 CMC, vol.58, no.1, pp.61-78, 2019

Differentially Private Real-Time Streaming Data Publication Based on Sliding Window Under Exponential Decay

Lan Sun[1], Chen Ge[1], Xin Huang[1], Yingjie Wu[1,*] and Yan Gao[2]

Abstract: Continuous response of range query on steaming data provides useful information for many practical applications as well as the risk of privacy disclosure. The existing research on differential privacy streaming data publication mostly pay close attention to boosting query accuracy, but pay less attention to query efficiency, and ignore the effect of timeliness on data weight. In this paper, we propose an effective algorithm of differential privacy streaming data publication under exponential decay mode. Firstly, by introducing the Fenwick tree to divide and reorganize data items in the stream, we achieve a constant time complexity for inserting a new item and getting the prefix sum. Meanwhile, we achieve time complicity linear to the number of data item for building a tree. After that, we use the advantage of matrix mechanism to deal with relevant queries and reduce the global sensitivity. In addition, we choose proper diagonal matrix further improve the range query accuracy. Finally, considering about exponential decay, every data item is weighted by the decay factor. By putting the Fenwick tree and matrix optimization together, we present complete algorithm for differentiate private real-time streaming data publication. The experiment is designed to compare the algorithm in this paper with similar algorithms for streaming data release in exponential decay. Experimental results show that the algorithm in this paper effectively improve the query efficiency while ensuring the quality of the query.

Keywords: Differential privacy, streaming data publication, exponential decay, matrix mechanism, sliding window.

1 Introduction

Currently, the rapid development of big data and the Internet of things (IoT) makes data easier to be collected, and leads to a serious concern about information security [Yang and Soboroff (2015)]. Specially, many applications require continuous statistical release of streaming data, such as real-time statistics of the sales amount on shopping sites, real-time statistics of high frequency phrases for search engines. In these applications, the release data is the accumulated value of streaming data in a certain sense. Statistical publication of streaming data not only brings information to people's life, but also brings

[1] College of Mathematics and Computer Science, Fuzhou University, Xue Yuan Road, No. 2, Fuzhou, 350116, China.

[2] Amazon, 410 Terry Avenue North, Seattle, Washington, 98109, USA.

[*] Corresponding Author: Yingjie Wu. Email: yjwu@fzu.edu.cn.

the risk of privacy disclosure [Fung, Wang, Chen et al. (2010)]. Differential privacy [Dwork (2006); Zhou, Li and Tao (2009); Xiong, Zhu and Wang (2014); Zhang and Meng (2014)] is recognized as a robust privacy protection model. In view of the privacy protection of streaming data publication, there are many related studies based on differential privacy model. There is a lot of related research [Dwork, Naor, Pitassi et al. (2010); Chan, Shi and Song (2010); Cao, Xiao, Ghinita et al. (2013); Zhang and Meng (2016); Bolot, Fawaz, Muthukrishnan et al. (2013)] for privacy protection of streaming data release is based on differential privacy model.

Dwork et al. [Dwork, Naor, Pitassi et al. (2010)] put forward a study of differential privacy under continual observation. It continuously releases the count value of single stream data from start to current time by piece-wise counting. Chan et al. [Chan, Shi and Song (2010)] improved the query accuracy and algorithm efficiency by binary tree. Zhang et al. [Zhang and Meng (2016)] realized the publication of count values in each sliding window with partitioning-based method.

There are two problems in existing correlation studies: 1. All of the above studies assume that the data items at all times are of the same importance. But in practical applications, it tends to pay more attention to the statistical release of recent data and low attention to historical data because the statistical monitoring of recent events has a stronger relevance to its purpose. To solve this problem, a usually method is to make the data item with weight which is inversely proportional to the distance from the data to the current time. Under exponential decay mode, Cao et al. [Cao, Xiao, Ghinita et al. (2013)] proposed differential privacy streaming data publishing algorithm with interval tree structure which is failure to make full use of the correlation between queries in continuous statistical publishing to further improve the accuracy of data release. 2. Authors in Dwork et al. [Dwork, Naor, Pitassi et al. (2010); Chan, Shi and Song (2010); Cao, Xiao, Ghinita et al. (2013); Zhang and Meng (2016); Bolot, Fawaz, Muthukrishnan et al. (2013)] mainly focused on how to improve the query precision of streaming data. However, here is a higher demand for the query efficiency of data release in many practical applications.

In this paper, we present a differential privacy real-time release algorithm for streaming data in exponential decay mode. This algorithm effectively improves the query efficiency on the premise of guaranteeing the quality of the streaming data.

The main works of this paper are as follows:

(1) Point at continuous statistical release of streaming data under exponential decay mode, we propose an algorithm to improve efficiency of range query with Fenwick tree.

(2) Furthermore, matrix mechanism is used to explore the relevance between queries. Algorithms for strategy matrix construction and diagonal matrix construction are proposed according to the characteristics of the load matrix under exponential decay mode. The matrix optimization method is used to further improve the query accuracy and efficiency.

(3) We present complete algorithm of streaming data range query response under exponential decay. Experimental results show that the algorithm is effective and feasible.

The organizational structure of the rest of this paper is as follows: The second section gives a brief introduction of the relevant concepts. The third section describes our approach and provides theoretical analysis. Section 3.1 proposes a fast range query algorithm based on

Fenwick tree. How to introduce the matrix mechanism to improve the release accuracy is explained in Section 3.2. Experimental results and analyses are presented in Section 4, and Section 5 is the conclusion.

2 Background

2.1 Differential privacy

Differential privacy [Dwork (2006)] has gradually emerged as the standard notion of privacy in data analysis. Informally, an algorithm is differential private if it is insensitive to small changes in the input. The formal definition of "small changes" is as follows:

Definition 1 (Neighboring data sets): Given two data sets D and D', if they are differing on at most one element:

$$\left|(D'-D)\bigcup(D-D')\right|=1 \tag{1}$$

Then we call them a pair of neighboring data sets.

