Distributed Data Streams

Jannik Castenow, Björn Feldkord, Jonas Hanselle, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide(✉)

Paderborn University, Paderborn, Germany
{jannik.castenow,bjoern.feldkord,jonas.hanselle,
manuel.malatyali,fmadh}@upb.de, tillk@mail.upb.de

Abstract. We consider a scenario where a server is wirelessly connected to countless sensor nodes that continuously measure data. The task of the server is to monitor the sensors' data. More precisely, at each time step the server calculates a function defined over the current measurements of the sensors. Since the sensors only have small computational power and tight battery constraints, the communication between the server and the sensors should be as small as possible, i.e., we aim at minimizing the total number of messages that is transferred.

There are various conceivable problems for the setting above. We demonstrate our approaches on the following three: In the Top-k-Value Monitoring Problem, the server aims at identifying the k largest values. The Top-k-Position Monitoring Problem shifts the task to identify the sensors observing these values. Finally, the Count Distinct Monitoring Problem obliges the server to determine the number of distinct values currently observed.

For all three problems, we not only present communication-efficient protocols for one time step, we also show how it can be exploited if the input at sensors is similar between consecutive time steps to reduce the total communication on the long term. Thereby, we utilize different techniques – involving sampling, dynamic data structures, filter-based approaches, and combinations of them – to demonstrate their potential and their limits in the broad setting described above.

Keywords: Top-k · Count distinct · Distributed monitoring · Distributed data streams

1 Introduction

Envision a scenario where a set of tiny, lightweight sensors is distributed in a hazardous area (e.g., an ocean, high mountains or in space) to monitor the environment. The sensors are connected to one or multiple central servers which have the task to track the measurements of the sensors, i.e., the servers have to compute a function of the sensor values at every point in time. This task is easy to solve as long as the sensors continuously send their current measurements to the servers and the latter ones have enough memory and computational power to do computations on the sensor data at every point in time. Realistic applications, however, require a huge number of sensors (e.g., because the area is very large,

H. Bast et al. (Eds.): Algorithms for Big Data, LNCS 13201, pp. 179–195, 2022.
https://doi.org/10.1007/978-3-031-21534-6_10

sensors are error-prone, sensors have only a limited battery lifetime, ...) that cannot be handled by modern server hardware, or the number of required severs might be uneconomically expensive. Additionally, sending the measured data of the sensors continuously to the servers leads to a rapid decrease of the sensors' batteries. Therefore, to build a feasible system, the communication between sensors and servers needs to be severely reduced.

We consider two types of randomized algorithmic approaches to reduce the communication: The first approach is based on Monte Carlo algorithms. Sensors decide randomly to communicate their current observed value to the server. The probability of sending a message depends on the significance of the current observed value: If the impact on the output function is small, the probability of sending a message is low; if the impact is high also the probability of sending a message is high. Thus, the server is not aware of all changes in the sensor values but with a high probability it gets to know all significant changes. With this approach, the server is able to compute a correct output with a high probability. In some scenarios (for instance in safety-critical systems), the application demands to always compute a correct output. Here, we exploit the idea of Las Vegas algorithms that reduce the number of sent messages with a high probability but always compute the exact output. With a low probability many messages may be sent, however the server can always be sure to compute the correct output. All in all, these two approaches build a trade-off between reducing the communication and computing correct outputs, while the randomization helps to keep the trade-off small.

Considering the scenario described above, we are interested in multiple problems. In the Top-k-Value Monitoring problem, the server is interested in the k largest values observed by the sensors at any time. In contrast to that, the Top-k-Position Monitoring problem tackles the case where the server is interested in the actual sensors measuring the k largest values, e.g., to track if large values and the set of sensors observing them are correlated. As in a lot of cases a rough estimate on the top k positions is sufficient, we also address the Approximate Top-k-Position Monitoring problem. Besides the largest values, the server might also be interested in how many different values are observed to get an overview on the global situation. This is captured in the (Approximate) Count Distinct Monitoring problem.

The aforementioned problems have in common that in practice a lot of communication can be avoided compared to a naïve approach that gathers all sensor data at every time step at the server. For example, consider the (Approximate) Count Distinct Monitoring problem. If a subset of the sensors observes identical values, not all of them need to communicate their observation to the server. See Fig. 1 for a depiction. Here, a horizontal block indicates a set of sensors observing the same value at the same time. Optimally, only one of them would need to communicate the value to the server. Additionally, if the value that is observed by a fixed sensor does not change significantly as time goes, the sensor does not need to notify the server all the time. Furthermore, observations of sensors that are not of interest should not be communicated. This can be seen when considering the Top-k-Value Monitoring problem. It would be best if all sensors not

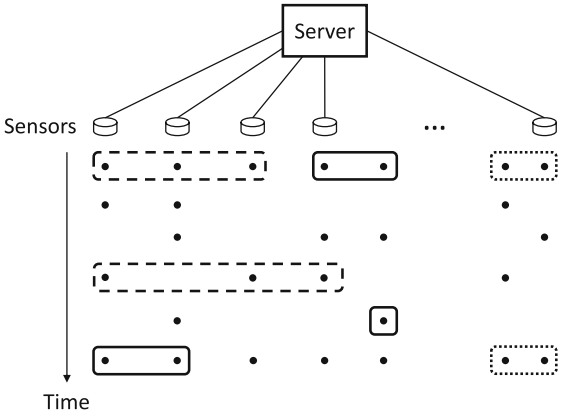

Fig. 1. We consider a central server that is connected to a set of sensor nodes. As time goes, each sensor observes a sequence of values (indicated by the dots below the sensors). Among others, communication can be avoided if a group of sensors observes the same value at the same time (horizontal blocks).

observing one of the k largest values do not communicate at all. Note, that for this it is required that all the sensors can receive information from the server. In our model, we allow the server to have a cheap broadcast channel. This can be assumed, as the central server has no need to reduce its power consumption as opposed to the sensors.

