Scalable Time-Decaying Adaptive Prediction Algorithm

Yinyan Tan[1], Zhe Fan[1], Guilin Li[1], Fangshan Wang[1], Zhengbing Li[1], Shikai Liu[1], Qiuling Pan[1]
Eric P. Xing[2], Qirong Ho[2]

[1]Research and Standard Department, Huawei Software Technologies CO. LTD
[2]School of Computer Science, Carnegie Mellon University
tanyinyan, fanzhe, liguilin, wangfangshan, zhengbing.li, liushikai, panqiuling@huawei.com
epxing, qho@cs.cmu.edu

ABSTRACT

Online learning is used in a wide range of real applications, *e.g.,* predicting ad click-through rates (CTR) and personalized recommendations. Based on the analysis of users' behaviors in Video-On-Demand (VoD) recommender systems, we discover that the most recent users' actions can better reflect users' current intentions and preferences. Under this observation, we thereby propose a novel time-decaying online learning algorithm derived from the state-of-the-art FTRL-proximal algorithm, called Time-Decaying Adaptive Prediction (TDAP) algorithm.

To scale Big Data, we further parallelize our algorithm following the data parallel scheme under both BSP and SSP consistency model. We experimentally evaluate our TDAP algorithm on real IPTV VoD datasets using two state-of-the-art distributed computing platforms, *i.e.,* Spark and Petuum. TDAP achieves good accuracy: it improves at least *5.6%* in terms of prediction accuracy, compared to FTRL-proximal algorithm; and TDAP scales well: it runs 4 times faster when the number of machines increases from 2 to 10. In our real running business cases, TDAP significantly increases the degree of user activity, which brings more revenue than existing ones for our customers.

1. INTRODUCTION

Online learning techniques [3, 9, 19, 24, 26, 30, 32] are powerful tools for a wide range of emerging applications, from online advertising [25] and personalized recommendation [2, 28], to unusual events detection [35] and suspicious URLs identification [23]. For example, in an online IPTV Video-On-Demand (VoD) recommender system for one of our customers, a giant Telcom company from Mainland China, the accuracy of prediction (*e.g.,* AUC) can be significantly improved at least 4% using online learning techniques compared to traditional batch (offline) learning techniques [12, 18, 37].

KDD '16, August 13–17, 2016, San Francisco, CA, USA

ISBN 978-1-4503-4232-2/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939714

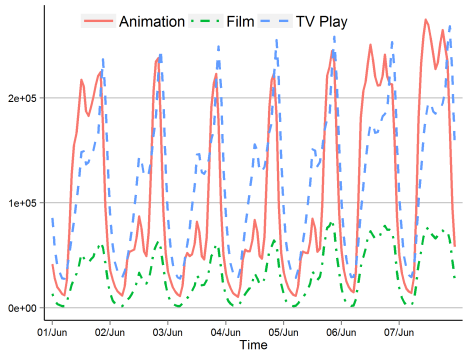

Figure 1: The fast changing users' watching counts on different types of videos (Film, TV play and Animation) from 2015-06-01 to 2015-06-07 on an online IPTV VoD service.

To date, people and devices (from smart phones, to VR Boxes, to coffee machines and cars) generate large volume of data every day, which are with *fast-varying* (or *fast-changing*) nature (*a.k.a.* concept drifting [11]). However, existing online learning techniques may *not* be a good fit for such fast-changing data, which are usually in form of examples with features and labels in practice. Let's first illustrate one of such data taken from a real-world application,

Example 1: Figure 1 shows features about users' watching counts on three different types of videos (Film, TV play and Animation) during the time period from 2015-06-01 to 2015-06-07 on an online IPTV VoD recommender system in a Telcom company from Mainland China.

From Figure 1, we observe that the watching counts on different types of videos vary a lot under different time period. In particular, in weekdays (from 2015-06-02 to 2015-06-05), the watching counts of TV play at working hours (08:00-18:00) on average are larger than those of the other two types of videos (*i.e.,* Animation and Film); Animation views are the most from 18:00 to 20:00; and the number of TV play increases again after 20:00. Besides, Animation views from 08:00 to 18:00 on weekends (from 2015-06-06 to 2015-06-07) and Children's day (2015-06-01) are larger than those of the other two, which typically differ from the case in weekdays (as above).

In short, from the above observation, we note that the varying of users' watching counts on different types of videos between day and night, or between weekdays and holidays, typically implies that the users' watching preferences are

changing rapidly along the time series, which in fact brings a big challenge to the design of the online learning algorithm for the recommender system. □

The above example raises one crucial concern to the existing online learning algorithm design: *how to fit and scale those big and fast-changing online data efficiently and practically*? To answer the question, a *scalable time-decaying online learning algorithm* is in need. However, two challenges need to be addressed.

The first challenge is that the time-decaying algorithm requires to *fit the fast-changing online data*. Unfortunately, up to date, previous works [19,24,25,30,32] have *not* addressed it well. On one hand, previous time-decaying algorithms either weight data (or examples) by time-decaying functions (*e.g.,* [16,17]) or just discard outdated examples (*e.g.,* [3]), which require full memory or keep only recent examples, respectively. On the other hand, existing online learning algorithms (*e.g.,* mirror descent (MD) algorithm like TGD [19] and FOBOS [30], and follow-the-regularized-leader (FTRL) algorithm like RDA [32] and FTRL-proximal [24,25]), which do *not* take the changes of the data into account, may *not* respond the changes correctly. And worse still, the target model generated from the existing online learning algorithms may no longer be suitable for current data distribution, since older history of models are equally weighted as the recent ones in the learning procedure, which may decrease its (*i.e.,* the target model's) effectiveness.

For the second challenge, the time-decaying algorithm needs to *scale the big fast-changing online data*. To the best of our knowledge, previous works [34,36,37] only study the parallelism of MD-like algorithms. And the parallelism of FTRL-like algorithms are mentioned in [25] only, *no* detailed solution is provided.

Contributions. This paper addresses scalable online learning problem with fast-changing data. The contributions are characterized as below.

(1) We define a problem called *time-decaying online convex optimization* TOCO problem (Section 2), where the target model approaches to the most recent models while older history of the models are deemphasized, following a user-defined time-decaying function.

(2) We propose a *time-decaying adaptive prediction* TDAP algorithm to solve TOCO problem (Section 3), which incorporates with both the state-of-the-art FTRL-proximal algorithm and an *exponential time-decaying* mechanism over model update. *Recursive closed forms* of the model update functions are proposed for computational efficiency and memory saving.

(3) To scale big data, we first parallelize the TDAP algorithm following the *data parallel* scheme [14] under *bulk synchronous parallel* (BSP) [6] consistency model (Section 4), in which recursive update methods considering time-decaying factors are proposed, and detailed implementations are provided. To avoid the unnecessary costs due to the synchronization policy ("blocking" of stragglers) of BSP, we further parallelize the algorithm under *stale synchronous parallel* (SSP) model, with accuracy guarantees from [13].

(4) Under real-world datasets, we experimentally evaluate our algorithm on top of two well-known platforms (Section 5): (a) general big data analytic platform Spark [27] under BSP model; and (b) specific machine learning platform Petuum [33] under SSP model. It is worth highlighting that TDAP achieves good accuracy: it improves *5.6%* prediction accuracy, compared to FTRL-proximal algorithm, on both platforms; and TDAP *scales* well: it runs 4x speedup when the number of machines increases from 2 to 10. We further observe that TDAP on Petuum runs faster than that on Spark, and the former enjoys better scalability with more machines.

In our production IPTV VoD service, when we switched from FTRL-proximal to TDAP, we observed *5% more users engaging* on our service, and *10% longer engagement time* per user, approximately. We also observed that TDAP on Petuum can process approximately *one order of magnitude* more users than Spark, with the same cluster setup. While both TDAP on Petuum and Spark are viable, we found that deployment was easier and more efficient using Petuum.

Organization. This paper is organized as below: In Section 2, we review some related work and formulate our TOCO problem; We propose our TDAP algorithm to solve the problem in Section 3; Section 4 then parallelize the TDAP algorithm under both BSP and SSP consistency model; We experimentally verify our algorithm in Section 5 and conclude this paper in Section 6.

2. ONLINE LEARNING ALGORITHM

In this section, we first introduce the related work. We then give some details of the FTRL-proximal algorithm since our algorithm are derived from the FTRL-proximal algorithm due to its superior performance. We end this section by formulating our problem.

