DPSAaS: Multi-Dimensional Data Sharing and Analytics as Services under Local Differential Privacy

Min Xu\textsuperscript{2}\textsuperscript{*}, Tianhao Wang\textsuperscript{1}\textsuperscript{†}, Bolin Ding\textsuperscript{1}, Jingren Zhou\textsuperscript{1}, Cheng Hong\textsuperscript{1}, Zhicong Huang\textsuperscript{1}
\textsuperscript{1}Alibaba Group \textsuperscript{2}University of Chicago \textsuperscript{3}Purdue University
\{bolin.ding, jingren.zhou, vince.hc, zhicong.hzc\}@alibaba-inc.com, xum@cs.uchicago.edu, tianhaowang@purdue.edu

ABSTRACT
Differential privacy has emerged as the de facto standard for privacy definitions, and been used, e.g., Apple, Google, Uber, and Microsoft, to collect sensitive information from users and to build privacy-preserving analytics engines. However, most of such advanced privacy-protection techniques are not accessible to mid-size companies and app developers in the cloud. We demonstrate a lightweight middleware DPSAaS, which provides differentially private data-sharing-and-analytics functionality as cloud services.

We focus on multi-dimensional analytical (MDA) queries under local differential privacy (LDP) in this demo. MDA queries against a fact table have predicates on (categorical or ordinal) dimensions and aggregate one or more measures. In the absence of a trusted agent, sensitive dimensions and measures are encoded in a privacy-preserving way locally using our LDP data sharing service, before being sent to the data collector. The data collector estimates the answers to MDA queries from the encoded data, using our data analytics service. We will highlight the design decisions of DPSAaS and twists made to LDA algorithms to fit the design, in order to smoothly connect DPSAaS to the data processing platform and analytics engines, and to facilitate efficient large-scale processing.

PVLDB Reference Format:
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION
Informed business decisions can be made from large volumes of data about user profiles and activities. In order to meet users’ expectation of their privacy, rigorous privacy guarantees need to be provided to them on how their sensitive data is collected, shared, and analyzed. To this end, the de facto privacy standard, differential privacy (DP) \cite{4}, is being used by, e.g., Apple \cite{1}, Google \cite{5}, Uber \cite{6}, and Microsoft \cite{2}. Informally, differential privacy requires that the output of a data sharing or analytics process varies little with any change in an arbitrary individual’s sensitive value.

The centralized DP model assumes that a trusted data collector maintains exact data from users, and injects noise in the analytical process to guarantee DP. For example, the solution by Uber \cite{6}, as well as PINQ and wPINQ \cite{8,9}, and PrivSQL \cite{7}, to answer SQL queries under DP is built on a SQL engine which is trustable, e.g., internally in Uber, and injects DP noise into query results.

In the absence of a trusted party, users prefer not to have their sensitive data leave their devices or workspaces in an unprotected form, and thus, the centralized DP model is no longer applicable. In such scenarios, one (e.g., enterprises \cite{1,5,2}) can adopt the local differential privacy model (LDP) \cite{3}. Each user’s sensitive data is encoded by a randomized algorithm before being sent to the data collector. LDP guarantees that the likelihood of any specific output of the algorithm varies little with input, i.e., the sensitive data. In this way, users do not need to trust the data collector.

An application scenario. LDP fits the class of analytical applications in this demo well. Suppose a number of individuals use an online shopping app. Users are anonymous. Each generates multi-dimensional data as in Table \ref{tab1}.

<table>
<thead>
<tr>
<th>Age</th>
<th>Salary</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>30K</td>
<td>50K</td>
<td></td>
</tr>
<tr>
<td>40K</td>
<td>150K</td>
<td></td>
</tr>
</tbody>
</table>

A data owner holds one or multiple rows of multi-dimensional data, or a mix of sensitive and non-sensitive data, and the rest are non-sensitive. Some attributes are measured to be aggregated in analytics, e.g., Salary, ActiveTime (how much time a user spent in the app) and Purchase.

Data owners (users in this example) prefer to have their sensitive data shared with a data collector (the app server) in a privacy-preserving way, e.g., under LDF. Note that the data collector is not trusted, and thus the privacy has to be preserved before data leaves each owner’s device or workspace. On the other side, the data collector wants to analyze how the app performs by issuing analytical queries, e.g.,

\[ Q_{\text{SUM}} = \text{SELECT SUM(Purchase) FROM } T \quad (1) \]
\[ \text{WHERE Age } \in [30, 40] \text{ AND Salary } \in [50K, 150K], \]

which aggregates Purchase under constraints on sensitive attributes.

To this end, our paper that appears in SIGMOD 2019 \cite{10} studies how to (approximately) answer a class of multi-dimensional analytical (MDA) queries, while each data owner shares the data under LDP. An MDA query is a SQL query with aggregation (e.g., COUNT, SUM, or AVG) on measure attributes and a predicate with multiple equality and range constraints on other attributes.

