Differentially Private Stream Processing at Scale*

Bing Zhang
Google
zhangbing@google.com

Vadym Doroshenko†
Google
dvadym@google.com

Peter Kairouz†
Google Research
kairouz@google.com

Thomas Steinke†
Google DeepMind
steinke@google.com

Abhradeep Thakurta†
Google DeepMind
athakurta@google.com

Ziyin Ma
Google
ziyinma@google.com

Eidan Cohen
Google
eidanch@google.com

Himani Apte
Google
himaniapte@google.com

Jodi Spacek
Google
jodes@google.com

ABSTRACT

We design, to the best of our knowledge, the first differentially private (DP) stream aggregation processing system at scale. Our system – *Differential Privacy SQL Pipelines (DP-SQLP)* – is built using a streaming framework similar to Spark streaming, and is built on top of the Spanner database and the F1 query engine from Google.

Towards designing DP-SQLP we make both algorithmic and systemic advances, namely, we (i) design a novel (user-level) DP key selection algorithm that can operate on an unbounded set of possible keys, and can scale to one billion keys that users have contributed, (ii) design a preemptive execution scheme for DP key selection that avoids enumerating all the keys at each triggering time, and (iii) use algorithmic techniques from DP continual observation to release a continual DP histogram of user contributions to different keys over the stream length. We empirically demonstrate the efficacy by obtaining at least 16× reduction in error over meaningful baselines we consider. We implemented a streaming differentially private user impressions for Google Shopping with DP-SQLP. The streaming DP algorithms are further applied to Google Trends.

PVLDB Reference Format:
Bing Zhang, Vadym Doroshenko, Peter Kairouz, Thomas Steinke, Abhradeep Thakurta, Ziyin Ma, Eidan Cohen, Himani Apte, Jodi Spacek. Differentially Private Stream Processing at Scale. PVLDB, 17(12): 4145 - 4158, 2024.
doi:10.14778/3685800.3685833

1 INTRODUCTION

Analysis of streaming data with differential privacy (DP) [16] has been studied from the initial days of the field [10, 17], and this has been followed up in a sequence of works that include computing simple statistics [41], to machine learning applications (a.k.a. online learning) [1, 29, 30, 44, 53]. While all of these works focus on the abstract algorithmic design for various artifacts of streaming data processing, to the best of our knowledge, none of them focus on designing a scalable stream processing system. In this work, we primarily focus on designing a scalable DP stream processing system, called Differential Privacy SQL Pipelines (DP-SQLP), and make algorithmic advances along the way to cater to the scalability needs of it. DP-SQLP is implemented using a streaming framework similar to Spark streaming [52], and is built on top of the Spanner database [12] and F1 query engine [43] from Google. We also present production applications with two use cases in Section 6. The first is a real world use case that deploys DP-SQLP in *Google Shopping* to generate streaming page-view counts. The second applies the streaming DP algorithm to *Google Trends*.

In this paper we consider a data stream to be an unbounded sequence of tuples of the form (`key`, `value`, `timestamp`, `user_id`) that gets generated continuously in time. We also have a discrete set of times (a.k.a. *triggering times*) $\texttt{Tr} = [t_1^{\text{tr}}, t_2^{\text{tr}}, ..., t_T^{\text{tr}}]$. The objective is to output the sum of all the values for each of the keys at each time t_i^{tr}, while preserving (ε, δ)-DP [16] over the entire output stream with respect to all of the contributions with the same `user_id`. Although most prior research has extended simple one-shot DP algorithms to the streaming setting [9, 10, 17], designing a scalable DP-streaming system using off-the-shelf algorithms is challenging because of the following reasons[1]:

(1) **Unknown key space:** A data stream processing system can only process the data that has already arrived. For example, keys for a `GROUP BY` operation are not known in advance; instead we discover new keys as they arrive. To ensure (ε, δ)-DP one has to ensure the set of keys for which the statistics are computed is *stable* to change of an individual user's data. That is, we can only report statistics for a particular `key` when enough users have contributed to it; to ensure DP the threshold for reporting a key must be randomized.

*The full version of the paper can be found at https://arxiv.org/abs/2303.18086
†Contributed equally.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685833

[1] Our work is most closely related to [9]. We defer a full comparison to Section 1.2.

(2) **Synchronous execution:** The execution of the streaming system is driven by the data stream. That is, we must process the data as it arrives, and cannot run asynchronously at times when there is nothing to trigger execution. We refer to the times when our system runs as *triggering times*. Furthermore, typically, at each triggering time, only the keys that appeared since the last triggering time are processed. However, this is problematic for DP – if we only output a key when it has appeared in the most recent event time window, then this potentially leaks information. Naively, to avoid this, one has to process all the keys at each triggering time, which is computationally prohibitive.

(3) **Large number of observed keys and events:** A fundamental challenge is scalability. The system should be able to handle millions of updates per second from a data stream with billions of distinct keys.

(4) **Effective user contribution bounding:** In real applications, each person may contribute multiple records to the data stream at different times. Providing "event-level DP" (where a single action by a person/user is protected) does not provide sufficient privacy protection. In this work, we provide "user-level DP", where *all* the actions by a person/user is protected simultaneously. To provide user-level DP, one has to bound the contribution of each user, and that eventually introduces bias on the statistics that get computed. But contribution bounding controls the variance of the noise we must add to ensure DP. A natural (and unavoidable [4, 32, 37]) challenge is to decide on the level of contribution bounding to balance this bias and variance.

(5) **Streaming release of statistics:** One has to output statistics at every triggering time. If we treat each triggering time as an independent DP data release, then the privacy cost grows rapidly with the number of releases. Alternatively, to attain a fixed DP guarantee, the noise we add at each triggering time must grow polynomially with the number of triggering times. This is impractical when the number of triggering times is large. Thus the noise we add to ensure DP is not independent across triggering times. This helps in drastically reducing the total noise introduce for a fixed DP guarantee.

In our design of DP-SQLP we address these challenges, either by designing new algorithms, or by implementing existing specialized algorithms. This is our main contribution. To the best of our knowledge, *we provide the first at-scale differentially private stream aggregation processing system.*

Motivation for DP-SQLP: Data streams appear commonly in settings like Web logs, spatio-temporal GPS traces [5], mobile App usages [34], data generated by sensor networks and smart devices [36], live traffic in maps [50], cardiovascular monitoring [47], real-time content recommendation [42], and pandemic contact tracing [11]. Almost all of these applications touch sensitive user data on a continual basis. Calandrino et al. [7] demonstrated that continuous statistic release about individuals can act as a strong attack vector for detecting the presence/absence of a single user (in the context of collaborative recommendation systems). Hence, it is imperative to a streaming system to have rigorous privacy protections. In this work we adhere to differential privacy. For more discussion on the type of streams we consider, see a detailed survey in [28].

Our Contributions: As mentioned earlier, our main contribution is overcoming multiple challenges to build a distributed DP stream processing system that can handle large-scale industrial workloads.

- **Private key selection:** A priori, the set of possible keys is unbounded, so our system must identify a set of relevant keys to track. To protect privacy, we cannot identify a particular `key` based on the contributions of a single user. The streaming setting adds two additional complications: (a) The existence of each `key` is only known when it is observed in a data record, and (b) the privacy leakage due to continually releasing information about any particular key increases the DP cost due to composition [19]. To address these challenges, we design a novel algorithm (Algorithm 1) that couples "binary tree aggregation" [10, 17, 27] (a standard tool for continual release of DP statistics that only accumulates privacy cost that is poly-logarithmic in the number of aggregate releases from the data stream) with a thresholding scheme that allows one to only operate on keys that appear in at least $\mu > 0$ user records. To further minimize privacy leakage, we employ a variance reduced implementation of the binary tree aggregation protocol [27].
- **Preemptive execution:** Since we may track a large number of keys in production systems, it is not scalable to scan through all of the state keys each time the system is invoked. Thus we design a new algorithm (Algorithm 3) that only runs on the keys that have appeared between the current and previous triggering times. The idea is to *predict* when a key will be released in advance, rather than checking at each triggering time whether it should be released now. That is, whenever we observe a given `key`, we simulate checking the release condition for the rest of triggering times assuming no further updates to `key`. In the future we only check for `key` at any triggering time if either of the two conditions happen: (i) `key` appears in a fresh microbatch in the data stream, or (ii) the earlier simulation predicted a release for that time. By doing so, we reduce the expensive I/O and memory cost, with little CPU overhead. This idea is motivated by the caching of pages in the operating systems literature.
- **Empirical evaluation:** We provide a thorough empirical evaluation of our system. We consider a few natural baselines that adopt one-shot DP algorithms to stream data processing (e.g., repeated differential privacy query). At $(\varepsilon = 6, \delta = 10^{-9})$-DP, we observed up to 93.9% error reduction, and the number of retained keys is increased by 65 times when comparing DP-SQLP with baselines. Through our scalability experiments, we

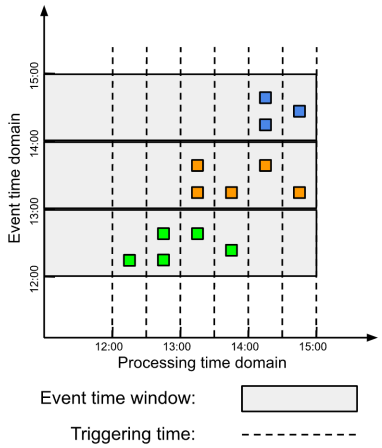

Figure 1: Event time domain and processing time domain

show that DP-SQLP can handle billions of keys without incurring any significant performance hit.

