WhereStore: Location-based Data Storage for Mobile Devices Interacting with the Cloud

Patrick Stuedi
Microsoft Research
Mountain View, CA 94043
pstuedi@microsoft.com

Iqbal Mohomed
Microsoft Research
Mountain View, CA 94043
iqbal@microsoft.com

Doug Terry
Microsoft Research
Mountain View, CA 94043
doug.terry@microsoft.com

ABSTRACT

In recent years, two major trends have changed the way mobile phones are used: smartphones have become a platform for applications, and 3G connectivity has turned them into ubiquitous Internet clients. Increasingly, applications on smartphones (such as document sharing, media players and map browsers) interact with the cloud as a backend for data storage and computation. We observe that, for many mobile applications, the specific data that is accessed depends on the current location of the user. For example, a restaurant recommendation application is often used to get information about nearby restaurants. In this paper, we present *WhereStore*, a location-based data store for smartphones interacting with the cloud. It uses filtered replication along with each device's location history to distribute items between smartphones and the cloud. We discuss the challenges of designing such a system, relevant applications, and a specific design and prototype implementation.

1. INTRODUCTION

During recent years, smartphones have evolved into a powerful platform for diverse applications. As of the time of writing, Apple's AppStore for the iPhone and iPod Touch boasts more than one hundred fifty thousand applications and over three billion downloads, while having been operational for less than two years. These staggering figures suggest that the smartphone has arrived as an application platform. The transformation of phones into ubiquitous Internet clients has been made possible due to improvements on the hardware and the user interface of phones as well as due to 3G network availability.

Today, many mobile applications are architected as client/server applications, and make use of the 3G connection to store data and perform computation in the cloud. Examples of such applications include document sharing, media players and map browsers. In this paper we particularly focus on applications sharing data with and through the cloud. Applications may store data in the cloud for multiple reasons. First, the storage space on a phone is limited (especially for rich media types such as high definition video), so applications may use the cloud for infinitely scalable permanent storage. Second, devices use the cloud as a rendezvous point to share data with other devices. Third, some applications (for instance, ones that provide voice recognition) offload heavy computation to the cloud, which requires the data to be close to the computation [10].

What makes the phone a different platform than the desktop is that its applications often are used in a location specific way [20]. For instance, consider an application allowing a user to find out about the ingredients needed for a given recipe. Such an application might be useful when in a grocery shop but not necessarily when at work. The schedule for a user's work day might be good to have available on the phone while on the way to work, and less important when driving home in the evening. Or think of an application that allows a user to listen to traffic news; The user may like to hear about the traffic news of a certain region before he actually gets there.

In this paper, we present *WhereStore*, a location-based data store for Smartphones interacting with the cloud. The key property of *WhereStore* is that it uses the phone's location history to determine what data to replicate locally. Our key insight is that a person's past is a good indication of his future locations and hence his future information needs. The main goal of caching cloud data on the phone is to decrease the overall data access latency and also reduce the probability of data becoming unavailable in periods of no connectivity. Furthermore, *WhereStore* is a shared resource for different applications and exchanges data with the cloud in batches, thus potentially reducing the overall energy consumption on the phone.

Replicating data accross multiple sites has been studied extensively for distributed systems. Some of this work applies well to mobile phones. For instance, *filtered replication systems* allow applications to select only a subset of items to be stored locally. In our system, we make use of filtered replication to replicate data between the phone and the cloud. The basic idea is to let a filter express the set of data items that are likely to be accessed in the near future, based on the user's current and predicted location.

The rest of this paper is structured as follows: In Section 2, we describe a set of applications that could benefit from a location-based data store. In Section 3 we show some preliminary results indicating the benefits of a caching system like *WhereStore*. We discuss the challenges of building a location-based caching system in Section 4. In Section 5

MCS'10, June 15, 2010, San Francisco, USA.

we give a brief overview of filtered replication and location prediction. The design of *WhereStore* is presented in Section 6. Section 7 talks about the implementation, Section 8 describes how applications interface with *WhereStore*, and a summary of related work is given in Section 9. The paper concludes with Section 10.

