FlashP: An Analytical Pipeline for Real-time Forecasting of Time-Series Relational Data

Shuyuan Yan
Alibaba Group
raul.ysy@alibaba-inc.com

Bolin Ding*
Alibaba Group
bolin.ding@alibaba-inc.com

Wei Guo
Alibaba Group
lengchuan.gw@alibaba-inc.com

Jingren Zhou
Alibaba Group
jingren.zhou@alibaba-inc.com

Zhewei Wei
Renmin University of China
zhewei@ruc.edu.cn

Xiaowei Jiang
Sheng Xu
Alibaba Group
xiaowei.jxw@alibaba-inc.com
xusheng.xs@alibaba-inc.com

ABSTRACT

Interactive response time is important in analytical pipelines for users to explore a sufficient number of possibilities and make informed business decisions. We consider a forecasting pipeline with large volumes of high-dimensional time series data. Real-time forecasting can be conducted in two steps. First, we specify the part of data to be focused on and the measure to be predicted by slicing, dicing, and aggregating the data. Second, a forecasting model is trained on the aggregated results to predict the trend of the specified measure. While there are a number of forecasting models available, the first step is the performance bottleneck. A natural idea is to utilize sampling to obtain approximate aggregations in real time as the input to train the forecasting model. Our scalable real-time forecasting system FlashP (Flash Prediction) is built based on this idea, with two major challenges to be resolved in this paper: first, we need to figure out how approximate aggregations affect the fitting of forecasting models, and forecasting results; and second, accordingly, what sampling algorithms we should use to obtain these approximate aggregations and how large the samples are. We introduce a new sampling scheme, called GSW sampling, and analyze error bounds for estimating aggregations using GSW samples. We introduce how to construct compact GSW samples with the existence of multiple measures to be analyzed. We conduct experiments to evaluate our solution its alternatives on real data.

PVLDB Reference Format:
Shuyuan Yan, Bolin Ding, Wei Guo, Jingren Zhou, Zhewei Wei, Xiaowei Jiang, and Sheng Xu. FlashP: An Analytical Pipeline for Real-time Forecasting of Time-Series Relational Data. PVLDB, 14(5): 721 - 729, 2021.
doi:10.14778/3446095.3446096

1 INTRODUCTION

Large volumes of high-dimensional data are generated on eCommerce platforms every day, from data sources about, *e.g.*, sales and browsing activities of online customers. Forecasting is among the most important BI analytical tasks to utilize such data, as sound prediction of future trends helps make informed business decisions.

Motivating scenario and challenges. An online advertising platform enables advertisers to target certain user visits and submit their bids for displaying advertisements on these visits during a time window (*i.e.*, *targeting ads campaigns*). To make the right decision about which customers to target and the bid prices, it is imperative for an advertiser to access reliable forecasts of important measures about the targeted customers and their activities, *e.g.*, Impression, the number of times an advertisement is showed to such customers, and ViewTime, the time a customer spends each visit.

Time-series data about customers can be very high-dimensional, with tens or hundreds attributes about customers' profiles and activities, including their demographics, devices (*e.g.*, mobile/PC), machine-learned tags (*e.g.*, their preferences and intents of visits), and so on. An advertiser may decide to target a group of customers for any combination of the attributes and values, based on the merchandise in the advertisement and fine-grained forecasts after different ways of slicing and dicing in the attribute space. For example, she may decide to target 20-30 year old females interested in sports and located in some cities for skirt sets in the advertisement.

Consider a time series of relation data in Figure 1. A forecasting task, *e.g.*, the one in Figure 2, asks to predict the total number of impressions by female customers under age 30.

Unlike in traditional forecasting applications such as airline planning, real-time response is critical in our scenario. First, it is an exploratory process to find the right attribute combination for targeting. An advertiser may try a number of combinations and read the forecasting results within a short period of time in order to support the decision. High latency can easily make advertisers impatient during the exploration. Second, real-time bidding for targeting ads campaigns is very competitive and dynamic market. Slow response may result in loss of displaying opportunities and higher prices. Thus, it is important to make our platform interactive.

It is challenging to provide real-time response to such forecasting tasks. The volume of input time-series data to the analytical pipeline is huge, with hundreds of millions of rows per day, and commonly months of historical data is used to forecast a future point. Moreover, as a result of the high-dimensionality of our data, there could be trillions of possible attribute/value combinations; a forecasting task is given online with any combination, and it would be too

*Bolin Ding is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 5 ISSN 2150-8097.
doi:10.14778/3446095.3446096

Age $\mathbf{a}^{(1)}$	Gender $\mathbf{a}^{(2)}$	Location $\mathbf{a}^{(3)}$	Impression $\mathbf{m}^{(1)}$	ViewTime $\mathbf{m}^{(2)}$	TimeStamp t
30	F	WA	5	1.6min	20200301
60	M	WA	1	1.8min	20200301
20	F	NY	10	3.2min	20200301
40	M	NY	20	6.3min	20200302

Figure 1: A time series of relational data (yellow cells are relevant to the task)

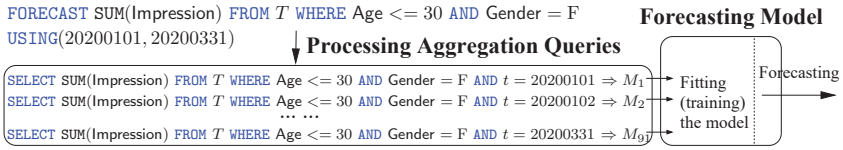


Figure 2: Processing a real-time forecasting task

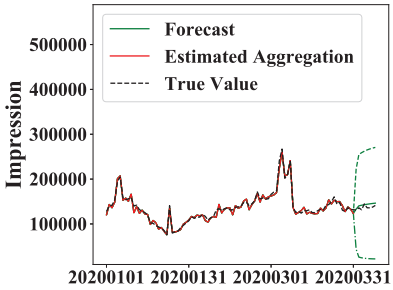


Figure 3: Forecasting example

expensive to precompute and store all possible combinations and the corresponding time series during an offline phase.

The task and solution phases. Our forecasting system FlashP is designed to process *real-time forecasting tasks*, in which we specify: i) a constraint specifying the portion of data to be focused on (*e.g.*, targeting customers "Age <= 30 AND Gender = F" in the example in Figure 2); ii) the aggregated measure to be predicted (*e.g.*, SUM(Impression)); iii) historical data points to be used to train a forecasting model (*e.g.*, from 20200101 to 20200331).

We give an overview of the solution. Consider the example in Figure 2. A forecasting task is specified in a SQL-like language, with the goal to predict the total number of impressions by female customers under age 30. To prepare training data for a forecasting model, we need to know the total number of impressions on each day (M_t), which can be written as an aggregation query. As the time-series data is partitioned on time, we can process these 91 aggregation queries with one scan of the data. After that, we have 91 data points to fit (train) the forecasting model. Processing the 91 aggregation queries (or, equivalently, a query with GROUP BY) is the bottleneck. A natural idea is to utilize sampling (*e.g.*, [21]) or sample-based approximate query processing (*e.g.*, [19, 34]) to estimate their answers. Figure 3 shows an example of prediction for the next 7 days: the red line shows estimated aggregations; the estimations are used to train the model, which then produces forecasts (green line) with confidence intervals (green dashed lines).

