Nihao: A Predictive Smartphone Application Launcher

Chunhui Zhang, Xiang Ding, Guanling Chen, Ke Huang,
Xiaoxiao Ma, and Bo Yan

Computer Science Department, University of Massachusetts Lowell,
1 University Avenue, Lowell, Massachusetts, USA, 01854
{czhang,xding,glchen,khuang,xma,byan}@cs.uml.edu

Abstract. Increasingly large number of the applications installed on smartphones tends to harm the application lookup efficiency. In this paper, we introduce Nihao, a personalized intelligent app launcher system, which could help the users to find apps quickly. Nihao predicts which app the user will likely open next based on a Bayesian Network model leveraging the contextual information such as the time of day, the day of week, the user's location and the last used app with the hypothesis that the users' app usage pattern is context dependent. Through the field study with seven users over six weeks, we first validate the above hypothesis by comparing the prediction accuracy of Nihao with other predictors. We found that the larger UI change did not necessarily yield longer app lookup time as the app lookup time highly depended on the app icon position on screen, which suggested the prediction accuracy was the most important factor in designing such a system. At the end of the study, we conducted a user survey to evaluate Nihao qualitatively. The survey results show that five out of seven users were quite satisfied with the prediction of Nihao and thought it could help to save both app lookup and management time by ranking the app icons automatically while Nihao did not help the other two users much since they used their phones primarily for calling and texting (not for apps).

Keywords: mobile applications, Android applications, application usage prediction, context awareness, adaptive UI.

1 Introduction

As of March 2012, the number of apps and games in Android market reached about 620,000 and the total number of downloads was estimated to be over nine billion [1]. Increasingly large number of the apps installed on smartphones tends to decrease the app lookup efficiency and more time is required to manage them. A typical Android phone launcher by default places all app icons in the *app drawer* and the apps are usually ordered according to their installation time or alphabetically. Furthermore, the users could create shortcut for their favorite apps on the homescreens directly or manage the shortcuts in folders. With a large number of apps installed, the users usually launch an app either by looking

D. Uhler, K. Mehta, and J.L. Wong (Eds.): MobiCase 2012, LNICST 110, pp. 294–313, 2013.

up the shortcut screen by screen (Android phone usually has more than one homescreens) or by going through the prohibitively long app list in the *app drawer*. In addition, the task of customizing and arranging the app shortcuts consumes considerable amount of time and effort.

To shorten the app lookup and management time, we developed Nihao, a personalized intelligent app launcher system, which could automatically rank the apps by predicting how likely they tend to be opened. The prediction is based on a Bayesian Network model leveraging various contextual information such as the time of day, the day of week, the user's location and the last used app, with hypothesis that the users' app usage pattern depends on those context. For instance, someone likes to read *Technology Review* in the morning and use *Gtalk* to contact his wife at the end of the day during weekdays while he likes to play games using his phone during weekends. Another example is that someone is likely to use *Yelp* [2] to read reviews for dishes when she is in a restaurant while she tends to use *Google Map* to find interesting places in the destination when waiting for the plane in the airport. Similarly for the context of the app correlation, the users may use *eBay* app after using *Amazon* app.

When designing such a system involving adaptive user interface, whether the UI change will hurt the usability is always a concern [3]. Some researchers believe that the users need a predictable UI while others argue that the accurate prediction could mitigate the confusion caused by the UI change. In this paper, we aim to find out the feasibility and efficiency of the dynamic app launcher UI based on the predictions.

With the aforementioned expectation, hypothesis and concern in mind, we conduct a field study with seven users for six weeks. As the data analysis results show, the performance of Nihao in term of app lookup time outperformed the default launcher when considering only downloaded apps. (Built-in apps such as *phone*, *text messaging*, *browser* are usually opened through the *quick launch bar* at the bottom of the homescreen in a Android phone and require little lookup time). Then, the hypothesis was proved by demonstrating that Nihao predicted more accurately than the benchmark predictor (app usage frequency based predictor) and other predictors. Finally, we found that the larger UI change did not necessarily yield longer app lookup time and the app lookup time highly depended on the app icon position on screen, which suggested the prediction accuracy was the most important factor in designing such a system.

At the end of the study, we conduct a user survey to evaluate Nihao qualitatively. The survey result shows that five out of seven users were quite satisfied with the prediction of Nihao and thought it could help to save both app lookup and management time by ranking the app icons automatically while Nihao did not help the other two users much since they used their phones primarily for calling and texting.

The contributions of this work include: 1) a new and effective UI design for intelligent smartphone app launcher, 2) an effective app usage prediction method based on a probabilistic model leveraging contextual information such as time,

location and app correlation, 3) a reference system design and implementation of such a context-ware personal assistant to speed up app lookup.

