Short Paper

Range Query Processing for Monitoring Applications over Untrustworthy Clouds

Hoang Van Tran
Univ Rennes 1, IRISA, Lannion, France

Tristan Allard
Univ Rennes 1, IRISA, Rennes, France

Laurent D'Orazio
Univ Rennes 1, IRISA, Lannion, France

Amr El Abbadi
UC Santa Barbara, California, USA

ABSTRACT

Privacy is a major concern in cloud computing since clouds are considered as untrusted environments. In this study, we address the problem of privacy-preserving range query processing on clouds. Several solutions have been proposed in this line of work, however, they become inefficient or impractical for many monitoring applications, including real-time monitoring and predicting the spatial spread of seasonal epidemics (e.g., H1N1 influenza). In this case, a system often confronts a high rate of incoming data. Prior schemes may thus suffer from potential performance issues, e.g., overload or bottleneck. In this paper, we introduce an extension of PINED-RQ to address these limitations. We also demonstrate experimentally that our solution outperforms PINED-RQ.

1 INTRODUCTION

Reducing the impact of seasonal epidemics (e.g., H1N1 influenza) is demanding for public health officials. Early detection of spatial spread of the epidemics could help alleviate severe consequences. It is thus important to track and predict the spread of such diseases in the population. To do that, an individual can interact with a website or mobile application to report personal data (e.g., age, phone, sex, symptoms, travel plan, social network user name, ...), to be utilized for real-time predicting analyses. Systems usually run in very short periods, a few days or weeks after an epidemic emerges. Due to the need of significant computing capacity and the high speed of incoming data, it is desirable to use cloud services for managing and exploiting submitted data. However, cloud computing suffers from privacy issues, e.g., sensitive information can be exploited by cloud's administrators.

Encrypting outsourced data is a common solution to handle privacy issues in clouds. In this study, we focus on range queries over encrypted data since it is a fundamental operation. Over the last years, different approaches have attempted to strike a trade-off between security and practical efficiency [2, 10, 11]. Index-based schemes [5, 13, 15] have also been proposed to increase query performance while ensuring strong security. Nevertheless, prior schemes cannot cope with the high rate of incoming data that occurs in a wide range of monitoring applications, especially in the proposed context.

To solve the drawback of existing works, we propose a solution based on PINED-RQ [15] that enables the building of a secure index over sensitive data at a trusted component (hereinafter referred to as a collector). The collector then publishes the secure index and the encrypted data to the cloud for serving range queries. Our choice is motivated by the fact that PINED-RQ offers strong privacy protection while it has significantly faster range query processing and requires less storage space, compared to its counterparts [5, 13]. Nonetheless, PINED-RQ has to publish data in batches and partially processes data at the collector. Consequently, a bottleneck may occur as incoming data and query requests arrive at a high rate. Moreover, since PINED-RQ partially evaluates queries at the collector, which often has limited resources, they may also confront scalability problems. Publishing small batches may help PINED-RQ ease those potential problems, however, because it is built upon differential privacy [6, 8], small batches would cause large aggregation noise, destroying index's utilities.

Therefore, in order to adapt PINED-RQ to the targeted context, we aim to shift heavy workload from the collector to the cloud, that is able to provide on-demand capacity. In particular, instead of publishing data in batches, when a new tuple arrives, the collector immediately sends it to the cloud. The challenge to our approach is how to build PINED-RQ's index for the new tuples that are previously moved to the untrusted cloud.

In this paper, we propose PINED-RQ++, an extension of PINED-RQ, to mainly prevent potential bottlenecks at the collector while ensuring a secure index for new data. The key idea behind our prototype is to reverse the process of constructing PINED-RQ's index. This allows the sending of new data to the cloud as soon as possible without sacrificing privacy. The experimental results give promising performance, e.g., the publishing time of the NASA dataset (~0.5M tuples) [1] is reduced up to ~35x while maximum data rate at the collector experiences a reduction of up to ~2.7x. In particular, our solution eliminates query processing at the collector, making the system more scalable. The contributions of this paper are as follows.