Contributions and organization. FlashP is implemented in Alibaba's advertising system. The first technical question is how the errors in sample-based estimations affect the fitting of forecasting models (Section 3). Error bounds of estimating aggregations using samples are well studied *e.g.*, for priority sampling [37] which has been shown to be optimal for aggregating SUM. However, it is unclear what the implications of such error bounds are in prediction results. We provide both analytical and experimental evidence showing that aggregation errors add up to the forecasting model's noise (from historical data points), and they together decide how confident we could be with the prediction. We give formal analysis for a concrete forecasting model (*i.e.*, ARMA(1, 1), defined later), and experimental results for more complicated models (*e.g.*, LSTM).

The second question is about the space budget needed for samples (Section 4). Uniform sampling provides unbiased estimations for aggregations, but the estimation error is proportional to the range of a measure (max – min) [28]. Weighted sampling schemes offer optimal estimations, with much better error bounds that are independent on the range [37]; however, as the sampling distribution is decided by the measure values [10, 21], we have to draw one weighted sample per measure independently. When there are a number of measures (as in our scenario), the total space consumption could be prohibitive. We propose a new sampling scheme called GSW (*Generalized Smoothed Weighted*) sampling, with sampling distributions that can be arbitrarily specified. We analyze estimation error bounds by quantify the correlation between the sampling distribution and measure values. We introduce how to use this scheme to generate a compact sample which takes care of multiple measures, using a sampling distribution that averages distributions of different measures, and analyze its estimation errors.

We describe how FlashP is implemented in Section 5. We report experimental results on real datasets in Section 6, and discuss related work in Section 7. All missing proofs are in the full version [41].

2 SOLUTION OVERVIEW

Time-series data model. The input to our forecasting pipeline is a *time series of relation* T, *i.e.*, a sequence of *observed rows* at successive time. We assume that time is a discrete variable here. A row in the table T is specified by a pair (i, t): an item i and a time stamp t. An item is associated with multiple *dimensions*, each denoted by $\mathbf{a}$, which are used to filter data (*e.g.*, Age and Location), and multiple *measures*, each denoted by $\mathbf{m}$, which we want to analyze and forecast (*e.g.*, Impression and ViewTime). We use $a_{i,t}$ and $m_{i,t}$ to denote the values of a dimension and a measure, respectively, on each row; or, simply, a_i and m_i, if the time stamp t is clear from the context or not important. The schema of T is $(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(d_a)}; \mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \ldots, \mathbf{m}^{(d_m)}; t)$.

Real-time forecasting task. A *forecasting tasks* is specified as:

$$\texttt{FORECAST SUM}(\mathbf{m})\ \texttt{FROM}\ T\ \texttt{WHERE C USING}\ (t_s, t_e) \quad (1)$$
$$\texttt{OPTION (MODEL = 'model\_x', FORE\_PERIOD = t\_future)}$$

$\mathbf{m}$ is the measure we want to forecast from data T on a given set of rows satisfying the constraint C. FlashP allows C to be any logical expression on the dimension values of $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(d_a)}$. We want to use historical data from time stamp t_s to t_e to fit the forecasting model. We can also specify the forecasting model we want to use in the OPTION clause with the parameter MODEL, and the number of future time stamps we want to predict with the parameter FORE_PERIOD (*e.g.*, 7 days). In most of our application scenarios, we care about SUM aggregation, *e.g.*, total number of impressions from a certain group users; FlashP can also support COUNT and AVG.

FlashP facilitates the following class of forecasting models. Let M_t be the value of metric M (= SUM($\mathbf{m}$) in our case) at time t. A

model is specified by a time-dependent function f_t with order K:

$$M_t = f_t(M_{t-1}, M_{t-2}, \ldots, M_{t-K}). \quad (2)$$

The model f_t is *fitted* on historical data $M_1, M_2, \ldots, M_{t_0}$ with *training data tuples*, each in the form of $(M_t; M_{t-1}, \ldots, M_{t-K})$ with $M_{t-1}, \ldots, M_{t-K}$ as the input to f_t and M_t as the output, for $t = t_0, t_0 - 1, \ldots, K + 1$. The fitted model can then be used to *forecast* future values $M_{t_0+1}, M_{t_0+2}, \ldots$ as $\hat{M}_{t_0+h|t_0}$ for $h = 1, 2, \ldots$, iteratively (*i.e.*, $\hat{M}_{t_0+1|t_0}$ can be used forecast M_{t_0+2}).

For the forecasting models supported by FlashP, we now give brief introduction to both classic ones, *e.g.*, ARMA [26], and those based on recurrent neural networks and LSTM [29].

Forecasting using ARMA. An ARMA (*autoregressive moving average*) [26] model uses stochastic processes to model how M_t is generated and evolves over time. It assumes that M_t is a noisy linear combination of the previous p values; each u_t is an independent identically distributed zero-mean random noise at time t, and historical noise at previous q time stamps impacts M_t too:

$$\text{ARMA}(p, q): \; M_t = \sum_{i=1}^{p} \alpha_i M_{t-i} + u_t + \sum_{i=1}^{q} \beta_i u_{t-i}. \quad (3)$$

For example, an ARMA model is: $M_t = 0.8M_{t-1} + 0.2M_{t-2} + u_t + 0.1u_{t-1}$. It falls into the form of (2), and is parameterized by $\alpha_1 \ldots \alpha_p$ and $\beta_1 \ldots \beta_q$. The model can be fitted on $M_1, \ldots, M_{t_0}$, using, *e.g.*, least squares regression, to find the values of α_i and β_j which minimize the error term, and used to forecast future values. We can estimate *forecast intervals*, *i.e.*, confidence intervals for forecasts $\hat{M}_{t_0+h|t_0}$: with a *confidence level* (probability) γ, the true future value M_{t_0+h} is within $[\hat{M}_{t_0+h|t_0} - l_\gamma, \hat{M}_{t_0+h|t_0} + r_\gamma]$.

When there are deterministic trends over time, the *differential method* can be used: the first order difference is defined $\nabla M_t = M_t - M_{t-1}$, and the second order $\nabla^2 M_t = \nabla M_t - \nabla M_{t-1}$, and in general, the d-th order $\nabla^d M_t = \nabla^{d-1} M_t - \nabla^{d-1} M_{t-1}$. If $\{\nabla^d M_t\}_t$ is an ARMA(p, q) model, $\{M_t\}_t$ is an ARIMA(p, d, q) model.

Forecasting using LSTM. LSTM (*long short-term memory*) [29] is a network architecture that extends the memory of recurrent neural networks using a *cell*. It is natural to use LSTM to learn and memorize trends and patterns of time series for the purpose of forecasting. In a typical LSTM-based forecasting model, *e.g.*, [35], an LSTM unit with dimensionality d in the output space takes $(M_{t-1}, \ldots, M_{t-K})$ as the input; the LSTM unit is then connected to a $d \times 1$ *fully-connected layer* which outputs the forecast of M_t. This model also falls into the general form (2). Here, the cell state is evolving over time and, at each time stamp t, is encoded in f_t, which can be learned from training data tuples $(M_t; M_{t-1}, \ldots, M_{t-K})$ in order.

2.1 Overview of Our Approach

Our system FlashP works in two online phases to process a forecasting task: *preparing training data points by issuing aggregation queries*, and *fitting the forecasting model using the training data.*

- *Aggregation query.* In a forecasting task, specified in (1), to predict SUM(**m**), we have $t_e - t_s + 1$ historical data points:

$$\begin{aligned} M_{t_s} &= \texttt{SELECT SUM(m) FROM } T \texttt{ WHERE C AND } t = t_s \\ &\ldots\ldots \\ M_{t_e} &= \texttt{SELECT SUM(m) FROM } T \texttt{ WHERE C AND } t = t_e \end{aligned} \quad (4)$$

  Each data point is given by an *aggregation query* with constraint C, which can be any logical expression on the dimension values of $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(d_a)}$. We would compute them in the online phase.
- *Forecasting.* In the next online phase, we use $M_{t_s}, \ldots, M_{t_e}$ as training data to fit the forecasting model. And we use the model to predict future aggregations, $M_{t_e+1}, \ldots, M_{t_e+\texttt{t\_future}}$.