- **Industry application:** We present two industry use cases of streaming differential privacy. The DP-SQLP is applied to Google Shopping that processes a very large scale data stream under production environment, which demonstrates the scalability of our approach. A differentially private product page-view count is generated by DP-SQLP that can be used to signal the product information update. In the second use case, the streaming private key selection is applied to Google Trends for analyzing popular search queries with differential privacy guarantee.

In the following, we formally define the problem, and delve deeper into relevant related works.

1.1 Problem Statement

Let D be a data stream defined by an unbounded sequence of records, i.e., $D = [d_1, d_2, ...)$, where each record d_i is a tuple (`key`, `value`, `timestamp` t_i, `user_id`). A common query pattern in data analytics is the unknown domain histogram query. For example, consider a web service that logs user activities: each record is a URL click that contains (`URL`, `user_id`, `timestamp`). An analyst wants to know the number of clicks on each URL for each day up to today. An SQL query to generate this histogram is presented in Listing 1.

```
SELECT URL,
       TO_DATE(timestamp) AS date,
       COUNT(*) AS count
FROM web_logs
GROUP BY URL, TO_DATE(timestamp)
```

Listing 1: Single histogram query

In Listing 1, the keys are (`URL`, `date`) denoted by `key`[2], and the `count` is an aggregation column denoted by m.

[2]Throughout the paper, `key` and k are used interchangeably.

When querying a growing database or data stream, the above-mentioned query only shows a snapshot at certain date. In stream data processing, we use event-time window to determine which records in the *event time domain* to process, and triggers to define when in the *processing time domain* the results of groupings are emitted [2]. Let W denote the event-time windows, $W = [w_1, w_2...)$. Each event-time window is defined by a starting time and an end time $w_i = (t_{i,s}, t_{i,e})$. $D_{w_i} \in D$ contains all records that can be assigned to window w_i so that the timestamp t of each record satisfies $t_{i,s} \leq t < t_{i,e}$. Let Tr denote a set of triggering times in the processing domain, $\texttt{Tr} = [tr_1, tr_2...)$. We assume triggering time is predefined and independent to dataset[3]. The streaming system incrementally processes D_{w_i} at triggering time $\texttt{Tr}_{w_i} = \{tr_{i,s}, ..., tr_{i,e}\} \subset \texttt{Tr}$. Due to the time domain skew[2], $t_{i,s} \leq tr_{i,s}$ and $t_{i,e} \leq tr_{i,e} \leq t_{i,e} + t$, where t is the maximum delay that system allows for late arriving records. Our goal is to release the histogram for all sub-stream D_{w_i}, at every triggering time $tr_i \in \texttt{Tr}_{w_i}$, in a differentially private (DP) manner (See Appendix B in the full paper for a formal DP definition). For a pictorial representations of various timing concepts, see Figure 1.

Privacy implication of input driven stream: In terms of privacy, we want to ensure that the stream processing system ensures (ε, δ)-user level DP [15, 16, 19] over the complete stream. Since we are operating under the constraint that the data stream is an input driven stream (Definition A.1), the timings of the system (e.g., event time (Definition A.2), processing time (Definition A.3), and triggering time (Definition A.5)) can only be defined w.r.t. times at which the inputs have appeared. Thus it forces us to define the DP semantics which considers the *triggering times to be fixed* across neighboring data sets (in the context of traditional DP semantics). For a given user, what we protect via DP is the actual data that is contributed to the data stream. We provide a formalism in Appendix B in the full paper.

1.2 Related Work

Stream processing has been an active research field for more than 20 years [24]. It is now considered to be a mature technology with various streaming frameworks deployed at scale in industry, including Spark Streaming [52], Apache Beam [2] and Apache Flink [8]. However, none of these systems offer differentially private streaming queries.

Our work builds on a long line of DP research that focuses on extending one shot applications of DP mechanisms to the continual observation (streaming) setting for both analytics and learning applications [9, 10, 13, 17, 22, 25, 27, 30, 35]. These mechanisms crucially leverage the tree aggregation protocol [10, 17, 27] or variants of it based on the matrix factorization mechanism [13, 25, 35]. All these approaches have the advantage of drastically reducing the error induced by repeated application of a DP mechanism, making the DP protocol itself stateful.

Of the above cited, our work is most related to [9], which investigates the problem of computing DP histograms with

[3]In practical application, the streaming system may choose trigger adaptively, with complicated implementation Akidau et al. [2]

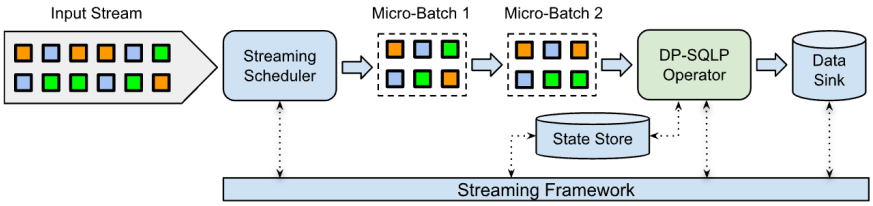

Figure 2: High-level overview for the DP-SQLP system

unknown-domains under continual observations. They leverage (extensions of) the tree aggregation protocol to build an efficient DP protocol for the continual observation setting. Our key selection algorithm (Algorithm 1) is heavily inspired by [9], with the main difference being the use of additional threshold μ for further privacy protection, and an algorithm to predict whether to select a key even if it has zero records (Algorithm 3). Algorithm 3 is crucial to the scalability of our approach to production workloads. From a system design point, our work further extend [9] as follows:

(1) We consider user level DP. As outlined in the introduction, this introduces interesting algorithmic and system challenges.
(2) We provide a concrete streaming system architecture for scalable production deployments whereas they focus on developing algorithms. More precisely, we develop and test an empty key prediction scheme that allows us to scale to millions of updates per second that contain billions of distinct keys.
(3) We test our algorithms and architecture on a number of large-scale synthetic and real-life datasets to demonstrate the efficacy of our approach relative to meaningful baselines.
(4) Our system is deployed to a extreme large scale production use case.

Another prior system that continuously releases streaming user count at scale is Google FLEDGE k-Anonymity server [21, 22]. It determines whether a given advertisement has been shown to at least k users over a sliding window with event-level differential privacy guarantee. While our algorithms and systems are more general that can be applied to arbitrary histogram query at scale, with a user-level guarantee. An extended list of real-world uses of DP is presented in Desfontaines [14].

1.3 Organization

The rest of the paper is organized as follows. In Section 2 we provide the necessary background on differential privacy and streaming systems; in Section 3 we describe the main algorithmic components of our DP streaming system; in Section 4 we provide details of the improvements needed to scale up the algorithms to large workloads; in Section 5 we provide a thorough experimental evaluation; and finally in Section 7 we provide some concluding remarks and outline a few interesting open directions. We also provide a glossary of terms from the streaming literature (used in this paper) in Appendix A in the full paper.

2 PRELIMINARIES

In this section, we describe the formalism that will be necessary for the rest of the paper: a) DP on Streams (Appendix B in the full paper), b) DP continual observation (Appendix C in the full paper), and c) System architecture for the streaming system (Section 2.1). For the purpose of brevity, we will defer (a) and (b) to the appendix in the full paper. For a comprehensive introduction to DP, please refer to [49].

2.1 Streaming System Architecture

The streaming differential privacy mechanism described in this paper can be generally applied to various streaming frameworks, including Spark Streaming[52], Apache Beam[2] and Apache Flink[8]. The DP-SQLP system we develop is implemented using a streaming framework similar to Spark Streaming[52], as shown in Figure 2.

The input data stream contains unordered, unbounded event records. The streaming scheduler will first assign each record to the corresponding event-time window w. Within each window, at every triggering timestamp tr, records are bundled together to create an immutable, partitioned datasets, called a *micro-batch*. After a micro-batch is created, it will be dispatched to the DP-SQLP operator for processing.