2. APPLICATIONS

In this section we describe existing applications that would benefit from our application framework. A common element of each of these applications is that different locations require varying data items to be accessed. By caching location-specific data, *WhereStore* allows applications to have improved latency characteristics (as some of the required content for a location is already available locally) and better availability (if the network connectivity is unavailable at a location).

Web Applications: At present, some popular web applications provide reviews of restaurants and local businesses (e.g. Yelp.com [5]), and classfied services (e.g. Craigslist [1]). However, these web applications require connectivity to the network in order to function. An offline client for a restaurant review service or a classified service can access reviews/advertisements that were previously downloaded based on the user's current and predicted future location.

Media Player: While mobile media players (Zune, iPod, etc.) have storage capacities that are relatively large, the storage requirements of large collections of rich media such as high-definition video far exceeds a device's resources. We observe that different sets of media files may be appropriate based on location. For example, while a person is walking or running outdoors, it is likely that he is not interested in video files. On the other hand, a person may deem web pages, video files and pod casts to be especially useful when he is commuting on the subway. The exact nature of what files are required is likely to vary across users. However, many users will benefit from a media player that provides ready access to a large variety of location-relevant data.

Live Traffic and Sensing Applications: Various applications utilize the latest information pertaining to traffic conditions and other environmental factors [16] (such as pollution levels and pollen count). The data presentation may be simple, such as rendering the information on a map or streaming relevant traffic reports via Internet radio protocols, or complex, such as deciding how to route a user based on the latest conditions. In all of these applications, different geographic locations have associated data values. If a mobile device caches the data values corresponding to the locations that a user is most likely to visit in the future, an application could function under disconnection or provide lower latency access to data that has been stored locally.

3. PRELIMINARY RESULTS

In this section, we provide some preliminary results indicating the benefits of *WhereStore*. First, we consider the amount of geo-tagged content in regions of various sizes in three cities: Manhattan, San Francisco and Palo Alto. Two of these are dense population centers while the third is a small tech-savy city. We first consider the two dense cities. Figure 1 and 2 show the number of items available on two popular content sites: Flickr and YouTube. Note that the vertical axis on the graphs is log-scale. While the results shown in these figures are for a specific query ("travel"), the overall trend is clear. For the dense population centers, we see that the absolute number of results returned for large geographic regions is far greater than what can be reasonably cached on a mobile device. For instance, in Manhattan, the largest region we considered (a circle of radius 3.2km at the center of the city) yielded 6127 photographs and 516 videos matching our query. On the other hand, for smaller regions, the amount of data returned from queries tagged with geographic locations is much smaller, and can conceivably be cached on a smartphone (57 photographs on Flickr and 49 videos on YouTube in the case of Manhattan). Our results indicate that while it is difficult for an application to cache query results for a large geographic region, an application might feasibly cache only those results for places a user is likely to visit.

In contrast to our observations for dense cities, figure 1 also shows that the number of results returned for Palo Alto (a relatively sparse city) is not too large. However, it should be noted that these are the results for just a single query.