In this paper, we examine how communication can be minimized in the problems above. Our focus is especially a theoretical analysis of techniques that allow to capture the idea that sensor data might not change arbitrarily between consecutive time steps. We examine, among other things, how to use dynamic data structures and restrictions on the adversary dictating the inputs at sensor nodes, such that an algorithm can keep/update an existing solution for more than one time step and reduce the overall communication.

We begin in Sect. 2 by a formal introduction of our model and the problems we consider. We also introduce a major technique that we use called *filters*. Afterwards, we establish computational primitives in Sect. 2.3. In Sect. 3 we deal with the Top-k-Value Monitoring problem followed by the Top-k-Position Monitoring problem in Sect. 4 and the (Approximate) Count Distinct Monitoring problem in Sect. 5.

This paper surveys results from [1 SPP, 4 SPP, 8 SPP, 9 SPP, 10 SPP] . We only give short sketches of algorithms and proofs. For technical details, please look at the papers above. A detailed description of the current state of the art is presented in [10 SPP].

2 Model

In our setting there are n nodes connected to a single server. The nodes are uniquely identified by IDs from the set $\{1, \ldots, n\}$ and each node i receives

$(v_i^1, v_i^2, v_i^3 \ldots)$ as a stream of data. At time t, a node i observes $v_i^t \in \mathbb{N}$ and does not know any $v_i^{t'}$, $t' > t$. The superscript t is omitted if it is clear from the context.

Following the model in [3], we allow that between any two consecutive time steps, a *communication protocol* exchanges messages between the server and the nodes. The communication protocol is allowed to use a number of rounds poly-logarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$. Nodes can only send messages to the server and they are able to store a constant number of integers, compare two integers and perform Bernoulli trials with success probability $2^i/n$ for $i \in \{0, \ldots, \log n\}$. The server can communicate to one node directly or utilize a broadcast-channel to send one message to all nodes simultaneously. All communication methods described above incur unit communication cost per message, the delivery is instantaneous, and we allow a message at time t to have a size which is logarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$.

A time step t defines a point in time at which the sensor nodes obtain a new piece of input (v_i^t for node i at time t). The protocol consists of multiple (communication) rounds: Each sensor node performs local computations and may send a message to the server. The server collects all messages, performs local computations and may send a message via the broadcast-channel to all sensor nodes.

Since all nodes are synchronized, the server can detect if no sensor sends a message and the sensor nodes can identify if the server did not send a message. Furthermore, the server has unrestricted capacity when receiving, i.e., it can always receive all messages that are send to it.

At the end of each round, when the communication protocol terminated, the server decides on the output of the function for the current time t and the whole network proceeds to the next time step $t + 1$.

We assume that all observed values are pairwise different for the (Approximated) Top-k-Value and Top-k-Position Monitoring Problems and coupe with a large number of duplicates considering the (Approximate) Count Distinct Problem.

2.1 Problems

Our focus here is on three problems; the Top-k-Value Monitoring problem, the Top-k-Position Monitoring problem and the Count Distinct Monitoring problem. In the Top-k-Value Monitoring problem, we are interested in the largest observed values, i.e., the ordering of the values is of special interest. Let $s_1^t, \ldots, s_n^t$ be the values observed at time t ($v_1^t, \ldots, v_n^t$) sorted in descending order.

Definition 1 (Top-k-Value Monitoring). *In the Top-k-Value Monitoring problem, the server has to output $s_1^t, \ldots, s_k^t$, $k \leq n$ at each time t.*

In contrast to keeping track of the values it might be more of interest to keep track of the nodes observing the largest values instead (for instance in safety critical applications). This is considered in the Top-k-Position Monitoring problem.

Definition 2 ((Approximate) Top-k-Position Monitoring). *In the Top-k-Position Monitoring problem, the server has to output at each time t the k nodes observing $s_1^t, \ldots, s_k^t$ – called the top-k. If we are interested in an approximation, we need some more notation. For any constant $\varepsilon \in (0,1)$ let $E(t) := \{i \mid v_i^t \in ((1-\varepsilon)^{-1} s_k^t, \infty)\}$ be the set of nodes observing values which are significantly larger than the kth largest one. In the Approximate Top-k-Position Monitoring problem, at each time t the server has to output $E(t)$ and $k - |E(t)|$ many nodes not in $E(t)$ observing a value which is at least $(1-\varepsilon)\, s_k^t$.*

In the case that multiple nodes observe the same value, one might be more interested in how many different values are observed. We approach this direction by the Count Distinct Monitoring problem. Note, that we do not assume all values to be distinct when discussing this problem.

