Sliding Window Top-K Monitoring over Distributed Data Streams

Ben Chen² · Zhijin Lv² · Xiaohui Yu¹ · Yang Liu²

Received: 9 September 2017 / Revised: 22 October 2017 / Accepted: 31 October 2017 / Published online: 15 November 2017
© The Author(s) 2017. This article is an open access publication

Abstract Most of the traditional top- k algorithms are based on a single-server setting. They may be highly inefficient and/or cause huge communication overhead when applied to a distributed system environment. Therefore, the problem of top- k monitoring in distributed environments has been intensively investigated recently. This paper studies how to monitor 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 adaptively reallocates numeric values of data objects among distributed 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. We also develop a framework that combines a distributed data stream monitoring architecture with a sliding window model. Based on this framework, extensive experiments are conducted on top of Apache Storm to verify the efficiency and scalability of the proposed algorithm.

Keywords Top- K · Distributed monitoring · Data stream · Stream processing

1 Introduction

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 is to track the exceptional large (or small) numeric attribute values (or numeric values for short) relative to the majority of numeric values of data objects. For example, consider a system that monitors a large network for distributed denial of service (DDoS) attacks. The DDoS attacks may issue an unusual 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 potential 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, which incurs huge communication overhead.

Existing algorithms for distributed top- k query processing 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 algorithms, it is wasteful to execute the 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. So, these studies are

✉ Xiaohui Yu
xhyu@yorku.ca

Ben Chen
cbstubborn@163.com

Zhijin Lv
allen3jin@163.com

Yang Liu
yliu@sdu.edu.cn

¹ School of Information Technology, York University, Toronto, ON M3J 1P3, Canada

² Present Address: School of Computer Science and Technology, Shandong University, Jinan 250101, Shandong, China

unsuitable to continuously monitor the top- k result over 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 the provided top- k result remains valid. Their algorithms assume that a single node may violate constraints each time, which is unrealistic. Moreover, it is not suitable for the case of top- k results that are defined over sliding windows, which focuses on the impact of recent data objects.

Motivated by this, in this paper we are going to study on a new problem of distributed top- k monitoring, which is to continuously query 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.

For continuous data monitoring, we adopt a time-based sliding window model [2], where the data objects generated within W timestamps from the current timestamp are target for monitoring. In this paper, we consider a model in which there is one coordinator node C and a set of m distributed monitoring nodes N connected to the coordinator node as shown in Fig. 1. The top- k set on the coordinator node and monitoring nodes are represented by the global top- k and local top- k , respectively. The distributed architecture is more scalable than a single-server setup when processing massive data streams.

The coordinator node tracks the global top- k result and assigns constraints to each monitoring node, where the

Fig. 1 Sliding window distributed monitoring architecture

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 the coordinator node. Then, the coordinator node tries to resolve the violations, which is 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 is not always required.

1.1 Motivational Observations

For effective management of urban traffic, governments often install high-definition cameras at road intersections to capture motor vehicle traffic records. We can use these data to implement road traffic monitoring and provide solutions for urban traffic congestion and future urban construction.

As shown in Fig. 2, a valid traffic record contains some attributes: location ID, passage time, plate number, etc. Location ID represents the location where this record is captured. Our goal is to get top- k locations such that the number of traffic records in these locations is the highest in the past 15 min. In this case, the monitoring window size W is 15 min.

There is a real case; Tables 1 and 2 show the traffic records information of three locations acquired by two monitoring nodes during the period of 07:00:00–07:15:10. Table 3 shows the traffic records information of three locations acquired by the coordinator node in the same time region. Based on these real data, we have two important observations.

Observation 1 Local top- k does not necessarily change with the sliding of the monitoring window.