Definition 2 (ε-differential privacy): Algorithm A is ε-differential private if for any neighboring data sets D and D', and any subset of outputs S⊆Range(A), the following holds:

$$\Pr(A(D)\in O)\le e^{\varepsilon}\times\Pr(A(D')\in O) \tag{2}$$

where the probability is taken over the randomness of the A. The smaller the ε, the stronger the private protection is.

2.2 Range Query in Sliding Window under Exponential Decay

Definition 3 (Sliding window): Arrange items in a streaming data according to their time stamp. The sliding window, with a fixed length W, always keeps the newest W th items, while discarding old ones.

Definition 4 (Range query in sliding window): Range query is to accumulate all items in a continuous interval in sliding window. Formally, for a data stream $S = \{D_1, D_2, ..., D_n\}$, and the current time is t. The answer of the range query q $[l, r]$ (t-$W \le l \le r \le t$) is:

$$result\,(l_q, r_q)=\sum_{i=l_q}^{r_q} D_i \tag{3}$$

Definition 5 (Range query under exponential decay): Based on definition 4, we add the weight coefficient $e^{-\lambda x}$ to each data item, where $p=e^{-\lambda}$ is called the decay factor which is fixed beforehand, the variable x represents the distance between the item's time stamp and t. The above formula is adjusted to:

$$result\,(l_q, r_q)=\sum_{i=l_q}^{r_q} e^{-\lambda|t-i|}D_i \tag{4}$$

Exponential decay associates longer items with smaller impact on final result.

2.3 Matrix mechanism

Given a workload of linear range queries, the matrix mechanism [Li, Hay, Rastogi et al. (2010); Yuan, Zhang, Winslett et al. (2012)] uses an alternative set of queries, named the strategy, which are answered privately by a standard mechanism. Answers to the workload are then derived from the strategy queries, which bringing a higher accuracy.

Definition 6 (Matrix mechanism): Decompose the workload matrix W into two matrices B and L. We call L the strategy matrix, which actually making a basis transformation on original data. Then noise is added to the result of transformation, and the disturbing result is converted to the final query result through matrix B. The formula form is as follows:

$$A_L(W,X) = B(LX + Lap_N(\Delta L/\varepsilon)) \tag{5}$$

Where ΔL is the global sensitivity of L, defined as:

$$\Delta L = \max_{D,D'} \|L(D), L(D')\| \tag{6}$$

Derived from formula (5) and formula (6), the estimation error of matrix mechanism is:

$$error_L(W) = \frac{2}{\varepsilon^2} trace(B^T B)\Delta^2{}_L \tag{7}$$

3 Real-time private streaming data publication under exponential decay

3.1 Real-time range query based on Fenwick tree

As proposed in Chan et al. [Chan, Shi and Song (2010)], one can use interval tree as the data structure to arrange streaming data in the sliding window, which improves the time efficiency of data release. Specifically, full binary tree architecture was used in Chan et al. [Chan, Shi and Song (2010)], and its structure is shown in Fig. 1. Assume the size of sliding window is W, and t represents the current moment. As shown in Fig. 1, there are two binary trees that each has its part included in sliding window. Gray nodes were slid out of the window and will never be concerned again in later process, while striped ones are about to be included. Once first node of the new tree has been included, the whole binary tree will be built beforehand. Nodes in the new tree will be activated one after another in pace with their sliding into the window.

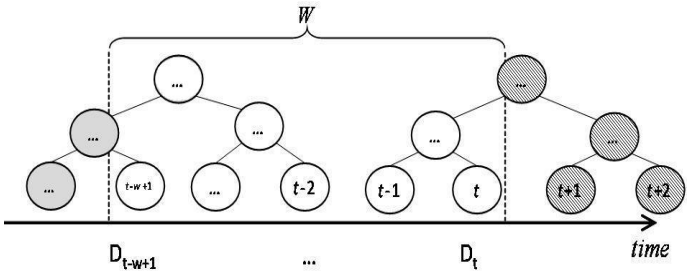

Figure 1: Building interval tree under sliding window

Using the full binary tree, we can answer any range query in the sliding window. However, as Fig. 2 suggested, all of the right sons (gray nodes) in the tree are unnecessary for this job, because we can obtain their value by their father nodes. By removing them, the number of remain nodes is equal to the size of sliding window, i.e. *W*. And we can still add every emerging data item in a constant time. After this simplification, the remained structure is called Fenwick tree [Fenwick (1994)].

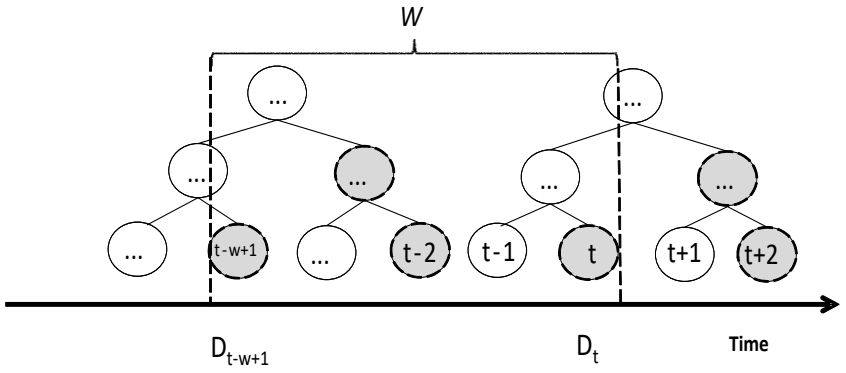

Figure 2: The gray nodes are redundant

Fenwick tree is such a data structure that supports query and modify operation with a $O(log_2(N))$ time complexity. For a given r, the prefix sum in range $[1, r]$ can be figure out quickly. We use $Sum(r)$ to represent the prefix sum, i.e. $Sum(r)=D_1+D_2+...+D_r$ (we neglect the exponential decay temporarily).