The rest of this paper is organized as follows. In Section 2, we compile the related works. Section 3 discusses the interface design and the prediction model. Section 4 presents details of the system design and implementation. Next, we show the system evaluation results in Sections 5. Finally, Section 6 concludes with planned future work.

2 Related Work

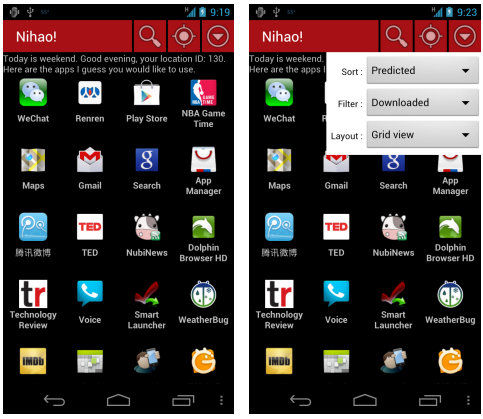
Several recent works have explored the smartphone usage and its correlation to contexts. H. Falaki et al. suggest that smart phone service should be customized due to the vast usage diversity among users [4]. M. Bohmer et al. find that app usage is correlated to contexts such as time of the day and the user's location using the dataset collected through their app recommendation tool *appazzar* in Android market [5] . T.M.T.Do et al. show two dependencies of app usage, i.e. place and social context based on data collected from Nokia phones [6]. Most recently, Ke Huang et al. proposed to use several contexts to predict app usage and found that the app correlation is the most influential factor [7]. So far, these mentioned researches are all based on the data collected from the smart phones. In contrast, Q. Xu et al. reveal day-time app usage pattern and show the app correlation through analysis based on a Tier1 ISP's dataset [8]. Instead of focusing on the context-based data analysis, our work emphases on the design of a complete system and the analysis of its effectiveness.

There is a rich body of literature regarding the adaptive user interface design. About twenty years ago, Andrew and Ben studied the *split menus* which is a menu comprises both a static and an adaptive part with a line separating them [9]. The adaptation was based on the selection frequency which could directly reflect the user's preference. Leah and Joanna showed that adaptive user interface is more beneficial in a small size screen than in a large size screen [10], which strengthens our confidence in designing Nihao. Melanie Hartmann summarized the challenges in designing adaptive user interface in his survey paper [3]. Recently, Santi. P. et al. have organized the outgoing call prediction result in form of *Intelligent Address Book* where the predicted callee IDs are ranked based on a computed score [11], while our system foresees the apps that are likely to be opened and organizes the app icons in a grid or list view.

Recently, several research groups have been focusing on the app recommendation systems for smart phone users. Bo Yan et al. developed a collaborative app recommendation system (*AppJoy*) considering the app usage history such as *frequency*, *duration* and *recency* [12]. M. Bohmer et al. present *Appazaar*, which is a context-aware recommender system [13]. In contrast to those works, our system focuses on predicting which already installed apps to be opened next.

3 Approach

If we had a good app launch prediction system, there are at least two usage scenarios to leverage this kind of app prediction. One use case is to rank the apps on launch pad based on the scores derived from the prediction model (i.e. Figure 1a). This use case falls into the category of intelligent user interface design and focuses on fast app lookup. Another use case is to pre-load apps that are likely to be launched into memory based on the prediction model [14]. These two use cases are different as the later focuses on launching the app quickly after the user finds the app, while the first one helps the user finds the app quickly. In this paper, we focus on the first use case, aiming to build an intelligent UI to assist app lookup. Thus, we propose a context-based app ranking method by leveraging the context such as *TimeOfDay*, *DayOfWeek*, *Location* and *LastUsedApp*.



(a) Screenshot1 (b) Screenshot2

Fig. 1. Nihao screenshots

3.1 Contextual Information

Based on the extensive data analysis result reported recently (see the related work), many contextual information have been found correlated to the mobile app usage in certain ways. Those contexts include time, the user's location, the correlation between apps and the social context etc. We believe that the app usage pattern is much more complex and the user's app usage behavior is influenced by much richer context. By balancing the importance of the context and the engineering effort, we chose the four aforementioned contexts.

First, *TimeOfDay* is defined as {morning(5am-11am): 0; noon(11am-2pm): 1; afternoon(2pm-6pm): 2; dinner(6pm-9pm): 3; evening(9pm-5am): 4}. Actually, we originally proposed a variable *HourOfDay* which has 24 values. But we

changed it to the less fine-grain variable *TimeOfDay* because it required shorter learning period and much less space for storing the historical data which is a concern due to the limited disk storage in a smartphone. Then, we introduced a variable *DayOfWeek*. For the same reason, we defined it as a binary variable with 0 representing the weekdays and 1 representing the weekends. Next, *Location* is represented by location ID computed through our significant place algorithm that is further explained in the next section. Finally, the correlation between apps is considered. For the sake of simplicity, an app is treated only correlated with the last used app (within a user interaction *session*, which is defined as the time between the screen is turned on and off) which follows the order-one Markov model. The value of the variable *LastUsedApp* is simply the string of the app name.