Location ID	→	3701126131
Passage Time	→	2016-10-29 08:00:01 456
Plate Number	→	鲁A123456
Direction	→	01
Brand	→	03
Color	→	03
Speed	→	80

Fig. 2 Sample of traffic record

Table 1 Monitoring node N_1 maintains the records information

Number \ Time	07:00:00-07:00:10	07:00:10-07:15:00	07:15:00-07:15:10
Location			
3701126131	12	1001	14
3701111002	10	920	9
3701022106	9	903	8

Table 2 Monitoring node N_2 maintains the records information

Number \ Time	07:00:00-07:00:10	07:00:10-07:15:00	07:15:00-07:15:10
Location			
3701126131	10	1015	14
3701111002	10	914	9
3701022106	8	915	11

By analyzing Table 1, we can see that when $k = 2$, the local top- k is $\{3701126131, 3701111002\}$ in the time slot of 07:00:00–07:15:00. As the monitoring window slides, we can get the same local top- k in the time slot of 07:00:10–07:15:10. In this case, although the monitoring window slides, there is no violation of the local constraint. Thus, the communication cost between the monitoring node and the coordinator node can be reduced.

Observation 2 Local top- k changes do not necessarily lead to global top- k changes.

By analyzing Tables 2 and 3, we can see that the local top- k on node N_2 changes from $\{3701126131, 3701111002\}$ to $\{3701126131, 3701022106\}$ from time slot 07:00:00–07:15:00 to slot 07:00:10–07:15:10, but the global top- k is still $\{3701126131, 3701111002\}$. In this case, the coordinator node can resolve the local violation of node N_2 by *partial resolve* and keep the global top- k still valid. (It is worth noting that the coordinator node does not maintain the object value in the actual case, here we make this assumption for convenience.)

Table 3 Coordinator node maintains the records information

Number \ Time	07:00:00-07:00:10	07:00:10-07:15:00	07:15:00-07:15:10
Location			
3701126131	22	2016	28
3701111002	20	1834	18
3701022106	17	1818	19

Based on the above two observations, we can effectively reduce the communication cost between the distributed nodes to maintain a valid global top- k when monitoring window slides.

1.2 Challenges and Contributions

Intuitively, distributed top- k monitoring is just about monitoring the data streams and then querying the highest value of the k data objects. However, in a distributed environment, the situation is much more complicated.

- *Data streams varying independently* Data streams vary independently on different monitoring nodes, which causes the difference between the local top- k of different monitoring nodes. Thus, tracking the top- k data objects with the largest aggregate numeric values from monitoring nodes will result in huge communication overhead, because the global top- k result is affected by local changes of data objects at monitoring nodes.
- *Data consistency and efficiency* In order to obtain a valid global top- k result, the local top- k at monitoring nodes must be in line with it. Meanwhile, it is imperative to find solutions that can effectively monitor the global top- k result, while minimizing the communication cost across the network.

For a more practical solution, we construct a framework that combines a distributed data stream monitoring architecture which is shown in Fig. 1 and as described earlier with a time-based sliding window model. When the window slides, the time-based sliding window model continues to provide the latest data streams for monitoring nodes.

In order to keep the local top- k at monitoring nodes consistent with the global top- k , objects monitored by each monitoring node must satisfy the local constraints. We propose a novel algorithm to resolve the violations of local constraints. It has three phases. Whenever local constraints are violated, it will initialize the first phase, namely *local alert phase*. In this phase, the monitoring nodes which violate local constraints send messages to the coordinator node. Then, it comes into the second phase, also known as *partial resolution phase*. In this phase, the coordinator node tries to resolve violations. If it failed, it will go to the third phase, called *global resolution phase*, which ensures all violations are resolved, but the cost is much higher in this phase.

We implement our distributed top- k algorithm on top of Apache Storm [20], an open-source distributed stream processing platform, on which we conduct extensive experiments to evaluate the performance of our solutions.

In summary, we make the following contributions:

- 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 it.
- We propose a novel algorithm for top- k monitoring over distributed data streams, which achieves a significant reduction in communication cost.
- We implement our algorithm on top of Apache Storm and conduct extensive experiments to evaluate the performance of our algorithm with synthetic data and real data, which demonstrates the efficiency and scalability of our algorithm.

