Spice: Socially-Driven Learning-Based Mobile Media Prefetching

Chao Wu†, Xu Chen‡, Yuezhi Zhou†, Ningyuan Li‡, Xiaoming Fu†, and Yaoxue Zhang‡
†Department of Computer Science and Technology, TNList, Tsinghua University
‡Institute of Computer Science, University of Goettingen
∗Software College, Zhongyuan University of Technology

Abstract—Mobile online social networks (OSNs) are emerging as the popular mainstream platform for information and content sharing among people. In order to provide Quality of Experience (QoE) support for mobile OSN services, in this paper we propose a socially-driven learning-based framework, namely Spice, for media content prefetching to reduce the access delay and enhance mobile user's satisfaction. Through a large-scale data-driven analysis over real-life mobile Twitter traces from over 17,000 users during a period of five months, we reveal that the social friendship has a great impact on user’s media content click behavior. To capture this effect, we conduct social friendship clustering over the set of user’s friends, and then develop a cluster-based Latent Bias Model for socially-driven learning-based prefetching prediction. We then propose a usage-adaptive prefetching scheduling scheme by taking into account that different users may possess heterogeneous patterns in the mobile OSN app usage. We comprehensively evaluate the performance of Spice framework using trace-driven emulations on smartphones. Evaluation results corroborate that the Spice can achieve superior performance, with an average 67.2% access delay reduction at the low cost of cellular data and energy consumption. Furthermore, by enabling users to offload their machine learning procedures to a cloud server, our design can achieve speed-up of a factor of 1000 over the local data training execution on smartphones.

I. INTRODUCTION

The past decade has witnessed the wide penetration of online social networks (OSNs) such as Facebook and Twitter into our daily lives. With the pervasivity and popularity of wireless communication such as WiFi and LTE, more and more users are accessing OSN services on mobile devices via wireless connection. It is reported that nowadays 68% of the OSN service consumptions occur on mobile devices [1], and on average a mobile user spends 2 hours and 25 minutes per day using OSN services, accounting for more than 20% of the overall mobile traffic [2].

Besides serving as the platform for social interaction, OSN is emerging as the mainstream channel for information and content sharing. For instance, over 52% and 47% of the users get news from Twitter and Facebook, respectively [3]. Moreover, a significant part of the shared content contains media files such as images and videos, which typically have much larger data size than that of the text content in users’ posts. The increasing popularity and ubiquity of such media content in OSN calls for a mobile-friendly design in order to provide QoE support for mobile devices.

A key factor of degrading the mobile user’s satisfaction in consuming the rich media content in OSN is the access delay. On one hand, limited network bandwidth, high wireless connection establishment latency and long roundtrip time of data transmission (varying from 3 seconds to 10 seconds or more [4]) would impair the real-time responsiveness of users’ daily social media usages, in particular when users try to access media files in social posts/tweets. On the other hand, time-varying network quality and sporadic network availability cause fluctuating connection and intermittent access. This would also incur excessive latency overhead for their social interaction engagement in OSNs.

To address this issue, an intriguing and promising approach is to leverage prefetching, i.e., to download the media content prior to user’s consumption whenever possible [5]. To embrace the profound benefit of prefetching, the key challenge is how to predict her media content click behavior in a precise manner. Achieving accurate content prediction can help to prefetch the most relevant content items which will be consumed by the user in the near future with high probability. This is useful to significantly reduce the access delay and meanwhile saving both energy and data traffic consumption by avoiding excessive content prefetching.

To boost the prediction accuracy of media content prefetching in OSNs on mobile devices, a very recent study in [6] proposed a framework of EarlyBird. The key idea is, by mining the user’s OSN usage pattern, to integrate tweet training features (e.g., image embedded or not, the specified recipient) into the linear regression model for prediction. A key drawback of the proposed approach in [6] is that it does not provide sufficient consideration for social influence among the users (i.e., social interaction patterns), which plays a critical role in media content consumption in OSNs [7]. Intuitively, if a tweet with an image is sent from her close friend rather than some acquaintance with infrequent contact, then she would click the image with a high probability (see Table II).