3.2 Nihao Model

Based on the variables introduced, we construct a Bayesian Network for Nihao (Figure 2) assuming that the app usage is dependent on the time of day, day of week, the user's location and which app was used before, where location is dependent on both temporal variables. Every time the user opens Nihao, each app is assigned a score calculated as the conditional probability of the app to be opened according to the following equation,

$$Score(A) = P(A|D, L, T, A') = \frac{P(A|D, L, T) * P(A|A')}{P(A)} \tag{1}$$

and then the app icons are arranged in a grid view (optionally a list view) from left to right and from top to bottom. When we actually calculate the scores, the posterior probability of an app, P(A), is assumed to be equal for all apps.

All the probabilities were computed against the past three weeks' historical data to capture the most recent app usage behavior. However, what is the optimal value for this window size is still open and we will try to make this value personalized and adaptive in the future.

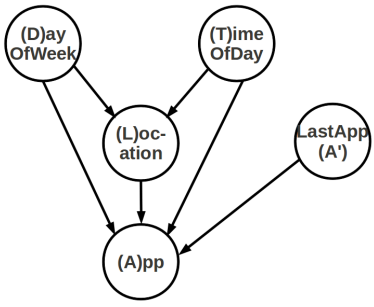


Fig. 2. Bayesian network model for Nihao

4 System Design and Implementation

Nihao follows the client-server model. The client runs on a smartphone and is responsible for providing the user interface, recording the app usage and the associated contextual data. The server runs on remote PCs for storing the historical data and conducting machine learning tasks such as learning new models. But, please note that the client could work when the server is down since all the UI required statistics are cached locally. For the communication between the client and the server, a customized protocol generated through Google ProtoBuf [15] over http is used. Details are discussed in the following subsections.

4.1 Client

The client software contains several components including the UI, a model manager, a local database and a background service (Figure 3).

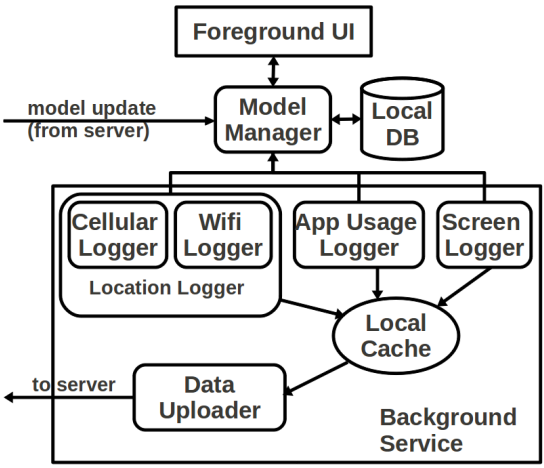


Fig. 3. Client software architecture

First, the major task of the UI is to present the app icons. Following the tradition of Android phone, Nihao uses the grid view as the default configuration and provides the list view as an option (Figure 1a). Users may like to switch between the grid view and the list view. Considering the code clarity and efficiency we put both views in the same activity (term in Android) and play with the view's visibility instead of assigning views to different activities. Furthermore, Nihao allows users to use popular static ranking methods i.e. *alphabetical* and *new app first* ranking. For both methods, installed apps will be showed thus it is particularly helpful when Nihao has not learned much at the beginning. In addition, the search function is also implemented within Nihao. However we found that it was seldom used by the subjects during the study which suggests that people prefer

to open apps by browsing instead of searching. Another function provided by Nihao UI is the place check-in. As a location-ware system, this function allows Nihao to learn place semantics opportunistically.

Second, as the name suggested, the model manager prepares the statistics required by the model in local DB and computes the score for the apps. For example, the model manager implements the significant place recognition algorithm to produce the location ID. According to the Bayesian Network model used, the statistics are organized in three local DB tables i.e. *location*, *context* and *correlation*. Each record in *location* table represents a location with a unique ID. *context* table maintains the statistics to calculate $P(A|D, L, T)$ and *correlation* table stores the data to compute $P(A|A')$.

