GeoFeed: A Location-Aware News Feed System

Jie Bao[1] Mohamed F. Mokbel[1] Chi-Yin Chow[2]

[1]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA
[2]Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
{baojie,mokbel}@cs.umn.edu, chiychow@cityu.edu.hk

***Abstract*—This paper presents the GeoFeed system; a location-aware news feed system that provides a new platform for its users to get spatially related message updates from either their friends or favorite news sources. GeoFeed distinguishes itself from all existing news feed systems in that it enables users to post message with spatial extent rather than static point locations, and takes into account their locations when computing news feed for them. GeoFeed is equipped with three different approaches for delivering the news feed to its users, namely, *spatial pull*, *spatial push*, and *shared push*. We design a smart model for GeoFeed to decide about using these approaches in a way that: (a) minimizes the system overhead for delivering the location-aware news feed, and (b) guarantees a certain response time for each user to obtain the requested location-aware news feed. GeoFeed also supports location-aware news feed function for its mobile users. Experimental results, based on real and synthetic data, show that GeoFeed outperforms existing news feed systems in terms of response time and maintenance cost.**

I. Introduction

Social networking systems, e.g., Facebook [6] and Twitter [27], and news aggregators, e.g., My Yahoo! [16] and iGoogle [12], are among the most popular web services nowadays. A common functionality shared by such web services is the *news feed* functionality, where users of social networks and news aggregators receive a set of news from their friends and favorite news sources, respectively. Due to the large volume of related news for each user, existing news feed systems opt to select a subset of k relevant news either based on the message timestamp, i.e., most recent k messages, or based on some diversity requirements. Unfortunately, such a selection ignores the spatial aspect of related messages, and hence, users may miss several important messages that are spatially related to them either because they are not so recent or do not satisfy diversity requirements. For example, when a traveling user logs on to a social network site, the user would like to get the news feed that match his/her new location, rather than sticking to the most recent news feed. The same concept can also be applied for users who keep logging on to the system from the same location, yet, they have a large number of friends. It is of essence for such users to limit their news feed to the ones related to their locations.

In this paper, we present GeoFeed; a location-aware news feed system that provides a new platform for its users to get spatially related message updates from either their friends or favorite news sources. GeoFeed complements the functionality of existing social networks and news aggregators to make them location-aware. Once a user u logs on to her favorite social network site that is equipped with GeoFeed, u will find the set of messages that are more relevant to her current location, e.g., a message about local news, a comment about a local store, or a status message targeting friends in a certain area. For a user u that has a set $\mathcal{F}_u$ of N friends (in a social network context) or follows a set $\mathcal{F}_u$ of N news sources (in a news aggregator context), GeoFeed abstracts the location-aware news feed problem to evaluating a set $\mathcal{Q}_u$ of N location-based queries posed by u. Each query $q_i \in \mathcal{Q}_u$ is posed to a friend $f_i \in \mathcal{F}_u$ to retrieve the set of messages that are issued by f_i and overlap with u's range of interest. u's range of interest could be the exact location of u, in which the location-based queries are point queries, or a range around u, in which location-based queries are range queries, e.g., get all the messages posed by my friends within r miles form my location. To limit the set of messages delivered to u, GeoFeed gets only k messages from each friend $f_i \in \mathcal{F}_u$. In the mean time, GeoFeed guarantees that each user u will get all the requested news feed within a response time threshold $\mathcal{T}_u$.

GeoFeed is equipped with three different approaches for evaluating each query $q_i \in \mathcal{Q}_u$, namely, (1) *spatial pull approach*, in which q_i is answered through exploiting a spatial index over the messages of friend f_i, (2) *spatial push approach*, in which q_i just retrieves the answer from a pre-computed materialized view maintained by friend f_i, and (3) *shared push approach*, in which the pre-computation and the materialized view maintenance at friend f_i are shared among multiple users that include u. Then, the main challenge of GeoFeed is to decide on when to use each of these three approaches to which queries. GeoFeed is equipped with an elegant decision model that decides about using these approaches in a way that: (a) minimizes the system overhead for delivering the location-aware news feed, and (b) guarantees a certain response time $\mathcal{T}_u$ for each user u to obtain the requested location-aware news feed. A better response time calls for using the *spatial push* approach for all queries, where all news feed are pre-computed. However, this results in a huge system overhead to maintain a massive number of materialized views, hence limit the scalability of the system to support more users. In contrast, favoring system overhead may result in evaluating more queries using the *spatial pull* approach as less views are maintained. However, users with large numbers of friends will suffer a significantly long delay when retrieving their news feed. GeoFeed takes these factors into account when deciding

This work is supported in part by NSF under Grants IIS-0811998, IIS-0811935, CNS-0708604, IIS-0952977, a Microsoft Research Gift, and grants from the City University of Hong Kong (Project No. 7200216 and 7002686).

on which approach to use to evaluate each query q_i in a way that minimizes the system overhead, i.e., supports more users, and guarantees a response time threshold.

A distinguishing characteristic in GeoFeed is that it builds its decision model for each single query q_i instead of the whole system or the set of queries $\mathcal{Q}_u$ for a given user. This means that for a certain user u that has two friends f_i and f_j, GeoFeed may opt to retrieve the messages from f_i through the *spatial push* approach while retrieving the messages from f_j through the *spatial pull* approach. Similarly, for a certain user f that feeds two users u_i and u_j, GeoFeed may opt to have u_i retrieve her messages from f with the *spatial push* approach while u_j retrieve her messages from f with the *spatial pull* approach. Extensive experimental results, based on real and synthetic data, show that (a) GeoFeed is favorable over existing news feed systems, and (b) the accuracy of the GeoFeed decision model in guaranteeing user response time while minimizing the total system overhead.

The closest work to ours is the feeding frenzy approach [23], which is a news feed system equipped with *pull* and *push* approaches to retrieve the most recent news feed items. Unfortunately, the feeding frenzy system cannot be directly applied to the location-aware news feed problem as it does not consider the message location aspect at any of its stages. The only way to turn feeding frenzy to be location-aware is to attach a wrapper around it in a form of a spatial filter, which is extremely inefficient as the spatial filter is applied afterthought. Our proposed location-aware news feed system, GeoFeed, distinguishes itself from feeding frenzy [23] and other systems in having all the following aspects: (1) GeoFeed is designed while having the location-awareness in mind, and thus makes use of the spatial extents of each message to early prune non-relevant messages, (2) GeoFeed modifies the traditional *pull* and *push* approaches to support spatial filters, (3) GeoFeed goes beyond the traditional *pull* and *push* approaches, and introduces the *shared push* approach as a third alternative that reduces the system overhead while not sacrificing the user response time, (4) GeoFeed builds its decision model to minimize the system overhead to support more users, while guaranteeing a certain user response time threshold for each user, (5) GeoFeed gives the message issuer the ability to determine the spatial validity of the each posted message, e.g., the weather service provider may decide a tornado warning is relevant only to followers residing in a certain area.

The rest of the paper is organized as follows: Sections II and III give related work and overview of GeoFeed. Section IV discusses the *spatial pull*, *spatial push*, and *shared push* approaches. The cost and decision models of GeoFeed are given in Sections V and VI. Section VII discusses GeoFeed with mobile users. Experimental results are given in Section VIII.

II. Related Work

This section highlights related work to GeoFeed in two main areas; news feed systems and location-aware social networks.

Location-based Social Network	Location Tag	Range Queries	Spatial Messages
Facebook Places [6]	√		
Renren [20]	√		
Sina Weibo [24]	√		
Loopt [14]	√	√	
Google Buzz [10]	√	√	
Foursquare [8]	√	√	
Twinkle [26]	√	√	
GeoFeed	√	√	√

TABLE I
A Taxonomy of Location-based Social Networks

News feed systems. Most of existing news feed systems work in a similar way to publich/subscribe services, e.g., [3], [4], [32], which use a push approach to fan out the message notices to all their users. However, such systems are not applicable to address the location-aware news feed, as (a) they do not consider the spatial relevance of each message, and (b) using the push approach does not scale up for large number of publishers and subscribers as it is the case for social networks. For commercial systems, our only knowledge is about the Feeding Frenzy system [23] from Yahoo!, which we consider as our closest work and compare with in Section VIII. The main idea of Feeding Frenzy is to build a cost model for deciding upon using the *pull* or *push* approaches as means for retrieving the news feed for a registered user. The only way to use the Feeding Frenzy system for the location-aware news feed problem is to attach a wrapper around it to filter any spatially irrelevant message from the users' news feed. However, that would be very inefficient as the spatial filter is applied as an afterthought solution. Our proposed system, GeoFeed, distinguishes itself from Feeding Frenzy in that it is built with the location-awareness functionality in mind. Thus, the query evaluation methods, the cost model, and decision model take into account the spatial aspect of each posted message along with the location of each user.