The rest of this paper is organized as follows. Section 2 provides a brief review of related work. Section 3 formally defines the top- k monitoring problem studied in this paper. We describe our top- k monitoring algorithm in detail in Sect. 4. Section 5 experimentally evaluates the performance of our algorithms, and Sect. 6 is the conclusion of this paper.

2 Related Work

Previous work on monitoring distributed data streams can be classified into two categories. One category is monitoring functions over the union of distributed data streams, and the other one 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 [24], and ratio queries [9]. Sharfman et al. [19] present a geometric monitoring (GM) approach for efficiently tracking the value of a general function over distributed data relative to a given threshold. Follow-up work [8, 11, 14] proposed various extensions to the basic method. Recently, Lazerson et al. [13] proposed a CB approach, which is superior to GM in reducing computational complexity, by several orders of magnitude in some cases. Cormode [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. Vlachou et al. [21] presented a SPEERTO approach, which utilizes a threshold-based super-peer selection mechanism based on the skyline points of each super-peer and returns the exact results progressively to the user. The algorithm proposed in [25] focuses on processing the reverse top- k query and solves by a geometric framework. In [1, 22, 23], top- k monitoring is performed with a sliding window model. Zhu et al. [27] introduce SAP, a self-adaptive partition

framework for supporting continuous top- k query over stream data.

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. Koudas et al. [12] introduce DISC, a technique for continuous monitoring approximate k -NN queries over multi-dimensional data streams. It can be applied to different data distributions to either optimize memory utilization or achieve the best accuracy. By indexing objects or queries, Yu et al. [26] proposed two grid-based algorithms to resolve the problem of continuously monitoring multiple k -NN queries over moving objects. Mouratidis et al. [17] 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 [18]. In [23], the candidate set is minimized and its computation time becomes faster than that of [17]. 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. [1] presented algorithms for distributed continuous top- k dominating query processing, which reduces both communication and computation costs. However, their algorithms are inappropriate for the top- k monitoring problem we studied, because their ranking functions are based on the dominance relationship of data objects rather than the aggregate numeric values from distributed data streams.

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 and Wang [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. Based on this work, Michel et al. [16] presented a novel algorithmic framework, called KLEE, which makes a strong case for approximate top- k algorithms over widely distributed data sources. It extends the TPUT algorithm and affords the query-initiating peer the flexibility to trade-off result quality and expected performance and to trade-off the number of communication phases engaged during query execution versus network bandwidth performance. Unfortunately, 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 has changed after an initial answer has been obtained.

A preliminary version of this work appeared as a conference paper [15]. This paper is an extension of the conference version, with the following new contributions:

- We modify the structure of introduction section to make it more reasonable and easier to understand. More importantly, we add an actual example to illustrate the issues we are studying, the motivations of research, and the challenges to help readers understand the paper.
- We conduct an additional experiment about the effect of the top- k set size on experimental results to evaluate the efficiency and scalability of the proposed method.
- We add a brief description of the two baseline algorithms involved in the paper and compare the differences between them and the method used in the paper.
- We rewrite many parts of the paper based on the valuable comments from the anonymous reviewers, which greatly improves the presentation as well as the content of the paper.

3 Problem Definitions

Formally, we define the problem studied in the paper. As described above, there is one coordinator node $\mathcal{C}$ and m distributed monitoring nodes $N_1, N_2, \dots, N_m$. Each monitoring node N_j continuously receives data records from a data stream. Collectively, the monitoring nodes track a set $\mathcal{O}$ of n logical data objects $\mathcal{O} = \{O_1, O_2, \dots, O_n\}$. Each data object is associated with a numeric value within the current monitoring window. For example, in the case of traffic field which we mentioned in Sect. 1.1, the object is location ID, and its associated numeric value is the number of traffic records in a period of time. The numeric value of each data object is updated as the monitoring window slides. For each monitoring node N_j , we define partial numeric values $C_{1,j}(t), C_{2,j}(t), \dots, C_{n,j}(t)$ representing node N_j 's view of the data stream DS_j within monitoring window W at time t , where