When processing a micro-batch, the DP-SQLP operator will interact with the system state store for a state update. Once the differentially private histogram is generated, it will be materialized to the data sink[4]. Similar to Spark Streaming, our streaming framework provides consistent, "exactly-once" processing semantic across multiple data centers. In addition, the streaming framework also provides fault tolerance and recovery.

There are multiple ways to schedule micro-batches based on certain rules[2], like processing timer based trigger, data arrival based trigger and combinations of multiple rules.

Based on the number of micro-batches received by operator at each time instance, we can also classify scheduling methods into two categories, sequential scheduling and parallel scheduling, as shown in the Figure 3. In sequential scheduling, input data stream is divided into a sequence of micro-batches. The operator will process one micro-batch at a time. Parallel scheduling is able to further scale up the pipeline, by allowing multiple micro-batches to be processed at the same time. The streaming scheduler will partition records by predefined key tuples, and create one micro-batch per key range.

[4]Data sink is the storage system used to store and serve output data, including file systems and databases.

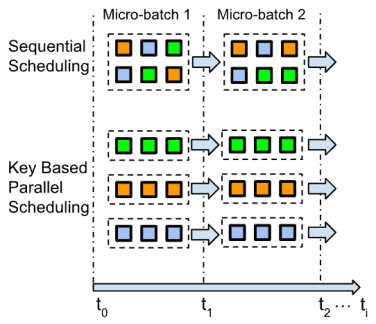

Figure 3: Two types of streaming scheduling

In the rest of paper, we will assume the sequential scheduling for the algorithm discussion. Given data stream D_{w_i} for window w_i, the sub-stream at triggering timestamp $tr_i \in \mathtt{Tr}_{w_i}$ can be denoted as $D_{tr_i} \subseteq D_{w_i}$, and the sub-stream for each micro-batch can be represented as the incremental data stream $\Delta D_{tr_i} = D_{tr_i} - D_{tr_{i-1}}$.

As Akidau pointed out in the unified dataflow model [2], batch, micro-batch, and pure streaming are implementation details of the underlying execution engine. Although the streaming differential privacy mechanism discussed in this paper is executed by a streaming system based on micro-batch, the mechanism and algorithm can be widely applied to batch, micro-batch, and pure streaming systems.

It is worth noting that different execution modes (batch, micro-batch and pure streaming) result in different trade-offs between data utility and pipeline latency. In differential privacy, the more frequent we repeat the process, the nosier results tend to be. Therefore, data utility is an additional factor when choosing execution modes.

3 STREAMING PRIVATE MECHANISM

In this section we will discuss the overall mechanism for streaming differential privacy. Our target is to perform aggregation and release histogram at every triggering timestamp in $\mathtt{Tr}$, while maintaining (ε, δ)-*differential privacy*. There are four main components within streaming differential privacy mechanism – user contribution bounding, partial aggregation, streaming private key selection and hierarchical perturbation, as shown in Figure 4.

To simplify the discussion, we will assume `Sum` is the aggregation function for `GROUP BY`. It is also possible to use other aggregation function within streaming differential privacy mechanism.

Let's declare the inputs, parameters and outputs that will be used in streaming differential privacy.

- Input: Data Stream D, event-time windows W, triggering timestamp per window $\mathtt{Tr}_w$, privacy parameters $\varepsilon, \delta > 0$, accuracy parameter $\beta > 0$, per record clamping limit L.
- System Parameters: Max. no. of records per user C.
- Output: Aggregated DP histogram at every triggering timestamp.

3.1 Non-Private Streaming Aggregation

The traditional streaming aggregation operator without differential privacy is shown on the top of Figure 4. Records within the micro-batch are grouped by key and aggregated by the *reduce* function [52]. After that, the partial aggregation result will be merged with the previous state [48] and the update histogram is emitted. There is no differential privacy protection within this process, and user privacy can be leaked from multiple dimensions, including histogram value, aggregation key and the differences between two histogram updates.

3.2 User Contribution Bounding

DP algorithms require that the sensitivity of each user contributions be limited, for example that a user can contribute up to C times. However, in reality, each user may contribute to many records and many keys, especially for heavy users. Therefore, we need to bound the maximum influence any user can have on the output in order to achieve a desired overall DP guarantee. This step is called "user contribution bounding".

Some one-shot mechanisms perform user contribution bounding by limiting contributed value per key and the number of contributed keys per user [3, 51]. However, this approach does not fit the streaming setting, since it requires three shuffle stages - shuffle by user, shuffle by (key, user) and shuffle by key.

User contribution bounding in streaming DP is performed on the user level for the entire data stream D[5]:

- Each user can contribute to at most C records in the data stream D.
- The value v for the aggregation column m in each record is clamped to L_m so that $|v| < L_m$.

The maximum number of records per user C and per record clamping limit L_m together determine the per-user ℓ_1 sensitivity in data stream:

$$L_1 = C \times L_m.$$

Choosing the right contribution bounding parameters C and L_m is critical for privacy-utility trade-off. When the bounding limit is small, the noise is small, but the data loss may be significant (e.g. if most/all users have a lot of data to contribute). On the other side, when the bounding limit is large, the data loss is small, but noise is large. It is possible to find a near optimal point from a heuristic study, which we discuss in Section 5. Indeed, one approach to choosing a good contribution bound is to inspect the data distribution. For example, we can pick C at 99^{th} percentile of per-user records (i.e. $< 1\%$ users have more than C records), which can be chosen in a DP way if computed on a fraction of the data stream, or in a non-DP way if it is based on proxy data.

In the following sections, we will introduce streaming private key selection and hierarchical perturbation, which are two main private operations in streaming differential privacy. Since the private operations are performed per window, we

[5] In a production streaming system, the DP guarantee is commonly defined with the minimum privacy protection unit (e.g., [user, day]). The maximum number of record per user C needs to be enforced within each privacy unit.

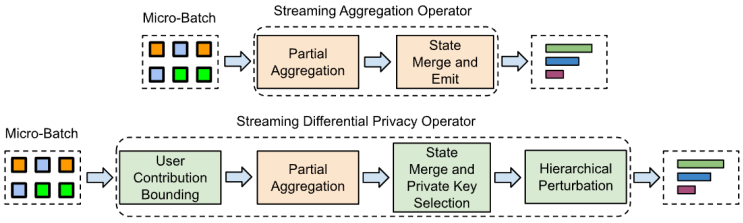

Figure 4: Overview of streaming differential privacy mechanism

will focus on D_{w_i} during algorithm discussion. However, the same private operations should be applied to all windows.

3.3 Streaming Private Key Selection

The main objective we consider in this section is to select the set of keys that exceed a certain threshold of user contributions $\mu \geq 0$. (These are the keys that are used later for releasing the aggregation columns in Section 3.4 below.) Recall that our streaming system is input driven by a growing data stream, meaning, one only sees the existence of a key if it has at least one user contribution. This poses a significant privacy challenge since the detectable set of keys is highly dependent on the data set. As a result, we design a novel thresholding scheme coupled with binary tree aggregation [10, 17, 27] that allows one to only operate with keys that deterministically have at least μ user contributions, and still preserve (ε, δ)-DP. In the following, we provide a description of the algorithm, along with the privacy analysis.

Data preprocessing: After user contribution bounding (discussed in Section 3.2), we first perform a regular key `GROUP BY` and aggregation for all records within the current micro-batch[6]. Then we merge the aggregated data on each key with a data buffer that is stored in a system state[7]. Beyond that we execute the key selection algorithm described in Algorithm 1.

Algorithm description: As mentioned earlier, the emitted key space is not predefined. A key may emerge when there is at least one user record with the key (due to the nature of input driven stream). Therefore, the streaming DP system must determine *when* and *what* keys to release or update, in a private manner. This is different from the non-private streaming aggregation, where updates will be emitted at every triggering timestamp after processing each micro-batch.

Remark: In the description of Algorithm 1, we use the following primitives implicitly used in Algorithm 4 (Appendix C): i) `InitializeTree` (T, σ): Initialize a complete binary tree $\mathcal{T}$ with $2^{\lceil T \rceil}$ leaf nodes with each node being sampled from $\mathcal{N}(0, \sigma^2)$, ii) `AddToTree` $(\mathcal{T}, i, c_i)$: Add c_i to all the nodes on the path from the i-th leaf node to the root of the tree, and iii) `GetTotalSum` $(\mathcal{T}_{\text{key}}, i)$: Prefix sum of the all the inputs $\{c_1, \ldots, c_i\}$ to the binary tree computed via Algorithm 4.

Our approach is an extension of thresholding algorithm in [33] to the streaming setting. In short, we carefully select a threshold and compute (with DP noise) the number of unique users that contribute to each encountered key. If this noisy number is greater than or equal to the chosen threshold, the key is released. We describe the algorithm in full detail in Algorithm 1, and provide the formal privacy guarantee in Theorem 3.1.