Definition 3 ((Approximate) Count Distinct Monitoring). *For a fixed time step t, let d_t be the number of distinct values observed by all nodes, i.e., $d_t = |\{v_i^t \mid i \in \{1, \ldots, n\}\}|$. At each time step t the server has to output d_t. In the approximation variant, the server has to output an (ε, δ)-approximation at each time step t, i.e.; for two constants $0 \leq \epsilon, \delta \leq 1$, the server has to compute a value $x \in [(1-\varepsilon) \cdot d_t, (1+\varepsilon) \cdot d_t]$ with probability at least $1 - \delta$.*

Explicitly, the values at times $t' < t$ do not matter for the output at time t.

2.2 Filter-Based Algorithms

One of our main techniques is the usage of *filters*. A filter defines for each sensor an interval of values that do not influence the output function. Filtering the input for an algorithm occurs in many different contexts. In algorithm engineering, filtering has turned out to be a valuable tool to decrease the input size to speed up the computation in certain cases. For instance the *Filter-Kruskal* algorithm can accelerate the computation of minimum spanning trees of graphs [12]. It improves the *qKruskal* algorithm which combines the original *Kruskal* algorithm with the partitioning idea of *QuickSort* – the edges are not sorted beforehand but a pivot edge is chosen, the problem is solved recursively on all edges with smaller weight and afterwards (provided the spanning tree is still incomplete) on all edges having a larger weight. *Filter-Kruskal* improves this by not using all edges of larger weight as an input for the second recursive call but only those edges which actually connect two different components of the graph, i.e., it filters all edges that cannot be part of the minimum spanning tree. This idea has also been applied to different problems later on, e.g., graph matching [11].

Another filtering approach with numerous applications is *Kalman Filtering*, also known as *Linear Quadratic Estimation*. Its goal is to predict the state of a system based on observations containing inaccuracies. It works in two steps: First, system parameters are predicted and afterwards, the predictions are updated as soon as the next observation (measurement with inaccuracies) arrives using a stochastic weighted average approach. Applications of Kalman

Filtering can be found in various areas, among others navigation control of vehicles, robot motion planning, and signal processing. It is also provably a valuable tool for data stream analysis. Similar to our goal, Kalman Filtering is used to reduce the communication in a sensor server architecture in [5]. Here, Kalman Filtering is applied on both the server and the sensor side (the sensors provide a data stream for the server). As long as the sensor observes values that are within a small deviation of its current prediction, the sensor does not communicate to the server. Once the deviation exceeds a certain threshold, the sensor updates the server.

Next, we introduce the formal notion of filters and necessary definitions for our model. A set of filters is a collection of intervals, one assigned to each node such that, as long as the observed values at each node are within the given interval, the value of the output function does not change.

Definition 4. *For a fixed time t, a* set of filters *is defined by an n-tuple of intervals $(F_1^t, \ldots, F_n^t)$, $F_i \subseteq \mathbb{N} \cup \{-\infty, \infty\}$ with $v_i^t \in F_i^t$, such that as long as the value of node i only changes within its interval,i.e., it holds $v_i^{t'} \in F_i^{t'} = F_i^t$ for $t' \geq t$, the value of the output function does not change. We use $F_i^t = [\ell_i^t, u_i^t]$ to denote the lower and upper bound of a filter interval, respectively.*

We assume that nodes are assigned filters by the server. If a node *violates* its filter, i.e., the currently observed value is not contained in its filter, the node may report the violation and its current value to the server. The server then computes a new set of filters and sends them to the affected nodes. To calculate a set of filters that works for the entire set of nodes, the server may need to probe some more nodes before sending out the new filters. At the end of each time step, no node is allowed to violate its filter. An algorithm following this approach is called *filter-based*.

The easiest way of defining a set of filters is to assign the value a node currently observes as its interval. In this case the usage of filters is not very beneficial, so we are looking for filters that are as large as possible to minimize the number of filter changes which is directly related to the number of exchanged messages.

Our analysis is based on the classical competitiveness approach first used in [13] and later on formalized by [6]; see also [2] for an overview. We compare the communication volume of our algorithms to one of an appropriately defined offline algorithm. In our model, a general offline algorithm knows all the input streams in advance and can trivially solve the aforementioned problems without any communication. To still get meaningful results regarding the quality of our algorithms, we assume the optimal offline algorithm OPT uses filters assigned by the server to the nodes. To lower bound the cost of OPT, we count the number of filter updates over time.

Definition 5 (Competitive Algorithms). *We call a (randomized) online algorithm* ALG c-competitive *if for every instance its (expected) communication volume is by a factor of at most c larger than the communication volume of* OPT.

2.3 Computational Primitives

This section is dedicated to three subroutines that will be used in later algorithms. Due to space constraints, we will use these protocols mainly as black-boxes, see the cited literature for more details. The first subroutine is a protocol for the Existence problem. In this problem, all nodes observe binary values, $\forall i \in \{1, \ldots, n\} : v_i \in \{0, 1\}$ and the goal for the server is to output the *logical disjunction*.