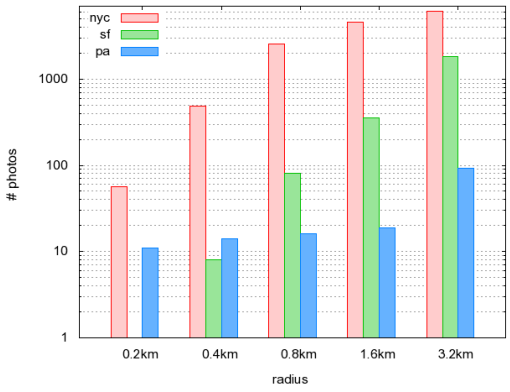

Figure 1: Amount of geo-tagged content at different locations for Flickr

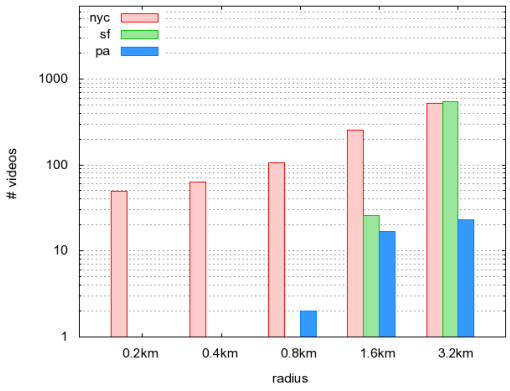

Figure 2: Amount of geo-tagged content at different locations for YouTube

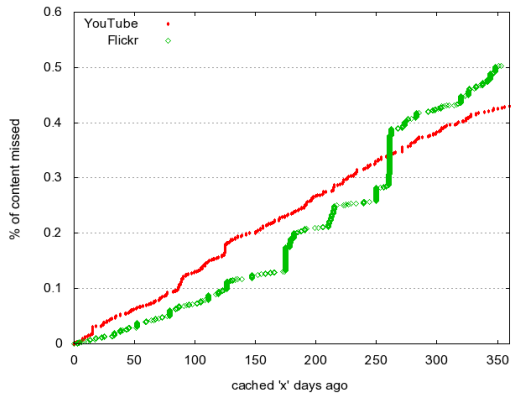

Figure 3: Percentage of geo-tagged content missed depending on the date the content is cached (YouTube and Flickr)

If an application composes the results of several queries, it would likely face a situation where all the results cannot be cached for even sparsely populated areas.

Apart from the amount of data resulting from geo-queries, it is important to consider the amount of data that will be missed by utilizing results cached in the past. Figure 3 shows the percentage of results missing from a cache constructed some number of days in the past. The query term is "travel" and the geographic region used is a circle of radius 300 meters at the center of Manhattan. We find that for a cache constructed 1 week in the past, we miss 0.3 and 0.8 percent of content for Flickr and YouTube, respectively. For a cache constructed 1 month in the past, we miss 1 and 4 percent of results for Flickr and YouTube. This suggests that results cached during an infrequent synchronization operation (syncing to a PC once in a while), can remain useful for a significant time duration. There are two caveats to this result. First, if a user's movement patterns change (e.g. the user goes to a different gym or changes her dry-cleaners), *WhereStore* may not have cached results for the new location. Until the next sync operation is performed, any data accessed in this region will have to be fetched on demand. The second caveat is that the search query may be related to a current event (e.g. Halloween). If so, the percentage of results missed will be much greater if the last sync operation occured prior to the event.

4. OPPORTUNITIES AND CHALLENGES

The high storage capacity on modern mobile devices presents a significant opportunity for caching and prefetching content. At the same time, a number of challenges arise when building a system like *WhereStore*. In this section, we discuss both aspects.

4.1 No such thing as a free meal?

Increasingly, smartphones are coming equipped with high capacity storage (8-64GB). While a significant fraction of this capacity is used for storing music, ebooks, videos and applications, it is interesting to consider the opportunity of utilizing unused storage space to cache and prefetch data. Further, mobile devices are regularly docked with a desktop computer in order to synchronize files and recharge batteries. When the docking period is sufficiently long (e.g. overnight), it is possible to prefetch and cache files for subsequent use. While this "opportunity" is not completely free (wear on hardware, power consumption, usage of the desktop's network resources, etc.), it is relatively inexpensive when compared to the time it takes to download large files on the go via 3G networks.

4.2 Challenges

First of all, as with any system that makes decisions on behalf of a user, there is a trade-off between automating decision making and giving control to the user. Ideally, the system should make autonomic decisions about what data to replicate at a given time/location. In practice, such decisions often require involvement by the application. Finding the right balance between those two principles is one challenge any system like *WhereStore* must face. Assuming well-defined application controlled boundaries within which data placement can happen, the question remains how to actually determine what data will be desired in the near future. The challenge here is to accurately predict future locations of users as well as the data which is likely to be used at those locations.

Predicting an application's data pattern is one problem, deciding the exact time at which data should be fetched from the cloud is yet another. Consider a smartphone user on his way back home from work. Even if the system correctly predicts the smartphone's next location to be the user's home, it may not be wise to fetch home relevant data while the user is in transit. The reason is that, while at home, access to data may be much faster and the drain on the phone's battery much lower than fetching data over a 3G connection. In general, a system like *WhereStore* can use information about current and future Internet access opportunities as well as information about different access technologies (e.g. 3G or WiFi) to make decisions about when to communicate with a remote cloud storage service. An intelligent decision may reduce the total power consumption of the device without data access penalties [15].