A crucial component of the algorithm is the choice of the threshold τ in Line 2 of Algorithm 1. One can instantiate τ with the bound in Theorem C.2. However, in our implementation (described in Section 4) we actually implement the tree aggregation via the "bottom-up Honaker" variance reduction described in Appendix C in the full paper. One can write the exact distribution of the differences between DP-Tree aggregated count and the true count, which is $\hat{q}_{tr_i,k} - \texttt{count}_k(D_{tr_i})$ in Line 2 of Algorithm 1, via equation (2). This allows us to get a tighter bound on τ based on the inverse CDF of the Gaussian distribution. Also, it should be obvious from equation (2) that the variance of the Gaussian distribution is dependent on the time step at which we are evaluating the cumulative sum. Hence, to obtain a tighter estimation of the threshold, we actually have a time dependent threshold τ_{tr_i} (based on equation (2)) instead of an universal threshold in Line 2.

Remark: For brevity, in Theorems 3.1 and 3.2, we provide the guarantees assuming each user only contributes once, i.e., in the language of Section 3.2 we assume that that user contribution bound $C = 1$. However, in our implementation we do allow $C > 1$. The idea is to use a tighter variant of advanced composition for (ε, δ)-DP [19] while ensuring that each user contributes *at most once to each key* in any instance of Algorithm 1. In the following we provide the privacy and the utility guarantees. For brevity, we defer the proofs to Appendix E and Appendix F in the full paper.

Theorem 3.1 (Privacy guarantee). *Algorithm 1 is $(\varepsilon, \delta + (e^{\varepsilon} + 1) \cdot \beta)$-DP for addition or removal of one element of the dataset.*

Theorem 3.2 (Utility guarantee). *For any fixed key k, there exists a threshold $\tau = O\left(\frac{\sqrt{\log(T/\beta)\log^2(T)\log(1/\delta)}}{\varepsilon}\right)$ such that w.p. at least $1 - \beta$, Algorithm 1 outputs k if at any one of the triggering time (in $Tr = [tr_1, tr_2, ..., tr_T]$) the true count is at least $\mu + \tau$.*

3.4 Hierarchical Perturbation

Once sufficient records of certain key are accumulated (i.e., following the notation from the previous section there are at

[6]This is a system level operation without any implication to the privacy guarantee.
[7]For every key, we accumulate aggregation column values, as well as the number of unique users. This process is called data accumulation.

Algorithm 1 Streaming Private Key Selection

Require: Data stream $D_{w_i} = \{d_1, \ldots, d_n\}$, where the event timestamp t_i of each d_i can map to event-time window $w_i = (t_s, t_e)$, $t_s < t_i < t_e$, triggering timestamps $\texttt{Tr}_{w_i} = [tr_1, tr_2, ..., tr_{T_i}]$. At each triggering time $tr_i \subseteq \mathbb{R}$, only a sub-stream $D_{tr_i} \subseteq D_{w_i}$ is available. Threshold $\mu \geq 0$, privacy parameters $\varepsilon, \delta > 0$, failure probability $\beta > 0$.

1: Compute the noise standard deviation σ for the tree aggregation based on $(T_i = |\texttt{Tr}_{w_i}|, \varepsilon, \delta)$. (See Appendix D in the full paper for more details.)
2: Compute the accuracy threshold τ of the tree aggregation such that for any fixed key and the corresponding binary tree $\mathcal{T}_{\texttt{key}}$,

$$\mathop{\mathbb{P}}_{\mathcal{T}_{\texttt{key}}} \left[\forall tr_i \in \texttt{Tr}_w, |\hat{q}_{tr_i,\texttt{key}} - \texttt{count}_{\texttt{key}}(D_{tr_i})| \leq \tau\right] \geq 1-\beta,$$

which depends on σ and β. (See Appendix D for more details.) Here, $\texttt{count}_{\texttt{key}}(D_{tr_i})$ denote the unique user count for key in D_{tr_i}, and $\hat{q}_{tr_i,\texttt{key}}$ denote the private estimate of $\texttt{count}_{\texttt{key}}(D_{tr_i})$.
3: **for** $i \in |\texttt{Tr}_{w_i}|$ **do**
4: $\mathcal{S}^{(i)} \leftarrow$ Set of all keys in the stream D_{tr_i} with count $> \mu$.
5: For all key $\in \mathcal{S}^{(i)} \backslash \mathcal{S}^{(i-1)}$, create a new tree $\mathcal{T}_{\texttt{key}}$ using $\texttt{InitializeTree}(T_i, \sigma)$, and execute Algorithm 4 till $(i-1)$-th step with all zeros as input.
6: **for** key $\in \mathcal{S}^{(i)}$ **do**
7: $\mathcal{T}_{\texttt{key}} \leftarrow \texttt{AddToTree}(\mathcal{T}_{\texttt{key}}, i, \texttt{count}_{\texttt{key}}(D_{tr_i}) - \texttt{count}_{\texttt{key}}(D_{tr_{i-1}}))$, i.e., Add the count at time stamp tr_i to $\mathcal{T}_{\texttt{key}}$.
8: $\hat{q}_{tr_i,\texttt{key}} \leftarrow \texttt{GetTotalSum}(\mathcal{T}_{\texttt{key}}, i)$.
9: **if** $\hat{q}_{tr_i,\texttt{key}} > \mu + \tau$, **then** output $(\texttt{key}, \hat{q}_{tr_i,\texttt{key}})$.
10: **end for**
11: **end for**

least μ unique user contributions), the objective is to select the key for *statistic release*. Statistic release corresponds to adding the value from user contributions to a main histogram that estimates the distribution of records across all the keys. Notice that in this histogram, a single user can contribute multiple times to the same key. In this section we discuss how to create this histogram while preserving DP.

The crux of the algorithm is that for all the set of keys detected during the key selection phase via Algorithm 1, we maintain a DP-tree (an instantiation of Algorithm 4) for every key detected. We provide a ρ-zCDP guarantee for each of the trees for each of the keys. Since each user can contribute C records in this phase, and in the worst case all these contributions can go to the same node of a single binary tree, we scale up the sensitivity corresponding to any single node in the tree to $L_1 = C \cdot L$ (analogous to that in Section 3.2), and ensure that each tree still ensures ρ-zCDP. We provide the details of the algorithm in Algorithm 2. The privacy guarantee follows immediately from Theorem C.1, and the translation from ρ-zCDP to (ε, δ)-DP guarantee. In Algorithm 2, we will use a lot of the binary tree aggregation primitives we used in Section 3.3.

Algorithm 2 Hierarchical Perturbation with DP-Tree

Require: Data stream: $D_{w_i} = \{d_1, \ldots, d_n\}$, where each d_i arrive at time t_i within event-time window $w_i = (t_s, t_e)$, $t_s < t_i < t_e$. Triggering timestamps $\texttt{Tr}_{w_i} = [tr_1, tr_2, ..., tr_{T_i}]$. At each triggering timestamp $tr_i \subseteq \mathbb{R}$, only a sub-stream $D_t \in D_{w_i}$ is available. Privacy parameters $\varepsilon, \delta > 0$, number of DP-Tree leaf nodes n.

1: Compute the noise standard deviation σ for the tree aggregation based on (n, ε, δ). (See Appendix D in the full paper for more details.)
2: **for** $i \in |\texttt{Tr}_{w_i}|$ **do**
3: $\mathcal{S}_i \leftarrow$ Set of keys output by the key selection algorithm (Algorithm 1) at tr_i.
4: **for** key $\in \mathcal{S}_i$ **do**
5: Let $D_{tr_i} \leftarrow$ data stream available at time stamp tr_i.
6: Last Release Time $\texttt{LRT}_{\texttt{key}} \leftarrow$ triggering timestamp of the previous statistic release.
7: $\Delta V_{\texttt{key}} \leftarrow$ Aggregated value for key in the sub-stream $D_{tr_i} - D_{\texttt{LRT}_{\texttt{key}}}$.
8: $\mathcal{T}_{\texttt{key}} \leftarrow \texttt{AddToTree}(\mathcal{T}_{\texttt{key}}, i, \Delta V_{\texttt{key}})$.
9: Output $\texttt{GetTotalSum}(\mathcal{T}_{\texttt{key}}, i)$.
10: **end for**
11: **end for**

4 SYSTEM IMPLEMENTATION AND OPTIMIZATION

When implementing the streaming differential privacy algorithms described in Section 3, one must take the system constraints into practical considerations. There are three main challenges:

- The streaming framework described in Section 2.1 discretizes the data stream into micro-batches. Therefore, streaming key selection and hierarchical perturbation, whose algorithms are defined based on data stream $D_{t_i^{\texttt{Tr}}}$, must be implemented using micro-batch $\Delta D_{t_i^{\texttt{Tr}}}$ (defined in Section 2.1) and the system state. In Section 4.1 we detail the complete state management of DP-SQLP.
- DP-SQLP is input data stream driven. The state loading and updating require the existence of a key in the current micro-batch. However, Algorithm 1 requires to test all keys that have appeared at least once. In Section 4.3 we discuss a new algorithm that avoids testing all the keys that have appeared at least once.
- There are multiple components to the DP-SQLP system which are individually (ε, δ)-DP. It is necessary to use appropriate forms of composition to account for the total privacy cost. In Section 4.4, we detail the complete privacy accounting for DP-SQLP.