Performance bottleneck. Suppose we have N rows in T for each time stamp, and use a history of $t_0 = t_e - t_s + 1$ time stamps to train a forecasting model with size (number of weights) s. The total cost of processing a forecasting task is $O(t_0 \cdot N) + \text{Train}(t_0, s)$, where $O(t_0 \cdot N)$ is the cost of processing t_0 aggregation queries by scanning the table T, and $\text{Train}(t_0, s)$ is the time needed to train a forecasting model with size s using t_0 training data tuples. $\text{Train}(t_0, s)$ is in the form of, *e.g.*, $(t_0 \cdot s \cdot \text{iter})$ where iter is the number of iterations for the model training to converge. We typically have t_0 in hundreds ($t_0 = 365$ if we use one year's history for training with one data point per day), N (number rows in T per day) in tens or hundreds of millions in our application, and s in tens. Therefore, as $t_0, s \ll N$, processing of aggregation queries is the performance bottleneck (even in comprison to training a complex model).

Real-time forecasting on approximate aggregations. In order to process a forecasting task in an interactive way in FlashP, we propose to estimate $M_{t_s}, \ldots, M_{t_e}$ as $\hat{M}_{t_s}, \ldots, \hat{M}_{t_e}$ from offline samples drawn from T, and use these estimates to form training data tuples and to fit the model (2). Several questions to be answered: i) If estimates $\hat{M}_i$, instead of M_i, are used to fit the forecasting model, how much the prediction would deviate. ii) How to draw these samples efficiently (preferably in a distributed manner). iii) How much space we need to store these samples for multiple measures.

3 REAL-TIME FORECASTING

We first analyze how sampling and approximate aggregations impact model fitting and, resultingly, forecasts. We give an analytical result for an ARMA model, and will conduct experimental study for more complex models, *e.g.*, LSTM-based ones in Section 6. It is not surprising that there is a tradeoff between sampling rate (or, quality of estimated aggregations) and forecast accuracy.

Required properties of sampling and estimates. In FlashP, we have no access to the accurate value of M_t; but instead, we have $\hat{M}_t$ estimated from offline samples. We require the estimates to have two essential properties for the fitting of forecast models:

- (*Unbiasedness*) $\hat{M}_t = M_t + \epsilon_t$ with $\mathbf{E}[\epsilon_t] = 0$;
- (*Independence*) ϵ_t's for different time stamps t are independent.

GSW *sampling* introduced in Section 4 offers unbiased estimates, with bounded variance of ϵ_t. GSW samplers are run independently on the data for each t—that is how we get independence.

Impact on ARMA(p, q)**.** When an ARMA(p, q) model has to be trained only on noisy metric values $\{\hat{M}_t\}$, we rewrite (3) as

$$\begin{aligned} \hat{M}_t = &\sum_{i=1}^{p} \alpha_i \hat{M}_{t-i} + (u_t + \epsilon_t) + \\ &+ \begin{cases} \sum_{i=1}^{p} (\beta_i u_{t-i} - \alpha_i \epsilon_{t-i}) + \sum_{i=p}^{q} \beta_i u_{t-i} & (1 < p \le q) \\ \sum_{i=1}^{q} (\beta_i u_{t-i} - \alpha_i \epsilon_{t-i}) - \sum_{i=q}^{p} \alpha_i \epsilon_{t-i} & (1 < q \le p) \end{cases} \end{aligned} \quad (5)$$

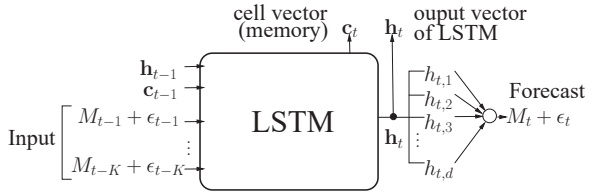


Figure 4: LSTM-based forecasting model with noisy inputs

The above model (with noisy inputs) is a combination of an autoregressive model of order p (the first line of (5)), and a moving average model of order $\max\{p, q\}$ (the two casese on the second line of (5)). It differs from the ARMA model in (3) only on the additional zero-mean error terms ϵ_t, which are independent on other terms in the model and have known variance (for fixed sampling and the estimation methods). Thus, the model can be fitted on $\{\hat{M}_t\}$ using, *e.g.*, maximum likelihood estimator, as normal ARMA models. ϵ_t increases the model's uncertainty, and, together with the model noise terms u_t, it decides the confidence of the model prediction.

For a fixed confidence level, the narrower the forecast intervals are, the more confident we are about the prediction. Their widths are proportional to the standard deviations of forecasts of $\hat{M}_t$, which in turn depends on the variance of noise terms u_t's and ϵ_t's.

In comparison to the original ARMA model in (3), the additional noise term ϵ_t will indeed incur wider forecast intervals. However, with a proper sampling-estimation scheme, if ϵ_t's variance is negligible in comparison to u_t's, ϵ_t will have little impact on the forecast error/interval, which will be demonstrated in our experiments later. Here, we give a formal analysis for the ARMA$(1, 1)$ model:

Proposition 1. *Suppose we have a time series $\{M_t\}$ satisfies* ARMA$(1, 1)$. *Let $\hat{M}_t = M_t + \epsilon_t$ be an estimation of M_t satisfying unbiasedness and independence. Then, $\mathbf{Var}[\hat{M}_t] = a \cdot \sigma_u^2 + \sigma_\epsilon^2$, where $\sigma_u^2 = \mathbf{Var}[u_t]$, $\sigma_\epsilon^2 = \mathbf{Var}[\epsilon_t]$, and $a = (1 + 2\alpha_1\beta_1 + \beta_1^2)/(1 - \alpha_1^2)$ is a constant decided by parameters in* ARMA$(1, 1)$.

We can use normal random variables to approximate u_t and ϵ_t, and estimate forecast intervals for a given confidence level.

Impact on LSTM-based model. LSTM can be applied for forecasting tasks thanks to its ability to memorize trends and patterns of time series. Figure 4 depicts such a forecasting model and illustrates where noise in the estimates impacts the model fitting.

At time stamp t, we want to forecast M_t with the previous K metric values $M_{t-1}, \ldots, M_{t-K}$ and the "memory" $(\mathbf{c}_{t-1}, \mathbf{h}_{t-1})$. However, we have only estimates $\hat{M}_{t-i} = M_{t-i} + \epsilon_{t-i}$ available; these estimates are fed into the LSTM unit as inputs. LSTM then generates an *output vector* $\mathbf{h}_t$ and update its memory *cell* from $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$. $\mathbf{h}_t$ can be interpreted as the current status of LSTM and it is used as the input to a fully-connected layer for forecasting M_t. Again, we have only $\hat{M}_t = M_t + \epsilon_t$ available as an approximation, which we learn to fit with LSTM and the fully-connected layer. $\mathbf{c}_t$ and $\mathbf{h}_t$ are used to deliver memory to the next time stamp.