Motivated by this insight, in this paper we propose a novel framework of Spice, which utilizes the unique characteristics of social interactions among users in OSNs for mobile media prefetching. To this end, we leverage the tools of socially-driven data mining and cluster-based machine learning, to infer a user’s potential interest in media content consumption based on her history content usage pattern and social friendship
follows: on the smartphones. and long processing latency when executing these tasks locally to a cloud server, in order to combat the high energy consumption of all personal data-related fields will be carried out before storing the data on the mobile device. Later, the locally stored data is uploaded to the cloud server only for further analysis when the mobile device is charging and connecting with WiFi.

The Data Aggregator also passes the received information to the Content Predictor component, where the learning-based content prediction model is trained for predicting the likelihood whether she would click the media in a new tweet. Specifically, this predictor would take the user’s new tweets and the relevant features of these tweets as an input to a machine learning model, in order to identify the relevant media content (e.g., image files) contained in these tweets as the prefetch candidates. These media files are then to be prefetched by the Content Prefetcher component. Note that, to speed up the whole process, we offload the machine learning procedure to a cloud server. When such a cloud server is not available, we can carry it out on the mobile device locally.

We comprehensively evaluate the performance of the Spice framework using trace-driven emulations on smartphones. Evaluation results show that an average Spice user can reduce her access delay by 67.2% at the low cost of cellular data and energy consumption, which is a significant improvement over the benchmark approaches.

Moreover, by enabling users to offload machine learning procedures to a cloud server, we can achieve a speed-up of a factor of 1000 over the local execution on smartphones.

Fig. 1. Spice Architecture

The rest of this paper is organised as follows: Section II outlines system overview of Spice, and describes how we conduct data collection through the Twidere app. Section III analyses the social interaction among users and reveals its impact on the user’s tweet click behavior. Section IV proposes the socially-driven learning-based prefetching prediction mechanism. Section V tackles the problem of prefetching scheduling. Section VI conducts the trace-driven emulation evaluation on smartphones. Section VII reviews the related work, and Section VIII concludes this paper.

II. SPICE OVERVIEW

A. System Architecture

We now introduce the system architecture of Spice for media content prefetching in mobile OSNs. As illustrated in Fig. 1, Spice works in a user-centric manner (i.e., implemented on a user’s mobile device), and collects traces about all tweets on the user’s feed when accessing Twitter with the Twidere app [8]. These traces were retrieved using the Twitter REST API [9], located in the Twitter Wrapper, which is controlled by the Task Scheduler component to periodically query for new tweets on her newsfeed (see Section V).

Then the retrieved tweets and user information are passed to the Data Aggregator component. To ensure the user privacy, text content in tweets are not recorded and the anonymization of all personal data-related fields will be carried out before directly storing the data on the mobile device. Later, the locally stored data is uploaded to the cloud server only for further analysis when the mobile device is charging and connecting with WiFi.
B. Data Collection

As mentioned above, we collect data traces from the users using Twidere app. This is because, although Twitter’s contents are publicly available, information about when, how, and where they access these social streams are not available in particular in the mobile environment. Therefore, we collected a large set of usage data from Twidere users \(^1\) who agreed to provide their information to us anonymously.

As the aim is to enable intelligent prefetching by identifying the tweets that the user is most interested in, a set of tweet attributes are collected as well. To this end, the Twitter Wrapper tracks the user interaction information (e.g., retweet, favourite, or mention) of the individual tweets. The source of a tweet is also recorded by identifying whether the tweet is obtained from a direct friend or propagated through friends of others’ friends. Furthermore, with the consent from the user, the Twidere app enables us to keep track of her activity events when reading the tweets, e.g., watching, liking, or commenting along the timeline. The collected trace items are shown in Table 1.

During five months from March 2015 to July 2015, we have collected data traces from more than 17,000 users from all over the world with a diverse demographic composition. The users refreshed 238 million tweets and clicked 9.6 million of them in total. Also, 72% of the clicked tweets contain media content (e.g., image). The volume and diversity of data reflects the real-life behavior of the mobile social app users, which is crucial for understanding and predicting media content in mobile social application network traffic. In our prediction algorithm, we are primarily interested in tweets with media content because the plaintexts of the tweets are simply short character strings, which are of very small size and would not play a significant role for prefetching design. Therefore, unless otherwise specified, we mainly consider media tweets with images \(^2\), and target at prefetching them in an appropriate manner.

