams.

Acknowledgment. This work was supported in part by the National Basic Research 973 Program of China under Grant No. 2015CB352502, the National Natural Science Foundation of China under Grant Nos. 61272092 and 61572289, the Natural Science Foundation of Shandong Province of China under Grant Nos. ZR2012FZ004 and ZR2015FM002, the Science and Technology Development Program of Shandong Province of China under Grant No. 2014GGE27178, and the NSERC Discovery Grants.

References

1. Twitter storm. `http://storm.apache.org/`
2. Amagata, D., Hara, T., Nishio, S.: Sliding window top-k dominating query processing over distributed data streams. Distributed and Parallel Databases 34(4), 535–566 (2016), `http://dx.doi.org/10.1007/s10619-015-7187-9`

3. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003. pp. 28–39 (2003), `http://doi.acm.org/10.1145/872757.872764`
4. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002. pp. 369–380 (2002), `http://dx.doi.org/10.1109/ICDE.2002.994751`
5. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004. pp. 206–215 (2004), `http://doi.acm.org/10.1145/1011767.1011798`
6. Cormode, G.: The continuous distributed monitoring model. SIGMOD Record 42(1), 5–14 (2013), `http://doi.acm.org/10.1145/2481528.2481530`
7. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA (2001), `http://doi.acm.org/10.1145/375551.375567`
8. Giatrakos, N., Deligiannakis, A., Garofalakis, M.N., Sharfman, I., Schuster, A.: Prediction-based geometric monitoring over distributed data streams. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012. pp. 265–276 (2012), `http://doi.acm.org/10.1145/2213836.2213867`
9. Gupta, R., Ramamritham, K., Mohania, M.K.: Ratio threshold queries over distributed data sources. In: Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA. pp. 581–584 (2010), `http://dx.doi.org/10.1109/ICDE.2010.5447920`
10. Kashyap, S.R., Ramamirtham, J., Rastogi, R., Shukla, P.: Efficient constraint monitoring using adaptive thresholds. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México. pp. 526–535 (2008), `http://dx.doi.org/10.1109/ICDE.2008.4497461`
11. Keren, D., Sharfman, I., Schuster, A., Livne, A.: Shape sensitive geometric monitoring. IEEE Trans. Knowl. Data Eng. 24(8), 1520–1535 (2012), `http://dx.doi.org/10.1109/TKDE.2011.102`
12. Lazerson, A., Keren, D., Schuster, A.: Lightweight monitoring of distributed streams. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. pp. 1685–1694 (2016), `http://doi.acm.org/10.1145/2939672.2939820`
13. Lazerson, A., Sharfman, I., Keren, D., Schuster, A., Garofalakis, M.N., Samoladas, V.: Monitoring distributed streams using convex decompositions. PVLDB 8(5), 545–556 (2015), `http://www.vldb.org/pvldb/vol8/p545-lazerson.pdf`
14. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006. pp. 635–646 (2006), `http://doi.acm.org/10.1145/1142473.1142544`
15. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. 30(1), 41–82 (2005), `http://doi.acm.org/10.1145/1061318.1061320`
16. Sharfman, I., Schuster, A., Keren, D.: A geometric approach to monitoring threshold functions over distributed data streams. In: Proceedings of the ACM SIGMOD

International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006. pp. 301–312 (2006), http://doi.acm.org/10.1145/1142473.1142508

17. Yang, D., Shastri, A., Rundensteiner, E.A., Ward, M.O.: An optimal strategy for monitoring top-k queries in streaming windows. In: EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings. pp. 57–68 (2011), http://doi.acm.org/10.1145/1951365.1951375
18. Yi, K., Zhang, Q.: Optimal tracking of distributed heavy hitters and quantiles. In: Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA. pp. 167–174 (2009), http://doi.acm.org/10.1145/1559795.1559820
19. Zipf, G.K.: Selected studies of the principle of relative frequency in language. Language 9(1), 89–92 (1932)