Noise terms ϵ_t's may affect how the weight matrices in LSTM and the fully-connected layer are learned and the values of $\mathbf{c}_t$ and $\mathbf{h}_t$ derived (via linear transformations and activation functions). We conjecture that the LSTM-based forecasting model performs well as long as the estimates $\hat{M}_t$'s are accurate enough (or ϵ_t's are small enough). It is difficult to derive any formal analytical result here, but we will evaluate it experimentally in Section 6.

4 GENERALIZED WEIGHTED SAMPLER

We now focus on how to draw samples for estimating results of aggregation queries. An aggregation query in (4) is essentially to compute the sum of a subset of measure values in a relation T at time t; the subset is decided online by the constraint C specified in the forecasting task. W.l.o.g., suppose the subset of rows satisfying C is $[n] = \{1, \ldots, n\}$, we want to estimate the metric $M = \sum_{i=1}^{n} m_i$, for a measure $\mathbf{m} = [m_i]$. The (offline) sampling algorithm should be independent on C, but could depend on $\mathbf{m}$.

Existing samplers. There are two categories of sampling schemes for estimating subset sums: *uniform sampling* and *weighted sampling*. In uniform sampling, each row in the relation is drawn into the sample with equal probability; an unbiased estimation for M is simply the rescaled sum of values in the sample. It has been used in online aggregation extensively, but the main deficiency is that the error bound is proportional to the difference between the maximum and minimum (or, the *range* of) measure values [28].

In weighted sampling, each row i is drawn with probability proportional to m_i, so that we can remove the dependency of the estimation's error bound on the range of the measure values. More concretely, [17] and [19] give efficient implementations of such a sampling distribution: for some fixed constant τ (deciding the sampling size), if $m_i < \tau$, the probability of drawing a row i is m_i/τ, and if $m_i \geq \tau$, multiple (roughly τ/m_i) copies of i are included into the sample. Threshold sampling [20] and priority sampling [10, 21] differ from the above one in that, if $m_i \geq \tau$, exactly one copy of i is included. Priority sampling has been shown to be optimal [37] in terms of the sampling efficiency with relative standard deviation $\sqrt{\mathbf{Var}[\text{estimation}]}/M = \sqrt{1/(\text{sample\_size} - 1)}$.

Requirements for the sampler in FlashP**.** Weighted sampling offers much better sampling efficiency than uniform sampling, especially for heavy tailed distributions, which is common in practice. However, in all the above weighted sampling schemes, the sampling distribution is decided by the measure's values. In FlashP, we have d_m different measures to forecast; and thus, we have to draw d_m such weighted samples independently; when d_m is large (*e.g.*, 10), the storage cost of all the samples (*e.g.*, even with sampling rate 1%) is prohibitive in memory. Therefore, the requirement here is: *whether we can generate a compact sample which takes care of multiple measures and still have accuracy guarantees.*

To this end, we propose GSW (*Generalized Smoothed Weighted*) sampling. We introduce its sampling distribution and analyze its sampling efficiency in Section 4.1. We utilizes the "generality" of its sampling distribution to generate a compact sample which takes care of multiple measures, and analyzes under which conditions it gives provable accuracy guarantees in Section 4.2.

4.1 Generalized Smoothed Sampling

The GSW sampling scheme is parameterized by a positive constant Δ and positive sampling weights $\mathbf{w} = [w_i]$: each row i in the given relation T is drawn into the sample S_Δ with probability $\frac{w_i}{\Delta + w_i}$, *independently*. For fixed sampling weights, Δ decides the sample size; and the choice of $\mathbf{w}$ decides the estimation accuracy.

Inspired by the Horvitz–Thompson estimator [30], we define the *calibrated measure* to be $\hat{m}_i = m_i(\Delta + w_i)/w_i$ if row i is drawn into the sample S_Δ, and $\hat{m}_i = 0$ otherwise. We would associate $\hat{m}_i$ with

each sample row i and store it in S_Δ. Formally, we have

$$\begin{cases} \mathbf{Pr}[i \in S_\Delta \wedge \hat{m}_i = \frac{m_i(\Delta+w_i)}{w_i}] = \frac{w_i}{\Delta+w_i} \\ \mathbf{Pr}[i \notin S_\Delta \wedge \hat{m}_i = 0] = 1 - \frac{w_i}{\Delta+w_i} = \frac{\Delta}{\Delta+w_i} \end{cases}. \quad (6)$$

We can estimate M as $\hat{M} = \sum_{i \in S_\Delta} \hat{m}_i$, which is unbiased since

$$\mathbf{E}\left[\hat{M}\right] = \sum_{i=1}^{n} \mathbf{E}[\hat{m}_i] = \sum_{i=1}^{n} \frac{m_i(\Delta + w_i)}{w_i} \cdot \frac{w_i}{\Delta + w_i} = M.$$

Simple and efficient implementations. A GSW sampler can be easily implemented in a distributed manner: each row i generates uniformly random number p_i from $[0, 1]$, independently; according to (6), if $p_i \le \frac{w_i}{\Delta+w_i}$, the row i is put into the sample S_Δ.

A GSW sample S_Δ can be maintained in an incremental way. Suppose we have drawn S_Δ from rows $[n]$ for some fixed Δ. If more rows $n+1, \ldots, n+k$ are coming, we want to increase Δ to Δ' and obtain a GSW sample $S_{\Delta'}$ from $[n+k]$ with size comparable to $|S_\Delta|$. Suppose rows in S_Δ are sorted by $(\frac{1}{p_i} - 1)w_i$ in an ascending order; we only need to delete those with $\Delta \le (\frac{1}{p_i} - 1)w_i < \Delta'$ from S_Δ, and insert those with $\Delta' \le (\frac{1}{p_i} - 1)w_i$, for $i = n+1, \ldots, n+k$. In this way, we update S_Δ to $S_{\Delta'}$ *without* touching any row in $[n] - S_\Delta$.

4.1.1 Accuracy Guarantee on Aggregations. We now analyze estimation errors of the class of GSW-based estimators $\hat{M}$, instantiated by $(\Delta, \mathbf{w})$. We consider *relative standard deviation* (RSTD) and *relative error* (RE). As $\hat{M}$ is unbiased,

$$\mathrm{RSTD}(\hat{M}) \triangleq \sqrt{\mathbf{E}\left[\left(\frac{\hat{M} - M}{M}\right)^2\right]} \ge \mathbf{E}\left[\frac{|\hat{M} - M|}{M}\right] \triangleq \mathrm{RE}(\hat{M}).$$

The goal of our analysis is to establish a relationship between the sample size and the (expected) RSTD and RE. Intuitively, when the sampling weight w_i is "consistent" with the measure m_i, the estimation error is minimum (for fixed sample size). We introduce the following notation to quantify the "consistency".

Definition 2. **($(\underline{\theta}, \overline{\theta})$-consistency)** *Sampling weights $\mathbf{w} = [w_i]$ are $(\underline{\theta}, \overline{\theta})$-consistent with measure $\mathbf{m} = [m_i]$, iff $\underline{\theta} = \min_i(m_i/w_i)$ and $\overline{\theta} = \max_i(m_i/w_i)$. $\theta \triangleq \overline{\theta}/\underline{\theta}$ is called the consistency scale.*

The above notation about "consistency" says, for any row i, $\underline{\theta} \le m_i/w_i \le \overline{\theta}$. It allows $\mathbf{m}$ and $\mathbf{w}$ to differ in scale (*e.g.*, both $\underline{\theta}$ and $\overline{\theta}$ could be large) but measures their similarity in patterns and trends. For example, suppose $\mathbf{m} = [100, 100, 200, 400]$ and $\mathbf{w} = [10, 10, 20, 50]$. We have $\underline{\theta} = 400/50 = 8$, $\overline{\theta} = 10$, and thus $\theta = 10/8 = 1.25$. In general, we have $\theta \ge 1$, and the following theorem shows that, the relative error has an upper bound that is proportional to $\sqrt{\theta}$.