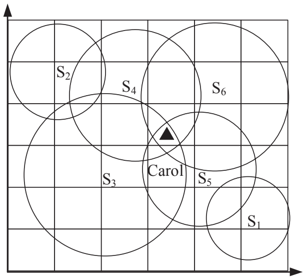
Location-aware social networks. Existing commercial location-based social networks fall in two categories, as summarized in Table I. The first category includes Facebook Places [6], Renren [20], and Sina Weibo [24], where they consider the location information of the message issuer as just an additional tag attached with the message. Then, a system user will get the same news feed (associated with location tags) regardless of the user location. The second category includes Loopt [14], Google Buzz Mobile [10], Foursquare [8], and Twinkle [26] where, in addition to having the location tags, they also give their users the ability to issue range queries to view the whereabouts of their friends. GeoFeed distinguishes itself from all these commercial products in two main aspects: (1) GeoFeed gives its users the ability to set the spatial validity range of each posted message., and hence give control to the message issuer to decide who should get the posted information. For example, the weather service provider may decide a tornado warning is relevant to followers located only in a certain area. (2) Unlike all existing systems that are built mainly to be used by mobile devices, GeoFeed offers a more flexible way for the users to share their geo-tagged messages. Users of GeoFeed can access their account in the same way they use Facebook, yet, they will retrieve more relevant location-aware news feed than that of Facebook users.

Message	*Timestamp*	*Spatial*	*Content*
M_5	14:30	S_5	Back to hotel.
M_3	14:10	S_3	A nice bar.
M_2	14:04	S_2	Eating at a bar.

(a) Location-based messages posted by user Alice

Message	*Timestamp*	*Spatial*	*Content*
M_6	15:00	S_6	Having coffee.
M_4	14:21	S_4	An accident.
M_1	11:40	S_1	Work finished.

(b) Location-based messages posted by user Bob



(c) Spatial areas of the location-based messages

Fig. 1. Location-based messages and news feed.

On the other side, related research prototypes in location-based social networks have either focused on: (a) message sharing [1], [2], where users can broadcast or receive public location-based messages, yet with no social awareness, i.e., there is no concept of friendship. Applying techniques form message sharing to the news feed problem is equivalent to having all queries evaluated with the (spatial) *pull* approach, which is very inefficient, (b) Privacy-aware search queries [9], [13], [21], [22], which enables private query search over users' friends, with no interest of the location-aware news feed functionality, or (c) location-aware recommender systems [17], [29], [30], [31], which suggests new places for their users. The functionality of recommender systems is fundamentally different from news feed systems, where the main goal is predict what the user would like rather than delivering the news feed from posted messages of users' friends.

III. GeoFeed: System Overview

This section gives an overview of GeoFeed as follows:

Location-based messages. GeoFeed users can either send or receive location-based messages to or from their friends with a spatial extent. A location-based message is represented by the tuple: (*MessageID*, *Content*, *Timestamp*, *Spatial*), where *MessageID* and *Content* represent the message identifier and contents, respectively, *Timestamp* is the message generation time, and *Spatial* indicates the spatial extent of the message. Figures 1a and 1b give examples of six messages, as messages M_2, M_3, and M_5 issued by user Alice and M_1, M_4, and M_6 issued by user Bob . The spatial extents of the six messages are represented by the areas S_1 to S_6 in Figure 1c.

System users. A user u, located in $u.location$ with range of interest $u.range$ has a list of friends $\mathcal{F}_u$, who are also considered to be system users, can log on to GeoFeed, and: (a) read all the messages posted from any friend $f_i \in \mathcal{F}_u$ where $u.range$ overlaps with the message spatial extents, and/or (b) post a new message M that should be broadcasted to all friends in $\mathcal{F}_u$ that have range of interest overlap with the spatial extents of M. Each user u has a response time threshold $\mathcal{T}_u$ where GeoFeed guarantees to provide the location-aware news feed for u within $\mathcal{T}_u$. The value of $\mathcal{T}_u$ for each user could be set by the system as either a default value for all users, or a value that reflects how valued and appreciated is the user.

Location-based news feed queries. GeoFeed abstracts the location-aware news feed functionality for a user u to a set $\mathcal{Q}_u$ of location-based queries posed to the user's friends $\mathcal{F}_u$. Each query $q_i \in \mathcal{Q}_u$ is posed to a friend $f_i \in \mathcal{F}_u$ to retrieve the k most recent spatially relevant messages to u's location. Then, u may opt to get all the received messages, or select only the k most recent and spatially relevant messages from *all* users. Similar to the feeding frenzy system [23], we will only focus on retrieving k message from *each* user, as an additional filter will be a trivial step. As an example, consider user Carol, depicted by a triangle in Figure 1c, which is a friend of Alice and Bob. As the spatial range of interest of Carol includes only her location, she issues two location-based point queries, with $k = 2$, one for Alice (returns M_3 and M_5) and one for Bob (returns M_4 and M_6). Carol can add an additional filter to get the most recent two messages, i.e., M_5 and M_6. The location-based news feed query can also use spatial range instead of the point location to retrieve the most recent k messages that overlap with the querying range.

Query evaluation methods. GeoFeed is equipped with three different approaches for evaluating each query $q_i \in \mathcal{Q}_u$, namely, *spatial pull*, *spatial push*, and *shared push* approaches. Details of these approaches will be discussed in Section IV.

Problem formulation. The decision model of GeoFeed can be formulated as follows: *For each location-based news feed query q posed by a user u, find out the best approach among spatial pull, spatial push, and shared push approaches, to evaluate q once u logs on to the system in a future time, such that: (a) the GeoFeed computational overhead for all system queries is minimized, and (b) the response time that u will encounter to get all the requested location-aware news feed is within the required threshold $\mathcal{T}_u$.*

IV. GeoFeed Query Evaluation

In this section, we present three different query evaluation approaches, *spatial pull*, *spatial push*, and *shared push*, to evaluate a location-based news feed query posed from user u, located at $u.location$, to a friend $f_i \in \mathcal{F}_u$, where $\mathcal{F}_u$ is the list of u's friends. GeoFeed employs these three approaches, monitors their cost (Section V), and then uses its decision model (Section VI), to decide on which approach should be used for which queries. The section starts by discussing the underlying data structure, then it goes on to the details of each approach. For simplicity, and without loss of generality, we use location-based point queries where the user range of interest is limited to the user location rather than a spatial area around the user location.

A. Data Structure

To support its three query evaluation approaches, GeoFeed maintains typical information for each user u that includes ID, name, and location. In addition, GeoFeed maintains the following two data structures for each user u.

List of friends. Each user u maintains a list of friends $\mathcal{F}_u$, where each friend $f_i \in \mathcal{F}_u$ is just another system user that has ID, name, location/querying spatial ranges, list of friends, and gird structure.

Grid index structure. Each user u maintains a grid index structure $\mathcal{G}_u$ that consists of $n \times n$ equal area grid cells, as it is efficient with frequent message updates and works seamlessly for both messages and friends' querying spatial range. Each

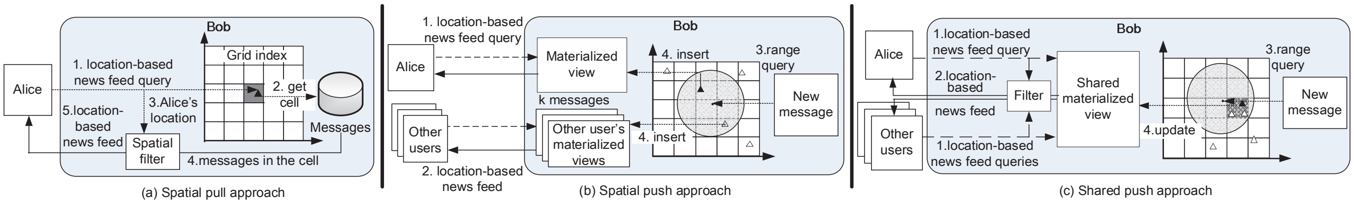

Bob
Grid index
Alice
1. location-based news feed query
2. get cell
3.Alice's location
5.location-based news feed
Spatial filter
Messages
4.messages in the cell
(a) Spatial pull approach
1. location-based news feed query
Bob
Alice
Materialized view
4. insert
3.range query
k messages
New message
Other users
Other user's materialized views
4. insert
2. location-based news feed
(b) Spatial push approach
Alice
1.location-based news feed query
Bob
2.location-based news feed
Filter
Shared materialized view
3.range query
New message
Other users
1.location-based news feed queries
4.update
(c) Shared push approach

Fig. 2. Three location-based news feed evaluation approaches in GeoFeed.

grid cell $C \in \mathcal{G}_u$ maintains: (1) The set of IDs and spatial ranges of u's friends, whose querying ranges overlap with C, and (2) The set of message IDs produced from u with spatial extents that overlap C. If a message overlaps with several grid cells, its ID will be stored in *every* grid cell it overlaps.