Finally, let us discuss the background service. As the most important component of Nihao, it further comprises several objects such as the cellular logger, the WiFi logger, the app usage logger, the screen logger, the data uploader and a local cache. The cellular logger and the WiFi logger is used together to provide the input of the significant place recognition algorithm such as the cell ID, the connected WiFi AP MAC address and the list of nearby WiFi AP MAC addresses. The app usage logger is used to monitor the app usage i.e. listen on the app open event. However, Android does not directly provide event-based APIs for such a task. Fortunately, the APIs to find the top activity, the activity the user is currently interacting with, is offered. So Nihao eventually generates app open events itself by polling the top activity for every one second. The screen logger is responsible for recording the screen on/off events which indicates the start and the end of the phone usage *sessions*. Depends on whether it is the first app used in a *session*, Nihao computes the scores for apps differently. App ranking is only based on $P(A|D, L, T)$ if no app is used so far within the session while $P(A|D, L, T) * P(A|A')$ is calculated if some apps has been opened. The data uploader is responsible for uploading the data generated by the loggers to the server. The uploading process is totally independent from the UI thread since the data needed for UI are stored in local DB in real time and at the same time the data are put into the local cache. To save the battery and reduce the 3G/4G network traffic, the data are uploaded when the WiFi network is available and several *records* are batched into a single HTTP POST when uploading. In addition, considering the user privacy, all the sensitive data such as Android ID, Wifi Mac address etc, are hashed.

4.2 Server

It is valid to ask why Nihao needs the server since the model required statistics are stored locally on the client and the client could work even when the server is down. Simply put, the server is used to preserve the app usage history for users and run the complex machine learning algorithms. Even though Nihao currently uses the fixed model for all users, more suitable model may exist for different users. To find that out, based on the historical raw data on the server, we could keep experimenting with the new models. When a better model is found, Nihao could push the updated code and the statistics needed for the new model to the clients.

Nihao server is implemented based on Play Framework [16] and MySQL database. Instead of storing the data received in database directly, Nihao server stores the data in daily files first. It is more efficient to do so thus the scalability of the server is increased. In addition, for the purpose of storing the historical raw data, the daily files are easier and more convenient to store and be moved around. Then, a scheduled job is designed to read the data from the files and store them to the database every 2:00AM in the morning. It is better to run various machine learning algorithms against database since search is often involved and it could benefit from the record indexing in the database.

4.3 Significant Place

Our significant place recognition method involves two steps. First, an automatic algorithm is used to compute the place ID (or the location ID, we use them interchangeably throughout the paper). Second, instead of guessing the semantic meaning of the place, the *check-in* function is implemented as the part of the UI to allow the user to teach the system opportunistically. But be noted that Nihao model is based on the place ID directly instead of the semantic label at this time.

Recognition Algorithm. Nihao jointly uses the identifiers of cellular network and WiFi network to recognize the significant places by assuming that the most in-door significant places have WiFi network. Cellular network is mainly used to recognize outdoor places. In Nihao, a significant place is represented by a tuple: (a set of cell IDs, a set of connected WiFi AP MAC addresses, a set of nearby WiFi AP MAC addresses). The first element in the tuple is a set of cell IDs instead of a single cell ID because a significant place may reside on the boundary of the adjacent cells in the cellular network. For the similar reason, the second element is also defined to be a set. The third element could be acquired through the WiFi scan and it is naturally a set.

The recognition process follows a decision tree (Figure 4). First of all, the algorithm checks whether the WiFi network is connected. If it is connected, it then tries to match a tuple in the local database and return the place ID. Here, by match, we mean the current connected WiFi MAC address is contained in the set of connected WiFi AP MAC addresses in an existing tuple. If the WiFi network is not connected, the algorithm will check whether the WiFi scan result (a set of nearby AP MAC addresses) presents. The scan is conducted by the WiFi logger and multiple consecutive scans are issued in the same scanning session to find as much APs as possible. If the WiFi scan result is none-empty, the algorithm will again try to compare the current WiFi scan result(A) to the the third variable of the tuples in DB (B). An existing place ID will be returned and the WiFi scan result will be merged into the nearby APs set in the tuple if there are at least two overlapped WiFi APs in (A) and (B). Otherwise, a new record will be created and a new place ID will be returned. Finally, the cell ID is used to match the record in a similar way if the WiFi scan result is empty.

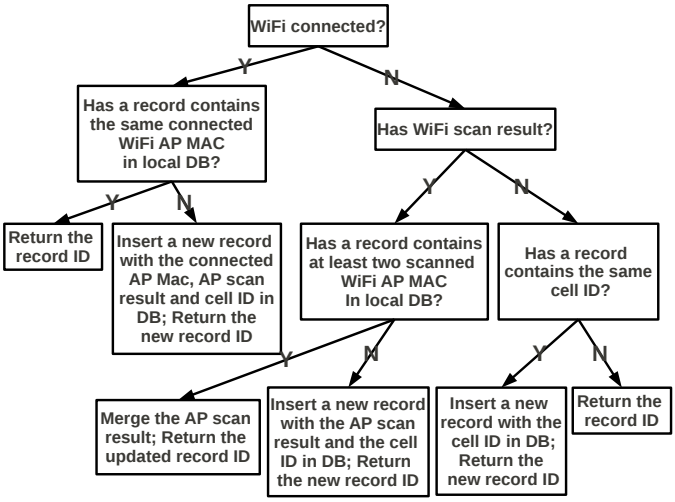


Fig. 4. Decision tree for significant place recognition

After the six weeks' filed study, the number of recognized places ranges between 4 and 60 for different users and the largest WiFi AP cluster contains 483 APs.