Demo overview. We propose and demonstrate a middleware solution DPSAaS, which enables differentially private data sharing and analytics as cloud services. Our vision with DPSAaS is to make differential privacy accessible to more “cloud users”, who can be categorized into data owners and data collectors in DPSAaS.

A data owner holds one or multiple rows of multi-dimensional data that are sensitive but to be shared with a data collector. S/he

\textsuperscript{2}Work done at Alibaba Group.
\textsuperscript{†}Work done at Alibaba Group.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx
uses our LDP data sharing service in DPSaaS to encode the sensitive data into an LDP version (with the privacy budget she desires), which does not leak much information about each row (see Definition 1). The encoding algorithm (e.g., via open source) and the encoding results are transparent to data owners. The LDP version of data can be submitted to the data collector for analytical tasks.

A data collector receives LDP version of data from a number of data owners (with the same data schema) and would like to conduct analytical study against them. Indeed, the normal query-processing engine gives meaningless output on the LDP-encoded data. Thus, the data collector uses our LDP data analytics service in DPSaaS, which is able to estimate answers to online MDA queries for the purpose of, e.g., data exploration and trend analysis. An arbitrary number of MDA queries can be issued as the privacy is guaranteed before each data owner submits LDP-encoded data.

The audience for our demo can play both roles of data owners and data collectors. For those with little background in differential privacy, they can have better intuitions on why (L)DP is a reasonable privacy notation by observing the LDP-encoded data generated by our data sharing service from a data owner’s perspective. For experts on private data analysis, they can check whether the execution of an LDP algorithm gives meaningless output on the LDP-encoded data. Thus, the data collector can estimate answers to online MDA queries for the purpose of data exploration and trend analysis.

3. ARCHITECTURE OF DPSAAS

We first introduce some design decisions of DPSaaS, in order to smoothly connect DPSaaS to the data processing platform and analytics engines, and to facilitate efficient large-scale processing.

**Overview of system design.** DPSaaS is built as a middleware (Figure 1) on top of a(n) any data processing platform, e.g., Spark. The first question is where we implement and deploy our LDP algorithms, i.e., LDP encoding algorithm A and estimation algorithm P. We have at least three alternatives: i) independent libraries, ii) inside of data processing engine, and iii) UDFs (user-defined functions). We eventually choose iii), i.e., LDP-Encoding_UDF for A and LDP_Analytics_UDAF for P in Figure 1 as it dominates the other two options in terms of both usability and efficiency. We only need to twist LDP algorithms to fit the UDF interfaces in the data processing platform. We will give more details in Section 3.1.

The second question is how these UDFs or UDAFs (user-defined aggregation functions) serve our users, i.e., data owners and data collectors. Writing SQL statements with these UDFs and UDAFs for the purposes of data sharing and analytics, respectively, would be convenient for experts; however, there is a steep learning curve for non-expert data owners and collectors. Therefore, a component called “Sharing Query Generator” in the LDP data sharing service...
3.1 Deploying LDP Algorithms via UDFs

We have considered three options to deploy LDP encoding algorithm $A$ and estimation algorithm $P$, in our data sharing and data analytics services, respectively: i) implementing $A$ and $P$ as independent libraries, ii) implementing them inside the data processing engine, and iii) implement them as user-defined functions (UDF) and user-defined aggregation functions (UDAF). We eventually choose iii), and implement LDP_Sharing_UDF for $A$ and LDP_Analytics_UDAF for $P$. There are two major reasons.

The first reason is about usability and extensibility. If we implement $A$ and $P$ as stand-alone libraries, we need to import these external libraries in the data processing platform and/or take care of data movement, which increases the complexity of our middleware. If we change the data processing engine to incorporate LDP, DPSaaS has to be tightly hooked up with a particular data platform, and is not easy to extended for others, not to mention the significant engineering efforts we have to spend for such deep integration.

The second one is about efficiency. More efforts have to be spent to make options i) and ii) able to process (including both sharing and analytics engineering efforts we have to spend for such deep integration.

Thus, we modify LDP algorithms to fit the following typical UDF and UDAF interfaces in the data processing platform.

class LDP_Sharing_UDF(BaseUDF):
    def evaluate(self, epsilon, data_tuple):
        # Implement the epsilon-LDP algorithm A
        # Output the epsilon-LDP encoded version of data_tuple

class LDP_Analytics_UDAF(BaseUDAF):
    def new_buffer(self):
        # Create a buffer to store partial sum of (3)
    def iterate(self, buf, ldp tuple, query)
        # Process one LDP-encoded tuple as in RHS of (3)
        # Update partial sum of (3)
    def merge(self, buf, pbuf):
        # Merge intermediate results (partial sums)
    def terminate(self, buf)
        # Final output (estimated answer)

It is straightforward to implement the LDP encoding algorithm $A$ in the method evaluate as the input to $A$ is also one tuple.