The Existence Protocol solves this problem in $\log(n) + 1$ rounds. In each round $r = 0, 1, \ldots, \log n$, all nodes that have observed the value 1 send a message with probability $2^r/n$ to the server. As soon as the first message reaches the server, the protocol ends (latest if $r = \log(n)$ holds).

Theorem 1 (Existence). *[9 SPP] There exists an algorithm* Existence Protocol *which uses* $\mathcal{O}(1)$ *messages in expectation and at most* $\log n + 1$ *communication rounds to solve the problem* Existence.

The Existence Protocol has several applications. Most important for our research is the detection of filter violations. The server can detect a filter violation using only a constant number of messages on expectation.

Corollary 1 (Filter Violation). *[9 SPP] There is a protocol* Existence Protocol *which uses* $\mathcal{O}(1)$ *messages in expectation to identify a filter violation. In case there are multiple filter violations one is drawn uniformly at random. If no filter violation occurs no message takes place.*

Additionally, the Top-k Protocol is able to solve the Top-k-Value Monitoring problem. The protocol uses similar ideas as the Existence Protocol: Nodes draw a height from a geometric distribution and a tree like structure is built. Initially, s_1 is determined by collecting a sample of all values, broadcasting the largest one and continuing until s_1 is determined. The same idea is used to find $s_2, \ldots, s_k$.

Theorem 2 (Top-k). *[4 SPP] The* Top-k Protocol *uses* $k + \log(n) + 2$ *messages in expectation and* $\mathcal{O}(k + \log n)$ *expected number of rounds to solve the Top-k-Value Monitoring problem.*

3 Top-k-Value Monitoring

In this section we consider problems regarding the k largest values at the current time step t. We design and analyze Las Vegas algorithms, i.e., we always output the correct values and can show that the total communication and number of rounds are polylogarithmic with high probability.

Consider a general input for the Top-k problem over time. The values might change over time as well as the nodes holding the Top-k values. As a consequence, in a worst-case situation we cannot reuse any information from previous

time steps and need to recompute the output from scratch. To counteract this possibility, we consider two different approaches.

In our first approach, we restrict the number of values which can change between queries, and parameterize the result in this number. We show that we can build up a data structure which preserves important information as long as there are not too many updates. This makes answering queries much more efficient, as we use the data structure to quickly reduce the number of candidate nodes which potentially hold the desired result.

In our second approach, we consider filter-based algorithms for the problem. These algorithms have the advantages discussed in Sect. 2.2, i.e., they are very effective if the changes in the output are not too large. To conduct a meaningful worst-case analysis, we consider the competitiveness of the algorithms against a filter-based offline algorithm.

Before we give the details of our solutions, we shortly mention that our protocols which compute the Top-k from scratch are essentially optimal with respect to the amount of communication. Intuitively, we can show that an algorithm cannot do much better than performing a binary search on n values. The algorithm can always ask a set of nodes for their value, and then broadcast the maximum to 'eliminate' all nodes with a smaller values for the process. Formally, Yao's minimax principle considering a random permutation as input can be applied. Each input occurs with probability $(1/n!)$ and it is shown that any deterministic algorithm needs at least $\Omega(\log n)$ messages on expectation which yields:

Theorem 3 ([8 SPP]). *Every comparison-based randomized algorithm requires at least $\Omega(\log n)$ messages on expectation to compute the maximum in our model.*

3.1 Dynamic Distributed Data Structure

In this section we consider a data structure for the rank related problems of Top-k and k-Select. The k-Select problem asks to identify the data item with rank k. We consider the approximate version, where we have to output an item with rank in $[(1-\varepsilon)k, (1+\varepsilon)k]$ with probability at least $1-\delta$. An approximate version with weaker conditions will also help us to solve the Top-k problem. For the bounds on communication, we consider the following setting: Only when there is a query for the Top-k or for k-Select, the output is determined. We allow the parameters to be different from query to query and, furthermore, we allow multiple k-Select queries at the same time step.

Our results are based on the idea of maintaining a (distributed) data structure which is used to answer a query and is informed about each update. More precisely, at every point in time, the data structure keeps track of an approximation of a data item with rank k. These approximations can be exploited by the protocols for a Top-k or k-Select computation to significantly decrease the communication and, interestingly, also the time bounds, rendering this approach very powerful.

The data structure supports the following operations: `Top-k`: Output $\{s_1^t, \ldots, s_k^t\}$, `StrongSelect`: Output $d \in \{s_{(1-\varepsilon)k}^t, \ldots, s_{(1+\varepsilon)k}^t\}$ and `WeakSelect`:

Output d with $s^t_{k\cdot\log^{c_1} n} \leq d \leq s^t_{k\cdot\log^{c_2} n}$, with $c_1, c_2 > 1$. The `Top-k` and `StrongSelect` operations answer queries for the Top-k and k-Select problems, while the operation `WeakSelect` supports the other two. Our data structure guarantees the following:

Theorem 4 ([4 SPP]). *There is a distributed data structure with expected amortized total communication cost for an update of $\mathcal{O}(1/\operatorname{polylog} n)$. The amortized number of rounds for an update is $\mathcal{O}(1)$. The data structure is able to answer a k-Select query correctly with probability at least $1-\delta$. For that, $\mathcal{O}(1/\varepsilon^2 \log 1/\delta + (\log\log n)^2)$ messages and $\mathcal{O}(\log\log \frac{n}{k})$ rounds are required on expectation. Additionally, the expected total communication cost to answer a Top-k query is $\mathcal{O}(k+\log\log n)$ and the expected number of rounds is $\mathcal{O}(\log\log n)$. The output is always correct.*