\(^1\)Twidere discloses the usage statistics information during its installation and update. Usage statistics only comes in effect after users choose to opt in. Up to now, there are around 23% of users grant us permissions, which indicates user privacy-awareness and the effectiveness of the privacy disclosure. In addition, we only keep the user data in a numerical and anonymous style. Last, twitter contents and social graph are publicly available information.

\(^2\)Note that the Twidere app currently just supports few video formats and the total number of video sharing is very limited, thus we mainly focus on image prefetching. We will extend our framework to the case with videos for other mobile OSNs in a future work.

### TABLE I

<table>
<thead>
<tr>
<th>Events</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Launch and Close</td>
<td>Timestamp</td>
</tr>
<tr>
<td>Network Availability</td>
<td>Timestamp, connection pattern</td>
</tr>
<tr>
<td>Tweets Click</td>
<td>Timestamp, tweet’s attributes, participant</td>
</tr>
<tr>
<td>Embedded Media Preview</td>
<td>Timestamp, tweet ID, preview URL</td>
</tr>
<tr>
<td>Media Click</td>
<td>Timestamp, tweets ID list, link URL</td>
</tr>
<tr>
<td>Coarse Location</td>
<td>Timestamp, coordinates</td>
</tr>
<tr>
<td>Others</td>
<td>Tweets’ favorite, retweet, and publish</td>
</tr>
</tbody>
</table>

Fig. 2. Number of clicked media tweets from user’s friends in log-log scale manner.

III. THE IMPACT OF SOCIAL FRIENDSHIP

In this section, we first conduct a data-driven analysis on the social interaction among users and reveal its impact on the user’s tweet click behavior.

The generation and re-share of a tweet on Twitter is simple: any user who generates or re-shares it will become a new host of the tweet content. Users can fetch these contents from their direct friends in the social network. Intuitively, the social relationships and interactions among a user and her friends have a significant impact on the Twittering behavior. A user may treat different friends differently, and interact with some close friends frequently, while having little contact or response with some unfamiliar friends on Twitter \(^10\).

In Fig. 2, we plot the number of one real-life user’s clicked tweets with media content (i.e., media tweets) from her friends (i.e., social neighbors) on Twitter in the log-log scale. We rank the set of friends in descending order according to the number of tweets sent by them. We observe a strong power law phenomenon, i.e., almost 85% of the tweets are from only a few friends (less than 5%), and most other friends have little contribution. This demonstrates that friendship (or social interaction strength) plays a critical role on shaping her usage behavior on Twitter. Thus, it motivates us to design the socially-driven prefetching mechanism by leveraging the social friendship among users.

With this insight, we further quantify the impact of social friendship on the user’s media tweet click behaviors. To proceed, we first carry out the social friendship clustering. The intuition is that in reality a user typically has very close relationships with a small set of people (e.g., family member, close friends), and is familiar with a group of people (e.g., colleagues). For many other people, the user would have little contact with them. With this observation, we conduct the friendship clustering using the commonly-adopted K-Means clustering algorithm \(^11\). As illustrated in Fig. 3, we utilize the number of tweets received from a specific friend and the number of tweets sent by the user to that friend as the clustering features, and cluster the set of her friends into three types: close friends (i.e., Cluster 1 in Fig. 3), familiar friends (i.e., Cluster 2 in Fig. 3), and acquaintances with infrequent relationships and interactions among a user and her friends

\(^10\)Twidere discloses the usage statistics information during its installation and update. Usage statistics only comes in effect after users choose to opt in. Up to now, there are around 23% of users grant us permissions, which indicates user privacy-awareness and the effectiveness of the privacy disclosure. In addition, we only keep the user data in a numerical and anonymous style. Last, twitter contents and social graph are publicly available information.

\(^11\)Note that the Twidere app currently just supports few video formats and the total number of video sharing is very limited, thus we mainly focus on image prefetching. We will extend our framework to the case with videos for other mobile OSNs in a future work.
contacts (i.e., Cluster 3 in Fig. 3). We will elaborate the impact of the number of friendship clusters later.