While we do not provide comprehensive solutions to each of these challenges in this paper, we establish a framework for future work in this area. In particular, we believe that the design of *WhereStore* represents a promising approach and is flexibile enough to incorporate different solutions to these problems.

5. BACKGROUND

In this section we give a short overview of filtered replication systems and location services, two technologies we use when designing *WhereStore*.

5.1 Filtered Replication

Replication systems are commonly used to keep data synchronized between a set of peers. The fundamental datastructure of any replication system is a collection. A collection consists of a set items. Each item contains user data (e.g., a picture) as well as some metadata (e.g., the size of the picture, type of picture, etc.). Replicas are local copies of either the entire collection or a subset of it. In a filtered replication system, a filter is used to describe which subset of a collection is stored in a given replica. A filter is a predicate over the metadata of the items in the collection. For instance, a filter can select all JPEG pictures, or all pictures

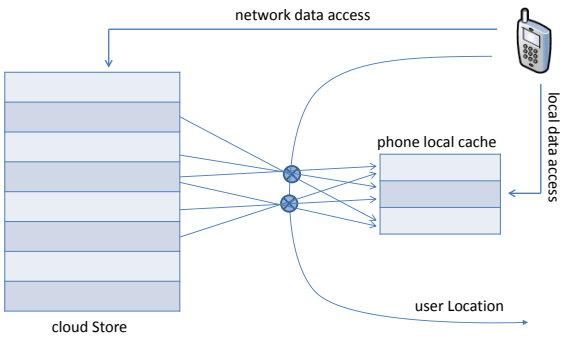

Figure 4: *WhereStore* conceptual view

whose geo-tag is inside a certain geographical region.

Replication systems typically implement efficient data structures and algorithms to synchronize one replica with another. In a filtered replication system, the goal of the synchronization process is to make sure that a) each replica stores exactly the items matching its filter and b) the version of the items stored on each replica are the same.

5.2 Location Prediction

The proliferation of GPS chipsets in mobile devices as well as the availability of high quality beacon databases for WiFi/cellular-based localization has resulted in a plethora of location-based services for mobile devices. While the majority of location-based applications take advantage of the current location of the user, some recent research efforts have considered long-term continuous location monitoring for mobile devices [9, 16]. With the availability of location histories for users, the ability to predict the future location of users becomes feasible [13]. The key idea in location prediction is to utilize the past location history of a user (along with additional features such as the user's current location or day of the week) to predict where they will be located at some future point in time. In this paper, we utilize a basic location prediction mechanism. From a user's location history, we mine a set of geographic regions or "places" where the user spends at least a given duration of time. In addition, we mine the transition probabilities between places using past history. It should be noted that factors such as time of day and day of the week affect the transition probabilities. Our initial prototype differentiates between morning and afternoon trips, and weekday versus weekend trips.

6. SYSTEM MODEL

WhereStore implements the idea of a data store for smartphones where the data placement is dynamic and based on the past and current location of the user (see Figure 4). In *WhereStore*, we give the application control over where the data is going to be replicated and when it is replicated. This is controlled using groups and regions. Data that is likely to be accessed within a common time frame can be grouped. And geographical regions where a user is likely to stay can be associated with a set of groups. In an ideal setting, the data placement strategy of *WhereStore* would then ensure that the data stored locally on a user's phone matches the groups associated with the geographical region(s) in which the user is currently located. By doing so, *WhereStore* gives control to the application to the define what data it wants available at a given location. At the same time, the data placement mechanism is fully transparent to the application.

6.1 Overview

WhereStore is layered on top of a filtered replication system and a location service. The replication layer maintains a collection per application and creates replicas both on phones and in the cloud. The data managed by *WhereStore* is stored directly within the replicas. Associated with each replica is a filter and a storage capacity. The filter linked to the replica on a phone is constantly adjusted to match items that are likely to be accessed at the current location. The filter associated with the replica in the cloud matches all items that are marked to be stored in the cloud. The storage capacity defines the maximum number of bytes each replica is allowed to store.

The semantics of *WhereStore* are similar to a cache. Access to a data item will be resolved locally if possible, causing the cloud only to be contacted if the item is not contained in the local storage. In the latter case, a data access is considered to have failed if the item cannot be found on the cloud site or if there is no network connectivity.