(1) We introduce a notion of index template within PINED-RQ to support frequently published data.
(2) We propose a mechanism, PINED-RQ++, for updating the index template while still retaining privacy protection for frequently published data comparable to PINED-RQ.
(3) We also develop a parallel version of PINED-RQ++ to improve the throughput of the system.
(4) We implement (non-)parallel PINED-RQ++ to show the superiority of our solutions compared with PINED-RQ.

The paper is structured as follows. In Section 2, we briefly review background. We then introduce our solution in Section 3. In Section 4, we present our experimental results before giving conclusion and future work in Section 5.

2 BACKGROUND

2.1 Related Work

Various schemes have been developed to preserve privacy for processing range queries in clouds over the last years. Hidden vector encryption approaches [4, 16] use asymmetric cryptography to conceal data's attributes in an encrypted vector. These

methods incur prohibitive computation costs. Many bucketization schemes [9–11] have been proposed for range query processing in clouds. These solutions partition an attribute domain into a finite number of buckets. The range query retrieves all data falling within the range. However, bucketing approaches disclose data distribution and suffer from large aggregation false positives. Agrawal *et al.* [2] and Boldyreva *et al.* [3] present order-preserving encryption schemes that preserve the relative order of plain data under encryption. A downside of these schemes is that they leak the total order of plain data to the cloud. This is vulnerable to statistical attacks. Meanwhile, Li *et al.* [13] and Demertzis *et al.* [5] propose index-based strategies for answering range queries over outsourced data. Unfortunately, both suffer from prohibitive storage cost. On the other hand, Sahin *et al.* [15] present PINED-RQ for serving efficient range query processing in clouds via secure indexes. Nonetheless, the high rate of new data is not discussed in this work. Based on PINED-RQ, we develop our solution to tackle the limitations of the current works.

2.2 Differential Privacy

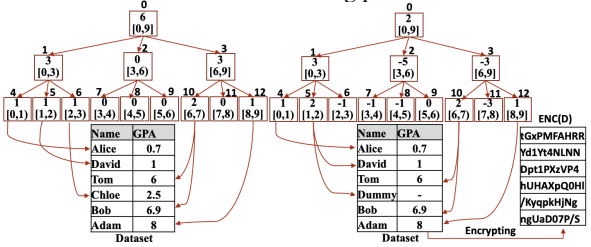
Definition 1 (ϵ-differential privacy [6, 8]): A randomised mechanism M satisfies ϵ-differential privacy, if for any set $O \in Range(M)$, and any datasets D and D' that differ in at most one tuple,

$$Pr[M(D) = O] \leq e^{\epsilon}.Pr[M(D') = O]$$

where ϵ represents the privacy level the mechanism offers.
Laplace Mechanism [7]: Let D and D' be two datasets such that D' is obtained from D by adding or removing one tuple. Let $Lap(\beta)$ be a random variable that has a Laplace distribution with the probability density function $pdf(x, \beta) = \frac{1}{2\beta}e^{-|x|/\beta}$. Let f be a real-valued function, the Laplace mechanism adds $Lap(max \parallel f(D) - f(D') \parallel_1 / \epsilon)$ to the output of f, where $\epsilon > 0$.
Theorem 1 (Sequential Composition [14]): Let $M_1, M_2, ..., M_r$ denote a set of mechanisms and each M_i gives ϵ_i-differential privacy. Let M be another mechanism executing $M_1(D), M_2(D), ..., M_r(D)$. Then, M satisfies $(\sum_{i=1}^{r} \epsilon_i)$-differential privacy.