Theorem 3. **(Sampling efficiency of** GSW**)** *Suppose the sampling weights $\mathbf{w}$ used in* GSW *sampling are $(\underline{\theta}, \overline{\theta})$-consistent with measure values $\mathbf{m}$, the estimate $\hat{M}$ is unbiased and has expected relative standard deviation and error bounded by: ($\theta \triangleq \overline{\theta}/\underline{\theta}$)*

$$\mathrm{RE}(\hat{M}) \le \mathrm{RSTD}(\hat{M}) \le \sqrt{\frac{\overline{\theta}/\underline{\theta}}{\mathbf{E}[|\mathcal{S}_\Delta|]}} = \sqrt{\frac{\theta}{\mathbf{E}[|\mathcal{S}_\Delta|]}}.$$

An open question raised by Alon *et al.* when introducing priority sampling [10] is: whether we can provide any error bound if subset sum on a measure ($\mathbf{m}$) is estimated using a priority sample drawn based on a different measure ($\mathbf{w}$). Theorem 3 answers this question in GSW sampling by specifying under what condition ($(\underline{\theta}, \overline{\theta})$-consistency)) there is an error bound.

4.1.2 A Special Case: Optimal GSW Sampler. We can choose $\mathbf{w}$ to minimize the variance of estimation, while the expected sample size is bounded. Please refer to [41] for a precise formulation and the optimal solution. From Theorem 3, a nearly-optimal solution is $\mathbf{w} = \mathbf{m}$, as in this case, $\mathbf{w}$ is $(1, 1)$-consistent with $\mathbf{m}$ ($\theta = 1$). We call it *optimal* GSW *sampler*. Directly from Theorem 3, we have

Corollary 4. **(Optimal** GSW **sampler)** *If we use $\mathbf{w} = \mathbf{m}$ ($w_i = m_i$ for each row i) as the sampling weights, we have*

$$\mathrm{RE}(\hat{M}) \le \mathrm{RSTD}(\hat{M}) \le \sqrt{\frac{1}{\mathbf{E}[|\mathcal{S}_\Delta|]}}.$$

The optimal GSW sampler has sampling efficiency that is comparable to the best known sampler for estimating subset sums, *e.g.*, priority sampling [10] with $\mathrm{RSTD} = \sqrt{1/(\mathrm{sample\_size} - 1)}$. We will compare their empirical performance in Section 6.

4.2 Compact Sample for Multiple Measures

The size of GSW sample can be controlled by the parameter Δ. When there is only one measure $\mathbf{m}$, we draw an optimal GSW sample (setting $\mathbf{w} = \mathbf{m}$). A more common scenario is that we have multiple measures (*e.g.*, in Figure 1) in one relation. We can draw one optimal GSW sample per each measure, which, however, increases the space consumption significantly. The question is whether we can use one sample to take care multiple measures.

Suppose there are k measures in a relation to be aggregated and predicted, $\mathbf{m}^{(1)}, \ldots, \mathbf{m}^{(k)}$. For each measure j, we want to estimate $M^{(j)} = \sum_{i=1}^{n} m_i^{(j)}$ for rows in a set $[n]$ (satisfying the constraint C in a forecasting task). A GSW sample S_Δ is drawn using weights $\mathbf{w} = [w_i]$. The *calibrated measure* on each sample row i for each measure j is $\hat{m}_i^{(j)} = m_i^{(j)}(\Delta + w_i)/w_i$. From Theorem 3, $\hat{M}^{(j)} = \sum_{i \in S_\Delta} \hat{m}_i^{(j)}$ is an unbiased estimation of $M^{(j)}$.

Intuitively, if the chosen sampling weight vector $\mathbf{w}$ centers around $\mathbf{m}^{(1)}, \ldots, \mathbf{m}^{(k)}$ and is not too far away from any of the k, from Theorem 3, the error can be better bounded. We can find such centers by taking the average of measures. For example, for $\mathbf{m}^{(1)} = [100, 100, 200, 400]$ and $\mathbf{m}^{(2)} = [1, 1, 2, 1]$, the *geometric mean* is $\mathbf{w}^\times = [\sqrt{100 \cdot 1} = 10, 10, 20, 20]$, and the *arithmetic mean* is $\mathbf{w}^+ = [(100+1)/2 = 50.5, 50.5, 101, 200.5]$. We now analyze how the error can be bounded for these two choices.

Geometric compressed GSW **sampling.** We can use the geometric mean of the k measures as the sampling weights:

$$w_i^\times = \left(\prod_{j=1}^{k} m_i^{(j)}\right)^{1/k}. \quad (7)$$

Among the k measures to be grouped, define the *trend deviation* between any two measures $\mathbf{m}^{(p)}, \mathbf{m}^{(q)}$ (for $p, q \in [k]$):

$$\overline{\rho}_{p,q} \triangleq \max_{i \in [n]} \frac{m_i^{(p)}}{m_i^{(q)}}, \ \underline{\rho}_{p,q} \triangleq \min_{i \in [n]} \frac{m_i^{(p)}}{m_i^{(q)}}, \text{ and } \rho_{p,q} \triangleq \frac{\overline{\rho}_{p,q}}{\underline{\rho}_{p,q}}, \quad (8)$$

which measures how similar the trends (instead of their absolute values) of the two measures are[1]. The smaller $\rho_{p,q}$ is, the more similar $\mathbf{m}^{(p)}$ and $\mathbf{m}^{(q)}$ are. For example, if $\mathbf{m}^{(p)} = c \cdot \mathbf{m}^{(q)}$ for a constant c, $\rho_{p,q} = c/c = 1$. Define $\rho \triangleq \max_{p,q\in[k]} \rho_{p,q}$ to be the *maximum deviation* among the k measures. From Theorem 3,

Corollary 5. *If we use* $\mathbf{w}^{\times} = [w_i^{\times}]$ *as the sampling weights for a relation with* k *measures, with a* GSW *sample* S_Δ*, we can estimate* $M^{(p)}$ *as* $\hat{M}^{(p)}$ *for each measure* p *with error*

$$\mathrm{RE}(\hat{M}^{(p)}) \le \mathrm{RSTD}(\hat{M}^{(p)}) \le \sqrt{\frac{\prod_{j:\ j\neq p} \rho_{p,j}^{1/k}}{\mathbf{E}[|\mathcal{S}_\Delta|]}} \le \sqrt{\frac{\rho^{\frac{k-1}{k}}}{\mathbf{E}[|\mathcal{S}_\Delta|]}}.$$

Arithmetic compressed GSW **sampling.** Another choice is to use the arithmetic mean as the sampling weights:

$$w_i^{+} = \frac{1}{k}\sum_{j=1}^{k} m_i^{(j)}. \tag{9}$$

Define the *range deviation* δ among k measures: for each row i, consider the ratio between the maximum measure and the minimum one; δ is the maximum ratio among all rows:

$$\delta \triangleq \max_{i\in[n]} \left(\frac{\max_{j\in[k]} m_i^{(j)}}{\min_{j\in[k]} m_i^{(j)}} \right). \tag{10}$$

From the definitions, for any measure p, we have $1/\delta \le m_i^{(p)}/w_i^{+} \le \delta$. Thus, directly from Theorem 3, we have the following bound.

