Efficient and Robust Top-k Algorithms for Big Data IoT

Ruifan Yang*, Zheng Zhou†, Lewis Tseng†, Moayad Aloqaily‡, Azzedine Boukerche§

*Cornell University, Ithaca, NY, USA. ry298@cornell.edu
†Boston College, Boston, MA, USA. {zhoupt; lewis.tseng}@bc.edu
‡Al Ain University, UAE maloqaily@ieee.org
§University of Ottawa, Ottawa, ON, Canada. boukerch@eecs.uOttawa.ca

Abstract—**Top-k considers as a technique to retrieve, from a hypothetically big data set, only the k ($k \geq 1$) best (most relevant/important) candidates. Top-k query processing is a decisive necessity in various collaborative environments that comprise big data such as the Internet of Things (IoT) networks. Particularly, efficient top-k processing in large-scale distributed systems has shown a positively noticeable effect on their performance. This paper considers the distributed approximate top-k processing algorithms dedicated to the IoT-based networks and improve the accuracy of algorithms introduced previously. We then propose a safety-based fault-tolerance notation and contribute to improving a known algorithm in terms of accuracy. Our algorithms have been evaluated using simulation and real-world data and show superiority over conventional methods.**

Keywords – IoT, Top-K query, fault-tolerance, distributed algorithms, approximate query

I. Introduction

Top-k queries have a wide spectrum of practical applications. One of the simplest and easy-to-grasp application is *election* which precisely capture the notion of "global" and "local" counts of votes. In a large-scale election, each "ballot box" has a *local* count of votes for each candidate (or item); however, what we really wants to find out is the *global* counts of votes, that is, the aggregation of all the local votes. More precisely, a top-k query should output the answer to the following question in a global sense:

Which are the k most popular candidates in the system?

This has many natural applications in large-scale big data systems. For example, find most popular files in a peer-to-peer file-sharing system (such as BitTorrent), identify potential DoS (Denial-of-Service) attack, and produce ranking in multimedia systems [1]. Due to its wide applications, efficient Top-k mechanisms have been a hot research topic, e.g., [1]–[6].

This paper focuses on the distributed and robust top-k algorithms in the setting of Internet-of-Things (IoT). IoT has emerged few years ago as an enabler to many applications and services. It has the ability to manage data and services in different environments [7] [8]. IoT devices of different types can cooperate among each others and the cloud/fogs to deliver simple/complex service to clients [9]. In addition, it has the convenience to manage mobile edge computing mobility in 5G settings [10]. IoT also contributed to the smart transportation systems in terms of path recommendation and planning [11]. Different algorithms and protocols using learning technologies have been proposed and recommended in such fertile environment [12] [13] [14].

Typically, the top-k query algorithms begin with the construction of a spanning tree, and then the aggregation tasks (such as counting and summing) can be completed using $O(\log n)$ bits per node by routing data over the spanning tree, where n is the number of nodes in the network [15]. If n is too large, then finding "exactly" k most popular items (or candidates) has a prohibitively high communication cost in IoT applications. Since IoT network scenario that requires ultra-low bit complexity, the focus of this work on the approximate algorithms. To reduce cost, approximate queries (or aggregation) mechanisms have been proposed, e.g., [15]–[18], which do *not* always output the k most popular items – instead, some randomized tricks are used to produce k items that are popular.

Motivation: In a large-scale network, e.g., sensor networks and IoT, Top-k queries is essentially a special case of aggregation [19], [20]. In this area of distributed aggregation, many efficient algorithms have been proposed [19], [20]. A typical performance measure is network cost, including total number of bit exchanged and total number of bit that each node sent/received. This motivates us to inspect the first part of the paper: a lightweight top-k algorithm with an improved accuracy. Here accuracy is defined with respect to the quality of the final answer (i.e., the k choices output by the algorithm).