Place Check-In. To acquire the location semantics, we design the *place check-in* function and the places include home, friend's home, work place, school, restaurant, shopping, gym, theater, outdoor and transportation related places. When the user checks in a place, the semantic label is assigned to the current place ID computed by the recognition algorithm and the place records with the same label will be merged. Basically, the place check-in function is used to group the places by their semantic meaning because for instance Nihao should predict in the same way when a user in different restaurants no matter where they are geographically.

4.4 Data Format and Protocol

Nihao uses Google ProtoBuf to manage the data format used to store and upload the data. Listing 1 shows the .proto file which is used to define the structure of the data and could be compiled to generate the code for serializing the data.

Listing 1. Nihao .proto file (1)

```
message Record {
  enum RecordType {
        APPUSAGE = 1;
        SCREEN = 2;
        WIFI = 3;
```

```
        CELLULAR = 4;
    }
    message AppUsage {
        required string package_name = 1;
    }
    message Screen {
        required bool on_off = 1;
    }
    message WiFi {
    ...
    }
    message Cellular {
    ...
    }
    required int64 timestamp = 1;
    required RecordType type = 2;
    optional AppUsage app_usage = 3;
    optional Screen screen = 4;
    optional WiFi wifi = 5;
    optional Cellular cellular = 6;
}
```

According to the message *Record*, four RecordTypes and their structure are defined and the corespondent variables are designed to be optional. Thus, four different loggers could share the same .proto file by using only the variable interests them. *timestamp* and *type* are two required variables since for each record Nihao needs to know when it is generated and what the type is.

The Nihao client uploads the data through Http Post with the body being the data serialized by ProtoBuf. To increase the data efficiency, we define message *RecordPool* (Listing 2) to upload the data in batch mode.

Listing 2. Nihao .proto file (2)

```
message RecordPool {
    required int32 version_number = 1;
    required string android_id_hash = 2;
    required string email_hash = 3;
    repeated Record records = 5;
}
```

5 Evaluation

We conducted a field study for six weeks with seven subjects, who are college students and professionals. At the beginning of the study, we installed Nihao on the participants' Android phones. The first three weeks were the learning period for Nihao, during which we asked the subjects to keep using their default device

UI to launch apps while Nihao was running in the background, collecting app usage data and related contextual data. In the following three weeks, we asked the subjects to use Nihao to launch apps as much as possible.

5.1 App Usage

Figure 5 and Figure 6 show the app usage statistics Nihao collected and demonstrate the usage diversity of the subjects. In both figures, for each user we separately show the usage of downloaded apps and built-in apps (e.g. phone, messaging, browser etc.) since we wanted to understand how much the downloaded apps, as the primary target of Nihao, were used. As Figure 5 shows, User1 used

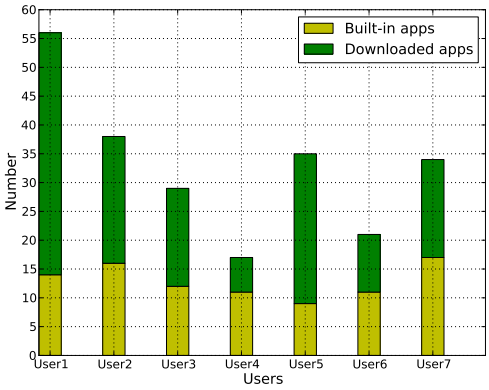


Fig. 5. Number of used apps for seven users

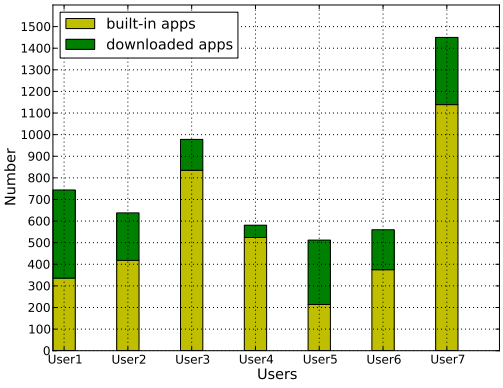


Fig. 6. App launch count for seven users

56 apps (the most) during the test period while User4 just used 17 apps (the least). Users used 33 apps on average. Considering the ratio of the downloaded apps to the built-in apps, User1 is 3.0 (highest) while User4 is 0.55 (lowest), which indicates that User1 had much broader app usage behavior than User4. In Figure 6, User7 launched apps for more than 1400 times among which about 3/4 times were for built-in apps. For User5, although the total launches were much less than that of User7, he mostly used downloaded apps.