Our data structure is designed as follows. We maintain a *Sketch(t)* about the data items received at time t in the server. The task of such a sketch is to maintain items to answer `WeakSelect` queries instantly. A *Sketch(t)* is a subset of data items denoted by $\{r^t_1, \ldots, r^t_m\}$, where $m \leq \log n$. We call *Sketch(t)* correct if it consists of a set of data items $\{r_1, \ldots, r_m\}$ such that, for each $k = 1, \ldots, n$, there exists r_k such that $s^t_{k\cdot\log^{c_1} n} \leq r_k \leq s^t_{k\cdot\log^{c_2} n}$. We say the data item r_k is the representative of the set of data items d with $s_{k\cdot\log^{c_1} n} \leq d \leq s_{k\cdot\log^{c_2} n}$. To answer the `WeakSelect` query for a specific rank in $[k\cdot\log^{c_1} n, k\cdot\log^{c_2} n]$, we simply output the representative r_{k+1}.

Computing a *Sketch* is somewhat expensive, hence we want it to be valid even after some values have been updated. It is easy to see that for appropriately chosen constants c_1, c_2, up to $\log^c n$ values can change without the property being lost. In conclusion, we can achieve the stated performance guarantees by computing a *Sketch* which is valid for $\log^c n$ updates, after which we recompute it from scratch. The `WeakSelect` operation simply returns an appropriate element from the *Sketch*.

Now, recall that there is a protocol for `Top-k` which uses $k + \log(n) + 1$ messages and $\mathcal{O}(k + \log n)$ rounds in expectation (Theorem 2). These bounds hold when the protocol is executed on n nodes without using any information from previous time steps. We can now utilize our *Sketch* in the following way: We execute a `WeakSelect` operation with input k, such that we receive a data item d of size at most $s^t_{k\cdot\log^{c_2} n}$. Then, we execute the `Top-k` protocol only for nodes which hold a data item smaller than d, i.e., we execute the protocol only on $\mathcal{O}(k\log n)$ nodes instead of n, yielding the desired bound. The bound on the `StrongSelect` operation can be obtained in a similar fashion.

3.2 Filter-Based Algorithm

We turn our attention to filter-based algorithms which we evaluate in the framework of competitive analysis. We are going to compare the algorithm against an optimal offline algorithm, which knows all of the future input in advance. To make this analysis meaningful, it is necessary to also restrict the offline algorithm to a filter-based approach. The important part of the filter-based approach

is that the offline algorithm has to communicate a set of valid filters to the nodes. In accordance to Definition 4, this means that the offline algorithm at least has to communicate each time the output changes.

The algorithm works as follows: First, the k largest values are determined using the Top-k-Protocol of Theorem 2 . Afterwards, the server broadcasts s_k such that all nodes i with $v_i \geq s_k$ define their filter to $F_i := [v_i, v_i]$ and the remaining nodes i with $v_i < s_k$ to $F_i := [-\infty, s_k]$. Whenever a node with one of the k largest values observes a different value, a filter violation occurs such that the node sends a message to the server. Each of the other nodes (those with filters $F_i := [-\infty, s_k]$) that observes a filter violation executes the Top-k-Protocol (to prevent that every node sends a message). The server unifies and outputs the k largest values of the nodes without a filter violation from the past time step and the new values of the current time step. This algorithm has the following guarantees.

Theorem 5 ([4 SPP]). *There is an online algorithm which monitors the Top-k-Values and is $\mathcal{O}(k + \log n)$-competitive against an optimal filter-based offline algorithm.*

4 Top-k-Position Monitoring

In this section we consider monitoring the IDs of the nodes which observe the Top-k values rather than the values themselves [8 SPP, 9 SPP]. The intuitive advantage is that small updates to the values of the nodes holding the Top-k do not necessarily mean a change in the Top-k positions. Hence, in a scenario where there are a lot of small fluctuations in the observed values but the overall ranking of nodes stays the same, we have to utilize much less communication if we monitor the nodes.

We only consider filter-based algorithms in this section. For the general approach as in Sect. 3.1, there is no further benefit from monitoring only the positions, as the entire data structure approach aims at optimizing cases in which only a fraction of nodes observe new values. On the other hand, it directly provides a solution for the positions since nodes can always send their IDs along with their values.

For the filter-based algorithm, we expect less communication due to the reason explained above. In fact, we observe an increase in the competitive ratio for the position monitoring: Under worst-case input sequences, the offline algorithm can gain a greater advantage in comparison to the online algorithm.

Theorem 6 ([10 SPP]). *Let each sensor node observe values from $1, \ldots, \Delta$. There is an online algorithm which monitors the Top-k-Positions and has a competitiveness of $\mathcal{O}(k + \log n + \log \Delta)$ compared to a filter-based offline algorithm.*

4.1 Filter-Based Top-k-Position Monitoring

The main observation for our approach is that for this problem it is sufficient to send only a single value v which divides the Top-k from the remaining nodes,

i.e., a value which is between the kth and the $(k+1)$st largest value. Based on this observation, the main task for the online algorithm is to decide where to set the value v which divides the Top-k and the remaining sensor nodes from each other. Since no information about the future is known, and the adversary has no restriction in the process of generating the values that the sensor nodes observe in future time steps, we simply take the median value.