2.1 Related Work

We characterize related work as follows.

Algorithms with time-decaying property. Numbers of time-decaying algorithms are proposed to address the concept drifting [11] problem. On one hand, for gradual concept drifting, *full* memory based approaches (*e.g.,* [16] and [17]) use exponential or linear time-decaying functions to weight data (or examples), *i.e.,* the older the example, the smaller the impact. For abrupt ones, on the other hand, *partial* memory based methods (*e.g.,* [3]) are proposed by discarding examples outside a pre-defined window, only recent examples in the window are taken into account.

In contrast, we differ from the above as: (a) unlike full memory based methods, we do *not* require to store all examples, which saves memory; (b) We consider all examples in model training, such that information in previous examples are not discarded; and (c) Our algorithms are able to tackle *both* gradual and abrupt concept drifting problems, with tuning the time-decaying factor (to be seen in Section 3).

Online convex learning algorithms. Various online learning algorithms are proposed to solve the online convex optimization problem. We characterize it as the following two types: (a) Mirror descent (MD) algorithms like truncated gradient descent [19] and FOBOS [30]; and (b) Follow-the-regularized-leader (FTRL) algorithms like RDA [32] and FTRL-proximal [26]. Moreover, [24] proves the equivalence between the mirror descent algorithms and the FTRL algorithms, and in particular, FTRL-proximal outperforms the others in terms of accuracy and sparsity [25], and are widely used in industry, *e.g.,* recommender system [2, 28] and Advertisement system [25].

However, our algorithms differ from the previous ones in the followings: (a) The effects of the older history of the models on target model are *scaled down* with an exponential time-decaying function, while existing methods do *not*; and (b) The time-decaying factors are embedded into the recursive closed form of model update functions, similar to the FTRL-proximal algorithm in [25].

Parallelism of online learning algorithm. In the literature, there are various existing works on parallelized machine learning algorithm [10, 12, 18]. Meanwhile, lots of machine learning platforms are proposed [1, 8, 22]. Among others, relate to online learning context, techniques for parallelizing mirror descent (MD) algorithms, in particular, stochastic gradient descent (SGD) algorithm have been intensively studied. [36] proves that parallelized SGD converges well with decayed updates. [37] presents the first parallelized SGD including a detailed analysis and experimental evidence. [34] proposes a user-friendly programming interface for parallelizing SGD. [25] also mentions the parallelism of the FTRL-like algorithms, but *no* detailed solution is given.

However, in contrast to previous works, we (a) propose *recursive* model parameters update functions embedded with exponential time-decaying factors in distributed (multi-machines) setting; (b) provide practical implementations of our algorithm (FTRL-like algorithm) for parallelism on both BSP [6] and SSP [13] consistency model in online learning scenario; and (c) compare the performance on two well-known platforms: (i) general big data analytic platform (*i.e.,* Spark [27]) for BSP model; and (ii) specific machine learning platform (*i.e.,* Petuum [33]) for SSP model. We conclude some interesting results in the experiments and hope that all practitioners get enlightened from it.

2.2 FTRL-Proximal Algorithm

In this subsection, we give the details of the FTRL-proximal algorithm. We start with the optimization problem that the algorithm is proposed for. We first establish some notations to be used in the following. We denote a vector $\mathbf{g}^{(t)} \in \mathbb{R}^d$, where t indicates the t-th training examples. The i-th entry in vector $\mathbf{g}^{(t)}$ is defined as $g_i^{(t)}$, and $\mathbf{g}^{(1:t)} = \sum_{s=1}^{t} \mathbf{g}^{(s)}$. We use data and examples interchangeably if the context is clear.

Optimization problem. Given a time step $T \geq 1$, a sequence of training examples with features $\mathbf{x}^{(t)} \in \mathbb{R}^d$ and labels $\mathbf{y}^{(t)}$, $t \in [1, T]$, the optimization problem that FTRL-proximal algorithm solved takes the form of

$$\mathbf{w}^{(T)} = \arg\min_{\mathbf{w}} \Big\{ \sum_{t=1}^{T} L(\mathbf{w}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)}) + R(\mathbf{w}) \Big\}, \quad (1)$$

in particular,

- $\mathbf{w}^{(T)} \in \mathbb{R}^d$ is the target model parameters to be computed;
- $L(,,)$ is a convex loss function of the prediction function, *e.g.,* least square loss in linear regression, or logistic loss in logistic regression; and
- $R(\mathbf{w})$ is a convex regularization term, *e.g.,* L_1-norm, L_2-norm or a linear combination of both.

Model update function. To solve the optimization problem in Equation 1, at each iteration $t \in [1, T]$, the FTRL-proximal algorithm updates the model parameters iteratively, *i.e.,* $\mathbf{w}^{(t+1)}$ takes the form of

$$\arg\min_{\mathbf{w}} \Big\{ \mathbf{g}^{(1:t)} \cdot \mathbf{w} + \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 + \frac{1}{2} \sum_{s=1}^{t} \sigma^{(s)} \|\mathbf{w} - \mathbf{w}^{(s)}\|_2^2 \Big\}, \quad (2)$$

where

- $\mathbf{g}^{(t)} \in \mathbb{R}^d$ is a vector of gradients of loss function $L(\mathbf{w}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)})$ in Equation 1;
- $(\lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2)$ is the regularization term corresponding to $R(\mathbf{w})$ in Equation 1; and
- $\frac{1}{2} \sum_{s=1}^{t} \sigma^{(s)} \|\mathbf{w} - \mathbf{w}^{(s)}\|_2^2$ is a *smoothing term*, importantly, such smoothing term does not change the optimum of the original problem in Equation 1, it aims to speed up convergence and improve accuracy.

  Moreover, the vector $\sigma^{(t)}$ is defined in terms of the learning-rate schedule, *i.e.,* $\sigma^{(t)} = \frac{1}{\eta^{(t)}} - \frac{1}{\eta^{(t-1)}}$, such that $\sigma^{(1:t)} = \frac{1}{\eta^{(t)}}$. And $\eta_i^{(t)}$ is a *per-coordinate learning rate* proposed in [25] as,

$$\eta_i^{(t)} = \frac{\alpha}{\sqrt{\sum_{s=1}^{t} (g_i^{(s)})^2}}, \quad (3)$$

  where α is set as twice the maximum allowed magnitude for $\mathbf{w}_i$ to give the best possible regret bound [25].

As reported in [25], the update function in Equation 2 can be transformed into a recursive closed form, *i.e.,* the value $\mathbf{w}^{(t)}$ can be recursively computed from $\mathbf{w}^{(t-1)}$. An efficient implementation with pseudocode is then provided based on the recursive form. And better still, FTRL-proximal algorithm outperforms the other classical online learning algorithms (*e.g.,* FOBOS and RDA) in terms of accuracy and sparsity, Therefore, in this paper, we derive our techniques from the FTRL-proximal algorithm. We consider a possible intuitive modification to FTRL-proximal, that lets it adapt faster to changes in data distribution.

2.3 Problem Definition

As motivated in Sec. 1, we want to tackle the data with fast-changing nature for online learning scenario, which indicates that the target model approaches to the most recent model while older history of the model can be deemphasized. Thus, we state the time-decaying online convex optimization TOCO problem as below.

Definition 1: Given a time step $T \geq 1$, a sequence of training examples with features $\mathbf{x}^{(t)}$ and labels $\mathbf{y}^{(t)}$, $t \in [1, T]$, the time-decaying online convex optimization, denoted as TOCO, is stated as

$$\mathbf{w}^{(T)} = \arg\min_{\mathbf{w}} \Big\{ \sum_{t=1}^{T} L(\mathbf{w}, \mathbf{x}^{(t)}, \mathbf{y}^{(t)}) + R(\mathbf{w}) + \sum_{t=1}^{T} F^{(T)}(t) S(\mathbf{w}, \mathbf{w}^{(t)}) \Big\}. \quad (4)$$

□

The objective function in Equation 4 is similar to that of Equation 1, except that we introduce an additional term as $\sum_{t=1}^{T} F^{(T)}(t) S(\mathbf{w}, \mathbf{w}^{(t)})$ to model the time decay of our target model. More specifically, (a) $F^{(T)}(t)$ is a *monotonic increasing time-decay function* (to be defined shortly) with independent variable t, and (b) $S(\mathbf{w}, \mathbf{w}^{(t)})$ is a smoothing term, *i.e.,* $\frac{1}{2} \|\mathbf{w} - \mathbf{w}^{(t)}\|_2^2$.