5.2 Launched App Position

First, we evaluated the model accuracy of Nihao using the metric *launched app position*, which is the icon position in the grid view when the user clicks it. The position starts with 1 and increases from left to right and top to bottom fashion. Currently we show 4 app icons in each row of the grid view. We compared Nihao with other predictors, such as frequency based predictor, app correlation based predictor, time based predictor and location based predictor. For Nihao, *launched app position* was recorded directly through Nihao UI. For other predictors, this metric was calculated based on the dataset collected through the six weeks user study. Simulating the real usage scenario, we used first three week's data as training dataset to predict the next three weeks' app launch. In addition, we progressively added the previous app launch records to the training dataset when predicting the next one.

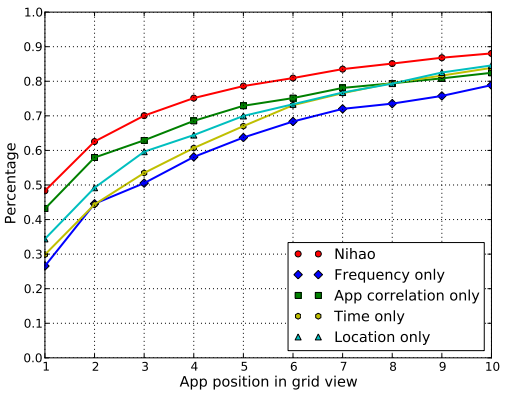


Fig. 7. CDF of launched app position (seven users combined)

As Figure 7 shows, 76% of the app launches were within first four positions (first row in the grid view) for Nihao while the number is only 58% for the frequency based predictor. Other predictors' performance were in between.

This result validates our hypothesis that app usage is correlated with the contextual information such as time, location and last used apps and a context-based predictor can be effective. However, Nihao model is not as accurate across all users. For User1, Nihao is the most accurate predictor (Figure 8), but for User3, Nihao does not obviously predict better than other predictors (Figure 9). The reason could be found by analyzing the most used apps of individual users. Take User3 for example, top five most used apps took up 81% of the total app launches (Figure 11) while the number was just 50% for User1 (Figure 10). In other words, User3 used top five apps for the most of times, thus the frequency-based predictor worked fairly good and that user can hardly take advantage of Nihao prediction. On the other hand, User1 had much broader app usage pattern and he benefited a lot from Nihao model. As a future work, we plan to develop a customized prediction model that uses different predictors depending on the user's app usage patterns.

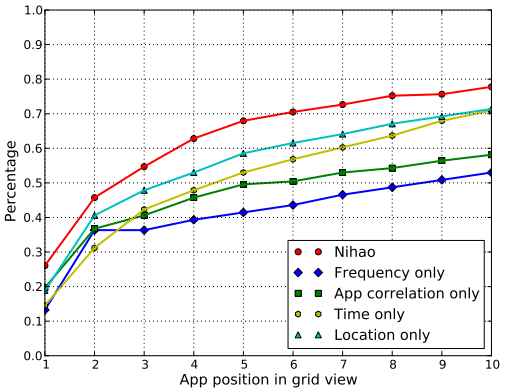


Fig. 8. CDF of launched app position for individual users (User1)

5.3 App Lookup Time

Next, we show the effectiveness of Nihao by analyzing how quick it could help its users to locate apps. We compare app lookup time using Nihao with the lookup time using the default device UI. We define the former as the time between when Nihao UI was showed and the first app was launched, and define the later as the time between when home screen was showed and the first app was launched. The default UI app lookup time is further separated into two variables, the lookup time for built-in apps and the lookup time for downloaded apps. As Figure 12 illustrated, the built-in apps in general had the shortest lookup time since apps like phone, messaging and browser were usually launched through quick launch bar at the bottom of the home screen, thus requiring little lookup time. But our focus is to compare the lookup time through Nihao with the time needed to launch downloaded apps through the default UI. In both cases, for 35% of

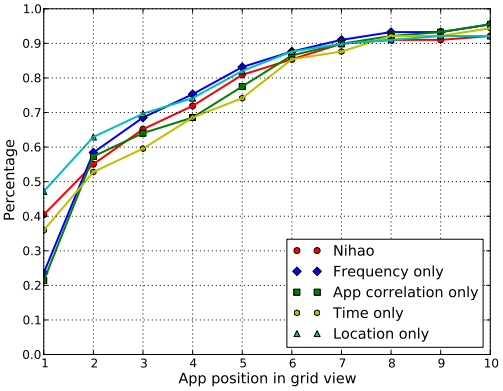


Fig. 9. CDF of launched app position for individual users (User3)