B. Spatial Pull Approach in GeoFeed

The *spatial pull* approach takes advantage of the grid index structure $\mathcal{G}_{f_i}$ maintained at friend f_i for early spatial pruning at the friend's side. This is the main distinguishing difference between the *spatial pull* approach employed in GeoFeed and the traditional *pull* approach employed in feeding frenzy [23]. With the early pruning pushed inside the *spatial pull* approach, GeoFeed avoids retrieving unnecessary messages as will be encountered in the traditional *pull* approach.

Figure 2a gives an example of the main idea of the *spatial pull* approach where user *Alice* needs to get her k spatially relevant messages from her friend *Bob*. The execution flow goes as follows: (1) *Alice* submits her location and the news feed query to *Bob*. (2) *Bob* exploits his grid index structure $\mathcal{G}_{Bob}$ to find out cell C that includes *Alice*'s location, and retrieves all the messages stored in C. (3) *Bob* applies a spatial filter over all the messages returned from $\mathcal{G}_{Bob}$ to only report those messages that include *Alice*'s location (depicted by a black triangle), as there could be messages in C that do not overlap *Alice* location. (4) If more than k messages are returned from the spatial filter, *Bob* forwards the k most recent ones to *Alice* as part of her news feed. *Alice* will get the rest news feed from the other friends.

C. Spatial Push Approach in GeoFeed

The *spatial push* approach in GeoFeed pre-computes and stores the answer of the location-based news feed query in a materialized view maintained by the friend f_i. Then, once the user u logs on, u only retrieves the news feed from the materialized view. Although the *spatial push* approach is very appealing to the user, it poses a large overhead over the system resources to continuously maintain the materialized view while the user is offline. Same as in the case of the *spatial pull* approach, the *spatial push* approach is distinct from the traditional *push* approach, used in the feeding frenzy system [23], in that it exploits the grid index structure $\mathcal{G}_{f_i}$ maintained at friend f_i to significantly reduce the overhead of pushing irrelevant messages to the materialized view.

Figure 2b gives an example of the main idea of the *spatial push* approach in GeoFeed where user *Alice* needs to get her k spatially relevant messages from *Bob*. The *spatial push* approach consists of two orthogonal parts, namely, *query processing* and *view maintenance*, detailed as follows: (1) *Query processing. Alice* registers her location with the grid index structure maintained by *Bob*, $\mathcal{G}_{Bob}$. In turn, *Bob* deals with *Alice* location as a continuous location-based point query [5], [15], [19] that needs to be maintained, in a materialized view, even if *Alice* goes offline. Once *Alice* logs on to GeoFeed, she just probes her materialized view, maintained by *Bob*, to retrieve her news feed. (2) *View maintenance.* The friend *Bob* uses his grid index structure, $\mathcal{G}_{Bob}$, to maintain a materialized view for each user employing the *spatial push* approach to retrieve his/her messages from him. Any new message M produced from *Bob* generates a range query over the grid index, $\mathcal{G}_{Bob}$, to retrieve the locations of all friends of *Bob* whose querying ranges overlap the spatial range of M and use the *spatial push* approach to retrieve their news feed from *Bob*. For each of these friends, M is forwarded to the designated materialized view. Figure 2b shows *Alice*'s location in $\mathcal{G}_{Bob}$ as a black triangle. The message M produced from *Bob* is depicted as a shaded circle over $\mathcal{G}_{Bob}$, which updates the materialized view of *Alice* among other views of those users who retrieve their news feed from *Bob* with *spatial push*.

D. Shared Push Approach in GeoFeed

The *shared push* approach in GeoFeed is designed to take advantage of the users locality to reduce the system overhead of the *spatial push* approach while only slightly increasing the response time of location-based news feed queries. The main idea of the *shared push* approach is to maintain one materialized view shared among different spatially co-located users within one grid cell. The benefit is that GeoFeed will maintain much less views than the *spatial push* approach. On the negative side, the messages returned by the shared view need an additional filter to split the answer among the shared views, imposing little overhead over the *spatial push* approach.

Figure 2c gives an example of how *Alice* gets her k relevant messages from her friend *Bob*, with the *shared push* approach. The scenario is very similar to the *spatial push* approach with the following two differences in query processing and view maintenance: (1) *Query processing.* As several users share the same materialized view with *Alice*, *Bob* needs to add an additional filter to filter out those news items that are not relevant to *Alice*. (2) *View maintenance.* Instead of maintaining one materialized view for each friend, *Bob* maintains one shared materialized view for all friends with the *spatial push* approach in each cell. Thus, a new message coming out from *Bob* will be inserted in much less materialized views.

The choice of which users to share views together will be discussed in Section VI as part of the GeoFeed decision model.

V. GeoFeed Cost Model

This section builds cost models for the *spatial pull*, *spatial push*, and *shared push* approaches in GeoFeed. The cost models are designed to measure: (a) The *system overhead* encountered by GeoFeed to process a location-based news feed query from a user u to a friend f_i, and (b) The *user response time* taken by GeoFeed to prepare the location-aware news feed for a user u from a friend f_i. These cost models will be used in the GeoFeed decision model (Section VI) to decide on what would be the best approach to evaluate each news feed query in order to: (a) minimize the GeoFeed system overhead, i.e., providing the ability to scale up GeoFeed to support more users, and (b) ensure that each user u will obtain the news feed within a threshold time $\mathcal{T}_u$. It is important to note that both the GeoFeed cost and decision models for any query issued by user u are designed when u is offline. Once u becomes an online user, GeoFeed sends the initial news feed to u based on the selected approach. As u remains online, new relevant messages to u are just pushed to u as in the *spatial push* approach. So, we ignore any maintenance cost for online users as it will be the same for all approaches. We will first discuss the required data structures to monitor the cost models. Then, we present the cost model.

A. Data Structure

In order to monitor the cost models, GeoFeed maintains a new data structure, termed *Stat Table*, as follows:

Stat Table. This is a global statistics table that includes the following four statistics for each user u: (1) *Total number of messages* (N_u). This is the total number of messages produced from u since joining the GeoFeed system; updated with every message posted from u. (2) *Response time requirement* ($\mathcal{T}_u$). This is the time threshold set for each user as a hard requirement for GeoFeed to produce the location-aware news feed for u within $\mathcal{T}_u$. The value of $\mathcal{T}_u$ could be set by the system as either a default value for all users, or a value that reflects how valued and appreciated is the user. (3) *Update frequency* ($\mathcal{UF}_u$). This is the average number of messages produced from u per hour; can be either computed from the time that u has joined GeoFeed, over the last hour, or as a weighted average over a certain time window. (4) *Predicated offline time* ($\mathcal{OT}_u$). This is set once u logs off the system, and it reflects the *predicted* offline time (in hours) before u logs on again to the system. $\mathcal{OT}_u$ can be either computed as an average offline time based on user history or similar to last observed offline time.

In addition, GeoFeed adds additional fields to the already maintained data structures, *list of fiends* and *grid structure*:

List of friends ($\mathcal{F}_u$). For each friend $f_i \in \mathcal{F}_u$, we maintain the total number of messages produced from u to f_i, $N_{u\to f_i}$, since u has joined GeoFeed.

Grid index structure ($\mathcal{G}_u$). With each grid cell $C \in \mathcal{G}_u$, we maintain the number of messages $N_{u\to C}$ that are produced from user u and overlap with cell C since u, has joined GeoFeed. This value is updated with each message produced from u.

B. The Spatial Pull Approach

The cost model for the *spatial pull* approach is developed for each news feed query issued from user u to her friends.

System overhead. As the *spatial pull* approach simply executes a query from u to exploit a grid index structure $\mathcal{G}_{f_i}$, we assume that the system overhead for the *spatial pull* approach is a constant cost, $Cost_{Pull}$, regardless of the user u and the friend f_i. This is mainly because every *spatial pull* query in GeoFeed will go through the same procedure, and hence they all have the same cost.

User response time. As everything in the *spatial pull* approach is done only when the user logs on to the system, the user response time will be the same as the system overhead to evaluate a location-based news feed query, which is $Cost_{Pull}$.

C. The Spatial Push Approach

Similar to the case of the *spatial pull* approach, the cost model for the *spatial push* approach is developed for each news feed query issued from user u to a friend $f_i \in \mathcal{F}_u$. Since, as we will see, this cost model depends on the number of messages produced from f_i while u is offline, the cost of the *spatial push* approach needs to be reevaluated every time u logs off the system. This is mainly because the GeoFeed decision model is all about what would be the best approach to evaluate u's query the next time u will log on to the system. Based on the decision, GeoFeed will decide whether to start maintaining a materialized view for u at f_i till u logs on the next time (i.e., use the *spatial push* approach), or simply do nothing for u till the next log on time (i.e., use the *spatial pull* approach).