3.2 Query Generating and Rewriting

For our early discussion, following is an example generated by “Sharing Query Generator” for sharing sensitive data in Table I[14]:

\[
\text{SELECT OVERWRITE TABLE ldp_T1;}
\text{SELECT LDP_Sharing_UDF(2.0, Age, Salary, State) as ldp_tuple, OS, ActiveTime, Purchase FROM T;}
\]

Now, “MDA Query Rewriter” rewrites $Q_{SUM}$ in Section II to invoke the analytics UDAF and estimate its answer from ldp_T1:

\[
\text{SELECT LDP_Analytics_UDAF(ldp_tuple, Purchase, Q_{SUM_Str}) FROM ldp_T1;}
\]

where $Q_{SUM_Str}$ is a string representation of $Q_{SUM}$ (input $q$ to $P$) and will be parsed inside LDP_Analytics_UDAF: ldp_tuple are the LDP version of attributes collected from data owners.

The efficiency of the two services in DPSaaS is further boosted by the ability of distributed processing of data platforms, e.g., Spark.

3.3 Estimating Error Bars

Recall that our LDP data analytics service gives an “estimated answer” to an MDA query. We have theoretical bounds [10] of the error metric introduced in Section II. However, the big-O notations in [10] hide some constants in the errors. When it comes to showing error bars on the estimated answers to users, confidence intervals or variances could be more intuitive and accurate.

Specifically, given a query and its estimated answer, we highlight an error bar which means that the true answer lies within this range with probability over, e.g., 90%. This is easy for COUNT queries, for which error bars can be calculated from variance of $(\alpha, \beta)$-accuracy; for SUM queries, we can rely on our variance analysis to derive approximate confidence intervals; and for an AVG query, whose answer is derived by dividing SUM with COUNT, we can divide the error bars of the two to obtain a confidence interval.

3.4 Privacy Budget and Implications

DPSaaS guarantees $\epsilon$-LDP for each tuple during the data sharing service against the data collector. There are two important notes about this guarantee. First, it is dangerous to run the $\epsilon$-LDP algorithm $A$ on the same tuple multiple times; no matter which output(s) of these runs is (are) submitted to the data collector, the privacy guarantee would be weakened – for example, if outputs of $k$
runs of $\epsilon$-LDP Algorithm \textit{A} are released, we can only guarantee $k\epsilon$-LDP overall. Secondly, if there are multiple tuples about the same individual in the fact table, \textit{e.g.}, a tuple is about an individual’s daily activity (one tuple per day), DPSaaS guarantees $\epsilon$-LDP per tuple, but not per individual (not protecting one’s long-term behavior).

Meanwhile, an arbitrary number of MDA queries can be issued without using up the privacy budget, because LDP is preserved for each tuple in the data sharing service, and the analytics service can be regarded as “post-processing” of the LDP encoded fact table.

4. DEMO OVERVIEW

We demonstrate DPSaaS for two different scenarios. Namely, data sharing, where the audience role plays as a data owner who uses the LDP data sharing service in DPSaaS to share sensitive data under LDP; and data analytics, where the audience pretends to be a data analyst who has collected LDP encoded data and wish to analyze them by issuing MDA queries via our LDP data analytics service. We use Spark as the underlying data processing platform here (DPSaaS can be easily plugged into other platforms, too).

4.1 Sharing Sensitive Data

The goals of our demo for this scenario are to (a) illustrate how easy it is to use our service to enforce LDP in the data to be collected/shared; and (b) show the LDP-encoded data to their owners to give intuitions on how to process them; and (c) show how privacy budget $\epsilon$ affects errors in the estimated answers.

Figure 2 shows a simple GUI used by the each data owner to browse all the LDP encoded tuples received, which do not leak any per-tuple information about sensitive attributes. S/he can write an MDA query as a SQL statement (against the original data), and click the “Issue Query” button; DPSaaS will then rewrite this query to invoke “LDP_Analytics_UIDAF”, execute it in the processing engine, and display the estimated answer. A drag-and-drop query builder and visualization of estimated answers can also be supported for analysts who are not familiar with SQL.

During the processing of the MDA query, the column “Contribution” in the “Fact Table” is updated. It shows the contribution of each LDP encoded tuple towards the final estimated answer on the measure (Purchase in this example). More precisely, it is the RHS of (3), and is query-dependent. It can be seen that even when two tuples have the same values on sensitive attributes and measures (\textit{e.g.}, #3 and #4), they may contribute differently to the estimation, which, again, is a good property for privacy protection.

We will have the exact answer to the same query on the side for attendees to compare and find out how the error is affected by $\epsilon$. The attendees are allowed to issue as many different queries as possible without further concerns about privacy.

5. REFERENCES