The second part of the paper is motivated by the observation that IoT devices are usually fragile. However, very few study addressed the fault-tolerance issue on top-k queries. In this part, we explore several reasonable notions of fault-tolerance, and proposed algorithms that are either optimal or effective. For the optimal algorithm, we provide a formal proof. For the others, we build a simulator to evaluate the efficacy of our novel algorithms. For one set of algorithms, we also use real-world sensor data set (the one from Array of Things) for evaluation. Array of Things are open-data set released by Chicago. The data is collected by a networked system of urban sensors, which demonstrate the applicability of our algorithms in a practical environment.

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

Contributions: More concretely, this paper has the following contributions:

- We improve the accuracy of the state-of-the-art distributed top-k algorithm proposed in [21]. We use simulation to validate our improvement.
- We propose a notion of fault-tolerance, *safety*, which roughly says that *no* faulty item will ever be selected. We modify the famous algorithm by Fagin [2] and formally prove that it achieves the best accuracy among all safe algorithms.
- We propose a family of algorithms that explore *unsafe* algorithms, and use an example to illustrate that these algorithms are necessary to avoid the ill-formed scenario which has accuracy equal to 0. We use both simulation and real-world data from Array of Things to evaluate our algorithms.

II. Related Work

Top-k query processing has been discussed extensively in the database applications. The Fagin Algorithm (FA) and the Threshold Algorithm (TA) [2] are two most popular algorithms in this area. Our fault-tolerant versions are based on these two algorithms. Top-k algorithms have also received attentions in other settings. Bruno et al. [6] explored the problem over web accessible databases. Klee et al. [4] are among the first to examine the problem of approximate top-k algorithms in distributed environments. Reference [5] is the closest to our work, where they also modified the TA algorithm for distributed sensor networks, and conducted an experiment where some nodes may fail. However, the authors did not provide much formal analysis. To the best of our knowledge, we are the first to formally explore ways to improve performance and fault-tolerance of top-k algorithms.

Some works have adopted the top-k algorithm in the notation of IoT-based networks such as the works in [22], [23] and [24]. The authors in [22] proposed a k-query-based dynamic real-time scheduling driven by IoT-based sensor data streams to dynamically achieve the waste collection and delivery by optimizing the garbage truck waste pick-up, size, and routes. The work presented in [23] employed the parallelism-edge computing idea in order to facilitate the top-k controlling process over numerous IoT data streams. The authors considered the issues of uncertainty, computation cost, and accuracy, then they validated their methods through the simulated experiments. The authors in [24] have developed a fixed-memory heuristic algorithm along its distributed-memory extension to the top-k issue. The proposed approach showed an efficient run-time complexity in distributed-memory settings.

Some works have proposed solutions that tackle the privacy perspective, such as the work presented in [25], where the researchers introduced a top-k query-based solution over an encrypted database for distributed system nodes. Some applications that involved continuous query processing over distributed environments, such as web-based systems and critical networks sensing and monitoring, has been addressed in [26], where the authors addressed the issue of a continuous top-k query over distributed traffic environment. They presented a solution that monitors the top-k data and eliminated the non-qualified objects from the outings that helped in minimizing the costs of communication and computation.

TABLE I: Notations and parameters of the algorithm

Parameter	Description
$A_j(e_i)$	The value of attribute A_j of record e_i
N_i	The sum of all attributes of record e_i
T	Sampling threshold, aka the sampling ratio
L	Number of "tickets" each nodes passes to others
c	Number of cycles that nodes exchange "tickets"
s	Number of runs that the main algorithm needs to be executed
m	Final cut-off for a value to appear in the final top-k items

III. An Approach to Improve Performance

System Model and Problem Formulation: The system model of this paper considers a network of n connected nodes, where each node have the ability to send a message to all other nodes. Since we focus on the design of top-k algorithms, we abstract away from the routing and networking. Some nodes may fail by *not* following the specification of the algorithm, and sending arbitrary data. These nodes are called *faulty* nodes; other nodes are called *non-faulty*. There is a fixed set of items that are known to every node, and each node has measurement (or vote) to a subset of those items. Each item has a unique identifier. The goal is to find an approximate answer to the k most popular items measured by all the *fault-free* nodes.