The intuition here is that with the help of time-decaying function, the target model $\mathbf{w}^{(T)}$ *cannot* differ too much compared to those recent model, while the effects from older history of the model can be decreased. Such corresponds to the objective of our problem.

3. TIME-DECAYING ADAPTIVE PREDICTION ALGORITHM

In order to solve TOCO problem, in this section, we present the details of the proposed time-decaying adaptive prediction (TDAP) algorithm. We first introduce the time-decaying function. We then give a formal definition of the model update function with recursive closed form. Details of the algorithm are finally presented.

Time-Decaying function. We note that there are numbers of time-decaying function in the literature, such as polynomial decay and exponential decay [7]. In this paper, we use *exponential* decay function as $F^{(T)}(t)$ in Definition 1. The reasons are two folds: (a) it is widely used in both industry [20, 31] and academia [5, 29]; and (b) the recursive closed form of the model update function (to be seen shortly) can be derived under the exponential term [1].

In particular, the exponential time-decaying function takes the form of

$$F^{(T)}(t) = \exp(-\frac{|T+1-t|}{(2\tau)^2}), \tag{5}$$

where $T \geq 1$ is current time step, and t is time step of the history model, $t \in [1, T]$. Note that $F^{(T)}(t)$ is a monotonic increasing function with independent variable t, *i.e.,* the larger the t, the larger the returned value.

Model update function. Based on the exponential decay function in Equation 5, we define the iterative model update function in each iteration t to solve TOCO, where $\mathbf{w}^{(t+1)}$ takes the form of

$$\arg\min_{\mathbf{w}} \Big\{ \mathbf{g}^{(1:t)} \cdot \mathbf{w} + \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 + \frac{1}{2} \sum_{s=1}^{t} \delta^{(s,t)} \|\mathbf{w} - \mathbf{w}^{(s)}\|_2^2 \Big\}. \tag{6}$$

Note that $\delta^{(s,t)} = \sigma^{(s)} \cdot F^{(t)}(s)$, which can be extended on per-coordinate base as

$$\begin{aligned} \delta_i^{(s,t)} &= \sigma_i^{(s)} \exp(-\tfrac{|t+1-s|}{(2\tau)^2}) \\ &= \tfrac{1}{\alpha} \Big(\sqrt{\textstyle\sum_{j=1}^{s} (g_i^{(j)})^2} - \sqrt{\textstyle\sum_{j=1}^{s-1} (g_i^{(j)})^2} \Big) \exp(-\tfrac{|t+1-s|}{(2\tau)^2}) \end{aligned} \tag{7}$$

We let $\gamma = \frac{1}{(2\tau)^2}$, then $\delta^{(s,t)} = \sigma^{(s)} \exp(-\gamma(t+1-s))$. Here, $\gamma > 0$ is a decay rate, and the bigger the γ, the faster decaying of the history model. It is easy to see that in case of $\lim_{\tau \to +\infty} \gamma = 0$, Equation 6 is equivalent to that of FTRL-proximal algorithm in Equation 2.

It is worth highlighting that the update function for TDAP algorithm differs from that of the FTRL-proximal algorithm as we introduce the time-decaying factor to the smoothing term, *i.e.,* the quadratic term $\frac{1}{2} \sum_{s=1}^{t} \delta^{(s,t)} \|\mathbf{w} - \mathbf{w}^{(s)}\|_2^2$.

Closed form. Similar to FTRL-proximal algorithm, we next derive a recursive closed form of the update function in Equation 6 following the flow in [25].

[1] In this work, we take the first step to study the TOCO problem with exponential decay, other time-decaying functions (*e.g.,* polynomial decay) are refered to the future work.

Algorithm Per-Coordinate TDAP algorithm with L_1 and L_2 Regularization for Logistic Regression
Input: Parameters α, λ_1, λ_2 and γ

1. $(\forall i \in \{1, \cdots, d\})$, initialize $u_i^{(0)} = v_i^{(0)} = \delta_i^{(0)} = h_i^{(0)} = 0$
2. **for each** $t = 1$ to T, **do**

/* features receiving */
3. receive feature vector $\mathbf{x}^{(t)}$ and let $I = \{i | x_i^{(t)} \neq 0\}$

/* model parameters update */
4. **for each** $i \in I$, **do**
5. update $w_i^{(t)}$ with closed form /* Equation 9 */

/* prediction */
6. predict $p^{(t)} = \frac{1}{1+\exp(-\mathbf{x}^{(t)} \cdot \mathbf{w}^{(t)})}$ /* logistic regression */

/* labels observation */
7. observe label $y^{(t)} \in \{0, 1\}$

/* recursive form update for future model update */
8. **for each** $i \in I$, **do**
9. $g_i^{(t)} = (p^{(t)} - y^{(t)}) x_i^{(t)}$ /* gradient of loss w.r.t. $\mathbf{w}$ */
10. $\sigma_i^{(t)} = \frac{1}{\alpha} (\sqrt{u_i^{(t-1)} + (g_i{}^{(t)})^2} - \sqrt{u_i^{(t-1)}})$ /* $\frac{1}{\eta_i^{(t)}} - \frac{1}{\eta_i^{(t-1)}}$ */
11. $u_i^{(t)} = u_i^{(t-1)} + (g_i^{(t)})^2$ /* sum of gradient */
12. $\delta_i^{(t)} = \exp(-\gamma)(\delta_i^{(t-1)} + \sigma_i^{(t)})$ /* Equation 10 */
13. $v_i^{(t)} = v_i^{(t-1)} + g_i^{(t)}$ /* sum of gradient square */
14. $h_i^{(t)} = \exp(-\gamma)(h_i^{(t-1)} + \sigma_i^{(t)} w_i^{(t)})$ /* Equation 11 */
15. $z_i^{(t)} = v_i^{(t)} - h_i^{(t)}$ /* Equation 12 */

Figure 2: TDAP Algorithm

We rewrite the Equation 6 *w.r.t.* the arg min over $\mathbf{w}$ as

$$\Big\{ (\mathbf{g}^{(1:t)} - \sum_{s=1}^{t} \delta^{(s,t)} \mathbf{w}^{(s)}) \cdot \mathbf{w} + \lambda_1 \|\mathbf{w}\|_1 + \frac{1}{2} (\lambda_2 + \sum_{s=1}^{t} \delta^{(s,t)}) \cdot \mathbf{w}^2 + (const) \Big\} \tag{8}$$

By storing $\mathbf{z}^{(t)} = \mathbf{g}^{(1:t)} - \sum_{s=1}^{t} \delta^{(s,t)} \mathbf{w}^{(s)}$, one can solve $\mathbf{w}^{(t+1)}$ in a recursive closed form on a per-coordinate base as follows,

$$\begin{cases} 0 & \text{if } |z_i^{(t)}| \leq \lambda_1 \\ -(\lambda_2 + \sum_{s=1}^{t} \delta_i^{(s,t)})^{-1} (z_i^{(t)} - \lambda_1 sign(z_i^{(t)})) & \text{if } |z_i^{(t)}| > \lambda_1 \end{cases} \tag{9}$$

From Equation 9, in iteration t, we are required to store $\sum_{s=1}^{t} \delta_i^{(s,t)}$ and $z_i^{(t)}$ in memory *only*. However, how can we update both values in t-th iteration with *only* the values of $(t-1)$-th iteration? Next, we focus on the *recursive form* for updating both $\sum_{s=1}^{t} \delta_i^{(s,t)}$ and $z_i^{(t)}$.