System overhead. The total cost encountered by GeoFeed to support the *spatial push* approach, $Cost_{Push}$, is the summation of two parts: (1) $Cost_{Query}$: The cost to answer the user query issued from u to f_i, and (2) $Cost_{View}$: The cost of maintaining the materialized view at f_i to serve u's query, while u is offline. The details of these two costs are as follows:

1) $Cost_{Query}$. This cost only includes retrieving the news items from the maintained materialized view. We denote this part of the cost as $Cost_{RView}$, which is a constant value as it is a simple selection operation, regardless of u and f_i.
2) $Cost_{View}$. This is the cost of the background processing, done by GeoFeed, to maintain a materialized view for the messages coming from f_i and including u's location. $Cost_{View}$ can be computed as the multiplication of the following two terms: (a) the cost of inserting a message posted by f_i in the materialized view for u, which is a constant cost, $Cost_{IView}$, regardless of f_i and u, and (b) the number of such messages posted from f_i while u is offline. This can be computed as $\frac{N_{f_i\to u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$, where $\frac{N_{f_i\to u}}{N_{f_i}}$ represents the ratio of messages produced from f_i to u to the total number of messages produced from f_i, $\mathcal{UF}_{f_i}$ is the frequency of messages coming from f_i (the number of messages per hour), and $\mathcal{OT}_u$ is the predicted offline time for the user u (in hours). Notice that $\mathcal{UF}_{f_i}$, N_{f_i}, and $\mathcal{OT}_u$ are

stored in the *stat table* while $N_{f_i \rightarrow u}$ is stored in the friend list of f_i.

Then, the cost of the *spatial push* approach, $Cost_{Push}$, is:

$$Cost_{Push} = Cost_{RView} + Cost_{IView} \times \frac{N_{f_i \rightarrow u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$$

User response time. Once u logs on to GeoFeed to retrieve the news feed from f_i through the *spatial push* approach, GeoFeed simply returns the already maintained materialized view, which has the same cost as the first part of the system overhead, $Cost_{RView}$.

D. The Shared Push Approach

Unlike the cost models for the *spatial pull* and *spatial push* approaches that were developed for a particular query issued from user u to a friend f_i, and reevaluated every time u logs off GeoFeed, the cost model for the *shared push* approach is: (a) developed for a set of queries $\mathcal{Q}_\mathcal{C}$ posed from different users in the same grid cell C to the same friend f_i, and (b) as the queries that share the view are from multiple users, the *shared push* approach is reevaluated every time any user of this shared view logs off.

System overhead. Similar to the *spatial push* approach, the total cost of the *shared push* approach, $Cost_{Shared}$, is the summation of two main parts: (1) $Cost_{SQuery}$: The cost to answer the user queries using the shared view, and (2) $Cost_{SView}$: The cost of maintaining the shared materialized view at f_i. The details of these two costs are as follows:

1) $Cost_{SQuery}$. This is similar to the case of the *spatial push* approach with two differences: (1) The cost for retrieving the query answer from the shared materialized view, denoted as $Cost_{RSView}$, which is slightly higher than the *spatial push* approach, due to an additional filter over the results returned from the shared materialized view. (2) As the cost of the *shared push* approach is computed for a set of queries $\mathcal{Q}_\mathcal{C}$, the query cost needs to be multiplied by the number of queries that share the view $|\mathcal{Q}_\mathcal{C}|$. Thus, $Cost_{SQuery} = |\mathcal{Q}_\mathcal{C}| \times Cost_{RSView}$.
2) $Cost_{SView}$. In a very similar way to the case of the *spatial push* approach, this cost can be computed as: $Cost_{ISView} \times \frac{N_{f_i \rightarrow C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q}_\mathcal{C})}{Area(C)} \times \mathcal{UF}_{f_i} \times \min_{\forall u \in Q_C}(\mathcal{OT}_u)$. This includes the following three differences from the case of the *spatial push* approach: (1) We use $Cost_{ISView}$ instead of $Cost_{IView}$ to reflect the cost to update one message to the shared view. (2) We use $\frac{N_{f_i \rightarrow C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q}_\mathcal{C})}{Area(C)}$ instead of $\frac{N_{f_i \rightarrow u}}{N_{f_i}}$ as the ratio of the messages produced from f_i in cell C to the total number of massages produced from f_i, multiplied by the ratio of the covered area of the minimum bounding rectangle (MBR) of the point queries sharing the same materialized view to the area of the grid cell C. (3) We use $\min_{\forall u \in Q_C}(\mathcal{OT}_u)$ instead of $\mathcal{OT}_u$ to reflect the time when the first user of the shared view in cell C logs off the system, in which the cost model needs to be reevaluated.

Then, the cost of the *shared push* approach, $Cost_{Shared}$, is:

$$Cost_{Shared} = |\mathcal{Q}_\mathcal{C}| \times Cost_{RSView} + Cost_{ISView} \times \frac{N_{f_i \rightarrow C}}{N_{f_i}} \times \frac{MBR(\mathcal{Q}_\mathcal{C})}{Area(C)} \times \mathcal{UF}_{f_i} \times \min_{\forall u \in Q_C}(\mathcal{OT}_u)$$

User response time. Similar to the case of the *spatial push* approach, the cost here will be to only read from the shared materialized view, which is $Cost_{RSView}$.

VI. GeoFeed Decision Model

In this section, we discuss the GeoFeed decision model that is applied for each query posed by a user u and is triggered every time u logs off from GeoFeed. The goal of this decision model is to find out the best approach, among the *spatial pull*, *spatial push*, and *shared push* approaches, to evaluate each news feed query issued from user u to a friend $f_i \in \mathcal{F}_u$, for the next time that u will log on to the system, i.e., after $\mathcal{OT}_u$ time units. The objective is to minimize the system overhead encountered by GeoFeed while ensuring that user u will get all the requested news feed within the time threshold $\mathcal{T}_u$. Once u logs on to the system, GeoFeed issues $|\mathcal{F}_u|$ news feed queries as one for each friend $f_i \in \mathcal{F}_u$. Each query will be evaluated using the approach that was selected by the GeoFeed decision model. As u remains online, GeoFeed keeps pushing new messages from any of u's friends to u as news feed. Once u logs off the system, the decision model will be reevaluated again based on the new expected $\mathcal{OT}_u$.

The GeoFeed decision model consists of three main steps, detailed in the rest of this section: (1) Step 1, *Response time guarantee*, finds out the maximum number of *spatial pull* queries, $NQPull_u$, that user u can afford while having the news feed within $\mathcal{T}_u$, (2) Step 2, *Pull vs. Push selection*, decides on what are these $NQPull_u$, out of all the queries posed by u that will be assigned to the *spatial pull* approach, other queries will be assigned to the *spatial push* approach, and (3) Step 3, *Refinement with shared push*, finds out if using the *shared push* approach for any grid cell C maintained at friend f_i can reduce GeoFeed system overhead.

A. Step 1: Response Time Guarantee

Objective. Since the *spatial pull* queries are mostly favorable to the system due to their lower overhead, yet, they result in high response time, this step finds out the number $NQPull_u$ as the maximum number of *spatial pull* queries that u can afford to guarantee that the news feed will be delivered to u within time threshold $\mathcal{T}_u$.

Main idea. In terms of *user response time*, the *spatial push* approach gives much lower response time than that of the *spatial pull* approach, i.e., $Cost_{RView} \ll Cost_{Pull}$, regardless of the user u and the friend f_i. In the mean time, the user may not actually feel the difference between the *spatial push* and the *shared push* approaches as their user response times are very close to each other, though the latter is slightly higher. Thus, from the user perspective, the user would always like to avoid using the *spatial pull* approach and have all the queries evaluated with either the *spatial push* or *shared push* approaches. However, a large number of views introduces significant system overhead to maintain them. To balance between these two contradicting factors, GeoFeed uses the response time requirement $\mathcal{T}_u$ for each user u to decide that u can tolerate having some queries evaluated with the *spatial pull* approach while the overall response time for all queries is less than $\mathcal{T}_u$.

Algorithm. Since user u has to issue $|\mathcal{F}_u|$ location-based news feed queries, we want to find the maximum number of queries $NQPull_u$ among $|\mathcal{F}_u|$ that can be evaluated with the *spatial pull* approach while ensuring that u will get the required news feed within the time threshold $\mathcal{T}_u$. If $NQPull_u$ queries will be evaluated using the *spatial pull* approach with response time $Cost_{Pull}$, then the rest of queries posed by user u, i.e., $|\mathcal{F}_u| - NQPull_u$, will be evaluated through either the *spatial push* or *shared push* approach. Given that these two latter costs are very close, with $Cost_{RSView}$ is slightly higher, then the following inequality should hold for any user u:

$$NQPull_u \times Cost_{Pull} + (|\mathcal{F}_u| - NQPull_u) \times Cost_{RSView} < \mathcal{T}_u$$

Thus, we can get $NQPull_u$ using the following equation:

$$NQPull_u = \left\lfloor \frac{\mathcal{T}_u - |\mathcal{F}_u| \times Cost_{RSView}}{Cost_{Pull} - Cost_{RSView}} \right\rfloor$$

B. Step 2: Pull vs. Push Selection

Objective. Now that GeoFeed finds out that u can afford having $NQPull_u$ queries with the *spatial pull* approach, it is the objective of this step to decide which $NQPull_u$ queries out of the total of $|\mathcal{F}_u|$ queries that will be evaluated using the *spatial pull* approach. The decision is taken to minimize GeoFeed system overhead.