Let $\{e_1 \cdots e_l\}$ be a set of items with unique identifiers (IDs) $\{1 \cdots l\}$ in a distributed system. Let $A_1 \cdots A_n$ be a set of unique attributes for each item. As found in most typical applications, we assume that each item e_i's attribute A_j is a non-negative value. Following the convention, we will use $A_j(e_i)$ to represent the value of attribute A_j of item e_i. Denote the sum of all attributes of e_i by $N_i = \sum_{j=1}^{n} A_j(e_i)$. Denote $N = \sum_{i=1}^{l} N_i$. We follow the assumption used in prior works that each attribute is stored at a distinct sensor node, i.e., node j has A_j. The goal is to find a lightweight algorithm to compute an approximate answer to the k items whose sum of attributes (or aggregated attributes) is the highest, i.e., the k items with highest N_i.

Note that some work also refer attribute as frequency. We will use both terms interchangeably.

Prior Approach: In 2014, Deolalikar and Eshghi proposed a lightweight mechanism, namely *Lottery Algorithm* (LA), to find top k items in an *approximate* fashion in distributed systems [21]. To our knowledge, LA is the first algorithm that only sends constant bits by each node with high probability. This feature makes LA an appropriate algorithm in the IoT setting. However, LA does not have a good performance in some setting. Here, the performance is measured by the precision defined below: $\frac{|O \cap K|}{k}$,, where O is the output of LA, and K is the set of correct top-k items.

In this section, we show how we improve LA, and use simulation to demonstrate the improvement. Please refer to Table I for the notations we use.

Our Algorithm: Improved LA: We present our algorithm below. The key observation is that we can modify the frequency by amplifying the probability that popular items would be chosen by our algorithm, particularly the step 1) below. Note that our algorithm is based on LA, so some steps are similar to their algorithms.

Improved LA:

1) *Multiply the frequency*: For each $A_j(e_i)$, replace it by $A_j(e_i)-c$ where $c \in (0, \frac{N}{pl})$ is a constant. By doing this, we amplify the relative value of N_i for those $N_i > \frac{N}{l}$, and decrease the relative value of N_i for those $N_i < \frac{N}{l}$.
2) *Generate a list of random variables*: In each node A_j, generate a list of "tickets" of form $\langle ID, r\rangle$ where ID denotes a item e_i, and r is a random variable generated from $Exp(A_j(e_i))$,.
3) *Prune the ticket list*: Each node prunes the list generated in the previous step. Tickets whose random variable is above the sampling threshold T are discarded.
4) *Merge list by minimum*: Each node exchanges the top L tickets of the list pruned in the previous step. During the exchange, each node keeps only the minimum random variable. Notice that after the exchange, every node have tickets with the same value , i.e. the minimum random variable generated by same node.
5) *Cropping the merged lists*: Each node sorts the lists merged in the previous step in the descending order, and crops it so that only L items left in the final list.
6) *Run the algorithm multiple times and merging*: Run previous algorithms s times. Obtain the final list according to the s lists (from each run of the steps above) by counting the number of appearance of a item in these s lists. If an item occurs in at lest m out of c lists, then the item is included in the final list.

Improvement: The following theorem formally proves that our approaches works in general. That is, increasing the probability an item will be picked if it has high frequency, and decreasing the probability if it has low frequency. The proof follows from simple algebraic operations; hence, is omitted for the sake of space.