After the social friendship clustering, we then explore the impact of friendship on the user’s media click behavior when accessing the media tweets. Table II summarizes the media click probability under different scenarios. In total, we observe that the user clicks the media file with a probability of 0.64 (0.28, 0.13), when the media tweet is sent by a close (familiar, unfamiliar) friend, respectively. This again show that social friendship has a significant impact on user’s media click behavior. We further explore the impact of social friendship on user’s media click behavior with different features. For example, for the network feature in Table II, if the user is on WiFi and the media tweet is sent by a close friend, then she would click the media file with a probability of 0.71. However, if the tweet is sent by a familiar friend, the click probability would decrease to 0.19. As another example, for the interaction feature, if the media tweet has been favored by a close friend, then the user would click the media file with a high probability of 0.98.

Inspired by the observation above that the social friendship has a significant impact, we then develop a socially-driven scheme for the prefetching prediction based on machine learning.

IV. LEARNING ALGORITHM FOR SOCIALLY-DRIVEN PREFETCHING PREDICTION

In this section we propose the learning-based socially-driven prefetching prediction mechanism as follows.

A. Social Friendship Clustering

As discussed above, we first carry out the social friendship clustering process to partition the set of the user’s friends into several friendship clusters. Different clusters represent different levels of social interactions between the user and her friends. Specifically, we utilize the number of tweets received from a specific friend and the number of tweets sent by the user to that friend as the clustering features, and use the commonly-adopted K-Means clustering algorithm [11] to carry out the social friendship clustering. In the following, we denote the number of friendship clusters as $K$, and the set of clusters as $\mathcal{C}$. For example, we can set $K = 3$ and $\mathcal{C} = \{\text{close friends}, \text{familiar friends}, \text{unfamiliar friends}\}$. Note that the number of friendship clusters $K$ generates a great impact on the accuracy of the proposed prefetching prediction mechanism later. Intuitively, when $K$ is too small, the impact of the social friendship is not well leveraged, which would impair the prediction accuracy. When $K$ is too large, the number of training parameters increases significantly due to the large number of friendship clusters. As a result, in order to train the prediction mechanism well, more fine-grained data traces with more detailed information with respect to all the friendship clusters are needed. However, in practice it is extremely difficult to obtain such fine grained data traces since many friends interact infrequently and reveal insufficient information. According to our experience (as illustrated in Fig. 4), typically the number of clusters $K = 3$ can provide a good balance and achieve the best prediction accuracy. This is also consistent with our daily life observation that people tend to classify their friends into three types: close friends, familiar friends, and acquaintances (with infrequent contact).

B. Tweet Training Features

After the social friendship clustering, we identify the set of important tweet training features for building up the learning model for prefetching prediction. As shown in Table II, we found that two types of features are critical: network and interaction features. For the network features, whether the user would click the media file depends on the network environment \(^3\) (on WiFi or cellular). For the interaction features, the events that the media tweet is published by, mentioning, favored by, retweeted by, or reply to by a friend are also taken into account. In the following, we denote the number of training features as $F$, and the set of all tweet training features as $\mathcal{F}$.

C. Cluster-Based Latent Bias Model

We then propose the learning model for prefetching prediction. Our algorithm design is based on the Latent Bias Model (LBM) [13] that aims to utilize proper bias terms to capture the importance of different features for prediction. Here we extend the standard LBM to our case with friendship clustering, and

Note that in this paper we utilize the widely-used Markov Chain model in [12] to predict user’s network environment during a prefetching prediction slot.

\(^3\)Note that in this paper we utilize the widely-used Markov Chain model in [12] to predict user’s network environment during a prefetching prediction slot.
The cluster-based LBM approach for socially-driven prefetching prediction.

Specifically, based on the social friendship clustering, we define $b_{f,k}$ as the cluster-based bias term to stand for the case that a media tweet is sent by a friend in the friendship cluster $k$ and contains the feature $f$. Moreover, for a given media tweet $c$, we first introduce an indicator function $I_{f,k}^c \in \{0, 1\}$ such that $I_{f,k}^c = 1$ if the media tweet $c$ is sent by a friend in the friendship cluster $k \in K$ and contains the feature $f \in F$; $I_{f,k}^c = 0$ otherwise. Then, we define the media click score for media tweet $c$ as follows:

$$\gamma_c = \alpha + b_0 + \sum_{f \in F} \sum_{k \in K} b_{f,k}I_{f,k}^c,$$

where $\alpha$ is the user's average click rate across all historical media tweets, and $b_0$ is the overall bias term for the user. In general, a higher media click score $\gamma_c$ implies a higher probability that the user will click the media file in the media tweet $c$.