6.2 Data types

WhereStore operates on data items, groups and regions. We discuss the implementation of those concepts in section 7. An item is a piece of data identified by a key and an associated priority. Groups are arbitrary, possibly overlapping, sets of items. The priority of items imposes an ordering over items of the same group. A region defines a geographical area. *WhereStore* provides interfaces for applications to create and maintain groups and regions, as well as to associate certain regions with certain groups. A region may be associated with multiple groups. An application may create regions referring to common geographical areas, create groups for items with similar properties, and associate the properties of the groups with certain regions. Examples of groups are 'movies', 'work related documents', 'text files', and 'reviews'. Examples of regions are '@home', '@work', 'shopping', and so on.

6.3 Location-based Filters

Filters used by the replication system express the preferred set of items to be stored at a given location. Instead of maintaining one single filter for an entire replica, *WhereStore* creates and updates a set of filters, one for each possible future location, including the current location. *WhereStore* makes use of the location prediction service to determine the set of future locations. We give details on how this set is computed in section 7. For now, assume the set of future locations to be given by $(l_1, l_2, l_3, \ldots l_n)$ together with a probability p_i for each location l_i. The probability is a measure of certainty for the prediction. *WhereStore* creates a filter f_i for each location l_i, considering all regions that include l_i. More precisely, the filter corresponds to the disjunctive ('or') concatenation of all groups associated with any of those regions.

In *WhereStore*, everytime the smartphone changes its location and hits the boundaries of one of the configured regions, the set of filters f_i is newly computed and passed to

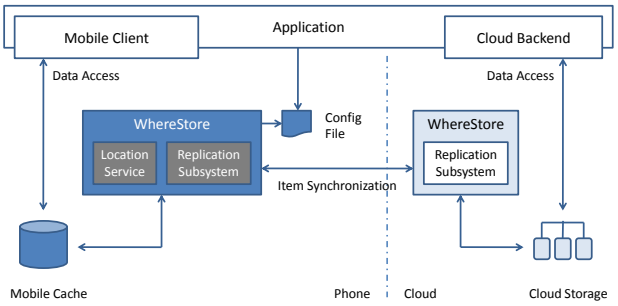

Figure 5: Architecture of *WhereStore*

the replication system together with the probabilities p_i and the maximum storage capacity specified for the replica.

6.4 Synchronizing with the cloud

The underlying replication platform exchanges items through synchronization between the smartphone and the cloud. After synchronization, the smartphone's replica should contain exactly those items that match its filter. This is done by evaluating the smartphone's filter at the cloud and only sending items to the smartphone that match the filter. Considering that the smartphone has limited space, not all matching items might fit on the phone. The problem boils down to choosing a subset of items for which the total capacity is below the specified storage capacity.

In *WhereStore*, this is done by ordering the items before they are sent to the smartphone. The cloud, for each filter f_i, computes the set of items matching the filter. For each matching item, the cloud computes a rank $c_j = p_i \times k_j$, where k_j is the priority of the item and p_i is the probability of the filter. If an item matches more than one filter its rank is the maximum of all the ranks computed. The maximum storage capacity is then enforced by ordering items according to their rank, and sending only the top n items whose total storage size remains below the specified capacity. Note that items which cannot be replicated on the phone will be fetched from the cloud upon request.

7. IMPLEMENTATION

7.1 Overview

In this section we give an overview of the system architecture of *WhereStore*. *WhereStore* is a client/server system with the client running on the mobile phone and the server running in the cloud. The *WhereStore* client on the phone is composed out of two basic building blocks, a *location service* and a *replication subsystem*. The location service provides information about possible future locations of the mobile phone. The replication subsystem keeps a local cache of data items synchronized with the cloud. The server part of *WhereStore* is almost identical to the client part, except that it does not contain a location service.

Applications interact with *WhereStore* by specifying a configuration file. The configuration file defines which items should be replicated locally at a certain location. We give a detailed example of such a configuration file in section 8. Based on the input of the configuration file and the location prediction provided by the location service, *WhereStore* updates the filter used by the replication subsystem. The replication subsystem then tries to maintain a synchronized copy of data items that match the current filter criteria.