In the calculating process, Fenwick tree has generated intermediate variables $S_i(i \in [1,r])$, given by the following formula:

$$S_i = \sum_{j=i-lowbit\ (i)+1}^{i} D_j \ , i \in [1, r] \tag{8}$$

D_j is the j-th data item, and $lowbit(x)$ means the lowest "1" in the binary form of x. For example, when x=12, we have $(12)_{10}=(1100)_2$. The lowest "1" is in the third position, so $lowbit\ (12)=(0100)_2=(4)_{10}$. We can get $lowbit(x)$ easily by bitwise operation: $lowbit(x)=x\ \&\ (-x)$. Fenwick tree then use these intermediate variables S_i to get the prefix sum:

$$Sum(r) = \sum_{i=\lfloor \log_2(lowbit(r)) \rfloor}^{\lfloor \log_2(r) \rfloor} S_{r-r \bmod i} \tag{9}$$

When considering about exponential decay, every data item should be weighted by the decay factor p^x. So the prefix sum becomes:

$$Sum(r) = \sum_{j=1}^{r} p^{r-j} D_j \tag{10}$$

Correspondingly, formula (8) and formula (9) are also adjusted to formula (11) and formula (12) as follows:

$$S_i = \sum_{j=i-lowbit\ (i)+1}^{i} p^{i-j} D_j, i \in [1, r] \tag{11}$$

$$Sum(r) = \sum_{i=\lfloor \log_2(lowbit(r)) \rfloor}^{\lfloor \log_2(r) \rfloor} p^{r \bmod i} S_{r-r \bmod i} \tag{12}$$

Fig. 3 shows the structure of Fenwick tree under exponential decay.

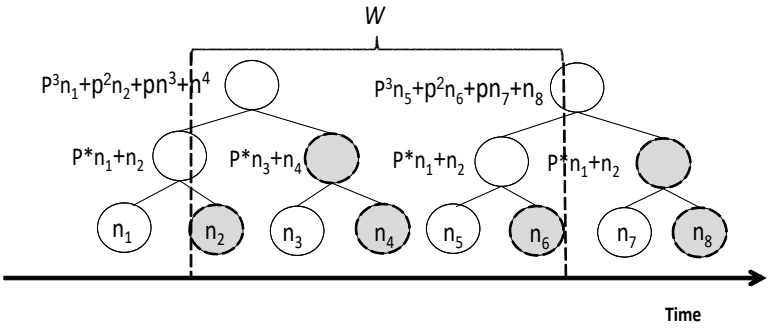

Figure 3: Fenwick tree under exponential decay

In Fig. 3, the size of sliding window, i.e. *W*, is 5, so we choose the tree's height to be 3. Nodes which have already slid out of the window will never be used again, so we can recycle them to save memory. And once after the latest node of old tree has been processed, the next data item will cause the building process of a new tree with preset height.

Then we give the detail about how the Fenwick tree works:

Firstly, we number activated nodes according to their time stamp. Then the white nodes in Fig. 4 form two Fenwick trees. One contains nodes ①~④, while nodes ⑤~⑦ stored in another. The value of node ① is the data item at time t_1, ②'s value is the sum of t_1 and t_2, and ④ is sum of [t_1, t_4]. (When considering about exponential decay, all the value mentioned above should be associated with their decay weight. For simplicity, we omit the statement about it. You can see Fig. 3 and algorithm 1 to get a better understanding).

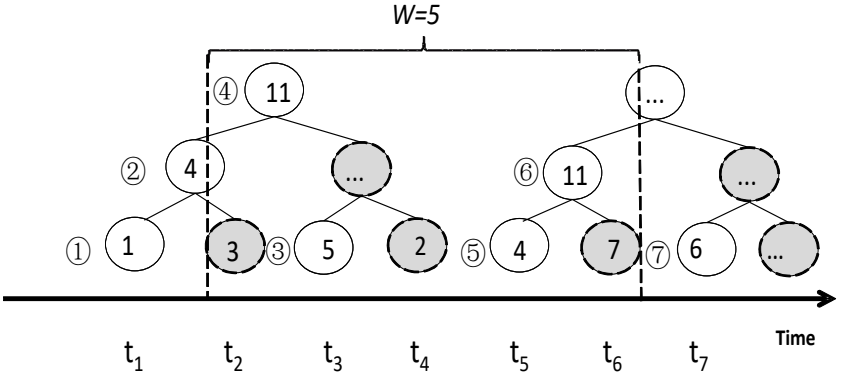

Figure 4: Details about Fenwick tree

Briefly, father's value is the sum of all its sons (We call node y is the father of node x, if $x + lowbit(x) = y$). And we just need one add operation for each node, because its father is unique. Let us take node ④ for example: It is only father of nodes ② and ③. So after calculating node ②, we can immediately add it to node ④, with $O(1)$ time cost. Therefore, we achieve time complicity linear to the number of data item for building a tree.

Then we can use such a tree structure to get the prefix sum. For example, $Sum(1)=t_1$ is the value stored in node ①, $Sum(2)=t_1+t_2$ is the value in node ②, and $Sum(3)=t_1+t_2+t_3$ is the sum of nodes ② and ③. As for $Sum(5)=t_1+...+t_5$, because it is split into two trees, we need to sum up nodes ④ and ⑤. We can further simplify this process. For example, since we has already get the prefix $Sum(6)$=⑥+④, we can get $Sum(7)$=⑦+⑥+④ by sum up $Sum(6)$ and node ⑦. By this way, we achieve $O(1)$ time complicity for inserting a single new node.

When W was given, the tree's height is then determined, written H. Based on formula (9), the global sensitivity is also H. Therefore, by adding Laplace noise with scale H/ε to each node in the tree, our algorithm satisfies ε-differential privacy.