Recursive form of $\sum_{s=1}^{t} \delta_i^{(s,t)}$. $\sum_{s=1}^{t} \delta_i^{(s,t)}$ can be computed in a recursive form:

$$\begin{aligned} \textstyle\sum_{s=1}^{t} \delta_i^{(s,t)} &= \textstyle\sum_{s=1}^{t} \Big(\sigma_i^{(s)} \cdot exp\big(-\gamma(t+1-s) \big) \Big) \\ &= \exp(-\gamma) \cdot (\textstyle\sum_{s=1}^{t-1} \delta_i^{(s,t-1)} + \sigma_i^{(t)}) \end{aligned} \tag{10}$$

Recursive form of $z_i^{(t)}$. Since $z_i^{(t)} = g_i^{(1:t)} - \sum_{s=1}^{t} \delta_i^{(s,t)} w_i^{(s)}$, we let $h_i^{(t)} = \sum_{s=1}^{t} \delta_i^{(s,t)} w_i^{(s)}$, $h_i^{(t)}$ can then be computed in a recursive form

$$\begin{aligned} h_i^{(t)} &= \textstyle\sum_{s=1}^{t} \delta_i^{(s,t)} w_i^{(s)} \\ &= \textstyle\sum_{s=1}^{t} \sigma_i^{(s)} exp\big(-\gamma(t+1-s) \big) w_i^{(s)} \\ &= \exp(-\gamma) \big(\textstyle\sum_{s=1}^{t-1} (\delta_i^{(s,t-1)} w_i^{(s)}) + \sigma_i^{(t)} w_i^{(t)} \big) \\ &= \exp(-\gamma)(h_i^{(t-1)} + \sigma_i^{(t)} w_i^{(t)}) \end{aligned} \tag{11}$$

Thus, the recursive form of $z_i^{(t)}$ becomes

$$\begin{aligned} z_i^{(t)} &= g_i^{(1:t)} - \textstyle\sum_{s=1}^{t} \delta_i^{(s,t)} w_i^{(s)} \\ &= g_i^{(1:t-1)} + g_i^{(t)} - \exp(-\gamma)(h_i^{(t-1)} + \sigma_i^{(t)} w_i^{(t)}) \end{aligned} \quad (12)$$

Detailed algorithm. Putting all together, in iteration t, from Equation 10 and 12, we note that our algorithm is required to keep track of $u_i^{(t)}$, $v_i^{(t)}$, $\delta_i^{(t)}$ and $h_i^{(t)}$ on per-coordinate base *only*. We show the pseudocode of TDAP algorithm using the example of logistic regression in Figure 2. The algorithm first initializes the parameters in Line 1. Then, upon receiving the features (Lines 2-3) for each iteration, TDAP updates the model parameters and does the prediction (Lines 4-6). It then updates the parameters for further usage by observing the labels (Lines 7-15).

Advantages. It is worth remarking that (a) the effects of older history models on target model are scaled down with the help of time-decaying function, which leads to adapt fast changes (gradual or abrupt) of data; (b) all examples are used for model training while there is *no* need to record them, which significantly saves the memory; and (c) it is efficient to update the models due to the recursive closed form, as shown in Section 5.

4. PARALLELISM OF TDAP ALGORITHM

In this section, we show the parallelism of TDAP algorithm. We first introduce the data model and the system model. We then present the parallel TDAP under data parallel scheme with *bulk synchronized parallel* (BSP) consistency model, as $\mathsf{TDAP}_{\mathsf{BSP}}$. We end this section by showing how we can extend the $\mathsf{TDAP}_{\mathsf{BSP}}$ to *stale synchronized parallel* (SSP) model, as $\mathsf{TDAP}_{\mathsf{SSP}}$.

4.1 Data Model and System Model

Data model. We follow the *data parallel* [14] model in online learning scenario. In particular, the data (examples with features and labels) take the form of stream, which is divided into a sequence of mini-batch $D^{(t)}$ in each iteration $t \in [1, T]$, $|D^{(t)}| \geq 1$. The mini-batch $D^{(t)}$ is then partitioned and assigned to computational workers indexed by p for further computations (to be seen shortly).

System model. The system model follows the state-of-the-art parameter server paradigm *e.g.,* [8, 13, 22], which are widely used to tackle the large scale iterative machine learning problem [15, 18, 37]. It comprises of the following three major components:

Driver. The driver is to initiate the algorithm and control the parameters updates via *parameter consistency controller* for each iteration $t \in [1, T]$ under the bulk synchronized parallel (BSP) or stale synchronized parallel (SSP).

Parameter servers. The parameter servers (PS) provide a global access to model parameters via a *parameter synchronization interface* like that of table-based or key-value stores. Note that in practice, the PS can also be the driver.

Workers. Let P be all workers. For each iteration t, each worker $p \in P$ receives the model parameters from PS via the parameter synchronization interface (*i.e.,* pull()), and updates the model locally *in parallel* using the received examples $D_p^{(t)}$ partitioned by mini-batch $D^{(t)}$, *i.e.,* $D_p^{(t)} \subset D^{(t)}$.

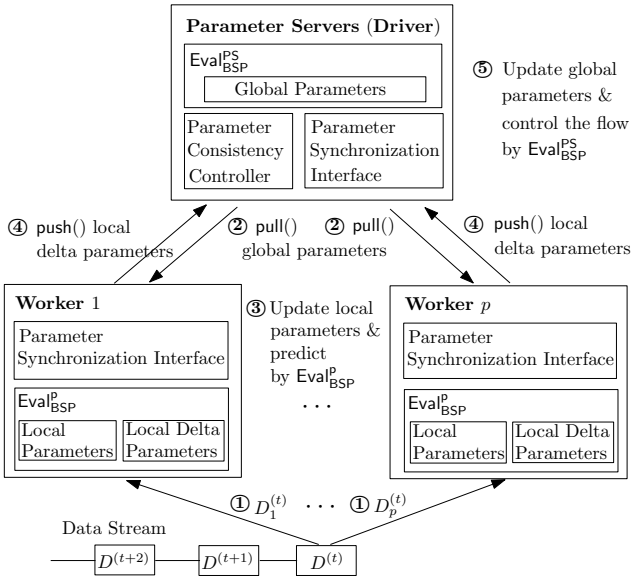

Figure 3: Data model, system model and working flow of $\mathsf{TDAP}_{\mathsf{BSP}}$ algorithm.

Note that each example in $D^{(t)}$ is allocated to one worker *only*. After the update is done, the worker p then synchronizes the newly updated model parameters to the PS via the synchronization interface push(). The driver (or PS) then generates future decisions.

Example 2: Figure 3 depicts an example of the data model and system model. Three parties of the system, *i.e.,* driver, PS and workers are shown. For simplicity, here, the PS also represents the driver. The examples coming to the system take the form of a stream of mini-batch $D^{(t)}$ in each iteration t. Collections of examples $D_p^{(t)} \in D^{(t)}$ are then sent to the worker p for later computations. □

4.2 Parallel TDAP Algorithm Under BSP

Data structure. We first introduce three types of data structures that facilitate $\mathsf{TDAP}_{\mathsf{BSP}}$.

Global parameters on PS. The global parameters are stored in PS, which keep track of the global model states. In particular, in iteration t, the global parameters to be saved are $(u_i^{(t)}, v_i^{(t)}, h_i^{(t)'}, \delta_i^{(t)'})$ on per-coordinate base, where

- $u_i^{(t)} = u_i^{(t-1)} + \sum_{p \in P} \sum_{d \in D_p^{(t)}} (g_{p,d,i}^{(t)})^2$;
- $v_i^{(t)} = v_i^{(t-1)} + \sum_{p \in P} \sum_{d \in D_p^{(t)}} g_{p,d,i}^{(t)}$;
- $h_i^{(t)'} = h_i^{(t-1)'} + \sum_{p \in P} \sum_{d \in D_p^{(t)}} \sigma_{p,d,i}^{(t)} w_{p,i}^{(t)}$; and
- $\delta_i^{(t)'} = \delta_i^{(t-1)'} + \sum_{p \in P} \sum_{d \in D_p^{(t)}} \sigma_{p,d,i}^{(t)}$.

Note that $g_{p,d,i}^{(t)}$ (*resp.* $\sigma_{p,d,i}^{(t)}$ and $w_{p,i}^{(t)}$) are gradient (*resp.* learning-rate schedule and model parameter) with i-th coordinate in t-th iteration computed under example $d \in D_p^{(t)}$ at worker p. And $h_i^{(t)'}$ and $\delta_i^{(t)'}$ do *not* introduce the time-decaying factor, which are different to $h_i^{(t)}$ and $\delta_i^{(t)}$.