Top-k Position protocol: Initially identify the kth and $(k+1)$st largest values and the respective sensor nodes (using the one-shot protocol). As long as the Top-k-Positions do not change, define the bound for the filters as the median value between the kth and the $(k+1)$st largest value.

In addition to the execution of the one-shot protocol from Theorem 2, this strategy yields additional $\mathcal{O}(\log \Delta)$ messages in expectation by applying the Existence Protocol from Theorem 1 for identifying a filter violation. Violations can occur until we have found the correct separation between the kth and $(k+1)$st largest value, which takes at most $\log \Delta$ steps, because by choosing the median value, we essentially perform a binary search for the correct value. Note, that since the adversary is offline adaptive, it is easy to see that every online algorithm needs at least $\Omega(\log \Delta)$ messages which easily translates to an overall lower bound of $\Omega(k + \log n + \log \Delta)$ on the competitiveness for any randomized online algorithm. By this, the bound in Theorem 6 is asymptotically tight.

While this strategy performs generally well under minimal changes to the input values, a lot of communication can occur if, e.g., the nodes holding the kth and $(k+1)$st largest values often switch positions, but these values are almost the same. In such a situation, it might be sufficient not to take note of the exact Top-k (e.g., for outdoor temperature one degree differences might not matter to us). We address this by proposing an algorithm calculation positions for Approximate Top-k as by Definition 2.

4.2 Filter-Based Approx. Top-k-Position Monitoring

In this section we allow the online algorithm to have some errors in its output and compare against an optimal offline algorithm which solves the exact problem. Recall that monitoring the Approximate Top-k-Positions allows (only) the online algorithm to choose nodes as an output which are 'close' to the kth largest value (see Definition 2). Observe that filters are allowed to overlap if we consider the relaxation of the Top-k-Position problem.

We want to make use of the allowed error in the following way: When solving the exact problem, we had to search the value domain for the correct separation between the kth and $(k+1)$st ranked value. Allowing an error means that we only need to find an approximation of this separating value, resulting in a faster search. In fact, if we introduce an additive error (say M), it is easy to see that the competitiveness compared to a filter-based offline algorithm which solves the exact problem is reduced from $\mathcal{O}(k+\log n+\log \Delta)$ to $\mathcal{O}(k+\log n+\log(\Delta-M))$.

However, if we use the standard notion of a multiplicative error the following disadvantage occurs: If the values we search for are smaller, the range of values

which lie within the margin of error also becomes smaller. So in a way, the criterion for a valid outputs becomes stricter when dealing with smaller values.

To circumvent this shortcoming, we first apply a binary search strategy on a logarithmic scale which terminates after $\log\log \Delta$ filter violations and stops with the property that the allowed error can only vary within constant factors. Applying the approach from the algorithm in Theorem 6 with an early stopping rule, the following can be achieved:

Theorem 7 ([10 SPP]). *Let each sensor node observe values from* $1, \ldots, \Delta$. *There is an online algorithm which monitors the Approximate Top-k-Positions with a competitiveness of* $\mathcal{O}(k + \log n + \log\log \Delta + \log 1/\varepsilon)$ *compared to a filter-based offline algorithm which monitors the exact Top-k-Positions.*

4.3 Approximate Offline Algorithm

In this section, we study a variant in which the optimal offline algorithm is allowed to introduce an error, i.e., both the online and offline algorithms monitor the Top-k-Positions approximately. It turns out that it is much more challenging for online than for offline algorithms to take advantage of the relaxed conditions for a correct output, resulting in a significantly higher competitive ratio. This fact is formalized in a lower bound of $\Omega(n)$ (for constant k) [9 SPP], which is much larger than previous upper bounds of $\mathcal{O}(k + \log n + \log \Delta)$ for the exact problem. Intuitively speaking, the online algorithm has to choose where to set filters, but also has to choose a subset of nodes the output is based on which significantly increases the lower bound:

Theorem 8 ([10 SPP]). *Any filter-based online algorithm which solves the approximate Top-k Position Monitoring problem cannot be better than* $\Omega(n + \log \Delta)$*-competitive.*

We consider two settings in which we compare to an approximate offline algorithm and design algorithms for the respective settings: First, an online algorithm has the task to solve the problem with the same error ε as the offline algorithm and second, an online algorithm is allowed to use 2ε, i.e., twice the error of the offline algorithm.

For the first setting, the online algorithm is allowed to make use of the same error ε as the offline algorithm, which results in a competitiveness of $\mathcal{O}(n^2 \log \Delta)$ (assuming reasonable values of ε or simply assuming to be constant). Intuitively speaking, in this scenario the online algorithm has to solve two questions at the same time: The bounds of the filter intervals, and the choice of the subset of nodes for the output.