2.3 PINED-RQ

We briefly describe PINED-RQ [15], which is built upon differential privacy [6]. We only focus on the insertion operation in this study. There are two main steps for building a PINED-RQ index.
(a) Building an index: Given a dataset at the collector, a PINED-RQ's index is constructed based on a B+Tree. In PINED-RQ, the set of all nodes is defined as a histogram covering the domain of an indexed attribute. For example, the students' GPA is used to build histograms (see Figure 1a). Each leaf node has a count representing the number of tuples falling within its interval. It also keeps pointers to those tuples. Likewise, the root and any internal node have a range and a count, combining the intervals and the counts of their children, respectively.
(b) Perturbing an index: All counts in the index are independently perturbed by Laplace noise [7]. The noise may be positive or negative, thus, after this step, the count of a node may increase or decrease, respectively. As shown in Figure 1b, the count of node 6 changes from 1 to -1 while the count of node 5 changes from 1 to 2. Such changes consequently lead to inconsistencies between a leaf node's noisy count and the number of pointers it holds. To address that issue, PINED-RQ adds dummy tuples (fake tuples) to the dataset as a leaf node receives positive noise. Otherwise, if a leaf node receives negative noise, real tuples are moved from the dataset to the corresponding overflow array. An overflow array [15] of a leaf node is a fixed-size array, which is randomly filled with dummy tuples at the publishing time to conceal removed tuples from the adversary. As illustrated in Figure 1b, the tuple (Chloe) belonging to node 6 is deleted from dataset while one dummy tuple is added and linked to node 5. Finally, the perturbed index is sent to the cloud along with the encrypted dataset.
Notably, PINED-RQ satisfies $(\epsilon, \delta)_n$-Probabilistic-SIM-CDP privacy model [15], a variant of differential privacy [8]. Intuitively, this variant results from the introduction of the encryption and the overflow arrays to the index building process.



(a) Clear index **(b) Secure index**

Figure 1: Example of PINED-RQ index

Clearly, an update directly to such published indexes would violate differential privacy [6, 8], thus, PINED-RQ cannot support live updates. Furthermore, PINED-RQ is also reluctant to publish very small datasets since the aggregation noise would destroy index's utilities. These properties make PINED-RQ impractical for high speed monitoring applications. In contrast, our proposal aims to send new data immediately to the cloud. Thus, one technical challenge is how to manage such dummy and removed tuples, which help protect the privacy of the index, as new data are stored at the cloud instead of at the collector.

3 PINED-RQ++

We focus on the architecture as depicted in Figure 2. Data generators produce raw data and send them to a collector. The incoming data are then pre-processed prior to being sent to a cloud. A consumer poses range queries to the cloud. In PINED-RQ++, we assume that the cloud is *honest-but-curious* while the other components are trusted. Thus, the adversary can use all information exchanged between the cloud and the other trusted components to deduce anything in a computationally-feasible way.

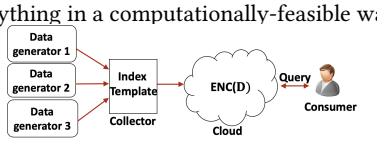


Figure 2: Proposed architecture

At the beginning, an index template is built at the collector. Whenever a new tuple arrives, the index template is updated with that tuple. Next, the tuple is encrypted and forwarded to the cloud. When the index template is published at a later time, the cloud associates it with unindexed data to produce a secure index as described in Section 2.3. The collector then initiates a new index template for future incoming data.
A query is only processed at the cloud which holds both indexed and unindexed data at time. As a result, as a consumer issues a query, it is first evaluated on indexed data (as in PINED-RQ's query processing [15]), the result of this evaluation and all the unindexed data are returned to the consumer. Finally, the consumer decrypts and filters the returned data for the final results.

3.1 Index Template

The process of building an index template is typically the same as in PINED-RQ (see Section 2.3). However, since initially there are

no data, its count variables only contain Laplace noise [7] and its leaves have no pointers. Such count variables and pointers are updated during a publishing time interval, which is defined as the period from when an index template is initiated to when it is published. This poses several challenges to our approach, for instance, how to publish dummy tuples generated during the index template building process or how to ensure that pointers between leaves and the new data do not leak privacy. Section 3.2, and 3.3 discuss these challenges as well as possible solutions.