Corollary 6. *If we use* $\mathbf{w}^{+} = [w_i^{+}]$ *as the sampling weights for a relation with* k *measures, with a* GSW *sample* S_Δ*, we can estimate* $M^{(p)}$ *as* $\hat{M}^{(p)}$ *for each measure* p *with error*

$$\mathrm{RE}(\hat{M}^{(p)}) \le \mathrm{RSTD}(\hat{M}^{(p)}) \le \sqrt{\frac{\delta^2}{\mathbf{E}[|\mathcal{S}_\Delta|]}}.$$

How to group measures? In the above, we have shown that, for chosen sampling weights, how the error can be bounded with some parameters (ρ and δ) decided by the data about a group of measures. When there are many (*e.g.*, $k = 100$) measures, ρ and δ could be huge and thus the above error bounds are not informative. We want to partition measures into small groups of size 4-5 based on their correlation, so that within each group one GSW sample gives good estimations for the measures. To this end, we establish a relationship between $(\underline{\theta}, \overline{\theta})$-consistency and L_1 distance as follows.

Proposition 7. **(consistency and L_1 distance)** *Suppose sampling weights* $\mathbf{w} = [w_i]$ *are* $(\underline{\theta}, \overline{\theta})$*-consistent with measure* $\mathbf{m} = [m_i]$*. We normalize* $\mathbf{w}$ *as* $\mathbf{w}' = [w_i' = w_i/\sum_j w_j]_i$*, and similarly,* $\mathbf{m}$ *as* $\mathbf{m}'$*. Let* $\theta = \overline{\theta}/\underline{\theta}$*. We have* $\|\mathbf{m}' - \mathbf{w}'\|_1 \le (\theta - 1)$.

$(\underline{\theta}, \overline{\theta})$-consistency is a "worst-case" notion about all the rows. It is not a good metric for grouping measures, because first, it is expensive to compute θ, and second, an aggregation query may not touch all the rows. Instead, we use L_1 distance (after normalization) as in Proposition 7 to quantify the *correlation* between two measures. Intuitively, it quantifies how two measures are similar in trends and patterns regardless of their absolute values.

[1]Or, equivalently, $\mathbf{m}^{(q)}$ is $(\underline{\rho}_{p,q}, \overline{\rho}_{p,q})$-consistent with $\mathbf{m}^{(p)}$.

We consider a formulation based on the KCenter problem [27]. The goal is to partition the measures into g groups so that the max L_1 distance (after normalization) between any measure to the center of the group it belongs to is minimized. The L_1 distance between two measures can be estimated using a sample of rows. We apply the standard greedy algorithm [27] to find a 2-approximation. For each group, we use the geometric/arithmetic mean as the sampling weight vector. It is a heuristic strategy without any formal guarantee, but from Proposition 7 and the triangle inequality in L_1, at least we know that we'd better not group two measures that are far way (*e.g.*, $> 2(\theta - 1)$) together, as there is no sampling weight vector that is consistent with both of them at the same time. A preliminary evaluation of this grouping strategy can be found in [41].

5 DEPLOYMENT

Offline Sample Preprocessor of FlashP is built on Alibaba's distributed data storage and analytics service, MaxCompute [1]. Time series of relations, partitioned by time, are stored in MaxCompute's data warehouse. Relations can be joined, *e.g.*, tables UserProfile and AdTraffic are joined on UserID. GSW sampler is implemented as UDFs (user-defined functions), and draws samples from one relation or the view of joined relation. A set of samples of different sizes (with increasing values of Δ) are drawn and stored as Multi-layer Samples of Relations for response time-accuracy tradeoff.

Online Forecasting Service first pulls Multi-layer Samples of Relations into Alibaba's in-memory OLAP engine, Hologres [1], to enable real-time response. There are 30 servers in the cluster to support this service, each with 96 CPU cores and 512G memory. Sample data is partitioned by time in the OLAP engine.

Users submit forecasting tasks to an Application Server via a Web UI. For a task, aggregation queries in (4) are generated from Query Rewriter and processed on samples in the OLAP engine to obtain estimated answers. These estimations are used as training data to fit the Forecasting Model, which is built on a Python server.

Two models are implemented. One is ARIMA. An open-source library [2] (built on X-13ARIMA-SEATS [3]) that trains ARIMA and automatically tunes for the best values of parameters p, d, q is used. We also support an LSTM-based model (in Figure 4), which is implemented using Keras [4]. We use the LSTM and fully-connected layers in Keras with $K = 7$ and $d = 4$ as the default parameter setting. Other forecasting models can be plugged in here, too. After fitting the model, we send forecasts back to users.

6 EXPERIMENTAL EVALUATION

We evaluate our system FlashP under the implementation and hardware specified in Section 5, on a real-life dataset with 11 dimensions about users' profiles and 4 numeric measures to be predicted, including Favorite, Impression, Click, and Cart. There are around 15 million rows per day, and 200 days of data.

We compare different samplers: **Uniform** sampling, which is also used in [7], is the baseline; **Priority**, the optimal weighted sampler [21]; our **Optimal** GSW and **Arithmetic/Geometric compressed** GSW introduced in Section 4. We also compare our sampling based methods with **PIM** (Partwise Independence Model) [7] based on a Bayesian model assuming partially independence.

Table 1: A summary of results (0.1% sample, Opt-GSW = Optimal GSW, C-GSW = Arithmetic compressed GSW)

	Full	PIM	Uniform	Opt-GSW	C-GSW
Favorite	0.105	0.695	0.248	0.131	0.196
Impression	0.140	0.374	0.147	0.142	0.144
Click	0.157	0.681	0.161	0.151	0.153
Cart	0.704	1.931	0.718	0.704	0.709

In the following, forecasting tasks are randomly picked with different measures to be predicted and different combinations of dimensions in their constraints, with some (approximately) fixed *selectivity* (the fraction of rows satisfying the constraint). By default, we use 150 days' data to fit the model and predict the next 7 days; we report *relative aggregation errors* (average of the 150 days), *relative forecast error*, and *forecast intervals* (average of the next 7 days), taking the average of 400 independent runs of different tasks, together with one standard deviation, for each measure and for each value of selectivity (on independent samples).

Exp-I: A summary of results. We first give a brief summary of experimental results. Table 1 reports the average forecast errors on 20 random tasks with selectivity from 0.5% to 10% using ARIMA. "Full" stands for the result when we use the full data to process aggregation queries for training. With a sampling rate 0.1%, our optimal GSW and compressed GSW in FlashP perform consistently better than Uniform and PIM in terms of forecast errors, and sometimes are very close to Full (the best we can do). These two also offer interactive response time (less than 100ms). In the rest part, we will report more detailed results about response time and performance of different sampling-based methods.