4.1 State Management

As mentioned above, both streaming key selection and hierarchical perturbation are defined based on data stream $D_{t_i^{\texttt{Tr}}}$. Therefore, they both require stateful operations. Furthermore, the global user contribution bounding that tracks the number of records per-user in data stream D is also stateful.

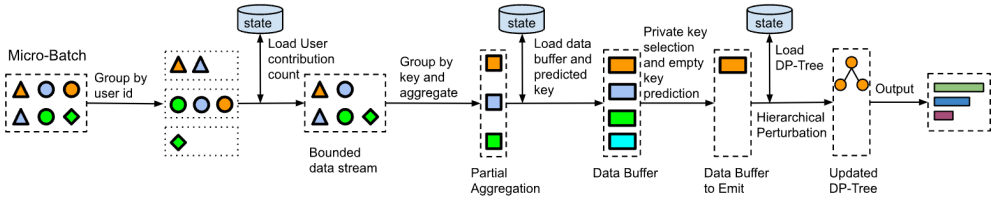

Figure 5: Execution of streaming differential privacy mechanism

In DP-SQLP, the system state store is a persistent storage system, backed by Spanner database [12] that provides high availability and fault tolerance. The system state store is co-managed by the DP-SQLP operator and the streaming framework for state update and maintenance. All the state information required by streaming differential privacy is stored in the system state store. For a pictorial depiction, see Figure 2.

State Store Structure: There are two main state tables in the system state stores, which are managed by the same streaming framework. Each state table is a key value storage containing *state key* and *state object*.

The first state table is keyed by user id to track per-user contribution within the data stream. The state object simply stores the count value.

The second state table is keyed by `GROUP BY` keys. The state object contains data buffer, `DP-Trees` for key selection and `DP-Trees` for aggregation columns. Data buffer is a data structure that temporarily stores the unreleased, aggregated data from new records, due to the failure in thresholding test (line 9, Algorithm 1). One `DP-Tree` is used by each round of Algorithm 1 execution, and one `DP-Tree` is used for hierarchical perturbation per aggregation column.

Execution Procedures: The execution of streaming differential privacy mechanism is shown in Figure 5. Different shapes represent different users and different colors represent different keys. Each step is described as following.

(1) **User contribution bounding**: Records in one *micro-batch* are grouped by user id. A map in *system state store* is maintained to track the number of records each user contributes. Once the number of contributions for a user reaches C, all the remaining records for that user in the data stream will be discarded. Furthermore, we clamp the value v of each *aggregation column* m, so that $|v| \leq L_m$.
(2) **Cross-user aggregation**: Records in one *micro-batch* are grouped by key and aggregated, which form a delta result [23]. After that, the delta result will be merged into the data buffer that is loaded from the system state store.
(3) **Streaming key selection**: The `DP-Trees` for streaming key selection are loaded from the system state store. Then we will perform Algorithm 1, which adds the incremental user count from the current micro-batch into `DP-Tree`, as a leaf node.
(4) **Hierarchical perturbation**: Once a key is selected, the `DP-Tree` for hierarchical perturbation is loaded from the system state store. After that, we will use Algorithm 2 to get DP aggregation results, and output the results.

The execution engine used to implement user contribution bounding and hierarchical perturbation will be discussed in section 4.2.

As mentioned in section 3.3, the `DP-Tree` estimator is implemented with the "bottom-up Honaker" variance reduction to get the DP sum. The estimated sum for `DP-Tree` root at node_{i^*} equals

$$\hat{\texttt{Sum}}(\text{node}_i) = \sum_{j=0}^{\mu-1} c_j \cdot \text{sum}(\texttt{level}_j).$$

In case more than one `DP-Trees` are used in key selection or hierarchical perturbation, we need to further sum the Honaker estimations from each tree together.

4.2 Parallel Execution

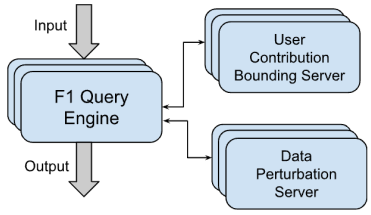

Figure 6: Parallel execution within DP-SQLP operator

When building the DP-SQLP operator, we leverage the F1 query engine [43] for its wide range of data sources and distributed query execution (Figure 6). The user contribution bounding step is executed by the *user contribution bounding server*. The privacy key selection and hierarchical perturbation are executed by the *data perturbation server*. Both servers contain thousands of workers that are horizontally scalable. Input data is first read by F1 query engine, partitioned, then sent to the user contribution bounding server through Remote Procedure Calls (RPCs). After that, the bounded data will stream back to F1, re-partitioned, then being sent to data perturbation server for key selection and hierarchical perturbation.

4.3 Empty Key Release Prediction

Since the data stream is unordered and unbounded, the existence of user contributions within each micro-batch can be arbitrary, as shown in Figure 7. It is possible that some keys do not have any user records in a micro-batch. In the traditional

streaming systems, states are not updated unless new records appear in the micro-batch[48]. Therefore, the system only needs to load states with keys from the current micro-batch.

However, the streaming key selection algorithm (Algorithm 1) requires us to perform the thresholding test for the entire key space, and it is possible for a key to be selected without new records. An naive approach is to load all the keys with their associated states from the system state store, and run Algorithm 1 directly. Unfortunately, when the key space is large, the I/O cost and memory cost of loading the entire state table is too high. Here, we propose the empty key release prediction algorithm, together with other operational strategies, to solve the scalability challenge.

Algorithm 3 Empty Key Release Prediction

Require: Data stream: $D_w = \{d_1, \ldots, d_n\}$, where each d_i arrive at time t_i within event-time window $w = (t_s, t_e)$, $t_s < t_i < t_e$. Triggering timestamps $\mathtt{Tr}_w = [tr_1, tr_2, ..., tr_T]$. At each triggering timestamp $tr_i \subseteq \mathbb{R}$, only a sub-stream $D_t \in D_w$ is available. privacy parameters $\varepsilon, \delta > 0$. Let j be the current round of key selection.

1: $\Delta D_{tr_j} \leftarrow D_{tr_j} - D_{tr_{j-1}}$ (which is the data stream for the micro-batch at tr_j.
2: $\mathcal{S}_{tr_j} \leftarrow$ All $\mathtt{key} \in \Delta D_{tr_j}$ selected via Algorithm 1 at time tr_j.
3: **for** $\mathtt{key} \in \mathcal{S}_{tr_j}$ **do**
4: **for** tr_p **from** tr_{j+1} **to** $tr_{|\mathtt{Tr}_w|}$ **do**
5: $D_{tr_p} \leftarrow D_{tr_j}$, which mimic the data stream at the next triggering timestamp tr_p, without any new record.
6: Perform streaming private key selection on D_{tr_p} for $\mathtt{key}$ via Algorithm 1 at time tr_p.
7: **if** $\mathtt{key}$ is selected, **then** write $(\mathtt{key}, tr_p)$ to state store and **break**.
8: **end for**
9: **end for**

Algorithm description for empty key release prediction: There are two scenarios that may trigger a key being selected: key is selected due to additional user contributions, or key is selected due to noise addition without user contributions. The first scenario is naturally handled by the streaming system, when processing micro-batch with new records. The secondary scenario is handled by Algorithm 3.

When a micro-batch ΔD_{tr_j} contains key k, the streaming private key selection algorithm for key k is applied to the sub-stream D_{tr_j}. In case k is not selected, we will simulate streaming private key selection algorithm executions from tr_{j+1} to $tr_{|\mathtt{Tr}|}$, using sub-stream D_{tr_j}, and predict if any future release is possible by adding leaf node with *zero* count. The predicted releasing time is tr_p, and it is written to the system state store.

After making a release prediction, the DP-SQLP will continue to process the next micro-batch. For key k with predicted releasing time tr_p, there are two cases:

(1) k appears in another micro-batch ΔD_{tr_n} before the predicted triggering timestamp ($j < n < p$). In this case, the prior prediction result is discarded. We will perform key selection algorithm for the micro-batch ΔD_{tr_n}. All the thresholding test for micro-batches that do not have k between tr_{j+1} and tr_{n-1} have been performed during the prediction phase in the prior micro-batch D_{tr_j}. In addition, we will make a new prediction for ΔD_{tr_n}.
(2) k appears in another micro-batch ΔD_{tr_n} after the predicted triggering timestamp ($p < n$). In this case, DP-SQLP loaded system states for k at the predicted time tr_p and released data from the data buffer. We will start a new round of streaming key selection from micro-batch ΔD_{tr_n}.