Main idea. The main idea of this step is to employ the following two concepts: (1) Two queries from the same user u to friends f_i and f_j may have different system overhead costs $f_i.Push$ and $f_j.Push$ for the *spatial push* approach based on the update frequency and spatial distribution of messages coming from f_i and f_j. Assuming that $f_i.Push > f_j.Push$, and that u can afford having only one query with the *spatial pull* approach, then, GeoFeed will select the *spatial push* approach for f_j, as it has lower system overhead, leaving f_i to be evaluated with *spatial pull*. Thus, if u would accept having $NQPull_u$ queries with the *spatial pull* approach, then the main idea here is to select the set of $NQPull_u$ queries that are the worst in terms of system overhead in *spatial push* to be assigned to the *spatial pull* approach. All other queries will be assigned to the *spatial push* approach. The main reason here is that from the user perspective, it does not really matter which queries will be selected as *spatial pull*, while this decision has significant impact on the system overhead. (2) In some cases, using the *spatial push* approach for a certain query may have less system overhead than using the *spatial pull* approach. Consider, for example, a user u with an expected very short offline time $\mathcal{OT}_u$. In this case it is better for GeoFeed to incrementally maintain a view of the last reported answer for u for a short time rather than reevaluating u's query with the *spatial pull* approach. This will be also favorable to the user for the much lower user response time. Thus, it may be desirable to have less than $NQPull_u$ queries using the *spatial pull* approach, if there are queries among the worst $NQPull_u$ that still have lower system overhead when using the *spatial push* approach.

Algorithm 1 Step 2: Pull vs. Push Selection

Input: (1) $NQPull_u$; the number of queries with the *spatial pull* approach that u can afford, and (2) The data structure used for computing the cost model.
Output: Set the decision $f_i.Decision$ for each query posed from u to a friend $f_i \in \mathcal{F}_u$ to either *spatial pull* or *spatial push*.

```
for each friend $f_i \in \mathcal{F}_u$ do
    $f_i.Push \leftarrow Cost_{RView} + Cost_{IView} \times \frac{N_{f_i \rightarrow u}}{N_{f_i}} \times \mathcal{UF}_{f_i} \times \mathcal{OT}_u$
    Insert $f_i$ in a Max Heap $\mathcal{MH}$ based on $f_i.Push$
end for
PullCount $\leftarrow 0$
$f_m \leftarrow$ The top element from the heap $\mathcal{MH}$
while PullCount $< NQPull_u$ AND $f_m.Push > Cost_{Pull}$ do
    Remove $f_m$ from the Max Heap $\mathcal{MH}$
    $f_m.Decision \leftarrow$ spatial pull
    $f_m \leftarrow$ The top element from the heap $\mathcal{MH}$
    PullCount $\leftarrow$ PullCount + 1
end while
for each remaining $f_h$ in the heap $\mathcal{MH}$ do
    $f_h.Decision \leftarrow$ spatial push
end for
```

Algorithm. Algorithm 1 gives the pseudo code for Step 2 in the GeoFeed decision model. The input to the algorithm is the number of queries $NQPull_u$ that u can afford having in the *spatial pull* approach, along with the data structure used to compute the cost model. The algorithm starts by calculating the system overhead cost for using the *spatial push* approach for each query posed by user u to a friend f_i. While calculating the cost, we insert u's friends in a maximum heap structure ordered by the calculated cost. Then, we keep removing friends from the maximum heap one by one, and assign them to the *spatial pull* approach, till any one of these two stopping conditions takes place: (a) the number of queries with the *spatial pull* approach has reached its maximum, which is $NQPull_u$. In this case, assigning more *spatial pull* queries will increase the user response time for u to be more than the threshold $\mathcal{T}_u$, or (b) we find a query that has lower system overhead cost with the *spatial push* approach than the cost of the *spatial pull* approach. In this case, it is better for both the system and the user to have this query, and all subsequent queries, with the *spatial push* approach. Note that all subsequent queries will have a lower *spatial push* overhead cost than the current query, and hence will also favor the *spatial push* approach. Once any of these two conditions is satisfied, we assign all the remaining queries in the maximum heap data structure to the *spatial push* approach.

C. Step 3: Refinement with Shared Push

Objective. Up to now, we have an initial decision for each query posed from u that satisfies the requirement $\mathcal{OT}_u$. In this step, we look further for all queries with the *spatial push* approach, and find out if some of them can be grouped together to use the *shared push* approach, and hence amortize the cost of maintaining a materialized view among several queries. As the response times of the *spatial push* and *shared push* approaches are similar, and we have already used the *shared push* cost in Step 1, the refinement step will never break the response time requirement for any user.

Main idea. The main idea here is to go through all the grid cells of each user in the system, and check if the *shared push* approach will be useful. Notice that u is located in $|\mathcal{F}_u|$ grid cells as one per each friend $f_i \in \mathcal{F}_u$. In any of these grid cells, using the *shared push* approach may be clearly favorable if most of the users in this cell are using the *spatial push* approach. In this case, the system overhead to maintain the shared view is amortized over these users. Unfortunately, there is no magic number for the number of *spatial push* queries in a cell that would call for going towards the *shared push* approach, as this depends on several factors that include the first offline time for any user in the cell, the frequency of users' updates, and the size of the minimum bounding rectangle of the shared view. So, a full computation of the gain/loss from using the *shared push* approach for all the users in the cell needs to be done before a decision is taken.

Algorithm. This step can be evaluated by having a loop over all the $|\mathcal{F}_u|$ grid cells that include u. For each such cell C at friend f_i, we will calculate two costs: (1) $Cost_{shared}$, as the cost of using the *shared push* approach for all the users in this cell. This is computed as described in Section V-D, and (2) $Cost_C$, as the current cost encountered for the users in C that currently use the *spatial push* approach. To compute this cost, we will need to go through all the users with the *spatial push* approaches and compute their current cost. At the end if $Cost_{shared} < Cost_C$, we set the decision to apply the *shared push* approach for all users in C with *spatial push* approach, otherwise, we do not change the decisions taken so far.

VII. GeoFeed for Mobile Users

All our previous discussions consider static registered locations for users logging on to GeoFeed. So, once a user logs off GeoFeed, we compute the cost and decision models for that user based on its registered location. User locations can be registered with GeoFeed either explicitly from the user or implicitly by detecting that a user is frequently logging on from a certain location. If a user has multiple registered static locations, e.g., home and work, GeoFeed treats each location separately, where the cost and decision models can be different for each registered location. As mentioned earlier, once a user is logged on, incoming news feed will just be pushed to the user based on the first location he/she logged on from.

In the case of mobile users, i.e., users keep moving during a login session, just pushing the incoming news feed based on the initial logging location is not applicable as users keep changing their locations. On the other hand, pulling the news feed for each new location has a prohibitive cost due to the high frequency of location changes. To this end, we modified the *spatial push* approach to support mobile users. The basic idea is to extend a user u's point location to be the whole grid cell C in which u is located in. Now, C is considered as the region of interest of user u where the news feed will be retrieved as those messages from u's friends and overlap with C. Then, a filter will be added to show only those messages that overlap with u' location. As long as u moves within its cell C, incoming messages that overlap with C are pushed to u as in the *spatial push* approach. Once u moves out of C to another grid cell C', GeoFeed employs the *spatial pull* approach to retrieves news feed overlap with C'.