Then, the critical task is to train the cluster-based LBM, i.e., learn the proper bias terms in (1) in order to well capture the user’s media click behavior. Suppose that the set of historical user data trace (i.e., historical media tweet set of the user) is denoted as $C$. For each media tweet $c \in C$, we have the ground truth $y_c \in \{1, -1\}$ such that $y_c = 1$ means that the user clicks the media file in the tweet $c$ and $y_c = -1$ otherwise. Next, to quantify the discrepancy between the prediction based on the media click score $\gamma_c$ and the desired ground-truth output $y_c$, we adopt the widely-used logistic loss measure, i.e.,

$$\mathcal{L}(\gamma_c, y_c) = \log[1 + \exp(-\gamma_c y_c)].$$

Thus, we learn the proper bias terms to minimize the total loss across the historical data trace $C$, i.e., $\sum_{c \in C} \mathcal{L}(\gamma_c, y_c)$. Following the common practice in machine learning, in order to avoid overfitting, we also impose $L_2$ regularization into minimization. That is, we minimize:

$$O = \sum_{c \in C} \mathcal{L}(\gamma_c, y_c) + \lambda \left( ||b_0||^2 + \sum_{f \in F} \sum_{k \in K} ||b_{f,k}||^2 \right),$$

where $\lambda$ is the regularization parameter to be manually tuned.

Since the objective function in (3) is convex, we can apply the first-order condition and derive the gradients as

$$\frac{\partial O}{\partial b_0} = -\sum_{c \in C} \left( \frac{\exp(\gamma_c y_c)}{1 + \exp(\gamma_c y_c)} \right) y_c + 2\lambda b_0,$$

$$\frac{\partial O}{\partial b_{f,k}} = -\sum_{c \in C} \left( \frac{\exp(\gamma_c y_c)}{1 + \exp(\gamma_c y_c)} \right) y_c I_{f,k}^c + 2\lambda b_{f,k}.$$

Similar to many machine learning studies, we adopt the Stochastic Gradient Descent (SGD) method [14] to learn optimal bias terms [14]. The key idea is to utilize the data samples to iteratively update the gradient as follows:

$$b_t^{f,k+1} = b_t^{f,k} - \epsilon_t \frac{\partial O}{\partial b_t^{f,k}},$$

where $b_t^{f,k}$ denotes a given bias term at the $t$-th iteration and $0 < \epsilon_t < 1$ is the smoothing factor for updating. As shown in [14], as long as a small enough $\epsilon_t$ is used, the SGD method converges to the optimum.

After learning the optimal bias terms from the historical data trace $C$, we then rank the new coming media tweet $\mathcal{H}$ for prefetching prediction during a given time slot (e.g., every 15 minutes). Specifically, for each media tweet $c \in \mathcal{H}$, we compute its media click score $\gamma_c$ using the learned bias terms. Accordingly, we rank the new media tweets in the set $\mathcal{H}$. Finally, as illustrated in Fig. 8, we determine the number of media tweets to be prefetched, based on the statistics of the average number of media tweet clicks during that specific time slot (e.g., weekend or weekday, and which slot).

Fig. 4 shows the performance of the proposed cluster-based LBM algorithm with different number of friendship clusters. It depicts the cumulative probability distribution (CDF) of the prediction accuracy for all the testing tweets. As discussed before, when the number of friendship clusters equals $K = 3$, it achieves the best performance with the average prediction accuracy of $0.845$ \(^4\). As the benchmark, we also implement the linear regression (LR) algorithm using tweet training features only [6]. Fig. 5 shows that the LR approach can only achieve the average prediction accuracy of $0.638$, which is even slightly worse than standard LBM without social friendship clustering (i.e., the case with one cluster in Fig. 4). This demonstrates the efficiency of the proposed cluster-based LBM approach. The gain of cluster-based LBM mainly stems from the fact that the defined cluster-based bias terms can well capture the impact of social friendship on user’s click behavior.