In summary, the algorithm for inserting new data item into Fenwick tree is described in algorithm 1:

Algorithm 1: Insert new data item into Fenwick tree

Input： S: Fenwick tree, H: tree's height, ε: privacy budget, p: decay factor
x: data item of current moment

Output：Updated Fenwick tree

1. Updating values of current node and its father:
   $L=2^{H-1}$; id=(i-1) % L+1;
   $S[id]=S[id]+x$;
   if ($id + lowbit[id]<=L$)
      $S[id + lowbit[id]] = S[id + lowbit[id]] + S[id]*p^{lowbit[id]}$;
2. Add Laplace noise to responding node: $S[id]=S[id]+Lap(H/\varepsilon)$;
3. Calculate the prefix sum for current moment:
   $Sum[id]=Sum[id-lowbit(id)] * p^{lowbit[id]}+S[id]$;
4. Return the updated Fenwick tree;

3.2 Accuracy optimization using matrix mechanism

To answer a range query, we may need to combine some nodes' values generated in last section. For example, in Fig. 4, we have to answer the range query [2, 5] by [1, 4]+[4,5]-[1, 2]. It causes the accumulation of noise, and then the loss of accuracy. In Section 3.1, we have generated intermediate variables S_i by Fenwick tree. This process can be written as a form of a matrix timing a vector. When the size of tree is 7, the matrix and vector is as follows:

$$S = L \times D = \begin{pmatrix} 1 & & & & & & \\ p & 1 & & & & & \\ & & 1 & & & & \\ p^3 & p^2 & p & 1 & & & \\ & & & & 1 & & \\ & & & & p & 1 & \\ & & & & & & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \\ 5 \\ 2 \\ 4 \\ 7 \\ 6 \end{pmatrix} \tag{13}$$

Where L is the strategy matrix, D is the original data set, and S is a vector combined by all intermediate variables S_i. That is to say, we transform the original data set into the intermediate variables vector, perturb it using Laplace noise, and finally restore it for answering the queries. This process is corresponding to formula (11) and Fig. 3.

Correspondingly, when given a certain Fenwick tree, we can restore it to the prefix sums that we need, using formula (12). We follow the example above, giving the matrix form for this restoring process:

$$WD = B \times S = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & p & 1 & & & & \\ & & & 1 & & & \\ & & & p & 1 & & \\ & & & p^2 & & 1 & \\ & & & p^3 & & p & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 4 \\ 5 \\ 11 \\ 4 \\ 11 \\ 6 \end{pmatrix} \tag{14}$$

Where W is the load matrix. In our setting of continual range query response, its form is as follows:

$$W = \begin{pmatrix} 1 & 0 & 0 & \cdots \\ p & 1 & 0 & \cdots \\ p^2 & p & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix} \tag{15}$$

In conclusion, the prefix vector can be written as matrix form $WD=BLD$, where $W=BL$.

After changing the tree into a matrix form, our method is actually a specific decomposition strategy of matrix mechanism. The matrix mechanism, in general, is to design an optimal decomposition strategy to decompose the workload W to improve accuracy. However, we aim to design a sub-optimal decomposing way, whose error is slightly greater than the optimal one. But we can figure out it quickly following Section 3.1's result. According to Yuan et al. [Yuan, Zhang, Winslett et al. (2012)], the mean square error of our method is:

$$error_L(W) = \frac{2}{\varepsilon^2} trace(B^T B)\Delta_L^2 \tag{16}$$

Let N be the size of matrix, concluded from formula (8), the number of nonzero elements in each row of reduction matrix B is not greater than $log_2(N)$, and total number of nonzero elements in B is not larger than $N \cdot log_2(N)$. Since the only possible value of nonzero elements in B is 1, we have $trace(B^T B) \leq N \cdot log_2(N)$. By formula (2) to formula (7), the global sensitivity of L, i.e. ΔL, is equal to the tree's height $H = log_2(N)$. The total error of all N times query is $error_L(W) = O(N \cdot log_2^3(N))$. Averaging to each query, the mean error is $O(log_2^3(N))$,

According to Cai et al. [Cai, Wu and Wang (2016)], there exist a diagonal matrix Λ which changes $W = BL$ to $W = B\Lambda\Lambda^{-1}L$, and formula (15) is adjusted to:

$$error_L(W) = \frac{2}{\varepsilon^2} trace(\Lambda^{-1} B^T B \Lambda^{-1}) \Delta_{\Lambda L}^2 \tag{17}$$

By choosing a proper Λ, we can further improve the accuracy. Wu et al. [Wu, Ge, Zhang et al. (2017)] gives the technical details. We show it in Algorithm 2, The key factor δ_i is given by following formulas:

$$\delta_m = \frac{\sqrt[3]{err_m}}{\sqrt[3]{err_m} + \sqrt[3]{2^m \times p^2}} \tag{18}$$

$$err_m = \begin{cases} 1, & m = 1 \\ (\sqrt[3]{err_{m-1}} + \sqrt[3]{2^{m-1}})^3, & m > 1 \end{cases} \tag{19}$$

Under our setting of sliding window, since the window's size is fixed, we can consider this calculation as a preprocessing part. We store the diagonal values beforehand, and invoke them directly during the publishing, avoiding time waste.

Algorithm 2: GetLambda (calculate k-th element λ_k in the diagonal of Λ)
Input: Upper time limit T, index k, decay factor p
Output: λ_k

---

1. Initialize λ_k to 1, calculate key coefficients $\delta_1 \sim \delta_{log2(T)+1}$
2. $kt \leftarrow k$, $m \leftarrow \log_2(T)+1$, $div \leftarrow 2^{m-1}$;
3. while div! $= kt$
   if $kt < div$ then $\lambda_k \leftarrow \lambda_k \times \delta_t$;
   if $kt > div$ then $kt \leftarrow kt - div$;
   $div \leftarrow div/2, m \leftarrow m-1$;
   wend
4. $\lambda_k \leftarrow \lambda_k \times (1-\delta_t)/p$;
5. return λ_k

From above analysis, we come to the conclusion that: Though this accuracy optimize method is deduced based on matrix mechanism, there is no need to explicitly calculate the matrices. We can simply adjust algorithm 1 by adding the coefficients from the diagonal matrix. The improved Fenwick tree building algorithm is as follows:

Algorithm 3 (Improved) Insert new data item in Fenwick tree

Input：Fenwick tree *S*, Tree's height *H*, privacy budget ε, decay factor *p* and data item of current moment *x*

Output：Updated Fenwick tree

1. Updating values of current node and its father:
   $L=2^{H-1}$; $id=(i-1)\ \%L+1$;
   $S[id]=S[id]+x$;
   if ($id+lowbit[id]<=L$)
      $S[id+lowbit[id]] = S[id+lowbit[id]] + S[id]*p^{lowbit[id]}$;
2. Add Laplace noise to responding node:
   $S[id]=S[id]+Lap(H/\varepsilon)$ / getLambda (L, id);
3. Calculate the prefix sum for current moment:
   $Sum[id]=Sum[id-lowbit(id)]*p^{lowbit[id]}+S[id]$;
4. Return the updated Fenwick tree;

Putting the Fenwick tree and matrix optimization together, we present our complete method in algorithm 4:

Algorithm 4: Real-time publishing algorithm for range query in sliding window under exponential decay (RTP_DMM)

Input：the origin data steam, the size of sliding window *W*

Output: the answer of each query

1. Initialize the height of tree according to *W*; Initialize the Fenwick tree *S* and the prefix sum array *Sum* (sized $L=2^{n-1}$); Call algorithm 2 to get the diagonal element.
2. Invoke algorithm 3 to insert new data item, delete the node in *Sum* which have already slide out, and recycle its memory.
3. Let *i* represent the current time-stamp
   if($i\%L==0$) jump to Step 4.
   else jump to Step 5.
4. Clear *S*, create a new array *Sum* while keeping the old one.
5. Use prefix sums in *Sum* to respond range queries. Jump to Step 2.

4 Experiments

We compare our method, named RTP_DMM, with two similar works in terms of efficiency and accuracy: The EX algorithm proposed in Zhang et al. [Zhang and Meng (2016)] using interval tree; and the LP algorithm proposed by Dwork, which directly calculate the weighted answers and perturb it. We set different privacy budget parameters

as 1.0, 0.1 and 0.01. To exclude randomness, each experiment was run 30 times to get the average.

4.1 Data sets and environment

We use the Search Logs and NetTrace in Hay et al. [Hay, Rastogi, Miklau et al. (2010)], along with WorldCup98 in Kellaris et al. [Kellaris, Papadopoulos, Xiao et al. (2014)] as our testing data sets. Search Logs collects the number of searches for the keyword "Obama" from 2004.01 to 2009.08. The NetTrace data sets contain the number of packet requests to a IP segment during a specific period. WorldCup98 records the visits to the World Cup official website during 1998.04 to 1998.07. Their scales are shown in Tab. 1.

Table 1: Scales of data sets

Data Set	Search Logs	NetTrace	WorldCup98
Size	35768	65536	7518579

We use the mean square error to measure the query accuracy of the published data, which is shown as follows:

$$error(Q) = \frac{\sum_{q \in Q} (q(D) - q(D'))^2}{|Q|} \tag{20}$$

Where $|Q|$ is the number of queries, D is the original data set, D' is the distributed data set and q represents a query.

The experimental environment is: Intel Core i5 4570 3.2 GHz, with 8 GB memory, and we use C++ under Windows 7.

4.2 Comparison and analysis of query efficiency

4.2.1 The effect of different query numbers on efficiency

In this experiment, we set different query numbers at each moment to compare the query efficiency of three algorithms. Because the query efficiency on small data set does not change significantly, the experiment only uses WorldCup98, and the size of the query range is set to 32768. The size of sliding window is fixed to 65536 and the decay factor p is 0.9995.

Different query time of different algorithm is shown in Fig. 5. As the number of queries increases, difference between several algorithms in the running time becomes more obvious. This is because when the number of queries is small, the main influence factor of the efficiency is the cost of model construction. When queries increase, the time spent in the query occupies the dominant position.

Compare with LP and RTP_DMM, the query time of EX increases most rapidly. This is because EX constructs an interval tree to achieve arbitrary range query within the sliding window. Although it reduces the number of nodes involved in a single query, it has to traverse the tree's height. So the time complexity for a single query is $O(log_2 W)$, which is log-linear to the size of the sliding window. So the query efficiency is low. RTP_DMM

uses the prefix sum to obtain an arbitrary range of query results within $O(1)$ time. LP also realizes constant time complexity. However, because of the large number of nodes involved in the calculation, its accuracy is terrible. Compared with LP, RTP_DMM needs more complex calculations to reduce the error which brings a larger constant. Therefore, as the number of queries increases, its running time is slightly larger than LP.

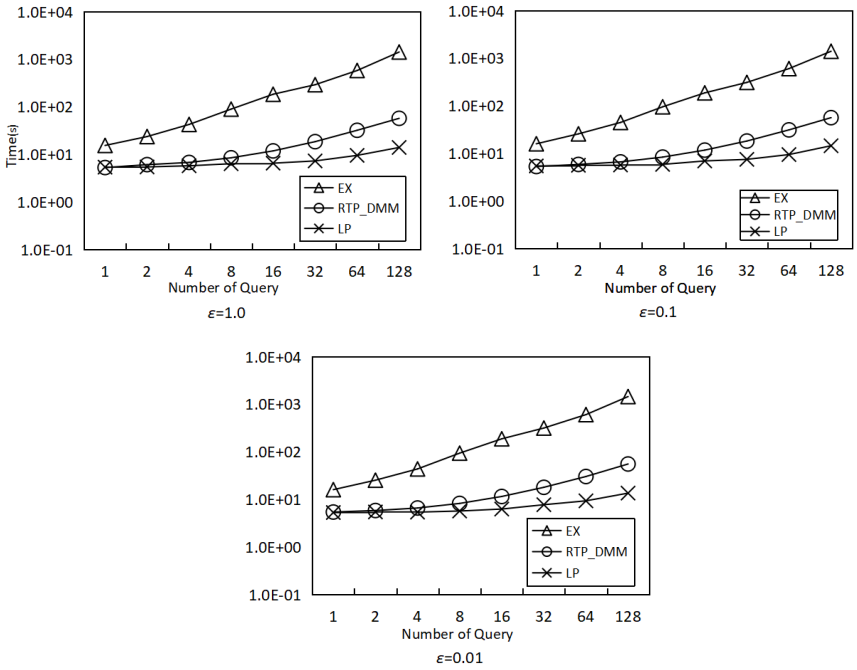

Figure 5: Efficiency at different query frequency

4.2.2 The effect of sliding window size on query efficiency

Then we set different sliding window sizes for further comparison. We still only use the WorldCup98 for the same reason. The size is set to 2^{15}, 2^{16}, ..., 2^{21}, respectively. Range size is set to be half of the window's size, so that this size increases with the window, and the query frequency is set to be once per time.