$$C_{i,j}(t) = |\{O'_i \in DS_j \mid t - t' \leq W\}|. \quad (1)$$

So the aggregate numeric value of each object O_i from distributed monitoring nodes is defined as

$$C_i(t) = \sum_{1 \leq j \leq m} C_{i,j}(t). \quad (2)$$

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:

$$\forall O_a \in \mathcal{T}, \forall O_b \in \mathcal{O} - \mathcal{T} : C_a(t) + \epsilon \geq C_b(t) \quad (3)$$

where $\epsilon \geq 0$ is a 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

We realize that the global top- k set is valid, if distributed monitoring nodes have the same top- k set locally. However, as we mentioned in Sect. 1.2, the numeric values of data objects vary independently at monitoring nodes. And in practical environments, the numeric values of the same object in different nodes may vary greatly. This makes the actual local top- k set at monitoring nodes may be differ from the global top- k set at a given time. Thus, we use *revision factors*, labeled $\delta_{i,j}$, to reallocate the numeric values of data object O_i to the monitoring node N_j to satisfy the following local constraint:

$$\forall O_a \in \mathcal{T}, \forall O_b \in \mathcal{O} - \mathcal{T} : C_{a,j}(t) + \delta_{a,j} \geq C_{b,j}(t) + \delta_{b,j} \quad (4)$$

If all data objects on each monitoring node meet the local constraint, local top- k of each monitoring node is consistent with the global top- k .

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:

$$\sum_{0 \leq j \leq m} \delta_{i,j} = 0. \quad (5)$$

Fig. 3 Sliding window unit structure

3.2 Slacks

In order to reallocate numeric values of data objects among distributed 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 objects that violate local constraints 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 :

$$\mathcal{P}_j = \max_{O_i \in \mathcal{O} - \mathcal{R}} (C_{i,j}(t) + \delta_{i,j}) \quad (6)$$

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

$$\forall O_i \in \mathcal{R} : \mathcal{S}_i = \sum_{1 \leq j \leq m} (C_{i,j}(t) - \mathcal{P}_j) = C_i(t) - \sum_{1 \leq j \leq m} \mathcal{P}_j \quad (7)$$

Use $\mathcal{S}_i$ to calculate the revision factors of data object O_i on each monitoring node and the coordinator node will discuss later in Sect. 4.1

3.3 Sliding Window Unit

The sliding window model [2] is a commonly used data stream model, which can improve the processing efficiency of the data stream by logical abstraction of the data stream. 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 units $s_0, s_1, \dots, s_{l-1}$ ($l = \frac{W}{w}$). 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 it will lead to higher communication and computation costs.

As shown in Fig. 3, 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 update within the new monitoring window W' ($W' = W + s_{\text{new}} - s_{\text{exp}}$). Obviously, changes in data objects may violate the current local constraints.

4 Top-K Monitoring Algorithm

In this section, we 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 $\mathcal{C}$ will send a message containing $\mathcal{T}$ and initial revision factors $\delta_{i,j} = 0$ corresponding to each monitoring node N_j . Upon receiving this message, monitoring node N_j will create local constraint (4) from $\mathcal{T}$ and revision factors. With the sliding of the monitoring window, monitoring nodes detect potential violations based on local constraint and revision factors.

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 violation of local constraints occurs at monitoring nodes. Our resolution algorithm consists of three phases, while the third phase is not always required. N_V represents all monitoring nodes that violate local constraints.