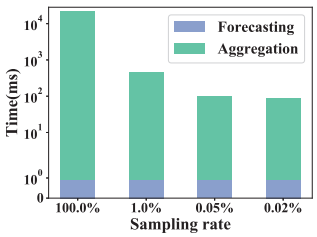
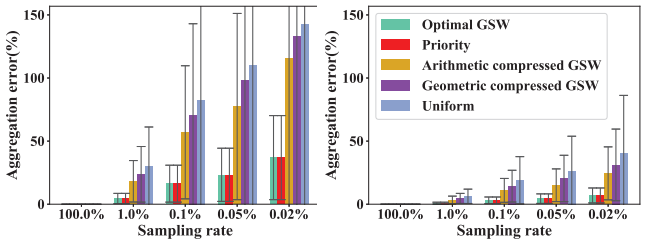


Figure 5: E2E response time (with ARIMA)

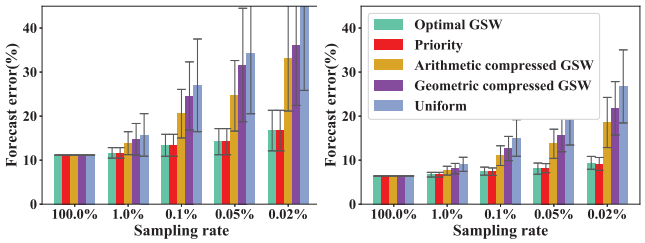
Exp-II: Real-time response. The end-to-end response time is reported in Figure 5, partitioned into the portion for processing (estimating) aggregation queries, and the portion for forecasting (model fitting + prediction using ARIMA). It can be seen that the portion for aggregation queries is the bottleneck, but with sampling, the response time can be reduced from around 20sec on the full data to 30ms on a 0.02% sample, which still gives reasonable prediction as will be shown later. If LSTM is used, the model fitting is much more expensive, but we still have an interactive response time around 1sec if a 1% sample is used.

Exp-III: Varying number of time stamps used in training. We consider different numbers of training data points used to fit the ARIMA and LSTM models. Please refer to the full version [41] for more details. It shows that the number of time stamps used to fit model has an obvious impact on the forecast error, with 150 (days) giving the most accurate and stable prediction for both ARIMA and LSTM. It motivates us to speedup the processing of aggregation queries, as more time stamps mean more aggregation queries.



(a) Selectivity 0.5% (b) Selectivity 5%

Figure 6: Aggregation error of different sampling methods for varying selectivity and sampling rate (Favorite**)**



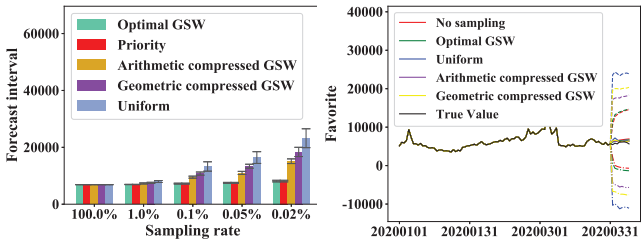
(a) ARIMA (b) LSTM

Figure 7: Forecast error of different sampling methods for varying sampling rate (selectivity 0.5%, Favorite**)**

Exp-IV: Varying sampling rate and selectivity. We compare different samplers in FlashP, for varying sampling rate and selectivity. Note that for Arithmetic/Geometric compressed GSW, one sample suffices for the relation; Priority are Optimal GSW are measure-dependent, and thus using either of them we need four samples (one per measure), with the total space consumption four times of Uniform and compressed GSW for a fixed sampling rate.

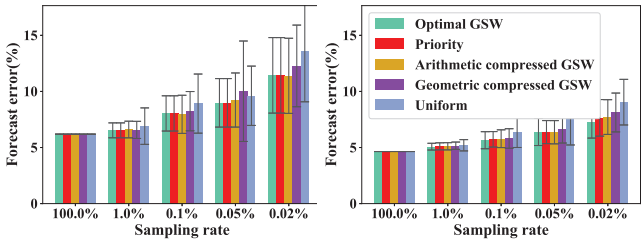
For tasks on Favorite with selectivity 0.5%-5%, Figure 6 reports aggregation errors, and Figure 7 reports forecast errors when using ARIMA and LSTM as forecasting models; the results on Impression are plotted in Figure 9 (those with selectivity 5% can be found in the full version [41]). In terms of both aggregation errors and forecasting errors, Priority and Optimal GSW are very close and better than the others (indeed, at the cost of storing four samples). In some cases, Optimal GSW is even slightly better than Priority (theoretically optimal). This is because, in Priority, if the measure is above some threshold, a row is included in the sample deterministically, which favors the long tail; however, the long tail may or may not satisfy the constraint specified online in the forecasting task. Uniform is the worst one which is consistent with its analytical error bound [28]. Arithmetic/Geometric compressed GSW needs only one sample, too; they are better than Uniform, and get very close to Priority and Optimal GSW when the sampling rate is close to 1%. For larger selectivity, with more rows satisfying the constraint included in the samples, every sampler gets better.

Figure 8(a) reports widths of forecast intervals of ARIMA with a confidence level of 90% on measure Favorite for varying sampling rate. With larger sampling rate, every sampler gives narrower forecast intervals, meaning predictions with more confidence. Uniform



(a) Varying sampling rate (b) Forecast intervals of a real query on 0.02% sampling

Figure 8: Forecast intervals (ARIMA) with different sampling methods for varying sampling rate (selectivity 0.5%, Favorite**)**



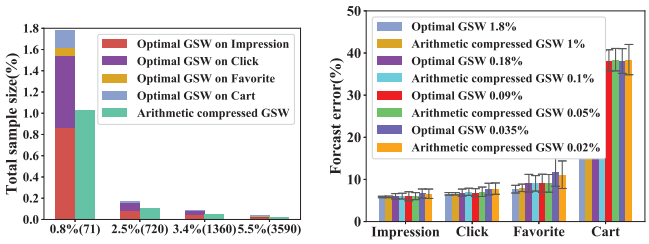
(a) ARIMA (b) LSTM

Figure 9: Forecast error of different sampling methods for varying sampling rate (selectivity 0.5%, Impression**)**

gives the widest one and Priority and Optimal GSW give the narrowest ones. Figure 8(b) shows the forecast intervals (dashed lines) for one particular task using different samplers.

In terms of forecasting, LSTM performs consistently better than ARIMA, at the cost of longer response time (as discussed in Exp-II). It is observed that, with increasing sampling rates, when the aggregation error is small enough (*e.g.*, when sampling rate = 1% in Figures 6-9), it will have little impact on the forecast error/interval in comparison to the case when we use the full data (sampling rate = 100%), because it is negligible in comparison to the model and data's noise (*e.g.*, u_t in (3) for ARIMA). Forecast errors and forecast intervals have similar trends as aggregation errors. Both observations are consistent with our analytical results in Section 3.

Exp-V: Space cost under the same accuracy requirement. We evaluate the space cost needed to achieve the same accuracy for different samplers. Since Priority and Optimal GSW perform similarly, we focus on Optimal GSW and compare it with Arithmetic compressed GSW. We fix the sample size of Arithmetic compressed GSW (from 0.02% to 1%). And for each measure, we choose the size of an Optimal GSW sample so that it gives approximately the same aggregation error as Arithmetic compressed GSW does. In Figure 10(a), we report the total size of the four Optimal GSW samples (the portions for different measures are stacked and labeled with different colors), and the size of the Arithmetic compressed GSW sample; x-axis is the max aggregation error in Arithmetic compressed GSW with the parameter Δ in brackets. It shows that, under the same accuracy requirement, the total size of Optimal GSW samples is around 1.8 times of the size of Arithmetic compressed GSW. Figure 10(b) reports forecast errors (of ARIMA) using



(a) Optimal v.s. Compressed (b) Forecast error (selectivity 5%)

Figure 10: Space cost for given accuracy requirement

the samples chosen above on different measures. Due to the way how we choose the sizes of Optimal GSW samples, Optimal GSW and Arithmetic compressed GSW give very close forecast errors.