3.2 Matching Table

Recall that when an index template is published, the cloud associates it with unindexed data to form a PINED-RQ's index for those data. To prepare for this association, the collector needs to keep the pointers between unindexed data and leaves. To do that, a simple way is to mark the ciphertext of a new tuple by the id of the leaf node to which the tuple belongs, and send the marked ciphertext to the cloud. Later, the cloud can rebuild pointers from marked ciphertexts when the index template is published. However, these marked ciphertexts reveal the real pointers between unindexed data and leaves during a time interval. PINED-RQ++ consequently discloses more extra information, e.g., the actual distribution of the incoming time of real data, when compared to PINED-RQ.

To prevent the leakage of such information, we use unique random numbers which are viewed as temporary ids of tuples and a matching table (see Figure 3). The first column in this matching table stores leaves' id while each row of the second column holds the temporary id of tuples belonging to the corresponding leaf node. For instance, tuples 1 and 7 belong to node 6 while tuple 5 belongs to node 9. In particular, when a tuple arrives, the collector encrypts it, generates a unique random number, and sends the *<random number, ciphertext>* pair to the cloud. This number is stored in the corresponding row in the matching table at the collector. The randomness guarantees that no useful information about the index template is leaked to the adversary. When an index template and its matching table are published, the cloud simply loops over the matching table and replaces random numbers with the leaves' pointers.

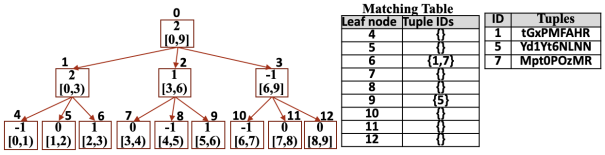


Figure 3: Perturbed index template and matching table

3.3 Noise Management

One challenge to our approach is that the collector initiates and perturbs the index template without any existing data. This means that no tuples are available to be deleted in case of negative noise. Also, when leaves receive positive noise, the collector can initially generate dummy tuples. However, when those dummy tuples are published to ensure privacy protection must be considered. To this end, we present two approaches as follows.

Regarding positive noise, the collector can immediately generate and send dummy tuples to the cloud at any random time point within a time interval. However, the arrival of all dummy tuples at the same time could be unsafe since the distribution of arrivals may be exploited by an adversary. Instead, we randomly release dummy tuples over the time interval. For instance, given 10 dummy tuples and a time interval of 100ms, then 10 discrete points can be randomly chosen between 1 and 100. At each chosen time point, a dummy tuple will be published along with a unique random number. It is, however, true that the privacy would be leaked as a dummy tuple might arrive at a chosen time at which real tuples are improbable. To avoid that case, the collector sends dummy data according to the actual distribution of the sending time of the real tuples. With this approach, when a *<random number, ciphertext>* pair comes to the cloud, the adversary cannot distinguish which pair is dummy or real data.

For negative noise, if a leaf node initially receives negative noise c, the collector moves the first c tuples (when they arrive) of that leaf node to the corresponding overflow array. At publication time, the collector randomly fills all overflow arrays with dummy tuples and sends these overflow arrays to the cloud. Notably, the movement of tuples only occurs at the collector and the adversary does not know which nodes receive negative noise. Thus, the privacy of such movements is also preserved.

3.4 Index Template Update Management

The main goal is to guarantee that the counts of PINED-RQ++'s index template are the same as those of PINED-RQ's index when the index template is published. In PINED-RQ, a leaf node's count represents the number of tuples falling within its interval. The count of internal nodes and the root is a summation of their children's counts. All counts are then perturbed by noise. In contrast, an index template only contains noise at first and it will increase its counts as soon as a tuple arrives at the collector.

Basically, when a tuple arrives at the collector, the leaf node to which the tuple belongs is determined. Then, the count of that leaf node and all its ancestors will be increased by 1. As shown in Figure 4, as the new tuple <Madison, 3> has GPA lying within the interval of node 7, the count of node 7 and all its ancestors (node 2 and node 0) are increased to 1, 2 and 3, respectively.