Algorithm 1 Partial Resolution Algorithm

Input: $\mathcal{T}, \delta_{i,j}, \{\mathcal{R}_j\}, \{C_{i,j}(t)\}$
Output: succeed or failed

```

1: for  $\forall N_j \in N_V$  do
2:    $bound_j \leftarrow \max(C_{i,j}(t) + \delta_{i,j}), O_i \in \mathcal{R}_j - \mathcal{T}$ 
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  $\forall N_j \in N_V$  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
   (4) */
11: for  $\forall O_i \in \mathcal{T}$  do
12:   if  $slack_i + \varepsilon < bound_0$  then
13:     return failed
14: return succeed

```

- **Local Alert Phase (LAP)** 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 (PRP)** The coordinator node determines whether all violations can be solved based on the messages from violated nodes N_V 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 (GRP)** The coordinator node requests the current partial values $C_{i,j}(t)$ of data objects O_i in overall resolution set $\mathcal{R} = \cup_{N_j \in N_V} \mathcal{R}_j$ 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 global top- k set $\mathcal{T}'$ and running Algorithm 2 to compute new revision factors of data objects in the resolution set $\mathcal{R}$, and then notifies all monitoring nodes of the new top- k set 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 the coordinator node.

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 , $C_{i,\mathcal{A}}(t) = \sum_{0 \leq j \leq m} (C_{i,j}(t) + \delta_{i,j})$. Similarly, we define the sum of the baseline values from the nodes set $\mathcal{A}$, $\mathcal{P}_{\mathcal{A}} = \sum_{0 \leq j \leq m} \mathcal{P}_j$.