\(^4\)We have elaborated the reason that $K = 3$ clusters offers the best performance in Section IV-A.
After studying the prefetching prediction, we next consider the prefetching scheduling problem of Spice. Intuitively, different users may show feature significantly different habits/patterns in the mobile social app usage. Motivated by this phenomenon, we schedule the prefetching to be adaptive to a user’s usage pattern. For example, in Fig. 6 and Fig. 8, through tracing a user’s app usage along different hours of a day during the period of two months, we observe that the user shows different patterns for weekday and weekend. For instance, the user tends to use the app more frequently from the time period from 7:00 to 12:59 on weekdays. Furthermore, if we partition the horizon of a day into 4 zones: from 01:00 to 06:59 (midnight), from 07:00 to 12:59 (morning), from 13:00 to 18:59 (afternoon) and from 19:00 to 00:59 (night), the user has different app usage frequencies during different zones. Thus, we can schedule Spice for media tweet prefetching with different frequencies with respect to different specific zones, based on the statistics of the average app usage frequencies in different zones. For instance, for the user in Fig. 6, we invoke Spice every 1 hour in the first zone, in the second zone Spice will be scheduled every 20 minutes on weekdays and every 30 minutes on weekends, in the third zone Spice will be invoked every 15 minutes on weekdays and every 20 minutes on weekends, and in the fourth zone we invoke Spice every 15 minutes on both weekdays and weekends.

To reduce the scheduling complexity and to achieve superior performance, in this paper we equally partition the day horizon into 4 zones and accordingly set different prefetching frequencies in the experiments. Our evaluation experience shows that for our collected dataset more fine-grained day horizon partition do not increase the performance of Spice significantly. In Fig. 7, we compare the prefetching scheduling settings of Spice with different numbers of zones for the day horizon partition. We focus on the primary metrics of delay reduction per day, monthly cellular data usage, and energy consumption per day. We observe that increasing the number of zones from 4 to 5 and 6 makes little improvement in terms of the access delay reduction. However, in addition to the increase of the scheduling complexity, it also results in increased monthly cellular data consumption of Twidere by 2.3 MB and 3.1 MB, respectively. What’s more, daily battery usage climbs from 5.3% to 9.3% and 12.7% respectively. This is due to the fact that increasing the number of zones for the day horizon partition tends to result in more (or even excessive) frequent prefetching scheduling during some busy periods, and hence leads to high energy consumption. This demonstrates the effectiveness of the 4 zone setup for our case. Nevertheless, for other cases (with different OSN datasets) the
number of zones for day horizon partition can be changed adaptively.

Note that to further reduce the cost of cellular data and energy consumption, we can set a lower scheduling frequency in each zone as needed. For example, we can integrate Spice with the Informed Mobile Prefetching (IMP) framework in [5], which is a prefetching scheduling library to enable a mobile app to control the amount of energy and cellular data usage. As an alternative, since WiFi coverage is pervasive and always available at regular locations such as home and workplace, we can also implement the solution of Spice with WiFi only such that we allow the scheduled prefetching to download media files only when the user is on WiFi. Evaluation results in Section VI show that Spice with WiFi only is still very efficient. With respect to the case without prefetching, it can achieve 58.4% access delay reduction with less cellular data consumption at the cost of an increase of daily battery usage by 1.4% only.

VI. EVALUATION

In this section, we conduct the trace-driven emulation evaluation on smartphones to investigate the performance of Spice. To evaluate the cluster-based LBM algorithm on a large-scale dataset, we run a trace-driven evaluation for 1000 long-term users such that each user keeps active for a long and consecutive period of at least 60 days with detailed trace records. The emulator runs on a Google Nexus 5 Android phone (with the CPU type of Qualcomm Snapdragon 800) connected to China Mobile TD-LTE cellular network and also has access to a campus WiFi network. The emulator reads and replays the usage events collected from real-life users, including connecting to or disconnecting from WiFi networks, accessing Twitter, and opening media files in tweets.