Theorem 1. *By replacing each $A_j(e_i)$ by $A_j(e_i) - c$ where $c \in (0, \frac{N}{mn})$, we amplify the relative value of N_i for those $N_i > \frac{N}{n}$, and decrease the relative value of N_i for those $N_i < \frac{N}{n}$. Furthermore, the higher the original N_i, the higher the amplification effect.*

Evaluation: We follow the same simulation used in [21]. We first generate nine datasets from Zipfian distribution by varying the following two parameter:

1) Number of items: $[700, 1500, 5000]$
2) The skewness of the distribution: $[1, 1.5, 2]$

Following the evaluation in [21], we have $N = L = c = 5$ in our experiments. We then execute LA (original algorithm in [21]) and our algorithm – Improved LA – on the dataset with different choices of (s, m) pairs. Each experiment was repeated 20 times. Finally, we output the average precision. Due to space limit, we only illustrate the result for one of the datasets. The following plot showed the precision and list length of the dataset of 700 items and 1.5 skewness. Our algorithm slightly outputs more items (with longer list), but we obtain a much higher precision in some cases.

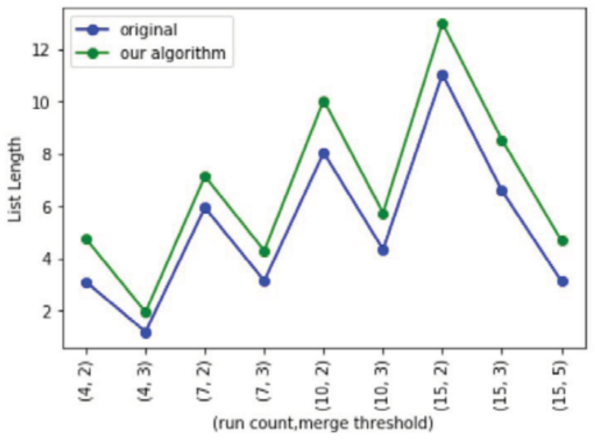


Fig. 1: Comparison of list length

IV. Fault-tolerant top-k Algorithms

Since IoT nodes are usually fragile, we are motivated to study the effect of faulty nodes. To start with, we need to introduce the notion of failure, and define appropriate properties for *fault-tolerant top-k algorithms*. Recall that the system may have at most f faulty nodes, which may not report the correct value of its attributes. Under such a scenario, we introduce the notion of a safe algorithm.

Definition 1. *An algorithm is **safe** if the algorithm will never output an item with no attribute stored in non-faulty nodes.*

Then, we show how to modify FA and TA introduced in [2] so that they are safe.

A. The proposed algorithms

Safe Fagin Algorithm

1) Do parallel access to each of the n sorted lists L_i. Stop when there are at least k "matches." That is, each of them have been accessed in all the lists.
2) For each item i that has been accessed: Retrieve all of its attributes $x_1, ..., x_n$. Compute $F(i) = F(x_1, ..., x_n)$. Set $F(x_1, ..., x_n) = 0$ if at least f of the fields are 0.
3) Return the top k answers.

Safe Threshold Algorithm

1) Do parallel access to each of the n sorted lists L_i. As each item is accessed: Retrieve all of its fields $x_1, ..., x_n$. Compute $F(R) = F(x_1, ..., x_n)$. Again, set $F(x_1, ..., x_n) = 0$ if at least f of the fields are 0. If this item is one of the top k answers so far (i.e., its global votes so far is among top k), store it into potential output O.

2) For each list L_i, let $\hat{x}_i$ be the value of the last item accessed.
3) Define the threshold value t as $F(\hat{x_1}, \hat{x_2}, \cdots, \hat{x_m})$.
4) When k items have been accessed whose global votes so far is at least t, then stop and return the top k answers in O.

Properties: The following two properties follow from [2] and our construction.

Lemma 1. *Safe TA always stops at least as early as Safe FA.*

Theorem 2. *If the aggregate function is monotone, then the algorithms find the top k safe answer.*

Note that since our algorithm takes the pessimistic approach by eliminating all the possible unsafe items, our algorithms might *not* output the exact top k items if there are failures. However, the next theorem proves that this is indeed the best we could do.