In Fig. 6, we observe that as the size of the sliding window increases, the RTP_DMM and LP are less affected than EX. The reason behind it is that the sliding window's size only affects RTP_DMM's space complexity and preprocessing time. For LP, the query efficiency is independent of the window size. However, as for EX, the time complexity is $O(log_2 W)$ which is log-linear to window size. So the curve of LP and RTP_DMM are below EX's. Both LP and RTP_DMM achieve $O(1)$ time complexity, so they have close performance in our experiment.

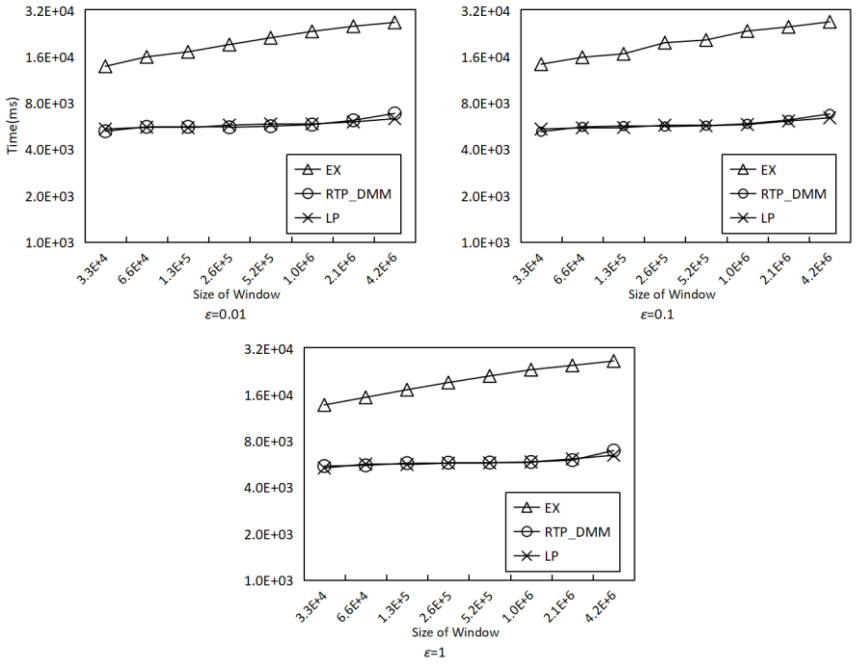

Figure 6: Efficiency under different sliding window sizes (WorldCup98)

4.3 Comparison of query accuracy

We use all three data sets to compare query accuracy between algorithms. Since the scale of Search Logs and NetTrace is small, the length of sliding window is set as the whole data set. As for WorldCup98, Setting the sliding window size to 65536 is suit for further comparison.

4.3.1 Compare accuracy for random range queries

We generate one range query of random size within the sliding window at each moment. Under exponential decay, small decay factor will make the decay speed too fast. So we fixed the decay factor p to 0.9995. The experimental comparison results are shown in Figs. 7-9.

It can be seen from Figs. 7-9 that RTP_DMM reaches significant higher accuracy than LP and EX. This is because RTP_DMM converts range queries into matrix representations, and applies diagonal matrix optimization to improve accuracy. For all three algorithms, the query error increases as the privacy budget decreases. It is because we need larger scale of Laplace noise with smaller privacy budget.

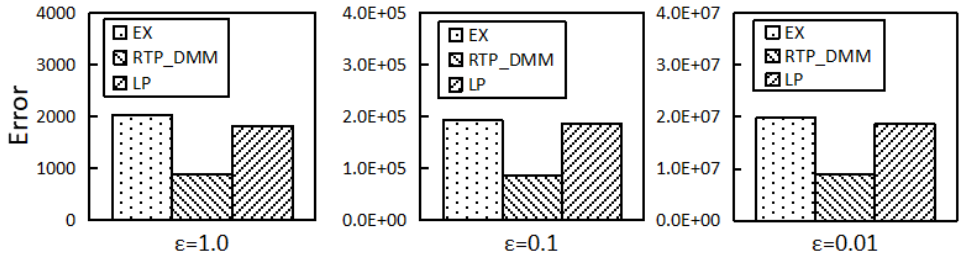

Figure 7: Comparison of query accuracy (Search Logs)

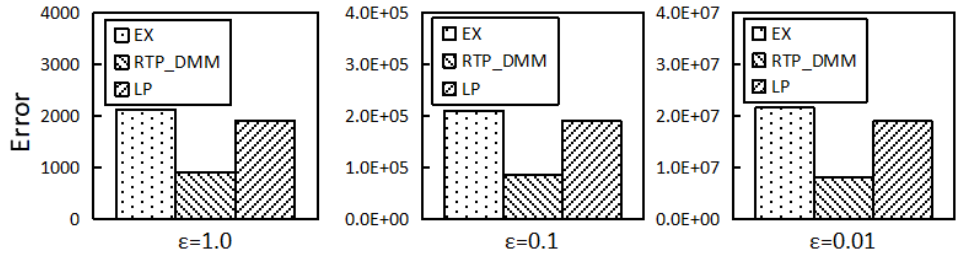

Figure 8: Comparison of query accuracy (NetTrace)