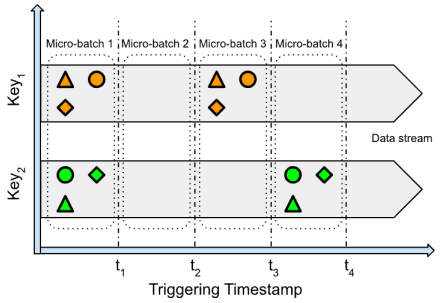

Figure 7: Some key may not have records within certain micro-batch

Within these operations, some computations might be wasted (e.g., Case 1). However, when the key space is large, the reductions in I/O cost and memory cost bring in more benefits than the CPU overhead.

The prediction result is stored in the system state. The step to load predicted results is shown in Figure 5. We also add a secondary index on the predicted timestamp to improve the state loading speed.

4.4 Privacy Accounting for DP-SQLP

In DP-SQLP, the privacy costs occur in streaming key selection (Algorithm 1) and hierarchical perturbation (Algorithm 2). Because each user is allowed to contribute at most C records, we use the combination of composition and sensitivity in privacy accounting.

- *Privacy accounting for streaming key selection:* When executing Algorithm 1 in DP-SQLP, the *value* added to each leaf node is the *unique user count*. Therefore, the per-user sensitivity for each $\mathtt{DP-Tree}$ is one. In addition, we restart the Algorithm 1 once a key is selected, and data accumulated in the data buffer is released immediately. As a result, each user may participate in at most C rounds of key selection. Given the (ε, δ) privacy budget for each round of Algorithm 1, the total privacy cost for streaming key selection is calculated using the empirically tighter variant of advanced composition [19] with C-fold.
- *Privacy accounting for hierarchical perturbation:* For each user, in the worst case, all C contributions can go to the same node of a single $\mathtt{DP-Tree}$, we scale up

the sensitivity corresponding to any single node in the tree to $L_1 = C \cdot L$, and ensure that each tree still ensures ρ-zCDP. After that, we use the conversion from [40, Proposition 3] to translate privacy cost from ρ-zCDP to (ε, δ)-DP guarantee [8].

Finally, the privacy costs of key selection and hierarchical perturbation are combined via advanced composition.

5 EXPERIMENTS

The experiments are performed using both synthetic and real-world data to demonstrate the data utility and scalability. The streaming DP mechanism is implemented in the DP-SQLP operator, as described in section 4.

Baselines: We compare DP-SQLP with two baseline approaches for data utility.

- Baseline 1 - Repeated differential privacy query. Most of the existing DP mechanisms does not have capacity to track user contributions across multiple queries. Therefore, when handling data streams, one common workaround is to repeatedly apply the one-shot DP query to the growing data set in order to get the histogram update. Thus, the overall privacy budget usage is the composition of all queries.
- Baseline 2 - Incremental differential privacy processing. The one-shot differential privacy algorithm is applied separately to each micro-batch, and we can get the final result by aggregating the outputs from each micro-batch. Compared with baseline 1, baseline 2 requires a similar global user contribution bounding system as DP-SQLP.

For both baseline 1 and baseline 2, the one-shot differential privacy mechanism is executed by Plume [3] with the Gaussian mechanism. Each one-shot differential privacy execution guarantees (ε, δ)-differential privacy. We also adopt the optimal composition theorem for DP [31] to maximize the baseline performance.

Metrics: The data utility is evaluated based on 4 metrics calculated between the ground truth histogram and the differentially private histogram. 1) Number of retained keys, which reflects how many keys are discovered during key selection process. It is also known as the ℓ_0 norm, 2) ℓ_∞-norm, which reflects the worst case error: $\max_{k \in \texttt{key space}} (|\hat{M}_k - M_k|)$, 3) ℓ_1-norm, which reflects the worst case error: $\sum_{k \in \texttt{ key space}} (|\hat{M}_k - M_k|)$, and 4) ℓ_2-norm: $\sqrt{\sum_{k \in \texttt{key space}} (|\hat{M}_k - M_k|^2)}$.

We choose $\varepsilon = 6$ and $\delta = 10^{-9}$ as the overall privacy budget for all experiments. Within DP-SQLP, the privacy budget used by the aggregation column is $\varepsilon_m = \varepsilon/2$, $\delta_m = \delta/3$, and the ones used by key selection is $\varepsilon_k = \varepsilon/2$, $\delta_k = \delta \times 2/3$. The parameter C is chosen based on the dataset property. In this experiment, we sampled 10% of one day's data and set C according to the 99 percentile of per user number of records. There are more discussions on choosing C in Section 5.4.

[8]The exact computation is from https://github.com/IBM/discrete-gaussian-differential-privacy/blob/master/cdp2adp.py#L123

For both synthetic data and real-world data, We assume the dataset represents a data stream within one day. We also shuffle users' records so that they are randomly distributed within the day. In experiments, the event-time window is also fixed to one day.

In addition to data utility, we also report the performance latency under various micro-batch sizes and number of workers.

5.1 Synthetic Data

We used the similar approach as [3] to generate the synthetic data to capture the long-tailed nature of real-world data. There are 10 millions unique users in the synthetic dataset. Each user draws a number of contributed records from a distribution with range $[1, 10^5]$ and mean 10 according to a Zipf-Mandelbrot distribution. The parameters[9] are chosen so that roughly 15% users contribute to more than 10 records. The key in each record is also sampled from a set of size 10^6, following a Zipf-Mandelbrot distribution[10]. This implies that roughly 1/3 of records have the first 10^3 keys.

The histogram query task we perform is a simple count query.

```
SELECT key, COUNT(*)
FROM SyntheticDataset
GROUP BY key
```

Listing 2: Histogram query for synthetic data

The per-record clamping limit $L = 1$ since the aggregation function is *COUNT*. We set $C = 32$.

All measurements are averaged across 3 runs. The experiments are performed with 100 micro-batches and 1000 micro-batches. Within one day, 100 micro-batches correspond to roughly 15 minute triggering intervals, and 1000 micro-batches correspond to roughly 1.5 min triggering intervals.

Table 1: Data utility measure with synthetic data ($\varepsilon = 6, \delta = 10^{-9}$)

Metrics	100 Micro-batches		
	DP-SQLP	Baseline 1	Baseline 2
Keys	28,338	435	191
ℓ_∞ Norm	1,391	18,077	21,913
ℓ_1 Norm	17,741,225	50,835,203	58,551,587
ℓ_2 Norm	50,039	430,547	576,425

Metrics	1000 Micro-batches		
	DP-SQLP	Baseline 1	Baseline 2
Keys	22,280	0	0
ℓ_∞ Norm	1,563	25,497	25,497
ℓ_1 Norm	19,395,721	59,052,062	59,052,062
ℓ_2 Norm	58,237	594,382	594,382

The results are shown in Table 1. There are significant data utility improvements comparing DP-SQLP with two baselines.

[9]In Zipf-Mandelbrot, the sampling probability is proportional to $(x + q)^{-s}$, where $q = 26$, $s = 6.738$.
[10]$q = 1000$, $s = 1.4$.

With 100 micro-batches, the number of retained keys is increased by **65** times; the worst case error is reduced by **92%**; the ℓ_1 norm is reduced by **65.1%** and the ℓ_2 norm is reduced by **88.4%**. The utility improvement is even more significant with 1000 micro-batches. The number of retained keys is increased from 0 to 22,280; the worst case error is reduced by **93.9%**; the ℓ_1 norm is reduced by **67.2%** and the ℓ_2 norm is reduced by **90.2%**.

Another observation we have for DP-SQLP is its stability when the number of micro-batches increases. The utility of one-shot differential privacy mechanisms in baseline 1 and baseline 2 degrade quickly, due to the privacy budget split (baseline 1) and data stream split (baseline 2). This degradation sometimes is not linear. The number of retained keys in baseline 1 and 2 is reduced to 0 when the number of micro-batches grows from 100 to 1000. On the contrary, the utility degradation for DP-SQLP is not as significant. Indeed, when the number of micro-batches increases from 100 to 1000, for DP-SQLP, the number of retained keys is reduced by 21%, the worst case error is increased by 12%, the ℓ_1 norm is increased by 9%, and the ℓ_2 norm is increased by 16%.

In summary, DP-SQLP shows a significant utility improvement over one-shot differential privacy mechanisms when continuously generating DP histograms.

5.2 Reddit Data

In the next step, we apply the same experiment to real-world data. Webis-tldr-17-corpus [46] is a popular dataset consisting of 3.8 million posts associated with 1.4 million users on the discussion website Reddit. Our task is to count the user participation per subreddit (specific interest group on Reddit).