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:    $ps \leftarrow \mathcal{S}_i / (1 + m)$ 
9:   for  $j = 0 \rightarrow m$  do
10:    if  $O_i \in \mathcal{T}'$  and  $j = 0$  then
11:       $\delta_{i,j} = \mathcal{P}_j - C_{i,j}(t) + ps - \epsilon$ 
12:    else
13:       $\delta_{i,j} = \mathcal{P}_j - C_{i,j}(t) + ps$ 
  
```

4.2 Correctness 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 algorithm finds that partial resolution phases are failed to resolve the violations, due to slack at coordinator node is zero. The global resolution phase computes the new revision factors assigned to monitoring nodes, at coordinator node $\delta_{2,0} = 1, \delta_{3,0} = 2$ 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 still valid.

Data objects not in the resolution set $\mathcal{R}$ cannot 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:

$$C_i(t) \geq C_l(t) \geq \mathcal{P}_{\mathcal{A}}. \quad (8)$$

And, the overall slacks of data objects in resolution set $\mathcal{R}$ satisfy the following in equation:

$$\forall O_a \in \mathcal{T}', \forall O_b \in \mathcal{R} - \mathcal{T}' : \mathcal{S}_a \geq \mathcal{S}_b \quad \text{and} \quad \mathcal{S}_a \geq 0 \quad (9)$$

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 (4) at distributed nodes are satisfied.

4.3 Cost Analysis

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 is not required. Thus, the communication cost at this phase is just assigning updated revision factors to the violated node, and the number of $2 * |N_V|$ messages are exchanged altogether. If all three phases are required, the total of $|N_V| + 3m$ messages are necessary to perform complete resolution.

It is important to reallocate the overall slack of data objects among the coordinator node and monitoring nodes. If the global slack retained at the coordinator node is tight, the probability of failure at partial resolution phase becomes high. That means it will increase the execution

probability of the global resolution phase to resolve violations. According to the cost analysis above, the algorithm has the largest communication cost in the third phase, so it will greatly increase the global communication cost. However, tight slacks at monitoring nodes result in frequent violations of local constraints. Our even policy for allocating additional slacks balances these two costs well.

4.4 Complexity Analysis

We now analyze the time complexity of Algorithms 1 and 2. With the sliding of the monitor window, we assume that n and a represent the number of monitoring nodes and all data objects that violate local constraints, respectively.

Lemma 1 *The time complexity of the Algorithm 1 is $O(nk + a)$, where k is the top- k set size.*

Proof The first for loop is used to find the boundary value for each monitoring nodes that violate local constraints, its time complexity is a . The time complexity of executing lines 4–8 is kn , and the algorithm exits the last for loop after executing lines 11–13 by k times. Thus, the time complexity of the Algorithm 1 is $O(k + nk + a) = O(n + a)$. $\square$

Lemma 2 *The time complexity of the Algorithm 2 is $O(a)$.*

Proof As described earlier, resolution set $\mathcal{R}$ contains data objects from global top- k set $\mathcal{T}$ and all data objects that violate local constraints, so its size is equal to $k + a$, and the algorithm exits the inner for loop after executing lines 9–13 by m times, m is the number of all monitoring nodes. Therefore, the time complexity of the Algorithm 2 is $O((k + a)m) = O(a)$. $\square$

5 Performance Evaluation

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 algorithm. We call them LSA and GSA, respectively. The differences between these two baseline

algorithms and the resolution algorithm used in this paper can be described as follows.

- **LSA** This algorithm retains zero slack at the coordinator node. In this case, all slack will be assigned to monitoring nodes. Using this algorithm, the phase of PRP will be invalid, and once there is a violation, it will directly come into the phase of GRP.
- **GSA** This algorithm retains all slack at the coordinator node. It helps to resolve violations in the phase of PRP, but it will increase the probability of constraint violation on monitoring nodes.
- **Resolution** This algorithm retains slack at both the coordinator node and monitoring nodes. It takes full account of the advantages and disadvantages of the former two to reduce communication cost effectively.

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 monitoring nodes. The monitoring node can exchange messages with the coordinator node, but cannot 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 real-time 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 size of top- k set k , number of monitoring nodes m , approximation parameter ε , and sliding window unit size w , respectively. The default values of the parameters are listed in Table 4. Parameters are varied as follows:

- Number of objects in top- k set k : 10, 20, 30, 40, 50
- Number of monitoring nodes m : 3, 5, 8, 10, 15
- Approximation parameter ε : 0, 25, 50, 75, 100
- Sliding window unit size w : 5, 10, 15, 20 s

5.2 Datasets

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

Table 4 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 (15 min)
w	Sliding window unit size (10 s)

Fig. 4 Varying number of nodes m using synthetic dataset