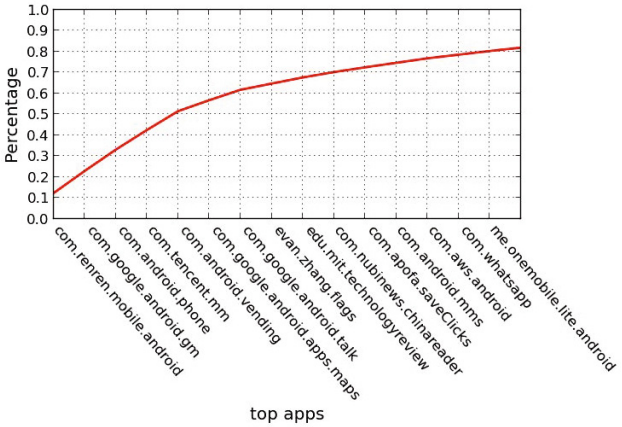


Fig. 10. Top apps launch count for individual users (User1)

times the apps were launched within 2.5 seconds. However, for about 70% of times, apps launched via Nihao required at most five seconds while only 60% of times required at most five seconds to launch a downloaded app. On average, lookup time for Nihao is 4.94s while the time for finding the downloaded apps through default launcher UI is 5.51s. Comparing to the default UI with high user familiarity, even the benefit in terms of app lookup time is not significant in general, (but it could be very significant for some users as Figure 13 shows), it is already a great achievement for Nihao to outperform within such a short period of learning time (three weeks). In addition, Nihao could potentially help save time required to manage apps (i.e. arranging app icons on home screen and assigning apps to different folders).

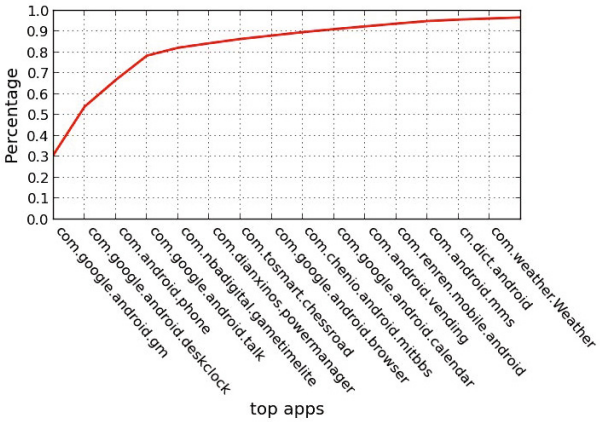


Fig. 11. Top apps launch count for individual users (User3)

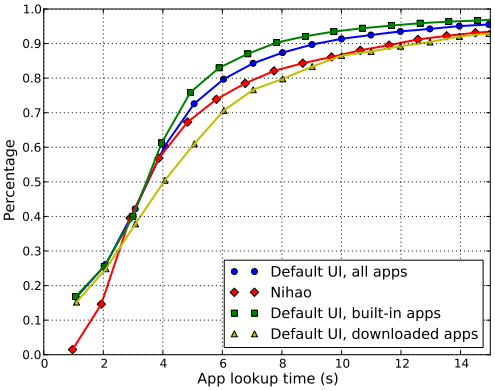


Fig. 12. CDF of app lookup time

5.4 Correlation between Launched App Position and App Lookup Time

After analyzing the *launched app position* and the *app lookup time*, it is interesting to see the correlation between them. In Figure 15, the x-axis represents the row number in a grid view and the y-axis represents the average time for finding the apps in a particular row with an error bar showing the standard deviation. As expected, the app lookup time is monotonically increasing as the row number in grid view increases which suggests that the more accurate prediction yields shorter app lookup time and it also proves that our eyes usually scan for apps row by row from the top. To further understand the correlation between the individual app position and the app lookup time, we show Figure 16 with the

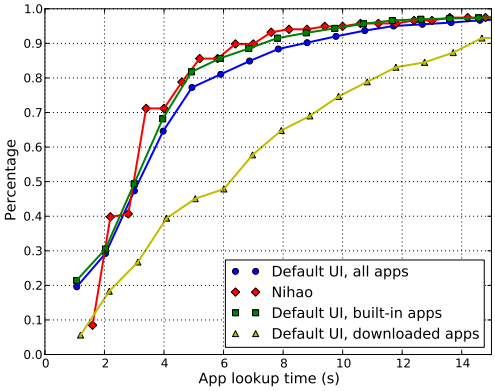


Fig. 13. App lookup time for individual users (User3)

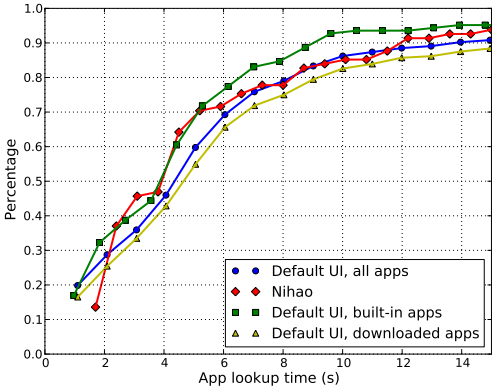


Fig. 14. App lookup time for individual users (User5)

x-axis representing the app icon position number in a grid view (This number increases with the position from left to right and from top to bottom). and the y-axis representing the average time for finding the app in that position. Unfortunately, we failed to find similar pattern as Figure 15. This result suggests that the users may start scanning the apps from left, right or even in the middle.