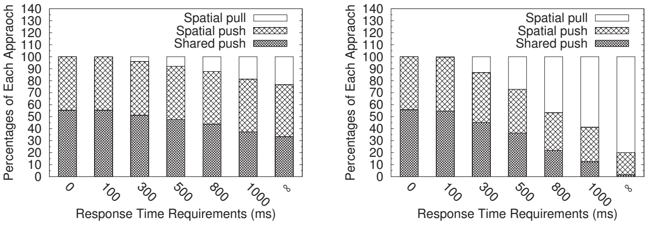
VIII. Experiments

This section gives experimental evaluation of GeoFeed based on an actual system implementation in PostgreSQL database management system [18]. We compare GeoFeed against an adaptation of the feeding frenzy approach [23], an industrial solution from Yahoo!, through an additional spatial filter as well as variations of the query evaluation approaches within GeoFeed. All experiments are based on a mixture of real and synthetic data. The real part comes from Twitter messages collected by our web crawler via Twitter Search API [28] by continuously issuing a query to the API with a spatial range of 150 × 150 miles space (approximately the size of the state of Minnesota). We query the API continuously for one week, and got 646,697 geo-tagged distinct tweets. The Twitter search API returns the location information with each tweet as either a longitude/lattitude coordinate or a semantic location, e.g., a city name, in which we use TinyGeoCoder [25] and Google GeoCoding [11] APIs to map it to longitude/latitude coordinates. Then, we randomly generate message spatial extents as circular areas centered at each message issuing location with a radius generated uniformly from 5 to 20 miles. We use a synthetic data set of 10,000 users randomly distributed over the map of Minnesota, where we randomly associate the real tweets to the synthetic users. Mostly taken from Facebook statistics [7], and unless mentioned otherwise, each user has an average of 150 friends, eight hours offline time before logging on, 500ms response time requirement, range of interest includes the user location only, and generates 90 messages per month. All average values are generated using Zipf distribution [33] (with skewness 0.5) that gives higher probabilities for smaller values. Experiments were evaluated on a server computer with Intel Core 2 Quad CPU 2.83GHz processor and 4 GB RAM with Ubuntu Linux 9.04.

To build the cost model, we ran 1,000 queries using each approach. We find that $Cost_{Pull}$=7.8ms, $Cost_{RView}$=0.5ms, $Cost_{RSView}$=0.8ms, $Cost_{IView}$=21.2ms and $Cost_{ISView}$=22.5ms. This confirms our earlier assumption that $Cost_{RView} \ll Cost_{Pull}$, and $Cost_{RView}$ is relatively close to $Cost_{RSView}$. We use these constant values in our GeoFeed decisions. The rest of this section is as follows: Section VIII-A gives an inside look on how GeoFeed takes its decisions. Comparison with other approaches is in Section VIII-B. Section VIII-C tests GeoFeed performance with respect to various parameters by simulating the workload for one day with 300 users.

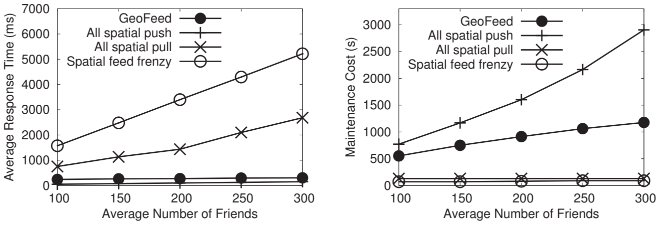
A. Inside GeoFeed

Figure 3 gives an inside view of how GeoFeed smartly takes its decisions in selecting the right query evaluation approach among *spatial pull*, *spatial push*, and *shared push* approaches



(a) Offline Time = 1 hour. (b) Offline Time = 8 hours.

Fig. 3. Inside GeoFeed Decision Model.



(a) Average Response Time. (b) Total System Overhead.

Fig. 4. Different Friends Numbers.

for each user query issued from a user u to a friend $f_i \in \mathcal{F}_u$. We vary the user response time requirements $\mathcal{T}_u$ from zero to infinity for a user offline time of one hour (Figure 3a) and eight hours (Figure 3b). For each value of $\mathcal{T}_u$, we plot the ratio of queries evaluated with *spatial pull*, *spatial push*, and *shared push* approaches.

This experiment gives the following four very interesting insights: (1) With the increase of $\mathcal{T}_u$, more *spatial pull* approaches are selected. This is mainly because GeoFeed feels that it has a relaxed $\mathcal{T}_u$, so, it goes for minimizing the system overhead by having more *spatial pull* approaches that does not cost much of system overhead. In the mean time, we have less *shared push* approaches with the increase of $\mathcal{T}_u$, which is natural as we have less *spatial push* approaches, which decreases the probability that GeoFeed finds queries that can be shared together. (2) When $\mathcal{T}_u$=0ms, (i.e., the user needs the news feed as fast as possible), no *spatial pull* approaches are applied, as they definitely pose more query response time for system users. However, it is interesting to notice that not all the queries are evaluated by the *spatial push* approach where a significant portion of the queries use the *shared push* approach. This is mainly due to our valid assumption that the query costs for both the *spatial push* and *shared push* approaches are similar. So, GeoFeed aims to reduce the system cost through having more of the *shared push* approach. (3) When $\mathcal{T}_u$=∞ (i.e., u does not have any response time requirement), GeoFeed aims to only minimize the system overhead through employing much of the *spatial pull* approach. However, it is interesting to notice that some queries are still evaluated with other approaches, especially with smaller offline time (Figure 3a). This is mainly because the GeoFeed decision model takes into account the case that the *spatial push* approach may be cheaper than the *spatial pull* approach, which takes place with low update frequency and/or short user offline time. (4) Comparing Figures 3a and 3b together shows that with a smaller offline time, more *spatial push* approaches are applied. As a smaller offline time means that the user will log soon again to GeoFeed, so, it is better to maintain the materialized view in the *spatial push* approach rather than executing the query from scratch upon the next log on time as in the *spatial pull* approach.

B. Comparison with Other Approaches

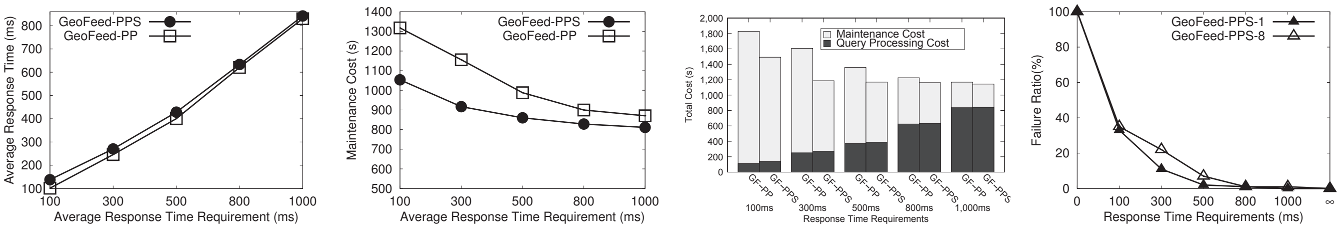
Figure 4 compares GeoFeed performance against: (a) an adaptation of *feeding frenzy* [23] to handle spatial data through an additional spatial filter, (b) having all the queries evaluated with the *spatial pull* approach, which is the state-of-the-art solution optimization in location-based systems, that pushes the spatial pruning to the bottom and (c) having all the queries evaluated with the *spatial push* approach, which is the state-of-the-art optimization in the publish-subscribe systems, that prunes out the unnecessary update notices early. We vary the average number of friends for each user from 100 to 300, and measure the average response time (Figure 4a) and system overhead cost (Figure 4b). It is clear that feeding frenzy has a very bad user response time as the spatial filter is applied afterthought, and it does not take the spatial aspects of the messages in its decisions. Though the *all spatial pull* approach gives better response time than feeding frenzy, but it is still unacceptable as it evaluates the query from scratch for each friend. The performance of GeoFeed and *all spatial push* are very similar in terms of response time, and almost have a steady performance with the increase of the number of friends, which is orders of magnitude better than that of both the feeding frenzy and *all spatial pull* approaches. With respect to system overhead, the *all spatial push* approach has a much higher cost than that of GeoFeed (about double cost for 300 friends). This is mainly due to the large number of materialized views maintained for the *all spatial push* approach.

From this experiment, we conclude that both feeding frenzy and *all spatial pull* approaches are completely impractical due to their unacceptable user response time, and hence we will not consider them later. Similarly, we will not consider the *all spatial push* approach later due to its clearly higher overhead cost than GeoFeed.

C. GeoFeed Performance

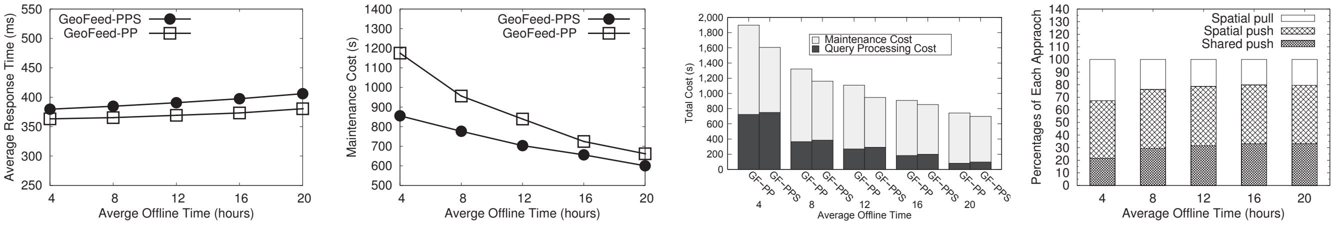
In this section, we compare the performance of two variations of GeoFeed; one with only the *spatial pull* and *spatial push* approaches (termed GF-PP or GeoFeed-PP), and the other one is our complete GeoFeed system (termed GF-PPS or GeoFeed-PPS) with different parameters: (1) user response time requirement $\mathcal{T}_u$, (2) offline time, (3) news update rate, (4) user spatial distribution, (5) grid cell granularity, (6) user querying range, and (7) GeoFeed with mobility. For a fair comparison, we use the same fixed scale in x-axis for the most experiments when comparing the average response time and total update cost between GF-PP or GeoFeed-PP.