- **Synthetic Dataset** The synthetic dataset includes random data records, which follow Zipf distribution [28]. The distribution parameter we used is 2. Each data record contains ID of data object and the time of generation. The goal of experiments conducted on this dataset is to continuous query the top- k data objects with the largest number of occurrences. Since the synthetic dataset is generated by a random function, the data generated each time is inconsistent. In order to ensure the accuracy of experimental results, the final cost of experiment is calculated by taking the average value of 10 experimental results.
- **Real Dataset** The real dataset consists of a portion of real vehicle passage records from the traffic surveillance system of a major city. Each passage record is as described in Sect. 1.1. The dataset contains 5,762,391 passage records, which are generated in 6 h, and it involves about 1000 detecting locations on the main roads. The goal of experiments conducted on this dataset is continuous monitor the top- k 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 min,

and the total communication cost is the number of messages exchanged for processing 100 sliding windows.

5.3 Results for Synthetic Dataset

As shown in 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 by using synthetic dataset.

Normally, as the number of monitoring nodes m increases, the overall communication cost of monitoring infrastructure is increased correspondingly. For one thing, due to the increase in the number of monitoring nodes, the data objects and their numeric values will be distributed more widely. So the numeric values of data objects will change more dramatically when the monitoring window slides, thereby increasing the probability of violating local constraints; for another, the global resolution phase in our algorithm needs to request information from all monitoring nodes to resolve violations of local constraints.

Besides, our resolution algorithm outperforms baseline algorithms (LSA algorithm and GSA algorithm) in all cases from the figure. This is because our resolution algorithm retains additional slack at both the coordinator

Fig. 5 Varying number of nodes m using real dataset

Fig. 6 Varying approximation parameter ϵ using real dataset, $w = 10s$, $m = 10$

node and monitoring nodes and reduces vast communication cost by solving the violated constraints detected at monitoring nodes successfully.

5.4 Results for Real Dataset

Figure 5 shows the communication cost as the number of monitoring nodes is varied when the size of sliding window unit w is 5, 10, 15 and 20 s. We can get the same results as using synthetic dataset. Besides, on the whole,

Fig. 7 Varying size of top- k set k using real dataset, $w = 10s$, $m = 10$

the communication cost of using real dataset is higher than using synthetic dataset under the same conditions. There are two reasons. First of all, as the monitoring window slides, the numeric values of data objects may vary greatly between the window units s_{exp} and s_{new} . For example, supposing a monitoring node N_1 current local top- k set contains an object O_1 , and its numeric value is 60 in the window unit s_{exp} . As time goes by, a new window unit s_{new} is added, and the numeric value of this object is 10 in this new window unit. In this case, O_1 may no longer belong to the local top- k set of monitoring node N_1 , and this situation

will result in a violation of local constraints. So the communication cost will be increased. The probability of such case is much greater in the real dataset than in the synthetic dataset. Then, unlike synthetic dataset, two adjacent data objects sorted by numeric values are very close in numeric terms in the real dataset. As we mentioned in Sect. 1.1, the global value of data objects 3701111002 and 3701022106 are 1834 and 1818, which results in a different local top- k on the monitoring node N_1 and the monitoring node N_2 in the time period of 07:00:10–07:15:10. Likewise, this case also increases the communication cost.

As shown in Fig. 6, 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 monitoring nodes, which results in lower communication overhead. However, the global top- k result is not accurate, and the error tolerance lies on the various application scenarios.

Figure 7 shows the communication cost as the size of top- k set is varied when the window unit and the number of monitoring nodes are each 10. From the experimental result, we can see that our resolution algorithm performs better than the two baseline algorithms. Besides, the change in the value of k does not significantly affect the communication cost. The cost is calculated by counting the number of messages exchanged between monitoring nodes and the coordinator node. It is worth noting that more resources will be required as the value of k increases, such as CPU, server memory, network bandwidth between monitoring nodes and the coordinator node, etc. But it is not the focus of this work.

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

6 Conclusions

In this paper, we have studied the problem of top- k monitoring over distributed data streams in sliding window case. Based on two motivational observations, we have proposed 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. We also have developed a hybrid framework and conducted our algorithm on top of Apache Storm. Furthermore, we have implemented two baseline algorithms and used two kinds of datasets to demonstrate the efficiency and scalability of our algorithm. Future work will