A. Comparison of Different Data Processing Approaches

The key component of Spice is to implement the cluster-based LBM algorithm for the data training (i.e., learning the optimal bias terms from the user data traces). In this paper we consider three data processing approaches: 1) processing on smartphone, i.e., we conduct the data processing on Google Nexus 5 phone locally; 2) processing on ordinary sever, i.e., we offload the data processing to the Intel i7-4790 CPU based server; 3) processing on a cloud cluster (e.g., using Spark), i.e., we offload the data processing to the cloud cluster server. The key motivation here is to leverage the strong parallel computing power to speed up the data processing. In Fig 9, we evaluate the average time overhead of these three data processing approaches when a user allows the data training with cellular connectivity. Note that from the perspective of a smartphone, the cellular data usage and energy consumption are the same for both the server and cloud cluster approaches. Thus, we only compare the cloud cluster approach with the smartphone approach. The results are shown in Fig. 10. We observe that the cloud approach consumes less energy (0.02% of total battery usage per day), while leading to a higher cellular data usage (15 MB per month). If a user is more sensitive to energy consumption, she should choose the cloud approach. Otherwise, the user can choose to process the data locally on the smartphone. Furthermore, since the user’s behavior tends to be stable, we can aggregate the training data for a while (e.g., one week), and carry out data training weekly by offloading the training to the cloud only when the user is on WiFi while charging, in order to save both cellular data and energy usage.

B. Comparison of Different Prefetching Strategies

After the data training has been conducted, we next evaluate the performance of the Spice framework running on the smartphones for media content prefetching. As illustrated in Fig. 4, the cluster-based LBM algorithm in Spice is very efficient, and achieves an average prediction accuracy of 0.845. Upon comparison, the linear regression (LR) algorithm using tweet training features can only achieve an average prediction accuracy of 0.638. In the following, we also compare different prefetching strategies given as:

3For simplicity, in this paper, we utilize NVIDIA GTX 970 GPU and Intel i7-4790 CPU to achieve the strong parallel computing power.

4The smartphone approach consumes 3 MB per month since additional data/information from Twitter server is needed for the training.
Spice with WiFi and cellular: We implement the Spice framework using both WiFi and cellular connections, i.e., we allow media prefetching when the user is on either WiFi or cellular.

LR-based prefetching: We replace the cluster-based LBM algorithm by the LR algorithm in the Spice framework using both WiFi and cellular connections.

Spice on launch: We run the cluster-based LBM algorithm in Spice for media prefetching only when the user launches the Twidere app. Otherwise, we do not schedule the media prefetching.

Spice with WiFi only: We run the cluster-based LBM algorithm in Spice for media prefetching using WiFi connection only, i.e., we allow the scheduled prefetching to download media files only when the user is on WiFi.

For the evaluation of each prefetching strategy, the same user events are replayed, and media links are fetched using the WebView component which is provided by the Android framework. WebView provides a notification when the media file downloading has finished, which is used to calculate the loading delay of a media link.

We compare different prefetching strategies in terms of the delay reduction per day (with respect to the case without any prefetching), cellular data usage of Twidere with prefetching (which includes the cellular traffic for both prefetching by Spice and on-demand fetching by the user), and energy consumption of the app per day (i.e., the percentage of the fully-charged battery capacity). The results are shown in Fig. 11. We see that Spice with WiFi and cellular achieves the best performance in terms of delay reduction, which can achieve 67.2% reduction per day. In terms of the overhead, Spice with WiFi and cellular uses 27.3 MB cellular traffic per month\(^7\) and 5.3% battery usage per day on average. Upon comparison, LR-based prefetching can only achieve 54.4% delay reduction with 31 MB cellular traffic per month and 6.8% battery usage per day on average\(^8\). This is due to the fact that LR-based prefetching has a much lower prediction accuracy than that of the cluster-based LBM in Spice. This demonstrates the efficiency of the proposed Spice framework. Furthermore, we observe that Spice with WiFi only achieves the second best performance for the delay reduction, with a 58.4% reduction per day. Spice with WiFi only achieves the lowest cost among all the prefetching strategies, with 21.4 MB per month and 4.6% battery usage per day on average. As per the evaluation result, compared with the case without prefetching, Spice with WiFi only consumes less cellular data traffic with an increase of around 1.4% battery usage per day on average. This is since the WiFi coverage is pervasive and always available at regular locations such as home and workplace, and Spice has a high prefetching prediction accuracy to download the most relevant media files in advance on WiFi. When a user is not sensitive with the cellular data usage, we can utilize Spice with WiFi and cellular to achieve a significant delay reduction. If the user is concerned about the cellular data usage, we can then adopt Spice with WiFi only, which can still achieve a superior performance with a low overhead.