7.2 Data access on the mobile phone

WhereStore itself does not store any the cached data. Instead, *WhereStore* operates in concert with existing data sources. Existing applications may already have their own way to store and cache data and *WhereStore* leverages this. For instance, an email client on the phone may store emails in the filesystem. It is also not unusual to use a database to store content on a phone. For instance a photo application may store photos in a database. Recently, the upcoming HTML5 [6] standard has proposed *web storage* [8], a way for browser applications to store application data in a database [7]. *WhereStore* can easily interoperate with this form of storage as well.

Most replication systems are capable of synchronizing external data sources. *WhereStore* uses Cimbiosys as the replication subsystem [17]. In Cimbiosys, data is accessed through callbacks that can be implemented specifically for the type of storage needed. Cimbiosys maintains its own metadata store containing a metadata item for each data item of the application. It uses this information to determine which items to transmit during a synchronization process. *WhereStore* creates Cimbiosys' metadata everytime a new data item is added to the cache by the application.

7.3 Cache Synchronization

The Cimbiosys synchronization protocol is based on a one-way, pull-style message exchange. A replica, called the target replica, initiates synchronization with another replica, called the source replica, by sending a request message. This message includes information about the items the target already knows about – the so called *knowledge* – and its filter. The source replica then checks its item store for any items that are not known to the target replica and whose contents match the target's filter. These items are then returned to the target.

WhereStore uses multiple filters per replica. Moreover, the replication platform is required to handle storage limitations. For instance, not all matching items might fit on the phone. Therefore, we have extended the current Cimbiosys model in two ways. First, we modified the sync request sent from the smartphone to the cloud to include the smartphone's knowledge, its storage capacity, the set of filters and the filter probabilities as described in section 6.3. Second, we modified the filter matching procedure of Cimbiosys to consider the ordering of the items discussed in section 6.3. The Cimbiosys platforms allows application-specific hooks to be inserted into the synchronization procedure. Both modifying the sync request as well as changing the matching procedure could be implemented using those hooks; thus, no modifications to the core Cimbiosys library were required.

7.4 Implementing Location Prediction

WhereStore utilizes a user's past location history to predict where he or she will be located at some time in the future. To easily maintain location history for users, we utilize the StarTrack location framework [9]. A user's personal mobile device runs a local service called the StarTrack client, which periodically captures the user's current location (e.g. using GPS) and relays it to the StarTrack server that runs as a service in the cloud. The StarTrack server processes the

```
<whereconfig>
  <region>
    <name>Bryant Park</name>
    <coordinates>40.754149N 73.984851W</coordinates>
    <radius>200m</radius>
  </region>
  <region>
          ...
  </region>
  ...
  <group>
    <name>Restaurants</name>
    <includes>Restaurants, Bars</includes>
  </group>
  <group>
    <name>Italian</name>
    <includes>Pizza,Pasta</includes>
  </group>
  ...
  <rule>
    <name>Rule1</rule>
    <region>Bryant Park</region>
    <intersection>
      <group>Restaurants</group>
      <group>Italian</group>
    </intersection>
  </rule>
</whereconfig>
```

Figure 6: *WhereStore* configuration file for a restaurant recommendation application

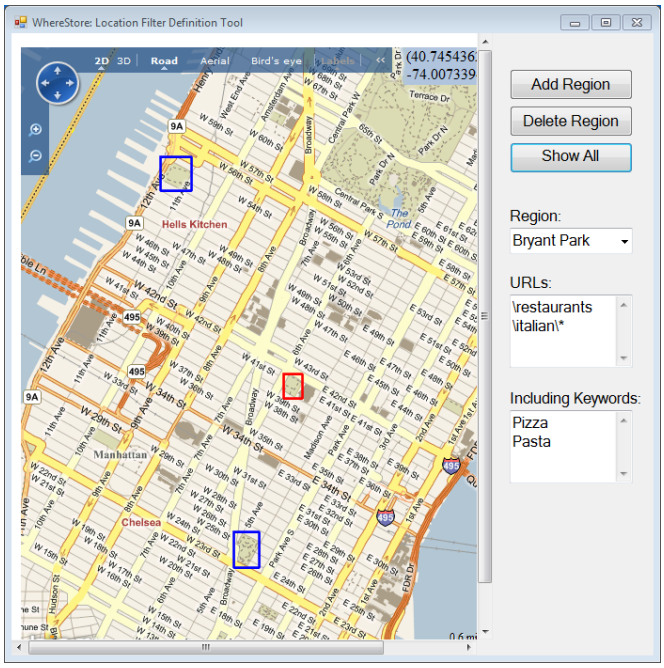

Figure 7: Screenshot of location filter definition tool

location data into tracks, which are discrete representations of trips taken by a user, and provides a high-level API to perform operations on tracks, such as retrieval.