concentrate on monitoring other functions over distributed data streams.

Acknowledgements 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.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Amagata D, Hara T, Nishio S (2016) Sliding window top- k dominating query processing over distributed data streams. *Distrib Parallel Databases* 34(4):535–566
2. Babcock B, Datar M, Motwani R (2002) Sampling from a moving window over streaming data. In: *Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms*, San Francisco, 6–8 January 2002, pp 633–634
3. Babcock B, Olston C (2003) Distributed top- k monitoring. In: *Proceedings of the 2003 ACM SIGMOD international conference on management of data*, San Diego, 9–12 June 2003, pp 28–39
4. Bruno N, Gravano L, Marian A (2002) Evaluating top- k queries over web-accessible databases. In: *Proceedings of the 18th international conference on data engineering*, San Jose, 26 February–1 March 2002, pp 369–380
5. Cao P, Wang Z (2004) 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, 25–28 July 2004, pp 206–215
6. Cormode G (2013) The continuous distributed monitoring model. *SIGMOD Rec* 42(1):5–14
7. Fagin R, Lotem A, Naor M (2001) Optimal aggregation algorithms for middleware. In: *Proceedings of the twentieth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems*, Santa Barbara, 1–23 May 2001
8. Giatrakos N, Deligiannakis A, Garofalakis MN, Sharfman I, Schuster A (2012) Prediction-based geometric monitoring over distributed data streams. In: *Proceedings of the ACM SIGMOD international conference on management of data*, SIGMOD 2012, Scottsdale, 20–24 May 2012, pp 265–276
9. Gupta R, Ramamritham K, Mohania MK (2010) Ratio threshold queries over distributed data sources. In: *Proceedings of the 26th international conference on data engineering*, ICDE 2010, Long Beach, 1–6 March 2010, pp 581–584
10. Kashyap SR, Ramamritham J, Rastogi R, Shukla P (2008) Efficient constraint monitoring using adaptive thresholds. In: *Proceedings of the 24th international conference on data engineering*, ICDE 2008, Cancún, 7–12 April 2008, pp 526–535
11. Keren D, Sharfman I, Schuster A, Livne A (2012) Shape sensitive geometric monitoring. *IEEE Trans Knowl Data Eng* 24(8):1520–1535

12. Koudas N, Ooi BC, Tan K, Zhang R (2004) Approximate NN queries on streams with guaranteed error/performance bounds. In: (e)Proceedings of the thirtieth international conference on very large data bases, Toronto, 31 August–3 September 3 2004, pp 804–815
13. Lazerson A, Keren D, Schuster A (2016) Lightweight monitoring of distributed streams. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, 13–17 August 2016, pp 1685–1694
14. Lazerson A, Sharfman I, Keren D, Schuster A, Garofalakis MN, Samoladas V (2015) Monitoring distributed streams using convex decompositions. *PVLDB* 8(5):545–556
15. Lv Z, Chen B, Yu X (2017) Sliding window top-k monitoring over distributed data streams. In: Web and big data—first international joint conference, APWeb-WAIM 2017, Beijing, 7–9 July 2017, proceedings, part I, pp 527–540,
16. Michel S, Triantafillou P, Weikum G (2005) KLEE: a framework for distributed top-k query algorithms. In: Proceedings of the 31st international conference on very large data bases, Trondheim, 30 August–2 September 2005, pp 637–648
17. Mouratidis K, Bakiras S, Papadias D (2006) Continuous monitoring of top-k queries over sliding windows. In: Proceedings of the ACM SIGMOD international conference on management of data, Chicago, 27–29 June 2006, pp 635–646
18. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. *ACM Trans Database Syst* 30(1):41–82
19. Sharfman I, Schuster A, Keren D (2006) A geometric approach to monitoring threshold functions over distributed data streams. In: Proceedings of the ACM SIGMOD international conference on management of data, Chicago, 27–29 June 2006, pp 301–312
20. Apache Storm. <http://storm.apache.org/>. Accessed Feb 2017
21. Vlachou A, Doukeridis C, Nørnvåg K, Vazirgiannis M (2008) On efficient top-k query processing in highly distributed environments. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2008, Vancouver, 10–12 June 2008, pp 753–764
22. Wang X, Zhang Y, Zhang W, Lin X, Huang Z (2016) SKYPE: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB* 9(7):588–599
23. Yang D, Shastri A, Rundensteiner EA, Ward MO (2011) An optimal strategy for monitoring top-k queries in streaming windows. In: EDBT 2011, 14th international conference on extending database technology, Uppsala, 21–24 March 2011, proceedings, pp 57–68
24. Yi K, Zhang Q (2009) Optimal tracking of distributed heavy hitters and quantiles. In: Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, PODS 2009, Providence, 19 June–1 July 2009, pp 167–174
25. Yu A, Agarwal PK, Yang J (2012) Processing a large number of continuous preference top-k queries. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2012, Scottsdale, 20–24 May 2012, pp 397–408
26. Yu X, Pu KQ, Koudas N (2005) Monitoring k-nearest neighbor queries over moving objects. In: Proceedings of the 21st international conference on data engineering, ICDE 2005, Tokyo, 5–8 April 2005, pp 631–642
27. Zhu R, Wang B, Yang X, Zheng B, Wang G (2017) SAP: improving continuous top-k queries over streaming data. *IEEE Trans Knowl Data Eng* 29(6):1310–1328
28. Zipf GK (1932) Selected studies of the principle of relative frequency in language. *Language* 9(1):89–92