Theorem 9 ([9 SPP]). *Assuming* ε *is a constant, there is an online algorithm for the approximate Top-k Position Monitoring problem which is* $\mathcal{O}(n^2 \cdot \log \Delta)$*-competitive.*

This interaction between the two questions leads to a gap between the lower bound and the upper bound stated above. To reduce the power of the adversary but still to consider the problem of choosing a subset of nodes for the output, we consider an augmented version which allows the online algorithm to use an error of 2ε compared to ε in the offline algorithm. The algorithm is $\mathcal{O}(n)$-competitive (again with reasonable assumptions on ε and also on the relation of n and Δ). In this setting with a constant number of filter violations it is possible to argue on the placement of filters and thus the combination of filter placement and the subset of the nodes do not take that much of a role expressed in the following:

Theorem 10 ([9 SPP]). *Assuming ε is a constant and $\log \Delta = \mathcal{O}(n)$, there is an online algorithm for the approximate Top-k Position Monitoring problem which is $\mathcal{O}(n)$-competitive against an optimal offline algorithm using an error of 2ε compared to the error of ε of the offline algorithm.*

5 (Approximate) Count Distinct Monitoring

In the following section, we consider the Count Distinct Monitoring problem where the server is tasked to count how many different values are observed at the sensors. More specifically, we establish an (ε, δ)-approximation of the number of distinct values d_t at time step t. On a high level, our approximation scheme shows how one can combine both a filter-based approach together with a sampling technique to shrink the required communication. Due to space constraints, we only explain our techniques on a high level. For details we refer to [1 SPP].

The key idea for estimating d_t is to follow a sampling approach on the values (not on the nodes). We create a sample out of all values and use the Existence Protocol (Theorem 1) to identify a representing node for each sampled value, i.e., one node per value of the sample set observing the value. Then, we monitor the identified representing nodes to keep track of d_t over time. For the monitoring, a filter-based approach is utilized, allowing us to compare the communication volume of our protocol to a minimal filter-based one as already done in previous sections.

We are able to achieve an (ε, δ)-approximation that is kept valid for multiple time steps depending on how much the values change in consecutive time steps (parameterized by σ). Using the filter-based approach, our analysis relates to the number of messages exchanged by an optimal filter-based approach (R^*). In total, we arrive at the theorem below.

Theorem 11 ([1 SPP]). *There is an (ε, δ)-approximation for the Count Distinct Monitoring problem for T time steps that uses $\mathcal{O}\left((\sigma+\delta)\frac{R^* \log n}{d_t} T \frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right)$ messages. Here, the change in the number of nodes observing a fixed value between consecutive time steps is upper bounded by a constant factor $\sigma \leq 1/2$ and R^* is the minimum number of changes of representatives for a given input.*

The bound stated above is comprised of different aspects which are reflected by factors stemming from a sampling approach $\Theta(1/\varepsilon^2 \log 1/\delta)$, the fact that

the number of domain changes is bounded $\Theta(\sigma + \delta)$ and the competitiveness of monitoring the representative for one domain ($\mathcal{O}(\log n \cdot R_v^*)$) with respect to R_v^*, the number of representatives to monitor that a value v was observed used by an optimal offline algorithm.

The bound of the algorithm can also be expressed as $\mathcal{O}(\log n \cdot R_S^* \cdot 1/\varepsilon^2 \log 1/\delta)$ where R_S^* denotes the optimal number of representatives for the sample set S throughout the time period T. Furthermore, focusing on the aspect of dynamic algorithms, the bound can also be expressed as $\mathcal{O}((\sigma + \delta) \cdot T \cdot 1/\varepsilon^2 \log 1/\delta)$. Note that these bounds are different bounds for the same algorithm and only reflect different input sequences more properly.

5.1 Computation for One Time Step

The computation for one time step takes place in two phases. First, a constant factor approximation for d_t is created. In the second phase, the constant approximation is used to determine a sufficiently large probability that is broadcasted to the sensors, which in turn create a sample out of all observed values that is reported to the server. Based on the size of the sample set and the previously calculated probability, the server can estimate d_t up to a factor of ε with a probability of at least $1 - \delta$.

It is crucial here that we do a random experiment for a value, i.e., all sensors observing the same value should see the same outcome of the random experiment. This can be achieved by a *public coin* [7]. A public coin is a random string consisting of fully unbiased bits that is common for all sensor nodes. It can be implemented by having the same pseudorandom number generator at each sensor initialized by a common seed that is broadcasted by the server at the beginning of each phase of the algorithm. Note that such an approach only increases the communication complexity by an additive constant. A set of sensors (observing the same value) is able to do a random experiment together by considering the same substring of the public coin (which is predefined by the value the sensors are observing).

For a constant factor approximation we first let the sensors draw a random number with the public coin based on a geometric distribution, i.e., we generate a random height h_v for each value v. Then the server triggers a communication of the values of largest height by polling the heights from largest to smallest in synchronous rounds. Thereby, for each value that is communicated, the Existence Protocol is used (cf. Theorem 1) to bring down the number of communicated messages to a constant.

After we have a constant factor approximation, we calculate a probability p which is broadcasted to the sensors. With probability p a value is communicated to the server in the second phase. Whether or not a value is communicated is again decided for all sensors observing the value using the public coin. For each of the values selected in this phase, the Existence Protocol is used again (cf. Theorem 1) to identify a representing sensor. Such a representing sensor witnesses that the sampled value is observed. p is chosen with respect to ε,

δ and the constant factor approximation such that the server can compute an (ε, δ)-approximation based on the number of received values of the second phase.