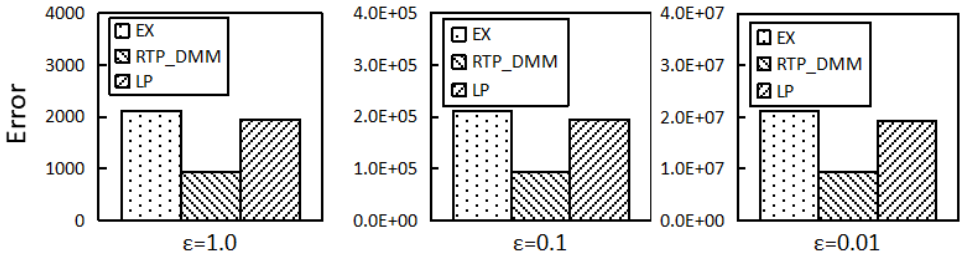

Figure 9: Comparison of query accuracy (WorldCup98)

4.3.2 Accuracy with different decay factors

In this experiment, we compare different decay factors to analyze their influence on the query error. We take the decay factors as 0.9991, 0.9992, ..., 0.9999 respectively.

As the comparison results in Figs. 10-12, the query error is positively correlated with the decay factor, this is because the increase of the decay factor will change the global sensitivity and thus affect the scale of added noise. The EX algorithm calculates the limit of noise scale according to the preset decay factor. Therefore, when the decay factor is close to 1, the average square error caused by the EX algorithm become large. As for the LP algorithm, when the decay factor is small, the weight of the long time nodes tends to be zero which leads to smaller error.

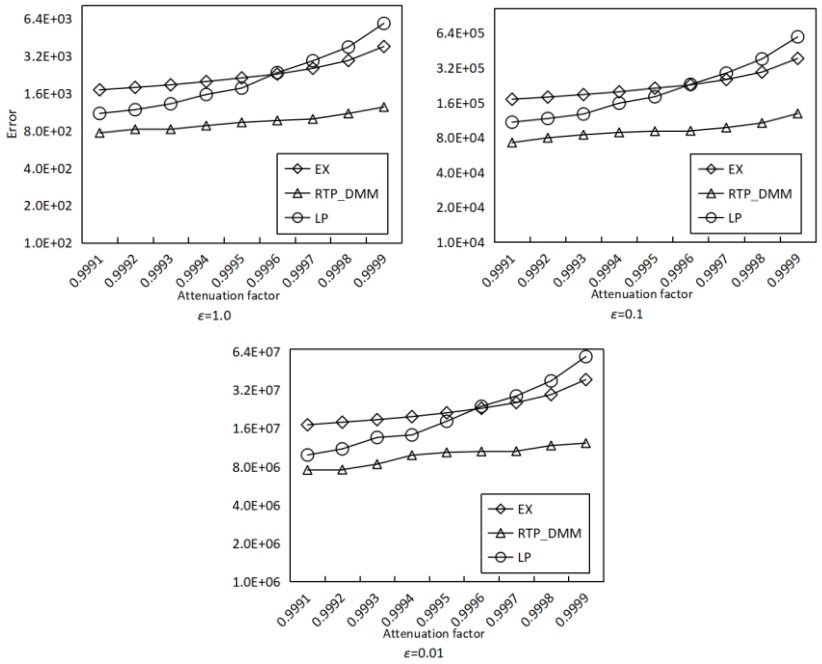

Figure 10: Accuracy with different decay factors (Search Logs)

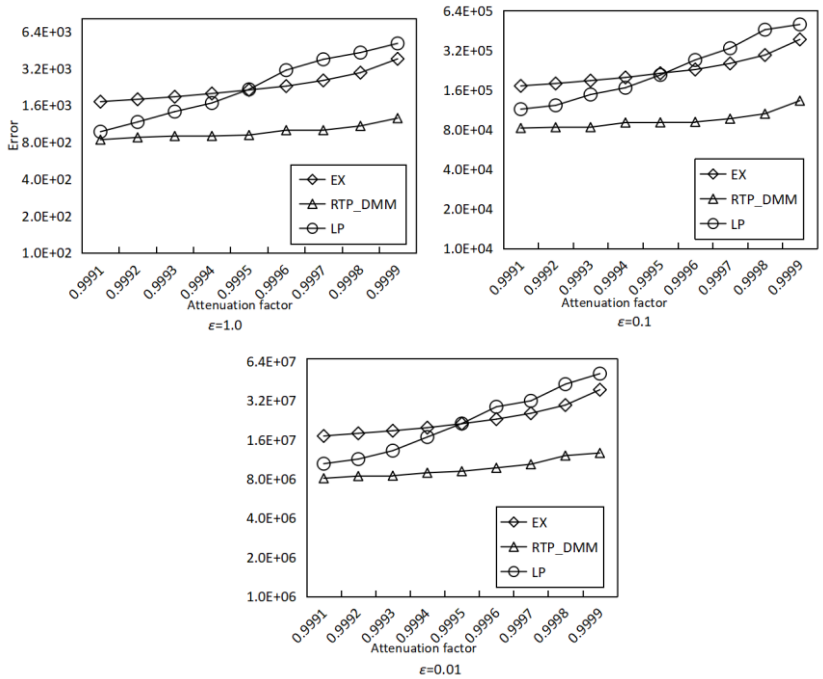

Figure 11: Accuracy with different decay factors (NetTrace)