Local parameters at workers. Each worker p stores its local parameters, which can be derived from global parameters on PS. Specifically, in iteration t, the local

Algorithm $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$
Input: A collection of examples $D_p^{(t)}$

/* receiving global parameters from PS */
1. $\mathsf{pull}(u_i^{(t)}, v_i^{(t)}, h_i^{(t)'}, \delta_i^{(t)'})$ on per-coordinate base

/* local parameters update */
2. **for each** coordinate i, **do**
3. $\quad u_{p,i}^{(t)} = u_i^{(t)}$
4. $\quad v_{p,i}^{(t)} = v_i^{(t)}$
5. $\quad h_{p,i}^{(t)} = \exp(-\gamma)\big(h_{p,i}^{(t-1)} + (h_i^{(t)'} - h_i^{(t-1)'})\big)$
6. $\quad \delta_{p,i}^{(t)} = \exp(-\gamma)\big(\delta_{p,i}^{(t-1)} + (\delta_i^{(t)'} - \delta_i^{(t-1)'})\big)$
7. $\quad$ store $h_i^{(t)'}$ and $\delta_i^{(t)'}$ locally for next iteration

/* computing model parameters */
8. **for each** coordinate i, **do**
9. $\quad$ compute $w_{p,i}^{(t)}$ with closed form /* Equation 9 */

10. **for each** $d \in D_p^{(t)}$, where $d = (\mathbf{x}, y)$, **do**
/* prediction using model parameters */
11. $\quad$ predict $p_{p,d}^{(t)} = \frac{1}{1+\exp(-\mathbf{x}\cdot\mathbf{w}_p^{(t)})}$
/* local delta parameters computation */
12. $\quad$ **for each** coordinate i, **do**
13. $\quad\quad$ compute $g_{p,d,i}^{(t)}$ and $\sigma_{p,d,i}^{(t)}$
14. $\quad\quad \Delta u_{p,i}^{(t)} = \Delta u_{p,i}^{(t)} + (g_{p,d,i}^{(t)})^2$
15. $\quad\quad \Delta v_{p,i}^{(t)} = \Delta v_{p,i}^{(t)} + g_{p,d,i}^{(t)}$
16. $\quad\quad \Delta h_{p,i}^{(t)'} = \Delta h_{p,i}^{(t)'} + \sigma_{p,d,i}^{(t)} w_{p,i}^{(t)}$
17. $\quad\quad \Delta \delta_{p,i}^{(t)'} = \Delta \delta_{p,i}^{(t)'} + \sigma_{p,d,i}^{(t)}$

/* sending local delta parameters to PS */
18. $\mathsf{push}(\Delta u_{p,i}^{(t)}, \Delta v_{p,i}^{(t)}, \Delta h_{p,i}^{(t)'}, \Delta \delta_{p,i}^{(t)'})$ to PS

Figure 4: $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$ at Worker p in iteration t

parameters to be saved at worker p take the form of $(u_{p,i}^{(t)}, v_{p,i}^{(t)}, h_{p,i}^{(t)}, h_i^{(t-1)'}, \delta_{p,i}^{(t)}, \delta_i^{(t-1)'})$ on per-coordinate base, where

- $u_{p,i}^{(t)} = u_i^{(t)}$;
- $v_{p,i}^{(t)} = v_i^{(t)}$;
- $h_{p,i}^{(t)} = \exp(-\gamma)(h_{p,i}^{(t-1)} + \sum_{p' \in P} \sum_{d \in D_{p'}^{(t)}} \sigma_{p',d,i}^{(t)} w_{p',i}^{(t)})$; and
- $\delta_{p,i}^{(t)} = \exp(-\gamma)(\delta_{p,i}^{(t-1)} + \sum_{p' \in P} \sum_{d \in D_{p'}^{(t)}} \sigma_{p',d,i}^{(t)})$.

It is worth highlighting that $h_i^{(t-1)'}$ is recorded in order to compute $\sum_{d \in D_{p'}^{(t)}} \sigma_{p',d,i}^{(t)} w_{p',i}^{(t)}$ (to be seen shortly). Similar for $\delta_i^{(t-1)'}$.

Local delta parameters. Intuitively, for each worker p, it computes the local delta parameters which is to incrementally update the global parameters on PS. More specifically, the local delta parameters for work p in iteration t are $(\Delta u_{p,i}^{(t)}, \Delta v_{p,i}^{(t)}, \Delta h_{p,i}^{(t)'}, \Delta \delta_{p,i}^{(t)'})$ on per-coordinate base, where

- $\Delta u_{p,i}^{(t)} = \sum_{d \in D_p^{(t)}} (g_{p,d,i}^{(t)})^2$;
- $\Delta v_{p,i}^{(t)} = \sum_{d \in D_p^{(t)}} g_{p,d,i}^{(t)}$;
- $\Delta h_{p,i}^{(t)'} = \sum_{d \in D_p^{(t)}} \sigma_{p,d,i}^{(t)} w_{p,i}^{(t)}$; and

Algorithm $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$
Input: Local delta parameters from all workers

/* global parameters update */
1. **for each** worker p, **do**
2. $\quad$ **for each** coordinate i, **do**
3. $\quad\quad u_i^{(t)} = u_i^{(t)} + \Delta u_{p,i}^{(t)}$
4. $\quad\quad v_i^{(t)} = v_i^{(t)} + \Delta v_{p,i}^{(t)}$
5. $\quad\quad h_i^{(t)'} = h_i^{(t)'} + \Delta h_{p,i}^{(t)'}$
6. $\quad\quad \delta_i^{(t)'} = \delta_i^{(t)'} + \Delta \delta_{p,i}^{(t)'}$

Figure 5: $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$ at PS in iteration t

- $\Delta \delta_{p,i}^{(t)'} = \sum_{d \in D_p^{(t)}} \sigma_{p,d,i}^{(t)}$.

$\mathsf{TDAP}_{\mathsf{BSP}}$ **algorithm**. We show the details of $\mathsf{TDAP}_{\mathsf{BSP}}$ using the logistic regression, which is controlled by a driver function $\mathsf{Driver}_{\mathsf{BSP}}$. $\mathsf{Driver}_{\mathsf{BSP}}$ initiates the $\mathsf{TDAP}_{\mathsf{BSP}}$, and triggers $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$ (Figure 4) and $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$ (Figure 5) functions in each iteration between workers and PS. All parameters are updated *iteratively* under BSP model, *no need* to compute from scratch.

In particular, all global parameters and local (delta) parameters are initiated as zero. Then given a collection of examples in iteration t, each worker p computes the local (delta) parameters by $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$, *in parallel*, based on the global parameters computed in previous iteration. The global parameters are then updated by $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$ at PS using the newly computed local delta parameters. $\mathsf{Driver}_{\mathsf{BSP}}$ *terminates* the whole procedure manually, or until there is no data coming.

$\underline{\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}}$. Given a collection of examples $D_p^{(t)} \subset D^{(t)}$ in t-th iteration at worker p, $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$ first receives all global parameters from PS by $\mathsf{pull}()$ function (Line 1). It then updates the local parameters and computes local delta parameters iteratively, based on the parameters in $(t-1)$-th iteration (Lines 2-7). The model parameters $\mathbf{w}^{(t)}$ are then constructed using the newly updated local parameters, and the prediction are conducted (Lines 8-11). $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$ finally computes the local delta parameters (Lines 12-17) and sends them to PS by $\mathsf{push}()$ function for incrementally updating the global parameters (Line 18).

$\underline{\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}}$. Upon receiving all local delta parameters from each worker p in iteration t, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$ increments global parameters by summing up all those received local delta parameters. The updated global parameters are used for next iteration (Lines 1-6).

Example 3: Figure 3 illustrates the working flow of $\mathsf{TDAP}_{\mathsf{BSP}}$ algorithm in t-th iteration given a mini-batch $D^{(t)}$. All local and global parameters are recored in PS and workers, respectively, which are initialized as zero.

Upon receiving a collection of examples $D_p^{(t)} \subset D^{(t)}$, the worker p invokes $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$. It pulls the global parameters from PS, and updates the local (delta) parameters accordingly. The prediction are conducted based on the newly computed model parameters. After the computations, $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{BSP}}$ pushes the local delta parameters to the PS.

Based on the local delta parameters received from all workers (required by BSP consistency model), the PS then invokes $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$ to augment the global parameters accordingly. Afterwards, the PS (*a.k.a.* the Driver) then invokes next iteration of the computation, or abort. □

4.3 Parallel TDAP Algorithm Under SSP

The performance of $\mathsf{TDAP}_{\mathsf{BSP}}$ may be significantly hampered by the *stragglers* in each iteration that may hold up the process, such costs are inherent to the synchronization policy of BSP. To reduce the costs, one possible solution is to use *asynchronous parallel strategy*, however, *no* performance guarantee is provided [33] in terms of correctness.

In this subsection, we introduce TDAP under SSP consistency model [33], denoted as $\mathsf{TDAP}_{\mathsf{SSP}}$, whose accuracy can be theoretically preserved by [13]. Since $\mathsf{TDAP}_{\mathsf{SSP}}$ is an extension from $\mathsf{TDAP}_{\mathsf{BSP}}$, for presentation simplicity, we avoid repeating the same details, but highlight the major differences between two.