We set $C = 17$ and the rest of the experiment settings are the same as the synthetic data. All measurements are averaged across 3 runs.

Table 2: Data utility measure with the Reddit data ($\varepsilon = 6, \delta = 10^{-9}$)

Metrics	100 Micro-batches		
	DP-SQLP	Baseline 1	Baseline 2
Keys	1,473	32	63
ℓ_∞ Norm	102,250	267,147	103,546
ℓ_1 Norm	989,249	2,721,349	2,376,937
ℓ_2 Norm	127,721	322,739	156,472

Metrics	1000 Micro-batches		
	DP-SQLP	Baseline 1	Baseline 2
Keys	1,181	9	3
ℓ_∞ Norm	102,218	266,391	108,124
ℓ_1 Norm	1,074,724	3,081,542	3,074,655
ℓ_2 Norm	127,830	341,557	242,482

The results are summarized in Table 2, and we have similar observations as in the synthetic data experiments. DP-SQLP demonstrates significant utility improvements in the number of retained keys, ℓ_1 norm and ℓ_2 norm, as well as the performance stability when the number of micro-batches grows from 100 to 1000.

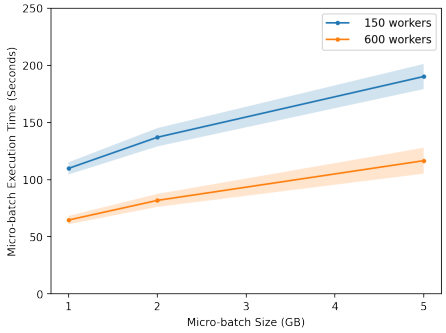

Figure 8: Micro-batch execution time ($\varepsilon = 6, \delta = 10^{-9}$)

5.3 Execution Performance

The end-to-end latency of a record consists of framework latency and micro-batch execution latency. The former is determined by the streaming framework. The latter is a critical indicator for the system scalability. In this section, we report the execution latency for each micro-batch, under different micro-batch sizes and number of workers.

The results are shown in Figure 8. All measurements are averaged across 2 runs, with shaded regions representing standard error. The execution latency grows sub-linearly as the micro-batch size increases. For example, with 150 workers, the execution latency grows 1.7 times while the data size increases 5 times from 1 GB to 5 GB.

Figure 8 also demonstrates horizontal scalability by trading machine resources with latency. When the total number of workers increases from 150 to 600, the execution latency is reduced by 41%, 40%, and 39% respectively for 1, 2, and 5 GB micro-batches.

To further test the scalability in terms of the size of key space, we generate another large synthetic dataset with 1 billion users. Each user draws the number of contributed records following Zipf-Mandelbrot distribution[11], generating 6 billion records in total. The key in each record is sampled from 1 billion keys following uniform distribution. When setting the micro-batch size equals 1GB and using 5500 workers, the average execution latency is 306 seconds. DP-SQLP easily handles the large load without incurring any significant performance hit.

5.4 Parameter Tuning

Tuning user contribution bounding is critical to achieve good data utility. If C is too small, lots of user records may be dropped due to user contribution bounding, which will lead to large histogram error. However, if C is very large, the noise and key selection threshold are scaled up accordingly. Choosing the right C is an optimization task.

In this section, we run the DP-SQLP with synthetic data using variable C from 1 to 50. Figure 9 shows how the change of C affects the number of retained keys, ℓ_1 norm, ℓ_2 norm and ℓ_∞ norm. The optimal value for C (naturally) varies under different metrics. For example, the optimal C for ℓ_1 norm is

[11] $q = 26$, $s = 6.738$.

around 25 whereas the optimal C for ℓ_2 norm is around 30. In comparison, the optimal C for the number of retained key is around 5. Therefore, the optimal value of C should be chosen according to the metric we care most about (e.g., ℓ_2 norm). In real applications, we could use the $P99$ percentile value or DP $P99$ percentile value from a data sample as the starting point and perform a few rounds of tests to search for the optimal point.

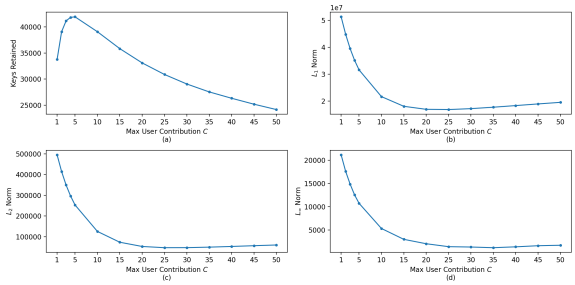

Figure 9: Metrics under different contribution limit ($\varepsilon = 6, \delta = 10^{-9}$)

6 INDUSTRY APPLICATIONS

6.1 Apply DP-SQLP to Google Shopping

We implemented a streaming differentially private user impressions for Google Shopping. A DP-SQLP pipeline was deployed that outputs differentially private product page-view counts. The page-view was used by some shopping systems as a signal for prioritizing the crawling of pages to update product price and availability information. When it comes to use cases such as price and availability, data freshness is critical for a good user experience. Long pipeline latency in case of using batch process, may incur obsolete search results. Therefore, a streaming pipeline with low latency is required by Google Shopping.

To maintain a low latency stream of page-view counts, while addressing the privacy risks, Google Shopping set up a DP-SQLP pipeline that outputs a histogram of view count per merchant product page. The DP-SQLP pipeline processes around 20GB/s data stream, and maintains a 20 mins end-to-end latency.

In terms of data utility after adopting DP-SQLP, we were able to retain 59% of the page-view. When focusing on the head pages, utility increases to 75% for pages with an average view rate of 1 view/hour, and to 99.9% for pages with an average view rate of 60 views/hour. When comparing noised impression counts with the raw counts, the relative error is around 11%. Each user is bound to contribute one event per day to ensure user level DP guarantee, per day. We use $\varepsilon = 1$ for streaming aggregation. For streaming key selection, when choosing $\varepsilon = 1$, the equivalent threshold is around 120; when choosing $\varepsilon = 3$, the equivalent threshold is around 43; and when choosing $\varepsilon = 10$, the equivalent threshold is around 16. The overall $\delta = 10^{-9}$. The max triggering window size is 150.

6.2 Streaming DP in Google Trends

Google Trends allows users to analyze the interest of search queries. The streaming private key selection (algorithm 1) is applied to it for selecting common Google Search queries with differential privacy in a streaming manner. Only queries chosen with differential privacy guarantee are shown on Google Trends website (e.g. as trending queries or related queries). The streaming pipeline ensures the 15 min end to end latency for highly searched queries to be selected with DP. Each user is bounded to contribute one event per query. We use $\varepsilon = 2$ and $\delta = 10^{-10}$ for a user-query level DP guarantee, i.e. each query has a budget $\varepsilon = 2$ and $\delta = 10^{-10}$. In addition, a deterministic pre-threshold $\mu = 50$ is used, to yield a stronger privacy protection. That means that DP streaming selection is applied only for queries with at least 50 unique users.

7 CONCLUSION AND FUTURE WORK

In this paper, we presented a streaming differentially private system (DP-SQLP) that is designed to continuously release DP histograms. We provide a formal (ε, δ)-user level DP guarantee for arbitrary data streams. In addition to the algorithmic design, we implemented our system using a streaming framework similar to Spark streaming, Spanner database, and F1 query engine from Google. The experiments were conducted using both synthetic data and Reddit data. We compared DP-SQLP with two baselines, and the results demonstrated a significant performance improvement in terms of data utility. In the end, we present two industry applications that apply DP-SQLP and the streaming private key selection algorithm to the production use cases.

There are three main ways in which our system can be further extended. First, in the design of the system we have used DP-tree aggregation [10, 17, 27] as the baseline DP algorithm. In recent research [13, 26], it has been shown that DP-tree aggregation is (significantly) sub-optimal in terms of privacy/utility trade-off, compared to general matrix factorization based mechanisms (DP-MF) [38]. However, DP-MF algorithms are not in general compatible with systems operating on data streams. Recently, following to our work, [39] provided a streaming variant of DP-MF. In future incarnations of our system, we plan to incorporate this.

Second, our algorithms are primarily designed to provide a centralized DP guarantee, where the final outcome of the system is guaranteed to be DP. It is worth exploring DP streaming system designs that allow stronger privacy guarantees like pan-privacy [18].

Third, we bound the contribution of each user globally by C. However, for higher fidelity, it is important to explore approaches to perform per-key contribution bounding. Naive approaches that address this issue can get complicated due to the fact that we are dealing with an input driven stream.

ACKNOWLEDGMENTS

We would like to thank Olaf Bachmann, Wei Hong, Jason Peasgood, Algis Rudys, Daniel Simmons-Marengo, Yurii Sushko and Sergei Vassilvitskii for the discussions and support to this project.