7 RELATED WORK

Approximate query processing and other samplers. An orthogonal line of work is approximate query processing (AQP), which has been studied extensively during last few decades. One can refer to [16] for a comprehensive review. There are two major lines of AQP techniques. i) Online aggregation [18, 28, 33] and online sampling-based AQP [31] either assume that the data is randomly ordered, or need to draw random rows from the data table via random I/O accesses which is inefficient in our setting. ii) Offline sampling-based AQP draws offline samples before queries come: some are based on historical workloads [5, 8, 14, 15, 22, 32, 36], and some are workload-independent [5, 6, 11, 13, 19, 34].

The most relevant part in the line of AQP techniques are the samplers proposed to estimate aggregations. We have reviewed such samplers at the beginning of Section 4. There are other samplers such as universe (hashed) sampling and stratified sampling introduced in AQP systems [8, 31, 34]. These samplers were proposed to handle orthogonal aspects such as missing groups in `GroupBy` and joins. They can also be used in our system if we want to extend the task class and data schema we want to support.

Precomputing aggregations. Another choice is to precompute aggregations or summaries using techniques such as view materialization [9], datacube [25], histograms [23], and wavelets [12, 23, 39]. These techniques either have too large space overhead (super-linear) to be applicable for enterprise-scale high-dim datasets, or cannot support complex constraints in our forecasting tasks.

Aggregation-forecasting analytics. [7] studies a very similar problem of forecasting multi-dimensional time series. It considers capturing correlations across the high-dimensional space using either Bayesian models or uniform sampling, and shows that the one based on uniform sampling (which is also evaluated in our experimentes) offers much better forecast accuracy. Another relevant line of work is about aggregation-disaggregation techniques [38, 40] in the forecasting literature. These techniques share some similarity with the Bayesian model in [7], but focus on one-time offline analysis with smaller scale and lower dimensional data.

For multi-dimensional time series, there are works about how to conduct fast similarity search [24], but they are less relevant here.

ACKNOWLEDGMENTS

We thank the reviewers for their suggestions that improved the quality of the paper. Zhewei Wei was supported by NSFC No.61832017, No. 61972401 and No. 61932001, by the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 18XNLG21, by Beijing Outstanding Young Scientist Program NO. BJJWZYJH01201910002009, and by Alibaba Group through Alibaba Innovative Research Program.

REFERENCES

[1] [n.d.]. https://www.alibabacloud.com. [Online; accessed 1/15/2021].
[2] [n.d.]. https://pypi.org/project/pmdarima/. [Online; accessed 1/15/2021].
[3] [n.d.]. https://www.statsmodels.org/stable/generated/statsmodels.tsa.x13.x13_arima_analysis.html. [Online; accessed 1/15/2021].
[4] [n.d.]. https://keras.io/. [Online; accessed 1/15/2021].
[5] Swarup Acharya, Phillip B Gibbons, and Viswanath Poosala. 2000. Congressional samples for approximate answering of group-by queries. In *SIGMOD*. 487–498.
[6] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The Aqua approximate query answering system. In *SIGMOD*. 574–576.
[7] Deepak Agarwal, Datong Chen, Long-ji Lin, Jayavel Shanmugasundaram, and Erik Vee. 2010. Forecasting high-dimensional data. In *SIGMOD*. 1003–1012.
[8] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Eurosys*. 29–42.
[9] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R Narasayya. 2000. Automated Selection of Materialized Views and Indexes in SQL Databases.. In *VLDB*. 496–505.
[10] Noga Alon, Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2005. Estimating arbitrary subset sums with few probes. In *PODS*. 317–325.
[11] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic Sample Selection for Approximate Query Processing. In *SIGMOD*. 539–550.
[12] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. 2001. Approximate query processing using wavelets. *VLDBJ* 10, 2-3 (2001), 199–223.
[13] Surajit Chaudhuri, Gautam Das, Mayur Datar, Rajeev Motwani, and Vivek Narasayya. 2001. Overcoming limitations of sampling for aggregation queries. In *ICDE*. 534–542.
[14] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2001. A robust, optimization-based approach for approximate answering of aggregate queries. In *SIGMOD*. 295–306.
[15] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *TODS* 32, 2 (2007), 9.
[16] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *SIGMOD*. 511–519.
[17] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On random sampling over joins. In *SIGMOD*. 263–274.
[18] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. 2010. MapReduce Online. In *NSDI*. 313–328.
[19] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. 2016. Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee. In *SIGMOD*. 679–694.
[20] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2005. Learn more, sample less: control of volume and variance in network measurement. *IEEE Trans. Information Theory* 51, 5 (2005), 1756–1775.
[21] Nick G. Duffield, Carsten Lund, and Mikkel Thorup. 2007. Priority sampling for estimation of arbitrary subset sums. *J. ACM* 54, 6 (2007), 32.
[22] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. 2000. ICICLES: Self-Tuning Samples for Approximate Query Answering.. In *VLDB*. 187.
[23] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Marin J Strauss. 2001. Optimal and approximate computation of summary statistics for range aggregates. In *PODS*. 227–236.
[24] Xudong Gong, Yan Xiong, Wenchao Huang, Lei Chen, Qiwei Lu, and Yiqing Hu. 2015. Fast Similarity Search of Multi-Dimensional Time Series via Segment Rotation. In *DASFAA*. 108–124.
[25] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.* 1, 1 (1997), 29–53.
[26] James D. Hamilton. 1994. *Time Series Analysis*. Princeton University Press.
[27] Sariel Har-peled. 2011. *Geometric Approximation Algorithms*. American Mathematical Society.
[28] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. *SIGMOD* (1997), 171–182.
[29] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[30] D. G. Horvitz and D. J. Thompson. 1952. A Generalization of Sampling Without Replacement From a Finite Universe. *J. Amer. Statist. Assoc.* 47, 260 (1952), 663–685.
[31] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex AdHoc Queries in BigData Clusters. In *SIGMOD*. 631–646.
[32] Christopher Olston, Edward Bortnikov, Khaled Elmeleegy, Flavio Junqueira, and Benjamin Reed. 2009. Interactive Analysis of Web-Scale Data. In *CIDR*.
[33] Niketan Pansare, Vinayak R. Borkar, Chris Jermaine, and Tyson Condie. 2011. Online Aggregation for Large MapReduce Jobs. *PVLDB* 4, 11 (2011), 1135–1145.
[34] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *SIGMOD*. 1461–1476.
[35] Jürgen Schmidhuber, Daan Wierstra, and Faustino J. Gomez. 2005. Evolino: Hybrid Neuroevolution/Optimal Linear Search for Sequence Learning. In *IJCAI*. 853–858.
[36] Lefteris Sidirourgos, Martin L. Kersten, and Peter A. Boncz. 2011. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *CIDR*. 296–301.
[37] Mario Szegedy. 2006. The DLT priority sampling is essentially optimal. In *STOC*. 150–158.
[38] Justin Tobias and Arnold Zellner. 2000. A note on aggregation,disaggregation and forecasting performance. *Journal of Forecasting* 19, 5 (2000), 457–469.
[39] Jeffrey Scott Vitter and Min Wang. 1999. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *SIGMOD*. 193–204.
[40] Mike West and Jeff Harrison. 1997. *Bayesian Forecasting and Dynamic Models*. Springer-Verlag.
[41] Shuyuan Yan, Bolin Ding, Wei Guo, Jingren Zhou, Zhewei Wei, Xiaowei Jiang, and Sheng Xu. 2021. FlashP: An Analytical Pipeline for Real-time Forecasting of Time-Series Relational Data. arXiv:2101.03298 [cs.DB]