In the end, most of the communication happens due to the chosen probability to have a sufficiently large sample of the observed value. Thus, we arrive at the theorem below.

Theorem 12 ([1 SPP]). *There is an (ε, δ)-approximation algorithm for the Count Distinct Monitoring problem for one time step using $\mathcal{O}(1/\varepsilon^2 \log 1/\delta)$ messages on expectation.*

5.2 Monitoring over Multiple Time Steps

In the worst case values might change arbitrarily between multiple consecutive time steps and a sensor that was used as a representative for a value might not be of use even after a single round. However, as argued before, one expects based on practical scenarios that the values that are observed at a fixed sensor are similar in consecutive time steps. To analyze the quality of our algorithm with regard to the significance of changes in consecutive time steps, we use a filter-based approach. The idea is to reuse the results of the (relatively) costly computation of one time step for consecutive time steps as long as the values are similar to a certain degree. The filter is implemented by the representing sensors that are identified, i.e., we compare how many times our protocol has to identify such a representing sensor compared to how many times an optimal filter-based algorithm has to do this (R^*).

Recapitulate that using a public coin, a sample set of values was determined. The server keeps track of the sample after an initial (ε, δ)-approximation is done. Thereby, any sensor sends a message to the server if it observes a value in the sample that has not been observed previously. The server estimates based on such messages how many values are in total newly observed. Similarly, if a representative for a value in the sample stops observing the latter, a new representative is searched (using the Existence Protocol, cf. Theorem 1) and if none is found, the server estimates how many values left in total. Since any filter-based algorithm has to communicate at some point when an optimal representative sensor stops observing its value, our result depends on the minimum possible number of such changes R^* as it can be seen in Theorem 11.

6 Conclusion

In this work we elaborated on models for dynamic input sequences and designed and analyzed algorithms which handle these settings. The respective bounds reflect this by comparing the communication to an optimal filter-based algorithm or by introducing parameters expressing how 'fast' an instance changes from time step to time step. We have also shown that there is an algorithm which combines these two techniques properly.

As a next step it would be interesting to see how these techniques perform in the presence of sliding windows. The fact that sensors are not capable of storing the entire history of the data stream has an influence on the output quality or the number of messages the sensors need to send to the server, although these values might not be relevant for the current time step.

Another aspect on the input streams might have a significant impact on communication bounds: Assuming the streams have a structured property, e.g., be provided by some random process and thus might be assumed to generate similar observations in consecutive rounds at a respective sensor node. With such an assumption in mind we assume to get bounds in return which reflect the communication complexity to be proportional in the ability of projecting future observations based on past observations.

References

1 SPP. Bemmann, P., et al.: Monitoring of domain-related problems in distributed data streams. In: Das, S., Tixeuil, S. (eds.) SIROCCO 2017. LNCS, vol. 10641, pp. 212–226. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72050-0_13

2. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
3. Cormode, G.: The continuous distributed monitoring model. SIGMOD Rec. **42**(1), 5–14 (2013). https://doi.org/10.1145/2481528.2481530

4 SPP. Feldkord, B., Malatyali, M., Meyer auf der Heide, F.: A dynamic distributed data structure for top-k and k-select queries. In: Böckenhauer, H.-J., Komm, D., Unger, W. (eds.) Adventures Between Lower Bounds and Higher Altitudes. LNCS, vol. 11011, pp. 311–329. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98355-4_18

5. Jain, A., Chang, E.Y., Wang, Y.: Adaptive stream resource management using kalman filters. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, 13–18 June 2004, pp. 11–22 (2004). https://doi.org/10.1145/1007568.1007573
6. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica **3**(1), 77–119 (1988). https://doi.org/10.1007/BF01762111
7. Kremer, I., Nisan, N., Ron, D.: On randomized one-round communication complexity. Comput. Complex. **8**(1), 21–49 (1999). https://doi.org/10.1007/s000370050018

8 SPP. Mäcker, A., Malatyali, M., Meyer auf der Heide, F.: Online top-k-position monitoring of distributed data streams. In: IPDPS, pp. 357–364. IEEE Computer Society (2015). https://doi.org/10.1109/IPDPS.2015.40

9 SPP. Mäcker, A., Malatyali, M., Meyer auf der Heide, F.: On competitive algorithms for approximations of top-k-position monitoring of distributed streams. In: IPDPS, pp. 700–709. IEEE Computer Society (2016). https://doi.org/10.1109/IPDPS.2016.91

10 SPP. Malatyali, M.: Big data: sublinear algorithms for distributed data streams. Ph.D. thesis, University of Paderborn, Germany (2019)

11. Osipov, V.: Algorithm Engineering for fundamental Sorting and Graph Problems. Ph.D. thesis, Karlsruhe Institute of Technology (2014). http://digbib.ubka.uni-karlsruhe.de/volltexte/1000042377
12. Osipov, V., Sanders, P., Singler, J.: The filter-kruskal minimum spanning tree algorithm. In: Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, New York, USA, 3 January 2009, pp. 52–61 (2009). https://doi.org/10.1137/1.9781611972894.5
13. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985). https://doi.org/10.1145/2786.2793