To make predictions of where a user will be located at some point in the future, we utilize a data structure called a *Place Transition Graph*. This graph is created from the tracks that are retrieved from the StarTrack service and is one particular way of doing location prediction (among others[13]). At the outset, we need to have a set of places that are frequently visited by the user; this is a list of latitude/-longitude pairs that we call *FrequentPlaces*. We compute this set of frequent places by interating over all the retrived tracks and considering their endpoints. If the end point of a track is close to one that is already in the FrequentPlaces list, we skip this point. Otherwise, it is added to the list.

Once we have the FrequentPlaces list, we construct the Place Transition graph. We represent the graph as an adjacency matrix. The number of rows and columns of this two-dimensional array correspond to the number of elements in the FrequentPlaces list, and upon creation, all elements are initialized to zero. Next, we interate over every possible combination of start and end places, and set the corresponding value in the array to be the frequency of trips between the particular start and end place. To set this value, we utilize a function from StarTrack called *QueryTracksetByStartAndEndpoint* to count the number of tracks in the relevant track set that match both the start and end point. Finally, we normalize the values in each row of the array to be a probability value (i.e. a value between zero and one). This is done by simply iterating over every row, for each row tallying the sum of the trip frequencies for that row, and finally dividing each of the frequencies by the sum of the frequencies for the row.

8. BUILDING ON WHERESTORE

After having described both our system model and implementation of *WhereStore*, we want to illustrate how *WhereStore* can be used to build applications accessing smartphone/cloud data.

Consider a restaurant recomendation application. The application provides information about restaurants close to a location specified by the user. The restaurent information may include a review, pictures, the menu card and in some cases even small video clips. A possible implementation of such an application on today's 3G enabled smartphones would be to store all the restaurant information in the cloud while providing a user interface on the phone to search for restaurant recommendations close to the current location of the user. Obviously, one downside of this approach is that the data access latency is high since, for each query, the restaurant information has to be fetched from the cloud using the phone's 3G connection. In the worst case, the information might not even be available if the network connection is temporarily broken or the phone's GPS is not working, a situation which is likely to happen when inside a building.

We believe that a restaurant recommendation application like this could be built more efficiently using *WhereStore*. Information about restaurants would be stored in *WhereStore* causing the data to be distributed across the cloud and the smartphone. On the smartphone, the *WhereStore* configuration file defines which restaurant information is preferably stored on the phone at a given location. Figure 6 illustrates how such a configuration file would look. Such a configuration file can be generated by a graphical front-end like the one shown in Figure 7. A *WhereStore* configuration file is an XML file including definitions about regions, groups and rules. For instance, the config file shown in Fig-

ure 6 defines a circular region called 'Bryant Park' which is specified by geographic coordinates and a radius. The config file also specifies two groups, one including restaurants at Bryant Park and one including italian specialities such as 'Pizza' or 'Pasta'. Finally, the config file contains one rule stating that whenever the phone's location hits the 'Bryant Park' region, then all the restaurant occuring in both the 'Restaurants' and the 'Italian' group should be stored on the phone. Rules can be intersections, unions or complements of groups. One can easily imagine how *WhereStore* can be used to build other types of applications. A music player may store different music files in *WhereStore*. A configuration file would define different groups of music such as 'jazz' or 'rock' and use rules to specify what type of music should be stored on the phone when in a given region.