Theorem 3. *This algorithm has the best precision over all safe algorithms.*

Proof. Suppose there exists a safe algorithm A that achieves better precision than our algorithms in some scenario. Then, we know there exist an item k with $\geq f$ 0 attributes with a higher sum. Consider another scenario with exactly the same attribute on each nodes. The difference is that in this scenario, all f nodes storing non-0 attributes for item k are all faulty. In this case, algorithm A has an unsafe output, which is a contradiction. $\square$

V. Occasionally Unsafe Algorithms

Under the realm of safe algorithms, the proposed safe algorithms perform very well. However, in some other cases, all safe algorithms perform poorly. In particular, our previous algorithm does not output any potential candidate for the top items and hence has precision zero. Therefore, we investigate another category of algorithms, i.e., occasionally unsafe algorithms, which are allowed to output unsafe items occasionally. That is, the algorithm uses randomization to produce outputs. In the worst case, these algorithms might output some unsafe items, but in general, they perform relatively stable, i.e., with high precision.

Algorithm 1: In the beginning of the algorithm, we randomly picked f nodes and assume them to be faulty. Then run the Threshold Algorithm.

Algorithm 2: Randomly pick one node to exclude from our system for this round. Then, run the TA (Threshold Algorithm) to find the top one item among the remaining node, and add it to the output. Then randomly pick another node. Repeat the process again for the remaining nodes. Add the highest ranking item that is not in the output yet. Repeat until we have k items.

Algorithm 3: We first find the median of each item over all nodes, and then give rank of each item according to their median.

Evaluation: Simulation: With such a randomized behavior, it is difficult to understand our algorithms analytically. Hence, we use simulation to verify the efficacy. We run the experiments on both uniformly distributed datasets and Zipfian distributed datasets with 50 nodes, 50 items and 5 faulty nodes. We choose Zipfian distribution, because it is known that Zipfian distribution models file distribution in peer-to-peer networks quite well [27]. Uniform distribution is also common in many natural events. For each dataset, we consider the following three faulty behaviors:

1) *ZERO*: The faulty nodes set all the attributes in it to zero.
2) *UNIFORM*: The faulty nodes set all the attributes in it to a random number between 0 and 10 times of the maximum value of all attributes.
3) *LARGE*: The faulty nodes set all the attributes in it to a random number between 5 times and 10 times of the maximum value of all attributes.

For each three faulty behaviors, we run all three algorithms with different $k \in [1, 3, 5, 7, 10, 15]$, i.e., different length of output, and for each k, we generated 500 Zipfian datasets with parameters 1.34/1.3/1.25. For each item, we generate the value for each 50 node using multinomial distribution. Then, we run the aforementioned algorithms in the 500 datasets to calculate the average precision. Results for Zipfian dataset with parameters 1.25 and 50 are presented in Figure 2. Note that the second parameter is the number of nodes, which is equal to 50 in our case. Figure 3 presents results for Zipfian dataset with ZERO faulty behavior with different value of the first parameter. Figure 4 presents results for uniform dataset.

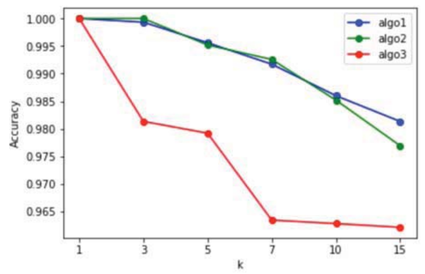
Evaluation: Discussion:

- All three algorithms are quite robust with Zipfian distribution under the failures that we consider. For uniform distribution, if we consider large enough k, then the algorithms have better accuracy.
- ZERO faulty behavior creates the worst inaccuracy.
- As the first parameter of Zipfian distribution approaches to 1, the precision increases.
- Uniform and Zipfian distributions have two different trends, since in Zipfian, the first few candidates/items dominate.