Response Time Requirements. Figure 5 gives the impact of user response time requirements on the performance of GeoFeed-PPS and GeoFeed-PP, where the required response time varies from 100 to 1,000 ms. When the response time requirement becomes less strict, the system can use more *spatial pull* approaches to process more location-based queries.



(a) Average Response Time. (b) Maintenance Cost. (c) Total Cost. (d) Fail Ratio.

Fig. 5. Response time requirements.



(a) Average Response Time. (b) Maintenance Cost. (c) Total Cost. (d) Inside GeoFeed Decisions.

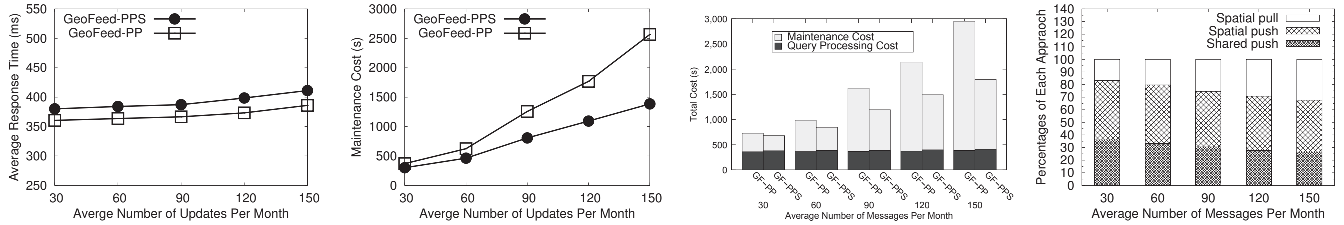
Fig. 6. User offline time periods.

Since the response time of the *spatial pull* approach is higher than that of the *spatial push* and *shared push* approaches, the response time of both GeoFeed-PPS and GeoFeed-PP gets worse with longer required response times (Figure 5a). When there are more location-based queries selected to use the *spatial pull* approach, i.e., a smaller number of queries is processed by the *spatial push* approach, the chance of grouping queries to use the *shared push* approach becomes slimmer. Thus, the improvement of GeoFeed-PPS on the view maintenance cost of materialized views reduces (Figure 5b). From Figure 5c, we can observe that with tight requirements, there is significantly large maintenance cost than query processing cost, as GeoFeed tends to avoid using *spatial pull* in order to be able to satisfy the user requirements, and thus has to pay maintenance cost for using *spatial push* and *shared push*. With relaxed requirements (increase of $\mathcal{T}_u$), GeoFeed tends to optimize its maintenance cost in favor of having more query processing cost, such that the overall total cost is decreased. Since maintenance cost optimization reduces the use of *spatial push*, it also reduces the possibility of having *shared push* refinement, and hence the performance of both GeoFeed variations becomes similar. This shows that the use of *shared push* refinement is more clear when the system is overloaded either with tight requirements, short offline times, or high update frequencies.

Figure 5d gives the ratio of user queries that exceed the user response requirements for GeoFeed-PPS with average offline time of one hour and eight hours. We vary $\mathcal{T}_u$ from zero to ∞. What is interesting here is the failure ratio is exponentially decreasing with relaxed time constraints. Also, we have less failure with smaller offline time, as more queries use the *spatial push* approach which has more deterministic user response time.

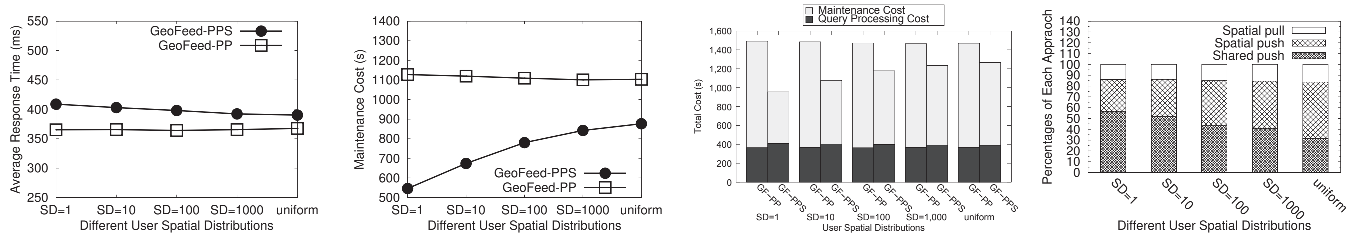
User Offline Time Periods. Figure 6 compares the performance of GeoFeed-PPS and GeoFeed-PP with respect to various user offline time periods from 4 to 20 hours. Increasing the user offline time period also increases the cost of the *spatial push* approach because the cost of maintaining materialized views during a user offline time period is higher. As a result, the system selects more queries to use the *spatial pull* approach. Since the response time of the *spatial pull* approach is higher than that of the *spatial push* approach, the response time of both GeoFeed-PPS and GeoFeed-PP gets higher when the user offline time period increases, as depicted in Figure 6a. Since the number of location-based queries using the *spatial push* approach decreases as the user offline time period gets longer, smaller numbers of queries can be grouped to use the *shared push* approach; and hence, the improvement of GeoFeed-PPS on the total view maintenance cost is smaller. From Figure 6c, we can observe that with the increase of offline time, both the query processing and maintenance cost decrease as less queries are posed to the system, and most of them are evaluated with *spatial pull*. Overall, with the increase of offline time, we have a similar performance of GeoFeed-PPS and GeoFeed-PP, because there are less *spatial push* queries, which is illustrated in Figure 6d, and hence the impact of the *shared push* refinement is reduced.

News Update Rates. Figure 7 compares the performance between GeoFeed-PPS and GeoFeed-PP by varying the average number of news update rates per month from 30 to 150. Since a higher news update rate results in more messages generated in the system, both GeoFeed-PPS and GeoFeed-PP need to use the grid index to prune more irrelevant messages; and thus, their response time increases (Figure 7a). When the news update rate increases, both algorithms have to update materialized views more frequently. Since the materialized view update cost of the *shared push* approach is shared by a group of users, it is more efficient than updating each materialized view of the *spatial push* approach individually. Therefore, GeoFeed-PPS performs much better than GeoFeed-PP in terms of the view maintenance cost of maintaining materialized views when the news update rate gets higher (Figure 7b). In Figure 7c, the average update frequency is varied from 30 to 150 per month. With the increase of the



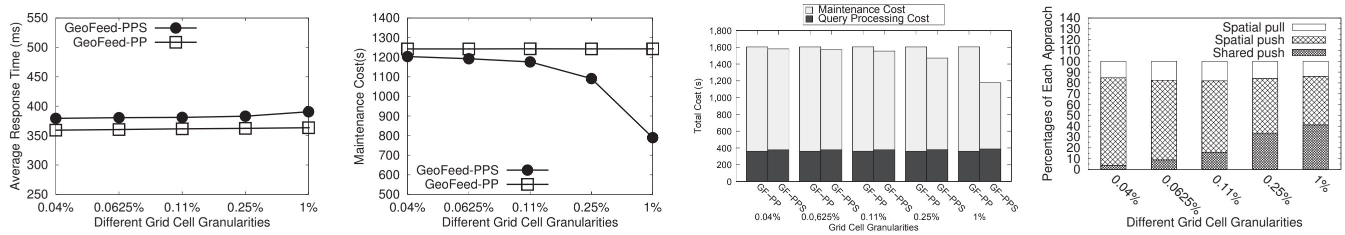
(a) Average Response Time. (b) Maintenance Cost. (c) Total Cost. (d) Inside GeoFeed Decisions.

Fig. 7. News update rates.



(a) Average Response Time. (b) Maintenance Cost. (c) Total Cost. (d) Inside GeoFeed Decisions.

Fig. 8. User distributions.



(a) Average Response Time. (b) Maintenance Cost. (c) Total Cost. (d) Inside GeoFeed Decisions.

Fig. 9. Grid cell granularities.

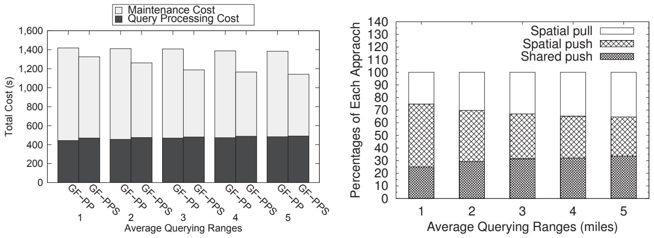
update frequency, the maintenance cost increases significantly for both approaches due to the need of more updates to the materialized views. However, GeoFeed-PPS has less total cost, as the *shared push* approach reduces the number of insertions to the materialized views. It shows the efficiency gained from sharing multiple views together to significantly reduce the maintenance cost. With respect to query processing cost, there is a slight increase for both approaches with the growth of the update frequency, which is illustrated by the decisions inside GeoFeed in Figure 7d. Overall, GeoFeed-PPS gives much lower total cost than GeoFeed-PP.