9. RELATED WORK

Disconnected operation is a well studied topic in mobile computing research. Coda [12], Ficus [11] and Bayou [19] are three seminal research efforts that strive to provide a mobile client with access to data when continuous connectivity between nodes is too expensive or unavailable. These systems work by replicating data across nodes. Coda and Ficus are "application-transparent" in that existing file system applications do not have to be recoded or recompiled to achieve the benefits of disconnected operation. Coda allowed users to construct "hoard profiles" to indicate the set of files that should be cached on clients in anticipation of future disconnections [18]. Other systems, like SEER [14], have provided more sophisticated hoarding policies based on file reference patterns. Bayou, on the other hand, provides an API to support fine-grain application-specific conflict detection and resolution for replicated database applications, but includes no support for partial replication. In contrast, our system provides applications with a key-value store API that allows fine-grain replication of data items. More importantly, applications can associate sets of items with locations, and the replication infrastructure automatically provides location-based hoarding of items.

Several recent industry projects have also considered disconnected operation in the context of web applications. The Gears [3] application framework provides web applications with the ability to store data locally on a web client and access to it via SQL. Recently, along the lines of HTML5 [6], *Web Storage* has been proposed [8, 7]. Web Storage is intended for web application to store data locally on end devices. HTML5 further provides facilities for web application to run in offline mode in cases no network access is available. Dojo Offline [2] is a toolkit that runs on top of Gears and gives synchronization functionality to web applications. While our system has similarities with Dojo Offline, there are some key differences. We allow applications to define the data which must be available at various locations and provide an automatic mechanism to hoard this data prior to the user reaching the location (based on past user history).

10. CONCLUSIONS

In this paper we presented *WhereStore*, a location-aware storage system for mobile devices. *WhereStore* provides mechanisms to allow application developers to specify what data would be typically accessed in various places. This information, along with the user's current and future predicted location, is used to improve data access latencies and availability. We discuss the design of *WhereStore* and provide details of a prototype implementation based on the Cimbiosys filtered replication platform and the StarTrack location framework.

Acknowledgement

We would like to thank Chandu Thekkath at Microsoft Research for helpful feedback on a draft of the paper.

11. REFERENCES

[1] Craigslist: An Internet classified service. http://www.craigslist.com.

[2] Dojo Offline. http://www.dojotoolkit.org/offline.

[3] Gears (formerly Google Gears). http://code.google.com/apis/gears/.

[4] Google Listen: Search and Listen. http://listen.googlelabs.com/.

[5] Yelp: An Internet restaurant and business review service. http://www.yelp.com.

[6] HTML5: W3C Working Draft 4 March 2010, 2010. http://www.w3.org/TR/html5/.

[7] Web SQL Database, 2010. http://www.w3.org/TR/webdatabase/.

[8] Web Storage, 2010. http://www.w3.org/TR/webstorage/.

[9] Ganesh Ananthanarayanan, Maya Haridasan, Iqbal Mohomed, Doug Terry, and Chandramohan A. Thekkath. Startrack: a framework for enabling track-based applications. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 207–220, New York, NY, USA, 2009. ACM.

[10] Byung-Gon Chun and Petros Maniatis. Augmented Smartphone Applications Through Clone Cloud Execution . In *USENIX Workshop on Hot Topics in Security (HOTSEC)*, August 2009.

[11] R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page, G. J. Popek, and D. Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71, Anaheim, CA, June 1990. USENIX.

[12] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[13] John Krumm and Eric Horvitz. Predestination: Where do you want to go today? *Computer*, 40(4):105–107, 2007.

[14] Geoffrey H. Kuenning and Gerald J. Popek. Automated hoarding for mobile computers. *SIGOPS Oper. Syst. Rev.*, 31(5):264–275, 1997.

[15] Du Li and Manish Anand. Majab: improving resource management for web-based applications on mobile devices. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 95–108, New York, NY, USA, 2009. ACM.

[16] Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, and Péter Boda. Peir, the

personal environmental impact report, as a platform for participatory sensing systems research. In *MobiSys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 55–68, New York, NY, USA, 2009. ACM.

[17] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall, and Amin Vahdat. Cimbiosys: a platform for content-based partial replication. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 261–276, Berkeley, CA, USA, 2009. USENIX Association.

[18] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. In *MLCS: Mobile & Location-Independent Computing Symposium on Mobile & Location-Independent Computing Symposium*, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.

[19] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSOP'15*, pages 172–183, Cooper Mountain, Colorado, December 1995.

[20] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Measuring serendipity: connecting people, locations and interests in a mobile 3g network. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 267–279, New York, NY, USA, 2009. ACM.