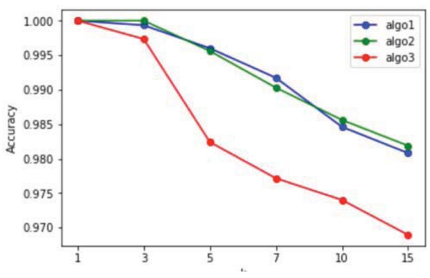
Real-World Data Set: We use sensor data from Array of Things, a networked urban sensor project.[1] Under subsystem of "chemsense," we extract HRF values (human readable values) of pollutant concentration (CO, SO_2, ground-level O_3, and NO_2) in the unit of ppm from Chicago complete daily data sets. We then calculate the AQI (air quality index) using concentration data, the breakpoints table, and linear interpolation equation from the EPA document.[2] According to US EPA, we set the highest AQI value between the 4 pollutants to be the AQI of a particular hour.

The proposed algorithms have been used to find out top 10 hour-long period with the worst air quality (i.e., top 10 hours

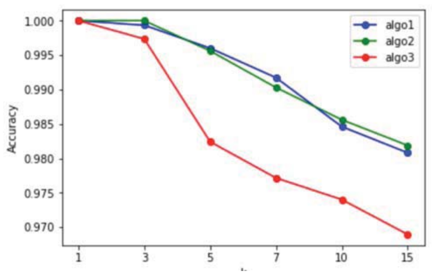
[1]https://arrayofthings.github.io/
[2]https://archive.epa.gov/ttn/ozone/web/pdf/rg701.pdf



(a) ZERO faulty behavior

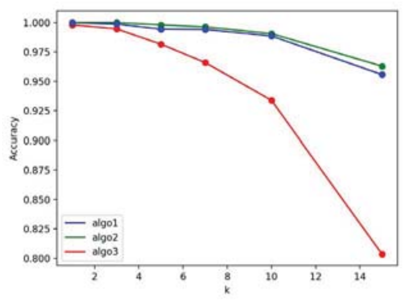


(b) UNIFORM faulty behavior

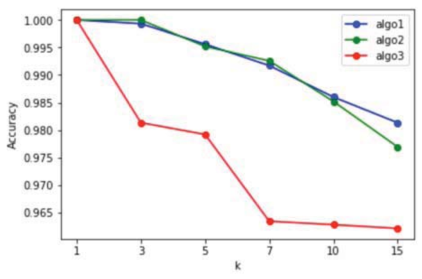


(c) LARGE faulty behavior

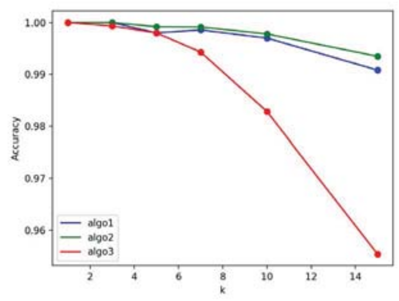
Fig. 2: Zipfian distributed dataset with first parameter 1.3



(a) Zipfian with first parameter 1.34

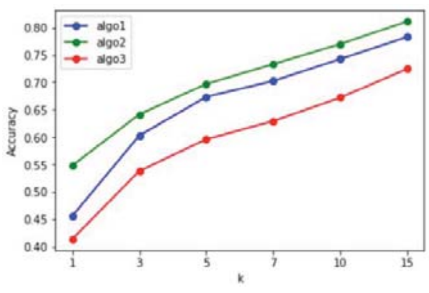


(b) Zipfian with first parameter 1.3

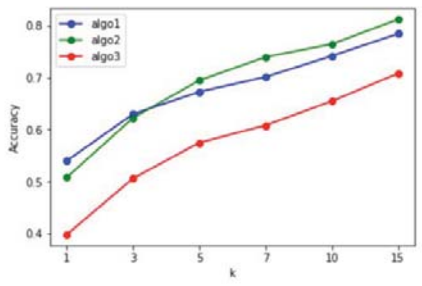


(c) Zipfian with first parameter 1.25

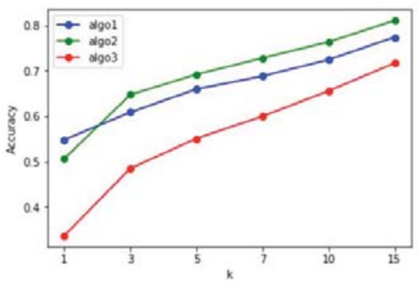
Fig. 3: Zipfian distributed dataset with ZERO faulty behavior