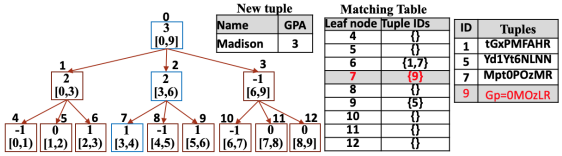


Figure 4: Index template and matching table are updated after the arrival of a new tuple

3.5 Parallel PINED-RQ++

Since the collector, that is assumed to be a small private node with a few cores, updates the index template, its throughput would be impacted as the rate of incoming tuples increases. We therefore parallelize the construction of the index template to improve the collector's throughput. Instead of keeping only one index template for updating, we create many clones of the original, each of which is independently updated. As a result, incoming tuples are equally distributed to clones for local updating. At publish time, all clones are merged together and sent to the cloud.

3.6 Privacy Analysis

As compared to PINED-RQ, both PINED-RQ++ and its parallel version only leak extra information of *<random number, ciphertext>* pairs and the time, when such pairs arrive at the cloud, to the adversary. Random numbers will not disclose any information about the data. Besides, as discussed in Section 3.3, when a pair arrives at the cloud, an adversary cannot distinguish between a dummy and a real tuple. Thus, PINED-RQ++'s privacy protection is similar to that of PINED-RQ.

4 EVALUATION RESULTS

4.1 Benchmark Environment

We ran our experiments on a cluster, whose configuration is illustrated in Table 1.

Table 1: Experimental environment

Component	CPU (2.4 GHz)	Memory (GB)	Disk (GB)
Collector	12	16	20
Cloud	16	16	40
Data generator	4	8	80
Consumer	4	16	10

We evaluate our proposal on four metrics namely network traffic, the time needed to publish an index (template), time response latency, and throughput. We use two real datasets NASA log [1] (1569898 tuples, five attributes) and Gowalla [12] (6442892 tuples, three attributes) for our experiments. We use the reply byte and check-in time as indexed attributes, respectively. Based on the values in the datasets, the reply byte's domain is divided into 350 bins while that of check-in time into 1502 bins. The fanout is set to 16. We use a time interval of 1 minute.

4.2 Results

(a) Network traffic: The network traffic metric gives an idea of the stability of the overall system, which is crucial for analytical processing. In this scenario, the data generator sends 3K tuples/second. Network traffic in terms of data rate is monitored at the collector over ten minutes. Figure 5 shows that the network traffic in PINED-RQ++ is much more stable than that in PINED-RQ. The maximum data rate is reduced by up to ~2.7x (NASA) and ~2.5x (Gowalla) in PINED-RQ++.

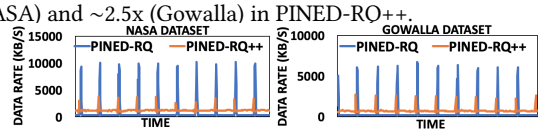


Figure 5: Network traffic over a ten minutes period

(b) Publishing time: We compare the time required to publish an index (template) according to dataset size. This metric is essential for monitoring applications since a long delay may cause bottlenecks at the collector. Different sizes of datasets are obtained by adjusting the incoming data speed and time interval parameter. As shown in Figure 6, when the dataset size increases, the time gradually rises in PINED-RQ while the publishing time in PINED-RQ++ remains almost unchanged. In particular, the publishing time is reduced by up to ~35x for NASA (514972 tuples) and ~16x reduction for Gowalla (1116907 tuples). Notably, when the dataset size rises, the gap between the two prototypes goes up.

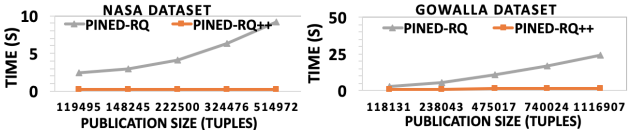


Figure 6: Average publishing time of 10 datasets

(c) Response time latency: We turn our attention to query latency. The data generator produces 2K tuples/second. The consumer sends one query per second. The query range is randomly chosen between 25% and 100%. In Figure 7, the results indicate that PINED-RQ has lower latency for small ranges compared to PINED-RQ++. However, our approach performs slightly better with large ranges (100%) because PINED-RQ's collector experiences higher workload for processing consumers' queries.