REFERENCES

[1] Naman Agarwal and Karan Singh. 2017. The price of differential privacy for online learning. In *International Conference on Machine Learning*. PMLR, 32–40.

[2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. (2015).

[3] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. 2022. Plume: Differential Privacy at Scale. *arXiv preprint arXiv:2201.11603* (2022).

[4] Kareem Amin, Alex Kulesza, Andres Munoz, and Sergei Vassilvtiskii. 2019. Bounding user contributions: A bias-variance trade-off in differential privacy. In *International Conference on Machine Learning*. PMLR, 263–271.

[5] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 901–914.

[6] Mark Bun and Thomas Steinke. 2016. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*. Springer, 635–658.

[7] Joseph A. Calandrino, Ann Kilzer, Arvind Narayanan, Edward W. Felten, and Vitaly Shmatikov. 2011. "You Might Also Like:" Privacy Risks of Collaborative Filtering. In *32nd IEEE Symp. on Security and Privacy*. 231–246.

[8] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38, 4 (2015), 28–38. http://sites.computer.org/debull/A15dec/p28.pdf

[9] Adrian Rivera Cardoso and Ryan Rogers. 2022. Differentially private histograms under continual observation: Streaming selection into the unknown. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2397–2419.

[10] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Trans. on Information Systems Security* 14, 3 (Nov. 2011), 26:1–26:24.

[11] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. 2020. Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs. *arXiv preprint arXiv:2003.11511* (2020).

[12] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 1–22.

[13] Sergey Denisov, Brendan McMahan, Keith Rush, Adam Smith, and Abhradeep Thakurta. 2022. Improved differential privacy for sgd via optimal private linear operators on adaptive streams. *arXiv preprint arXiv:2202.08312* (2022).

[14] Damien Desfontaines. 2024. A list of real-world uses of differential privacy. https://desfontain.es/blog/real-world-differential-privacy.html

[15] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology—EUROCRYPT*. 486–503.

[16] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proc. of the Third Conf. on Theory of Cryptography (TCC)*. 265–284. http://dx.doi.org/10.1007/11681878_14

[17] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential Privacy Under Continual Observation. In *Proc. of the Forty-Second ACM Symp. on Theory of Computing (STOC'10)*. 715–724.

[18] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N Rothblum, and Sergey Yekhanin. 2010. Pan-Private Streaming Algorithms.. In *ics*. 66–80.

[19] Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[20] Cynthia Dwork and Guy N Rothblum. 2016. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887* (2016).

[21] Alessandro Epasto, Kevin Graney, Jieming Mao, Andres Munoz Medina, Martin Pál, and Miroslava Sotakova. 2023. Differentially Private Algorithms for k-Anonymity Server. https://github.com/WICG/turtledove/blob/main/DP_kanon_server.pdf

[22] Alessandro Epasto, Jieming Mao, Andres Munoz Medina, Vahab Mirrokni, Sergei Vassilvitskii, and Peilin Zhong. 2023. Differentially private continual releases of streaming frequency moment estimations. *arXiv preprint arXiv:2301.05605* (2023).

[23] Leonidas Fegaras. 2016. Incremental query processing on big data streams. *IEEE Transactions on Knowledge and Data Engineering* 28, 11 (2016), 2998–3012.

[24] Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. 2020. A survey on the evolution of stream processing systems. *arXiv preprint arXiv:2008.00842* (2020).

[25] Monika Henzinger and Jalaj Upadhyay. 2022. Constant matters: Fine-grained Complexity of Differentially Private Continual Observation Using Completely Bounded Norms. *arXiv preprint arXiv:2202.11205* (2022).

[26] Monika Henzinger, Jalaj Upadhyay, and Sarvagya Upadhyay. 2023. Almost tight error bounds on differentially private continual counting. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 5003–5039.

[27] James Honaker. 2015. Efficient use of differentially private binary trees. *Theory and Practice of Differential Privacy (TPDP 2015), London, UK* (2015).

[28] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, and Shahzad Khan. 2019. A survey of distributed data stream processing frameworks. *IEEE Access* 7 (2019), 154300–154316.

[29] Prateek Jain, Pravesh Kothari, and Abhradeep Thakurta. 2012. Differentially Private Online Learning. In *Proc. of the 25th Annual Conf. on Learning Theory (COLT)*, Vol. 23. 24.1–24.34.

[30] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. 2021. Practical and private (deep) learning without sampling or shuffling. In *International Conference on Machine Learning*. PMLR, 5213–5225.

[31] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2017. The Composition Theorem for Differential Privacy. *IEEE Trans. Inf. Theory* 63, 6 (2017), 4037–4049. https://doi.org/10.1109/TIT.2017.2685505

[32] Gautam Kamath, Argyris Mouzakis, Matthew Regehr, Vikrant Singhal, Thomas Steinke, and Jonathan Ullman. 2023. A Bias-Variance-Privacy Trilemma for Statistical Estimation. *arXiv preprint arXiv:2301.13334* (2023).

[33] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. 2009. Releasing search queries and clicks privately. In *Proceedings of the 18th international conference on World wide web*. 171–180.

[34] Sufian Latif, Yu Hao, Hailong Zhang, Raef Bassily, and Atanas Rountev. 2020. Introducing differential privacy mechanisms for mobile app analytics of dynamic content. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 267–277.

[35] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal* 24, 6 (2015), 757–781.

[36] Jianqing Liu, Chi Zhang, and Yuguang Fang. 2018. Epic: A differential privacy framework to defend smart homes against internet traffic analysis. *IEEE Internet of Things Journal* 5, 2 (2018), 1206–1217.

[37] Yuhan Liu, Ananda Theertha Suresh, Wennan Zhu, Peter Kairouz, and Marco Gruteser. 2022. Histogram Estimation under User-level Privacy with Heterogeneous Data. *arXiv preprint arXiv:2206.03008* (2022).

[38] Jiri Matousek, Aleksandar Nikolov, and Kunal Talwar. 2014. Factorization norms and hereditary discrepancy. *arXiv preprint arXiv:1408.1376* (2014).

[39] H Brendan McMahan, Krishna Pillutla, Thomas Steinke, Abhradeep Thakurta, et al. 2024. Efficient and near-optimal noise generation for streaming differential privacy. *arXiv preprint arXiv:2404.16706* (2024).

[40] Ilya Mironov. 2017. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 263–275.

[41] Victor Perrier, Hassan Jameel Asghar, and Dali Kaafar. 2018. Private continual release of real-valued data streams. *arXiv preprint arXiv:1811.03197* (2018).

[42] Owen Phelan, Kevin McCarthy, and Barry Smyth. 2009. Using twitter to recommend real-time topical news. In *Proceedings of the third ACM conference on Recommender systems*. 385–388.

[43] Bart Samwel, John Cieslewicz, Ben Handy, Jason Govig, Petros Venetis, Chanjun Yang, Keith Peters, Jeff Shute, Daniel Tenedorio, Himani Apte, et al. 2018. F1 query: Declarative querying at scale. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1835–1848.

[44] Adam Smith and Abhradeep Thakurta. 2013. (Nearly) optimal algorithms for private online learning in full-information and bandit settings. In *Advances in Neural Information Processing Systems*. 2733–2741.

[45] Utkarsh Srivastava and Jennifer Widom. 2004. Flexible time management in data stream systems. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 263–274.

[46] Shahbaz Syed, Michael Voelske, Martin Potthast, and Benno Stein. 2017. Webis-tldr-17 corpus. *Zenodo, November* (2017).

[47] Liang Tan, Keping Yu, Ali Kashif Bashir, Xiaofan Cheng, Fangpeng Ming, Liang Zhao, and Xiaokang Zhou. 2021. Toward real-time and efficient cardiovascular monitoring for COVID-19 patients by 5G-enabled wearable medical devices: a deep learning approach. *Neural Computing and Applications* (2021), 1–14.

[48] Quoc-Cuong To, Juan Soto, and Volker Markl. 2018. A survey of state management in big data processing systems. *The VLDB Journal* 27, 6 (2018), 847–872.

[49] Salil Vadhan. 2017. The complexity of differential privacy. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 347–450.

[50] Leye Wang, Daqing Zhang, Dingqi Yang, Brian Y Lim, and Xiaojuan Ma. 2016. Differential location privacy for sparse mobile crowdsensing. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1257–1262.

[51] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2019. Differentially private SQL with bounded user contribution. *arXiv preprint arXiv:1909.01917* (2019).

[52] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 423–438. https://doi.org/10.1145/2517349.2522737

[53] Úlfar Erlingsson, Ilya Mironov, Ananth Raghunathan, and Shuang Song. 2019. That which we call private. arXiv:1908.03566 [cs.LG]