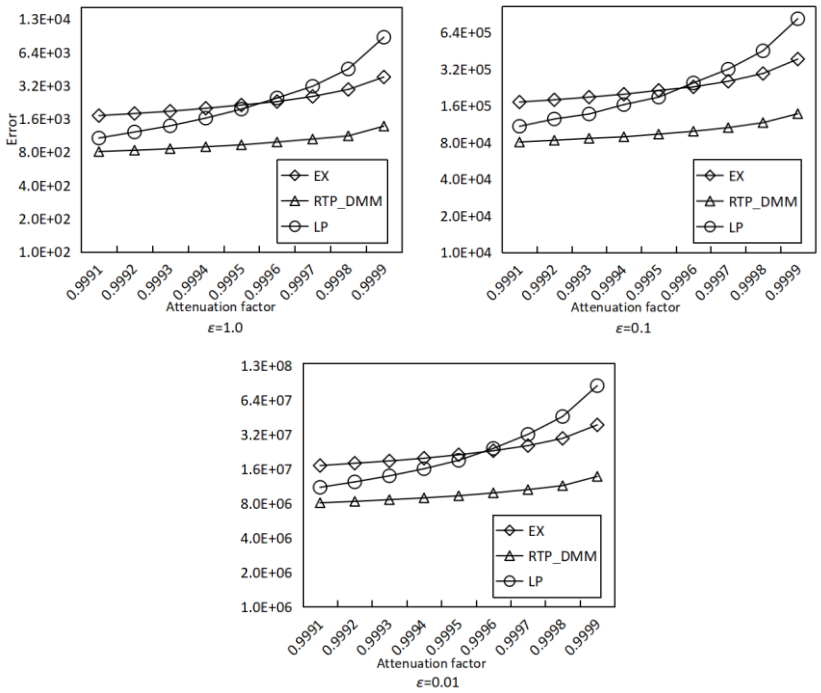

Figure 12: Accuracy with different decay factors (WorldCup98)

Based on the above experiments, it can be concluded that the algorithm RTP_DMM is scalable for different applications with various decay factors and privacy budget. It can reduce query error, and achieve real-time publishing under the sliding window.

5 Conclusion

In this paper, we propose an efficient method for real-time differential privacy streaming data publishing under exponential decay. It answers any range query within the sliding window in $O(1)$ time. We further convert the model to a matrix form, using matrix mechanism to optimize the accuracy. Comparison experiments with similar methods show that our RTP_MM guarantees the query accuracy while achieving higher time efficiency. In future studies, it is worth investigation to adapt our method to practical applications with other decay modes.

Acknowledgement: This work is supported, in part, by the National Natural Science Foundation of China under grant numbers 61300026; in part, by the Natural Science Foundation of Fujian Province under grant numbers 2017J01754 and 2018J01797.

References

Bolot, J.; Fawaz, N.; Muthukrishnan, S.; Nikolov, A.; Taft, N. (2013): Private decayed predicate sums on streams. *Proceedings of the 16th International Conference on Database Theory*, pp. 284-295.

Cai, J. P.; Wu, Y. J.; Wang, X. D. (2016): Method based on matrix mechanism for differential privacy continual data release. *Journal of Frontiers of Computer Science & Technology*, vol. 10, no. 4, pp. 481-494.

Cao, J.; Xiao, Q.; Ghinita, G.; Li, N.; Bertino, E. et al. (2013): Efficient and accurate strategies for differentially-private sliding window queries. *Proceedings of the 16th International Conference on EDBT/ICDT*, pp. 191-202.

Chan, T. H.; Shi, E.; Song, D. (2010): Private and continual release of statistics. *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*, vol. 14, no. 3, pp. 405-417.

Dwork, C. (2006): Differential privacy. *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, vol. 26, no. 2, pp. 1-12.

Dwork, C.; Naor, M.; Pitassi, T.; Rothblum, G. N. (2010): Differential privacy under continual observation. *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pp. 715-724.

Fenwick, P. M. (1994): A new data structure for cumulative frequency tables. *Software Practice & Experience*, vol. 24, no. 3, pp. 327-336.

Fung, B. C. M.; Wang, K.; Chen, R.; Yu, P. S. (2010): Privacy-preserving data publishing: a survey of recent developments. *ACM Computing Surveys*, vol. 42, no. 4, pp. 1-53.

Hay, M.; Rastogi, V.; Miklau, G.; Dan, S. (2010): Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the 36th International Conference on VLDB*, vol. 3, no. 1-2, pp. 1021-1032.

Kellaris, G.; Papadopoulos, S.; Xiao, X.; Papadias, D. (2014): Differentially private event sequences over infinite streams. *Proceedings of the 40th International Conference on VLDB*, vol. 7, no. 12, pp. 1155-1166.

Li, C.; Hay, M.; Rastogi, V.; Miklau, G.; Mcgregor, A. (2010): Optimizing linear counting queries under differential privacy. *Proceedings of the 29th ACM Sigmod-Sigact-Sigart Symposium on Principles of Database Systems*, vol. 24, pp. 123-134.

Wu, Y. J.; Ge, C.; Zhang, L. Q.; Sun, L. (2017): An algorithm for differential privacy streaming data publication based on matrix mechanism under exponential decay mode. *Scientia Sinica*, vol. 47, no. 11, pp. 1493-1509.

Xiong, P.; Zhu, T. Q.; Wang, X. F. (2014): A survey on differential privacy and applications. *Chinese Journal of Computers*, vol. 37, no. 1, pp. 101-122.

Yang, H.; Soboroff, L. (2015): Privacy-preserving IR 2015: When information retrieval meets privacy and security. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1157-1158.

Yuan, G.; Zhang, Z.; Winslett, M.; Yang, Y.; Yang, Y. et al. (2012): Low-rank mechanism: optimizing batch queries under differential privacy. *Proceedings of the 38th International Conference on VLDB*, vol. 5, no. 11, pp. 1352-1363.

Zhang, X. J.; Meng, X. F. (2014): Differential privacy in data publication and analysis. *Chinese Journal of Computers*, vol. 37, no. 4, pp. 927-949.

Zhang, X. J.; Meng, X. F. (2016): Stream histogram publication method with differential privacy. *Chinese Journal of Computers*, vol. 27, pp. 381-393.

Zhou, S. G.; Li, F.; Tao, Y. F. (2009): Privacy preservation in database applications: a survey. *Chinese Journal of Computers*, vol. 32, no. 5, pp. 847-861.