Data structure. We first present the difference on data structure. The local delta parameters for $\mathsf{TDAP}_{\mathsf{SSP}}$ are the same to those of $\mathsf{TDAP}_{\mathsf{BSP}}$, while local and global parameters involve more components, as described below.

Global parameters on PS. Besides the global parameters used in $\mathsf{TDAP}_{\mathsf{BSP}}$, $\mathsf{TDAP}_{\mathsf{SSP}}$ introduces two more types of global parameters on PS: (a) The time step t_p for each worker p, which encodes the *latest* time step that worker p synchronizes (via push interface) its local delta parameters to the PS; and (b) The minimum time step t_{PS} for all workers p, *i.e.,* $t_{PS} = \min\{t_p | \forall p \in P\}$.

Local parameters at workers. At worker p, except for those parameters in $\mathsf{TDAP}_{\mathsf{BSP}}$, $\mathsf{TDAP}_{\mathsf{SSP}}$ requires to record (a) the *latest* time step t_p that the local parameters been updated; and (b) the *latest* time step $t_{(p,PS)}$ that worker p synchronizes global parameters (via pull interface) from PS.

$\mathsf{TDAP}_{\mathsf{SSP}}$ **algorithm**. Based on the newly introduced parameters, we are ready to show the algorithm. Similar to $\mathsf{TDAP}_{\mathsf{BSP}}$, $\mathsf{TDAP}_{\mathsf{SSP}}$ is also controlled by a driver $\mathsf{Drivers}_{\mathsf{SSP}}$, which is for initialization and triggering $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$ and $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ functions under the SSP consistency model. In particular, compared to BSP, workers may compute advance ahead of each other up to s iterations apart, where s is called *staleness threshold* (in short stale), under SSP. Workers that go too far away (s iterations faster than the slowest one) are forced to wait, until slower workers catch up. Note that SSP is equivalent to BSP when $s = 0$.

$\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$. Given a collection of examples $D_p^{(t)}$, $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$ first checks whether the local parameters are delayed or stale, by evaluating the value of $(t_p - t_{p,PS})$. (a) If $(t_p - t_{p,PS}) > s$ (*i.e.,* local parameters at worker p are delayed at least s iterations), $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$ yields to synchronize global parameters from PS via pull interface (like Line 1 in Figure 4) and conducts the procedures that are the same to Lines 2-17. The time step $t_{p,PS}$ is then updated as t_{PS}, *i.e.,* $t_{p,PS} = t_{PS}$, where t_{PS} is within those returned global parameters; (b) Otherwise, *no need* to update the local parameters, $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$ just computes the local delta parameters, the same to Lines 12-17 in Figure 4.

Note that the time step t_p for worker p is incremented by 1 after the above procedure, and all local delta parameters are then sent to PS via push interface, as Line 18 in Figure 4.

$\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$. Compared to $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{BSP}}$, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ is invoked *once* receiving (a) the local delta parameters, or (b) a pull request for global parameters, from a worker p.

- (a) Upon receiving the local delta parameters from the worker p, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ conducts the update functions as Lines 2-6 in Figure 5. The time step t_p for worker p is added by 1, and t_{PS} is updated accordingly;
- (b) Otherwise, *i.e.,* receiving a pull request for global parameters from the worker p, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ is required to evaluate the value of $(t_p - t_{PS})$. (a) If $(t_p - t_{PS}) \leq s$, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ returns the global parameters directly; (b) Otherwise, which implies that worker p runs too far away, $\mathsf{Eval}^{\mathsf{p}}_{\mathsf{SSP}}$ has to wait until the slower workers catch up. Once $(t_p - t_{PS}) = s$, $\mathsf{Eval}^{\mathsf{PS}}_{\mathsf{SSP}}$ proceeds the request again and returns the global parameters.

5. EXPERIMENTAL STUDY

In this section, we experimentally study the performance between FTRL-Proximal and TDAP using real-world datasets. In short, we evaluate (a) the *accuracy* of TDAP; (b) the *scalability* of TDAP; and (c) the *efficiency* between $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$. The results exhibit the algorithm with good accuracy, scalability and efficiency.

5.1 Experimental Setup

Datasets. We benchmarked 8 real-world datasets in total, which can be classified into the following categories. Some detailed statistics of the datasets are illustrated in Table 1.

Public non-time series datasets. We used 6 datasets without time series, namely Books, DVD, Electronics, Kitchen, RCV1 and News. Among them, Books, Dvd, Electronics and Kitchen are first used in [4], representing Amazon product reviews of four different product types. RCV1 and News are scaled versions of rcv1.binary [21] and news20.binary [15] datasets for binomial classification, respectively.

Public time series dataset. We used a public dataset with time series on suspicious URLs detection, called URLs which is available from [23]. It is an anonymized 120-day subsets with around 2.4 million URLs detection records. Each day contains about $6,580$ positives and $13,223$ negatives examples on average.

IPTV VoD time series dataset. We collected a dataset with 7-day IPTV VoD (Video On Demand) program (*e.g.,* movies and TV plays) views, named as IPTV, from a giant Telecom company in Mainland China. It involves $72,350,479$ view records from $4,446,247$ users.

To generate the examples, we construct a feature label pair $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$ for each VoD program view record at time t. In particular, $\mathbf{x}^{(t)}$ is a sparse feature vector, with size of $1,321,393$ (as shown in Table 1), encoding the characteristics of both program and user, *e.g.,* program information, user properties, user behaviors and time stamp of user's view. The label $\mathbf{y}^{(t)}$ is 1 if the program was viewed by the user at least 20 minutes or half of the total length of the program; and 0 otherwise.

Algorithm settings. We implemented the following algorithms: (a) Algorithms on Spark 1.5.1 under BSP model: (i) $\mathsf{FTRLProx}_{\mathsf{Spark}}$, the FTRL-Proximal algorithm [25], where α, λ_1 and λ_2 are all set as 0.1 [2]; and (ii) $\mathsf{TDAP}_{\mathsf{Spark}}$ of Sec. 3, where the settings for α, λ_1, λ_2 are taken from $\mathsf{FTRLProx}_{\mathsf{Spark}}$. The parameter γ is carefully chosen for different datasets (to been seen shortly); and (b) Algorithms on

[2]We have varied α, λ_1 and λ_2 from 0.0001 to 1 on all the above datasets, and 0.1 brings the best accuracy on average.

Table 1: Statistics of the datasets

Dataset	# of features	# of positive examples	# of negative examples
Books	332,439	2,264	2,201
DVD	282,899	1,807	1,779
Electronics	235,796	2,857	2,824
Kitchen	205,664	2,954	2,991
RCV1	47,236	355,460	321,939
News	1,355,192	9,999	9,997
URLs	3,231,961	786,182	1,593,948
IPTV	1,321,393	32,869,901	39,480,578

Petuum 1.1 under SSP model [3] (i) $\mathsf{FTRLProx}_{\mathsf{Petuum}}$; and (ii) $\mathsf{TDAP}_{\mathsf{Petuum}}$, the *stale* is set to 2 (as recommended in [33]), other parameter settings are the same to those on Spark (BSP).

Evaluation framework. We conducted the evaluation of the algorithms using three different frameworks in terms of different datasets, where the accuracy metrics used include AUC (Area Under the ROC Curve) and AER [23] (Accumulative Error Rate). Details are as below.

Framework for public non-time series datasets. The framework is designed for public non-time series datasets, where the size of mini-batch is set as one, *i.e.,* once coming one example, the framework conducts prediction and trains the model. The metric, AUC, is calculated after all examples have been traversed.

Framework for URLs dataset. We follow the evaluation framework for URLs as [23], where each mini-batch consists of one example, similar to the above framework. However, the AUC for *both* FTRL-Proximal algorithm and TDAP are very close, which approach to 0.999 (0.999214 for TDAP and 0.999057 for FTRL-Proximal), we thereby use AER instead. The AER is then aggregated and reported in terms of days.

Framework for IPTV dataset. This framework, on the other hand, performs prediction and updates the model on IPTV over each mini-batch formed by time slot with 15 minutes, which follows our business requirements. And still, the AUC is then aggregated and compared in terms of days.