VII. RELATED WORK

In this section, we describe prior work in mobile prefetching and in socially-driven network analysis, and highlight the key differences of Spice from the existing studies.

A. Mobile Prefetching

For the mobile prefetching, [5] presents the Informed Mobile Prefetching (IMP) framework as a prefetching scheduling library that a mobile app is able to link to control the energy and cellular data consumption. In IMP, a strong assumption is that the whole procedure works on the basis that mobile apps provide precise prediction information through mining users’ content usage pattern. In [15], the authors illustrate that inappropriate prefetching can be worthless to mobile users. They adopt Procrastinator to decide whether prefetching tasks should be invoked by considering different constraints, including the network environment (on WiFi or cellular), the user’s data plan, and battery life. Note many related works in the literature (e.g., [5], [15], [16]) target at designing mobile prefetching mechanisms of generic purpose, which can be used for different types of mobile apps. Similar to our work, a recent study in [6] considers the media content prefetching in mobile OSN services, which adopt the linear regression model for prediction by utilizing the tweet training

---

\(^7\)This is the total usage for the whole app. By comparing with the case of Spice with WiFi only later, we see that the extra cellular data usage for prefetching is around 6 MB for Spice with both WiFi and cellular.

\(^8\)In [6], LR achieves the promised performance with strict usage budgets. Otherwise, it consumes more energy and cellular data.
features through mining the user’s OSN usage pattern. Along a different line, motivated by the insight that social friendship plays a critical role on users’ media tweet click behavior, in this paper we propose a novel socially-driven learning-based prefetching prediction based on the generalized cluster-based Latent Bias Model.

B. Socially-Driven Network Analysis

For the socially-driven network analysis, [17] and [18] identify the social graphical structure as a key influence on the interactions of users with social ties using Flickr dataset. A number of recent papers addresses the problem of computing influence in Twitter-like networks and finding leader users whose tweets are influential. [19], [20] detect the influential users by applying the PageRank ranking algorithm based on the number of retweets among users, and [21] utilizes the user attributes such as number of friends, number of followers and past influence of seed users. [22] proposes a variation of PageRank algorithm, accounting for topic specific ranking to measure the influence. Our work does not aim at finding users who are influential directly. Instead, we incorporate the feature that the different social friends make significantly different impact on a user’s likelihood behaviors on media tweet consumption. [23] proposes a tree-based algorithm to mine user-friend graphs to discover strong friends of a user. In contrast to our work, [23] does not consider how to utilize the social friendship structure to facilitate the information and content sharing among users in particular under the rich media content.

VIII. Conclusion

Aiming at designing an intelligent mobile prefetching mechanism, in this paper we first identified the unique features of user’s social behavior in OSN, and then proposed a novel framework of Spice based on the cluster-based LBM learning mechanism for prefetching prediction. We also developed an adaptive prefetching scheduling scheme by mining user’s mobile OSN app usage pattern. We further evaluated the performance of Spice through trace-driven emulation on smartphones. Evaluation results corroborate that the proposed Spice approach can achieve superior performance with a significant access delay reduction at the low cost of cellular data and energy consumption. Moreover, our design enables users to offload their machine learning procedures to a cloud server, and achieves a speed-up of up to 1000 over the local execution on smartphones.

Note that in this paper we propose the Spice framework by using Twitter as a case study. Nevertheless, the proposed techniques can be applied to other OSNs as well. For instance, by integrating the Spice prefetching mechanism, Spice could benefit the Moment module (which contains rich media content for information sharing among friends) of WeChat [24], a popular mobile OSN service with 600 million active users. Moreover, we will consider a comprehensive design to integrate the prefetching techniques enabled by Spice at the mobile side with cloud computing techniques at the content server side in a synergetic manner.

ACKNOWLEDGMENT

We thank the anonymous reviewers, Prof. Wenwu Zhu, Prof. Stephan Sigg, and Dr. Hao Liu for their valuable and insightful comments; Jie Zhang, Bowen Yang, Zipri Huang and others who ever provided feedback on earlier drafts of this paper; we also thank all the 17,000 anonymously worldwide Twidere data providers for their contributions in this paper.

This work is supported by the International Science and Technology Cooperation Program of China under Grant No. 2013DFB10070 and the EU FP7 IRSES MobileCloud Project under Grant No. 612212.

REFERENCES