(a) ZERO faulty behavior



(b) UNIFORM faulty behavior



(c) LARGE faulty behavior

Fig. 4: Uniform distributed dataset

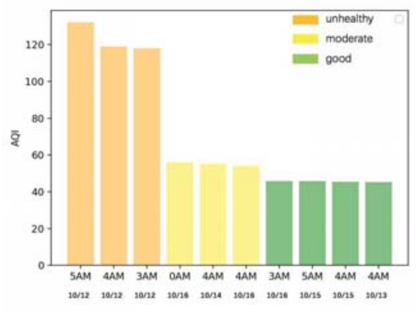
with highest AQI). That is, the top-k queries have been tested with $k \leq 10$ using the dataset from Chicago. The sensors record the four pollutant values from midnight to 6 AM during October 12nd, 2019 to October 18th, 2019. There are total 42 candidates (7 days with 6 hours per day) and 48 sensor nodes. The original dataset contains some NA values; we exclude those sensor. The top six periods without introducing any faults are presented in Figure 5a.

Based on the original dataset, we create 20 new datasets with the following two common and natural faulty behaviors:

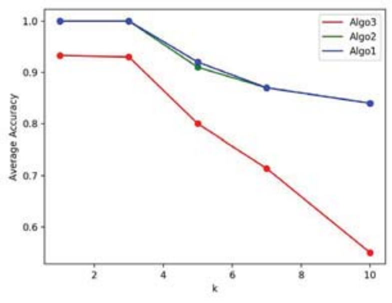
- *RANDOM*: Create 10 new data sets with random error.
  - Randomly pick five sensors to be faulty.
  - For each value of the faulty sensor, change the value to some random number.
- *LARGE*: Create 10 new data sets with large error.
  - Randomly pick five sensors to be faulty.
  - For each value of the faulty sensor, change the value to some large number, i.e., AQI = 1000.

Then, we run the proposed fault-tolerant algorithms on new datasets with synthetic faulty values, and calculate the average accuracy over the 10 datasets for each faulty behavior. The results are presented in figures in Figure 5. In the dataset, there are large number of negative values (wrong sensor readings); hence, the median is not a reasonable estimator of algorithm 3. For these datasets, we use mean instead of median.

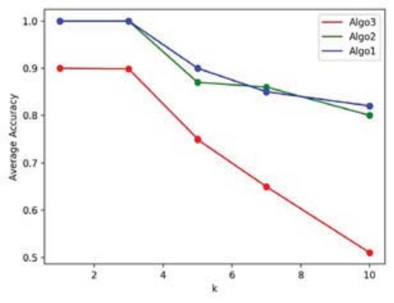
Overall, the proposed algorithms perform well when k is small. For example, when $k = 5$, all proposed three algorithms have accuracy at least 0.7. The reason is that after top-6 hours



(a) Top 10 Hours with Worst Air Quality in Our Dataset



(b) RANDOM faulty behavior



(c) LARGE faulty behavior

Fig. 5: Results with Array of Things dataset

with worst air quality, the quality is similar for the other hours. Therefore, it is reasonable that algorithms do not perform well under those scenarios.

VI. Conclusion

This paper considers distributed top-k algorithms for big data IoT networks. Our evaluation results show that the proposed algorithms are quite promising, and interesting future work include consideration of other fault-tolerant techniques and more comprehensive evaluation with different distribution and faulty behaviors.

References

[1] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, October 2008.

[2] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[3] William Kokou Dedzoe, Philippe Lamarre, Reza Akbarinia, and Patrick Valduriez. *As-Soon-As-Possible Top-k Query Processing in P2P Systems*, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[4] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Klee: A framework for distributed top-k query algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 637–648. VLDB Endowment, 2005.