User Spatial Distributions. Figure 8 gives the experimental results of GeoFeed-PPS and GeoFeed-PP with respect to two different user distributions: (1) Uniform distribution where users are randomly distributed in the map and (2) Gaussian distribution where users are distributed in the map according to different standard deviations SD that indicate different user densities in a certain range distance of a given location. The Gaussian distribution is mainly used to simulate a hot spot area in the map, i.e., the downtown area in the city, to further explore the impact of user spatial locality and the advantage of applying the *shared push* approach. A smaller value of SD indicates that users are concentrated in a denser area. In this experiment, SD varies from 1 to 1,000.

Since the user density does not affect the selection between the *spatial pull* and *spatial push* approaches, varying the user density only slightly affects the performance of GeoFeed-PP (Figures 8a to 8c). However, the response time of GeoFeed-PPS increases as the users are distributed more densely. This is because the system with a higher user density has a higher probability to use the *shared push* approach to process user queries, as given in Figure 8d, which increases the query response time. On the positive side, using the *shared push* approach to process more user queries leads to a lower update cost of maintaining shared materialized views (Figure 8b). As a result, the view maintenance cost of GeoFeed-PPS performs much better than that of GeoFeed-PP when the user distribution becomes denser.

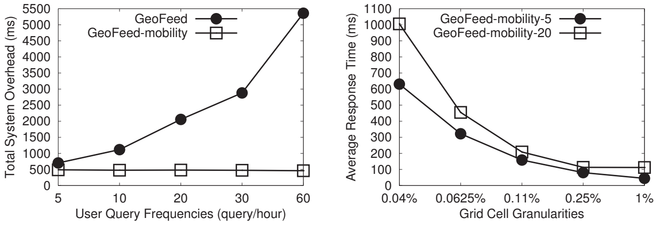
Grid Cell Granularities. Figure 9 depicts the performance of GeoFeed-PPS and GeoFeed-PP with respect to various cell granularities of the grid index which is defined as a ratio of the cell area to the total system area. This experiment considers five different grid cell granularities 0.04%, 0.0625%, 0.11%, 0.25% and 1%. The results give that the grid cell size only slightly affects the performance of GeoFeed-PP (Figures 9a to 9c). On the other hand, when the grid cell size gets larger, there is a higher chance for GeoFeed-PPS to use the *shared push* approach to share a materialized view with a group of user queries, as confirmed by Figure 9d. Due to the fact that the response time of the *shared push* approach is slightly higher than the *spatial push* approach, the response time of GeoFeed-PPS increases as the grid cell size gets larger.

User Querying Ranges. Figure 10a depicts the performance of GeoFeed-PPS and GeoFeed-PP with different user querying ranges with respect to varying from 1 mile to 5 miles. Figure 10b demonstrates the decisions inside GeoFeed system.



(a) Total Cost. (b) Inside GeoFeed Decisions

Fig. 10. User Querying Ranges.



(a) Query Frequencies. (b) Grid Cell Granularities.

Fig. 11. GeoFeed with Moving Users.

With the increase of the user querying ranges, more queries are executed with *spatial pull* approach, because the larger querying range has more chance to get the message updates, which increases *spatial push* cost to maintain the materialized view. As a result, the total querying process cost in Figure 10a increases with the growth of user querying ranges. Moreover, more *shared push* approaches are adapted in the system, because the higher cost of *spatial push* makes *shared push* approach more efficient. As a result, our GeoFeed-PPS becomes more efficient with a lower overall system overhead than GeoFeed-PP with the increase of user querying ranges.

GeoFeed Mobility. Figure 11a depicts the performance of GeoFeed and GeoFeed-Mobility for mobile users with respect to varying their query frequencies from 5 to 60 queries per hour with the user's traveling speed at 5 miles per hour, and the grid cell granularity as 0.25%. The results show that GeoFeed-Mobility is much more scalable than the original GeoFeed. In fact, the system overhead of GeoFeed-Mobility is not dependent of the user query frequency because GeoFeed-Mobility only needs to push new messages to a user when the user moves across grid cells.

Figure 11b gives the results of GeoFeed-Mobility for mobile users moving at two mobility speeds (i.e., 5 and 20 miles per hour) in different grid cell granularities, increasing from 0.04% to 1%, with the default query frequency at 10 queries per hour. The results show that the response time of GeoFeed-Mobility improves as the grid cell size gets larger because the users take longer time to move outside a grid cell; thus, the users can compute their query answers locally without enlisting the server for help. Similarly, GeoFeed-Mobility with the lower mobility speed provides the better response time, because the users can stay in a grid cell for longer time.

IX. Conclusion

We have presented the GeoFeed system; a location-aware news feed system that takes into account the spatial extents of messages and user locations when deciding upon the selected news feed. GeoFeed is equipped with three different approaches, namely *spatial pull*, *spatial push* and *shared push* for delivering the news feed to its users. Based on an accurate developed cost model for each approach, GeoFeed employs a smart decision algorithm that decides about using these approaches in a way that: (a) minimizes the system overhead for delivering the location-aware news feed, and (b) guarantees a certain response time for each user to obtain her location-aware news feed. GeoFeed further extends the *spatial push* approach to support the moving users. Experimental results, based on real and synthetic data, show that GeoFeed is favorable over existing news feed systems, with a minimal system overhead.

References

[1] L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala. Bluetooth and wap push based location-aware mobile advertising system. In *MobiSys*, 2004.

[2] Y. Cai and T. Xu. Design, analysis, and implementation of a large-scale real-time location-based information sharing system. In *MobiSys*, 2008.

[3] B. Chandramouli and J. Yang. End-to-end support for joins in large-scale publish/subscribe systems. *PVLDB*, 1(1), 2008.

[4] B. Chandramouli, J. Yang, P. K. Agarwal, A. Yu, and Y. Zheng. ProSem: scalable wide-area publish/subscribe. In *SIGMOD*, 2008.

[5] R. Cheng, Y. Xia, S. Prabhakar, and R. Shah. Change Tolerant Indexing for Constantly Evolving Data. In *ICDE*, 2005.

[6] Facebook. http://www.facebook.com/.

[7] Facebook Statistics. http://www.facebook.com/press/info.php?statistics, 2010.

[8] FourSquare. http://www.foursquare.com/.

[9] D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, 2010.

[10] Google Buzz Moblie. http://www.google.com/buzz.

[11] Google GeoCoding. http://code.google.com/apis/maps/documentation/geocoding/.

[12] iGoogle. http://www.google.com/ig.

[13] A. Khoshgozaran and C. Shahabi. Private Buddy Search: Enabling Private Spatial Queries in Social Networks. In *SIN*, 2009.

[14] Loopt. http://www.loopt.com.

[15] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[16] MyYahoo! http://my.yahoo.com/.

[17] M. Park, J. Hong, and S. Cho. Location-based recommendation system using bayesian users preference model in mobile devices. *UIC*, 2007.

[18] PostgreSQL . www.postgresql.org.

[19] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Trans. Computers*, 51(10), 2002.

[20] Renren. http://www.renren.com/.

[21] L. Siksnys, J. Thomsen, S. Saltenis, and M. L. Yiu. Private and Flexible Proximity Detection In Mobile Social Networks. In *MDM*, 2010.

[22] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A Location Privacy Aware Friend Locator. In *SSTD*, 2009.

[23] A. Silberstein, J. Terrace, B. F. Cooper, and R. Ramakrishnan. Feeding Frenzy: Selectively Materializing User's Event Feed. In *SIGMOD*, 2010.

[24] Sina Weibo. http://www.weibo.com/.

[25] TinyGeoCoder. http://tinygeocoder.com/.

[26] Twinkle. http://tapulous.com/twinkle/.

[27] Twitter. http://www.twitter.com/.

[28] Twitter Search API. http://search.twitter.com.

[29] M. Ye, P. Yin, and W. Lee. Location recommendation for location-based social networks. In *ACM SIGSPATAIL GIS*, 2010.

[30] V. Zheng, Y. Zheng, X. Xie, and Q. Yang. Collaborative Location and Activity Recommendations with GPS History Data. In *WWW*, 2010.

[31] Y. Zheng, Y. Chen, X. Xie, and W.-Y. Ma. GeoLife2.0: A Location-Based Social Networking Service. In *MDM*, 2009.

[32] Y. Zhou, A. Salehi, and K. Aberer. Scalable delivery of stream query results. *PVLDB*, 2(1), 2009.

[33] G. K. Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, 1949.