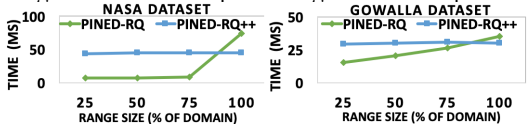


Figure 7: Average response time latency of 1000 queries

(d) Throughput: We compare the parallel PINED-RQ++'s throughput with the non-parallel version. The incoming data rates are chosen to be higher than the maximum throughput of the non-parallel version, 6.5K and 18K tuples/second for NASA and Gowalla, respectively. The different data rates are chosen since the tuple size of NASA is larger than that of Gowalla. The parallel version always has better throughput than the non-parallel version. The results in Figure 8 show that when the number of clones increases, the throughput is improved. The two-clone setting gives the best throughput, increasing by about 18% (to ~3K tuples/second) for NASA and about 47% (to ~8.3K tuples/second) for Gowalla when compared to the non-parallel version. Using a larger number of clones is not meaningful due to the merging process's costs.

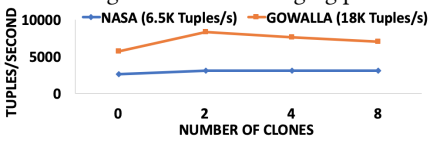


Figure 8: Parallel PINED-RQ++

5 CONCLUSION

We developed PINED-RQ++ to address the challenges of high rates of incoming data for processing range queries in clouds and the scalability problems of the prior schemes. The experimental results show that our solution provides better performance than PINED-RQ while privacy is also protected. This proves that PINED-RQ++ is appropriate for real-world monitoring applications. Besides, we introduce the parallel version, that helps enhance throughput.

Future work includes improving query performance, caching techniques and a dynamic adaptation to the query workload.

REFERENCES

[1] 1996. NASA Log. (1996). http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order Preserving Encryption for Numeric Data. In *SIGMOD*. 563–574.

[3] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In *CRYPTO*. 578–595.

[4] Dan Boneh and Brent Waters. 2007. Conjunctive, Subset, and Range Queries on Encrypted Data. In *TCC*. 535–554.

[5] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. 2016. Practical Private Range Search Revisited. In *SIGMOD*. 185–198.

[6] Cynthia Dwork. 2006. Differential Privacy. In *ICALP*. 1–12.

[7] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*. 265–284.

[8] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (2014), 211–407.

[9] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad . 2002. Executing SQL over Encrypted Data in the Database-service-provider Model. In *SIGMOD*. 216–227.

[10] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure Multidimensional Range Queries over Outsourced Data. In *VLDB*. 333–358.

[11] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. 2004. A Privacy-preserving Index for Range Queries. In *VLDB*. 720–731.

[12] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).

[13] Rui Li, Alex X. Liu, Ann L. Wang, and Bezawada Bruhadeshwar. 2014. Fast Range Query Processing with Strong Privacy Protection for Cloud Computing. *In VLDB* (2014), 1953–1964.

[14] Frank D. McSherry. 2009. Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis. In *SIGMOD*. 19–30.

[15] Cetin Sahin, Tristan Allard, Reza Akbarina, Amr El Abbadi, and Esther Pacitti. 2018. A Differentially Private Index for Range Query Processing in Clouds. In *ICDE*. 857–868.

[16] M. Wen, R. Lu, K. Zhang, J. Lei, X. Liang, and X. Shen. 2013. PaRQ: A Privacy-Preserving Range Query Scheme Over Encrypted Metering Data for Smart Grid. *IEEE Transactions on Emerging Topics in Computing* 1, 1 (June 2013), 178–191.