[5] Zeinalipour-Yazti and et. al. The threshold join algorithm for top-k queries in distributed sensor networks. In *Proceedings of the 2Nd International Workshop on Data Management for Sensor Networks*, DMSN '05, pages 61–66, New York, NY, USA, 2005. ACM.

[6] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings 18th International Conference on Data Engineering*, pages 369–380, 2002.

[7] Moayad Aloqaily and et. al. Data and service management in densely crowded environments: Challenges, opportunities, and recent developments. *IEEE Communications Magazine*, 57(4):81–87, 2019.

[8] Haythem Bany Salameh and et. al. Secure routing in multi-hop iot-based cognitive radio networks under jamming attacks. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 323–327, 2019.

[9] Ismaeel Al Ridhawi and et. al. A profitable and energy-efficient cooperative fog solution for iot services. *IEEE Transactions on Industrial Informatics*, 2019.

[10] Venkatraman Balasubramanian and et. al. A mobility management architecture for seamless delivery of 5g-iot services. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[11] Awais Ahmad, Sadia Din, Anand Paul, Gwanggil Jeon, Moayad Aloqaily, and Mudassar Ahmad. Real-time route planning and data dissemination for urban scenarios using the internet of things. *IEEE Wireless Communications*, 26(6):50–55, 2019.

[12] Saniya Zafar and et. al. Qos enhancement with deep learning-based interference prediction in mobile iot. *Computer Communications*, 148:86–97, 2019.

[13] L. Tseng, Y. Wu, H. Pan, M. Aloqaily, and A. Boukerche. Reliable broadcast in networks with trusted nodes. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2019.

[14] Safa Otoum, Burak Kantarci, and Hussein Mouftah. Empowering reinforcement learning on big sensed data for intrusion detection. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[15] Boaz Patt-Shamir and Allon Shafrir. Approximate top-k queries in sensor networks. In *Proceedings of the 13th International Conference on Structural Information and Communication Complexity*, SIROCCO'06, pages 319–333, Berlin, Heidelberg, 2006. Springer-Verlag.

[16] Suman Nath and et. al. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 250–262, New York, NY, USA, 2004. ACM.

[17] Boaz Patt-Shamir and Allon Shafrir. Approximate distributed top-k queries. *Distributed Computing*, 21(1):1–22, Jun 2008.

[18] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 648–659, 2004.

[19] X. Y. Li, Y. Wang, and Y. Wang. Complexity of data collection, aggregation, and selection for wireless sensor networks. *IEEE Transactions on Computers*, 60(3):386–399, March 2011.

[20] Samuel Madden and et. al. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.

[21] V. Deolalikar and K. Eshghi. Lightweight approximate top-k for distributed settings. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 835–844, Oct 2014.

[22] T. Anagnostopoulos, A. Zaslavsy, A. Medvedev, and S. Khoruzhnicov. Top – k query based dynamic scheduling for iot-enabled smart city waste collection. In *2015 16th IEEE International Conference on Mobile Data Management*, volume 2, pages 50–55, June 2015.

[23] Chuan-Chi Lai, Tien-Chun Wang, Chuan-Ming Liu, and Li-Chun Wang. Probabilistic top-k dominating query monitoring over multiple uncertain iot data streams in edge computing environments, 06 2019.

[24] Keith Henderson and Tina Eliassi-Rad. Solving the top-k problem with fixed-memory heuristic search. 2009.

[25] Sakina Mahboubi, Reza Akbarinia, and Patrick Valduriez. Top-k query processing over distributed sensitive data. In *Proceedings of the 22Nd International Database Engineering & Applications Symposium*, IDEAS 2018, pages 208–216, New York, NY, USA, 2018. ACM.

[26] Ben Chen, Zhijin Lv, Xiaohui Yu, and Yang Liu. Sliding window top-k monitoring over distributed data streams. *Data Science and Engineering*, 2(4):289–300, Dec 2017.

[27] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, January 2002.