Distributed settings. To evaluate the accuracy of the algorithms, we deployed all algorithms on a mini cluster with $p = 3$ machines, each of which has 24 cores, 96GB memory and 18TB disk. To further study the scalability, we deployed a cluster with $p = 10$ machines, where each is with 8 cores, 64GB memory and 500GB disk. Each experiment is repeated 10 times and the average is reported.

5.2 Experimental Results

Parameter Choosing for TDAP. We first study the accuracy of TDAP by varying the parameter γ, in order to choose good γ for each dataset. We fix the parameters α, λ_1 and λ_2 as mentioned above, for fair comparison.

Figure 6 shows the AUC on Kitchen dataset using $\mathsf{TDAP}_{\mathsf{Spark}}$. The x-axis is γ and y-axis represents the AUC. In particular, by varying γ from 0.1 to 0.000001, the AUC of TDAP becomes stable (approaching 94%) once $\gamma > 0.0005$, and similar for $\mathsf{TDAP}_{\mathsf{Petuum}}$. Hence, we determine $\gamma = 0.0005$ for Kitchen. Note that we conducted the same sets of experiments to determine the value of γ for other datasets (not shown), which are used in later evaluations.

[3]We use Petuum for the implementation of SSP as it is not known how to realize it under Spark.

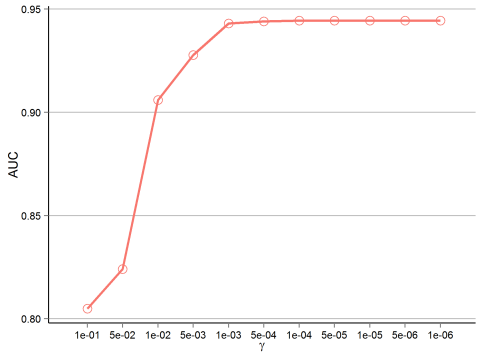

Figure 6: The AUC of TDAP by varying parameter γ on Kitchen dataset using $\mathsf{TDAP}_{\mathsf{Spark}}$; Similar for $\mathsf{TDAP}_{\mathsf{Petuum}}$.

Accuracy Comparison Between FTRL-Proximal and TDAP. After determining γ of TDAP for each dataset, we are ready to compare its accuracy to FTRL-Proximal. As the comparison between $\mathsf{FTRLProx}_{\mathsf{Petuum}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$ shows similar results, for space constraints, we only report the results on $\mathsf{FTRLProx}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Spark}}$.

Accuracy on public non-time series datasets. Table 2 reports the AUC comparison between $\mathsf{FTRLProx}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Spark}}$ on public non-time series datasets. We note that $\mathsf{TDAP}_{\mathsf{Spark}}$ performs nearly the same as $\mathsf{FTRLProx}_{\mathsf{Spark}}$ on Books, Dvd, Electronics, Kitchen and RCV1. However, for News, $\mathsf{TDAP}_{\mathsf{Spark}}$ improves the accuracy by at most 12%. The hypothesis held by FTRL-Proximal is that the effects of all historical models are equivalent. Although this might be suitable for most non-time series datasets, we still believe that, according to "no free lunch" theorem, treating historical models differently can lead to better performance in some datasets, *e.g.,* News in our experiments.

Table 2: The AUC of $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{FTRLProx}_{\mathsf{Spark}}$ on all public non-time series datasets. The best value for each dataset is in bold.

Dataset	$\mathsf{FTRLProx}_{\mathsf{Spark}}$	$\mathsf{TDAP}_{\mathsf{Spark}}$
Books	**0.8944**	0.8914 ($\gamma = 0.0005$)
Dvd	**0.8838**	0.8792 ($\gamma = 0.0005$)
Electronics	**0.9278**	0.9249 ($\gamma = 0.0005$)
Kitchen	**0.9448**	0.9440 ($\gamma = 0.0005$)
RCV1	0.9904	**0.9936** ($\gamma = 0.05$)
News	0.8885	**0.9952** ($\gamma = 0.1$)

Accuracy on URLs dataset. Figure 7 shows the AER, *i.e.,* the percentage of misclassified examples for all URLs encountered up to date, for $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{FTRLProx}_{\mathsf{Spark}}$ on URLs. The x-axis is the number of days and y-axis is the AER. From the figure, we find out that the AER of $\mathsf{TDAP}_{\mathsf{Spark}}$ approaches 1.4%, which outperforms $\mathsf{FTRLProx}_{\mathsf{Spark}}$ (AER with 1.6%) by 12.5% on average. It is worth remarking that the AUC for both algorithms are very close, both approach 0.999, and the value of $\mathsf{TDAP}_{\mathsf{Spark}}$ increases 0.016% compared to that of $\mathsf{FTRLProx}_{\mathsf{Spark}}$. Hence, we only report the AER in this part.

Accuracy on IPTV dataset. Figure 8 shows the AUC on

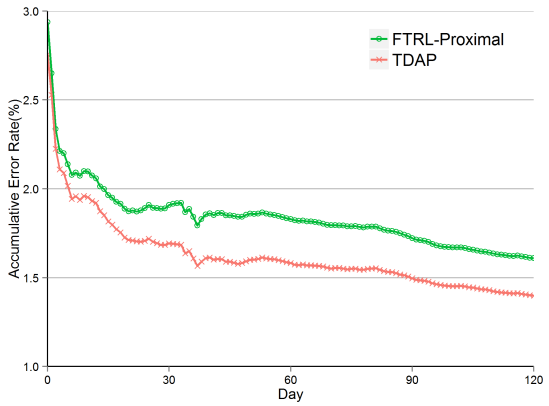

Figure 7: The AER of $\mathsf{TDAP}_{\mathsf{Spark}}$ (with $\gamma = 0.00001$) and $\mathsf{FTRLProx}_{\mathsf{Spark}}$ on URLs dataset.

IPTV time series dataset. Not surprisingly, the results show that $\mathsf{TDAP}_{\mathsf{Spark}}$ outperforms $\mathsf{FTRLProx}_{\mathsf{Spark}}$ by around 5.6% on average. Such indicates that our strategy to deemphasize the effects of historical models in model training can definitely improve the accuracy, when the data are changing fast as motivated in Figure 1.

In our running production service, when we switched from FTRL-Proximal to TDAP, TDAP significantly increased the degree of user activity, *i.e.,* there were approximately 5% more users engaged and each user took 10% more time than ever, due to the accuracy improvement compared to the former one. Such accordingly brought more revenue than the existing one (using FTRL-Proximal) for our customers.

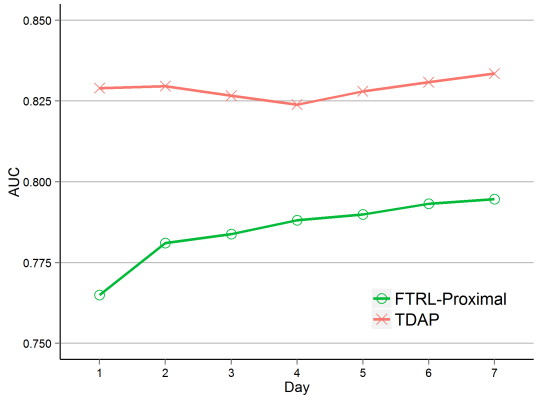

Figure 8: The AUC of $\mathsf{TDAP}_{\mathsf{Spark}}$ ($\gamma = 0.0005$) and $\mathsf{FTRLProx}_{\mathsf{Spark}}$ on IPTV dataset.

Scalability of TDAP. As TDAP shows good accuracy, we next verify its scalability. We conducted the experiments on IPTV dataset by varying $p \in [2, 10]$ using $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$, the results are shown in Figure 9.

As expected, $\mathsf{TDAP}_{\mathsf{Petuum}}$ shows an excellent scalability, which achieves 4x speedup when p increases from 2 to 10. However, $\mathsf{TDAP}_{\mathsf{Spark}}$ can only achieve 3x speedup, no matter how we increase the machines, *i.e.,*$\mathsf{TDAP}_{\mathsf{Petuum}}$ enjoys better scalability with more machines than $\mathsf{TDAP}_{\mathsf{Spark}}$. The major reasons are two folds: (a) SSP avoids the "blocking" of stragglers in each BSP round, by setting the staleness (as revealed in [13]); (b) Petuum uses fine-grained caching strategies and distributed shared memory to further reduce the costs of network communication [33], whereas Spark's RDD system performs caching at a coarser granularity [27] and thus requires more communication.

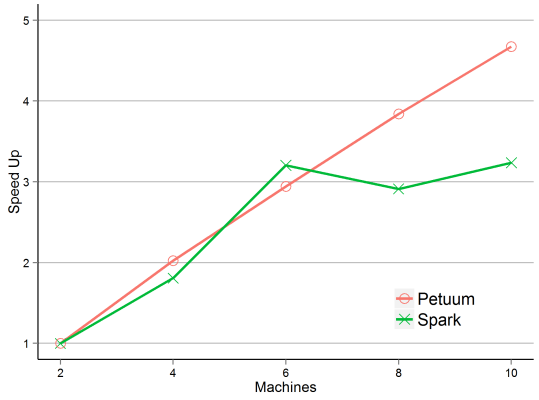

Figure 9: Speed-Up of $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$. The higher the slope of curve, the better the scalability. Both of them show good scalability, but $\mathsf{TDAP}_{\mathsf{Petuum}}$ is better.

Efficiency Comparison between $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$. We next focus on the efficiency comparison between $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$. Figure 10 shows the training time of both algorithms under the same setting (using $p = 3$ machines). We highlight that the training time of $\mathsf{TDAP}_{\mathsf{Petuum}}$ is *at least* 1/20 of that of $\mathsf{TDAP}_{\mathsf{Spark}}$, which is also exhibited in [33] due to the same reasons explained above.

Still in our production service, we observed that Petuum can process approximately *one order of magnitude* more users than Spark, with the same cluster setup, *i.e.,* the throughput of Petuum was at least 10 times larger than that of Spark. Hence, TDAP on Spark was possible, but the deployment was easier and more efficient using Petuum.

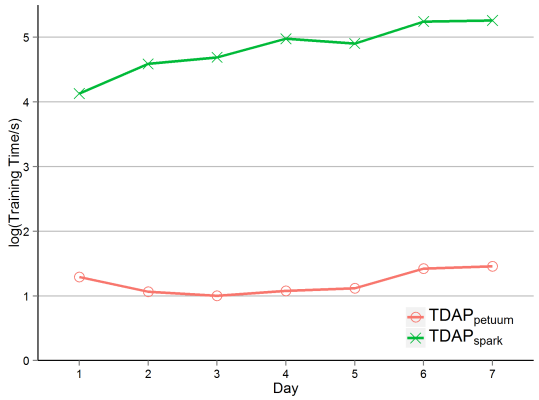

Figure 10: Training time (s) of $\mathsf{TDAP}_{\mathsf{Spark}}$ and $\mathsf{TDAP}_{\mathsf{Petuum}}$.

Summary. We find the followings: (a) TDAP gives better accuracy than FTRL-Proximal: TDAP improves 5.6% prediction accuracy on average over our datasets; (b) TDAP has good scalability: $\mathsf{TDAP}_{\mathsf{Petuum}}$ (resp. $\mathsf{TDAP}_{\mathsf{Spark}}$) is 4 (resp. 3) times faster when p increases from 2 to 10; and (c) $\mathsf{TDAP}_{\mathsf{Petuum}}$ outperforms $\mathsf{TDAP}_{\mathsf{Spark}}$ in terms of efficiency: $\mathsf{TDAP}_{\mathsf{Petuum}}$ runs at least 20 times faster than $\mathsf{TDAP}_{\mathsf{Spark}}$.

6. CONCLUSIONS AND DISCUSSION

Conclusions. In this paper, we have defined time-decaying online convex optimization TOCO problem, where the target model approaches to the most recent models while older history of the models are deemphasized, to tackle the fast-changing data. We have proposed a time-decaying adaptive prediction TDAP algorithm to solve TOCO problem, where recursive closed forms of the model update functions have been designed. To scale big data, we have parallelized the TDAP algorithm under both BSP and SSP consistency model. Using real-world datasets, we have experimentally verified that TDAP achieves good accuracy, scalability and efficiency, on both models. We further observe that TDAP on Spark is possible, but it is easier and more efficient for deployment on Petuum in practice.

Discussion. We note that the proof of *regret bounds* for TDAP, as well as $\mathsf{TDAP}_{\mathsf{BSP}}$ and $\mathsf{TDAP}_{\mathsf{SSP}}$, are *not* the focus of this paper. We refer these to the future work.

7. REFERENCES

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *JMLR*, 15(1):1111–1133, 2014.

[2] D. Agarwal, B.-C. Chen, and P. Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *SIGKDD*, pages 703–712, 2010.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[4] D. M. Blitzer J and P. F. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. *ACL*, 7:440–447, 2007.

[5] H.-I. Choi and W. J. Williams. Improved time-frequency representation of multicomponent signals using exponential kernels. *TSP*, 37(6):862–871, 1989.

[6] T. H. Cormen and M. T. Goodrich. A bridging model for parallel computation, communication, and i/o. *CSUR*, 28:208, 1996.

[7] S. V. S. D. Cormode, G. and B. Xu. Forward decay: A practical time decay model for streaming systems. In *ICDE*, pages 138–149, 2009.

[8] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, and E. P. Xing. High-performance distributed ml at scale through parameter server consistency models. In *AAAI*, 2015.

[9] C. B. Do, Q. V. Le, and C.-S. Foo. Proximal regularization for online and batch learning. In *ICML*, pages 257–264, 2009.

[10] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic control, IEEE Transactions on*, 57(3):592–606, 2012.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46, 2014.

[12] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *SIGKDD*, pages 69–77, 2011.

[13] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *NIPS*, pages 1223–1231, 2013.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *SIGOPS Review*, volume 41, pages 59–72, 2007.

[15] S. S. Keerthi and D. D. A modified finite newton method for fast solution of large scale linear svms. In *JMLR*, volume 6, pages 341–361, 2005.

[16] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8:281–300, 2004.

[17] I. Koychev. Gradual forgetting for adaptation to concept drift. ECAI Workshop, 2000.

[18] A. Kyrola, D. Bickson, C. Guestrin, and J. K. Bradley. Parallel coordinate descent for l1-regularized loss minimization. In *ICML*, pages 321–328, 2011.

[19] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. In *NIPS*, pages 905–912, 2009.

[20] K. B. Lee, J. Siegel, S. Webb, S. Leveque-Fort, M. Cole, R. Jones, K. Dowling, M. Lever, and P. French. Application of the stretched exponential function to fluorescence lifetime imaging. *Biophysical Journal*, 81(3):1265–1274, 2001.

[21] R. T. G. e. a. Lewis D D, Yang Y. Rcv1: A new benchmark collection for text categorization research. In *JMLR*, volume 5, pages 361–397, 2004.

[22] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, pages 583–598, 2014.

[23] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *ICML*, pages 681–688, 2009.

[24] H. B. McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, pages 525–533, 2011.

[25] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *SIGKDD*, pages 1222–1230, 2013.

[26] H. B. McMahan and M. Streeter. Adaptive bound optimization for online convex optimization. *COLT*, 2010.

[27] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015.

[28] T. Moon, L. Li, W. Chu, C. Liao, Z. Zheng, and Y. Chang. Online learning for recency search ranking using real-time user feedback. In *CIKM*, pages 1501–1504, 2010.

[29] S. Provencher. A fourier method for the analysis of exponential decay curves. *Biophysical journal*, 16(1):27, 1976.

[30] Y. Singer and J. C. Duchi. Efficient learning using forward-backward splitting. In *NIPS*, pages 495–503, 2009.

[31] W. Windig and B. Antalek. Direct exponential curve resolution algorithm (decra): a novel application of the generalized rank annihilation method for a single spectral mixture data set with exponentially decaying contribution profiles. *Chemometrics and Intelligent Laboratory Systems*, 37(2):241–254, 1997.

[32] L. Xiao. Dual averaging method for regularized stochastic learning and online optimization. In *NIPS*, pages 2116–2124, 2009.

[33] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: A new platform for distributed machine learning on big data. *SIGKDD*, 2015.

[34] Y. Zhang and M. I. Jordan. Splash: User-friendly programming interface for parallelizing stochastic algorithms. *arXiv preprint arXiv:1506.07552*, 2015.

[35] B. Zhao, L. Fei-Fei, and E. P. Xing. Online detection of unusual events in videos via dynamic sparse coding. In *CVPR*, pages 3313–3320, 2011.

[36] M. Zinkevich, J. Langford, and A. J. Smola. Slow learners are fast. In *NIPS*, pages 2331–2339, 2009.

[37] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *NIPS*, 2010.