5.5 UI Predictability

In order to understand how dynamic/predictable the Nihao UI is (we assume that more dynamic UI yields less predictable UI), we introduce a metric *UI difference* between Nihao launches that is calculated as the number of positions on which the app changed between Nihao launches. For example, the difference

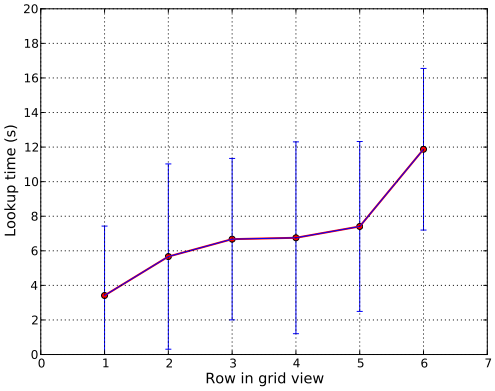


Fig. 15. Correlation between launched app position(in term of row in grid view) and lookup time

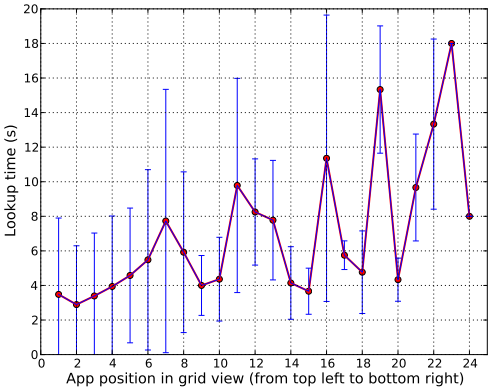


Fig. 16. Correlation between launched app position and lookup time

between (A B C) and (A C B) is two. Then, we use this metric to compare Nihao with other predictors (same as the predictors being compared with when we analyze the *launched app position*). As Figure 17 shows, for about 80% of the times, the number of the changed apps was at most five for the frequency based predictor, while for less than 30% of the times Nihao was as dynamic. In other words, Nihao UI was significantly more dynamic than the UI of the frequency based predictor. The app correlation only based predictor produced quite dynamic UI and the curve was fairly close to the curve of Nihao which means that the app correlation is the primary contributor to the highly dynamic UI of Nihao. The time only based predictor and the location only based predictor produce much less dynamic UI than Nihao, though its UI was still more dynamic than the frequency based predictor.

To understand whether the larger UI difference will produce longer app lookup time, we plotted Figure 18 with x-axis representing the number of the UI difference between the Nihao launches and y-axis representing the average app lookup time given the number of UI difference with the error bar showing the standard deviation. The figure shows that the larger UI difference does not necessarily require longer app lookup time. At the same time, we already know that the app lookup time is highly correlated to the app position (Figure 15). Thus, it allows us to conclude that the prediction accuracy is the most important factor than the UI predictability in designing such a system.

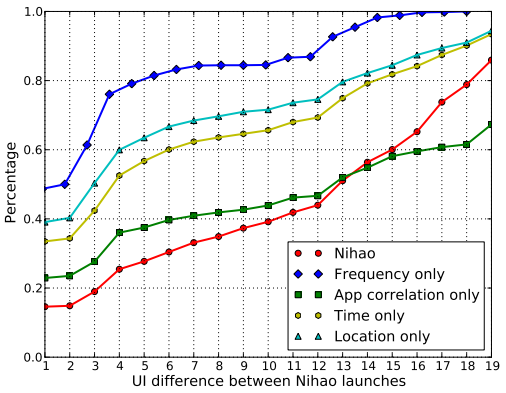


Fig. 17. CDF of UI difference between Nihao launches

5.6 User Survey

Finally, we conducted a user survey to better understand Nihao qualitatively. We first asked questions about how the users were satisfied with the predicted apps. Given the scale of one to seven, the average score is 6.3 which indicates that the users were quite satisfied. Furthermore, five out of seven users would like to keep using Nihao regularly after the test since they had become familiar with Nihao and chose it as their personal app management assistant. The other two users mainly used their phones for calling and texting thus Nihao was not as helpful to them. Next, we obtained user opinions on Nihao's UI design. First, six out of seven users preferred Nihao to be configured as a widget in one of the homescreens instead of a standalone app since one more click could be saved in the form of a widget. Second, all users preferred the grid view to the list view. Finally, by considering two typical usage scenarios for Nihao: 1) to find a specific app in mind quickly, 2) to browse and open some interesting apps to kill time), five out of seven users chose the first option which suggested that Nihao was largely treated as a serious app lookup tool instead of a time killer.

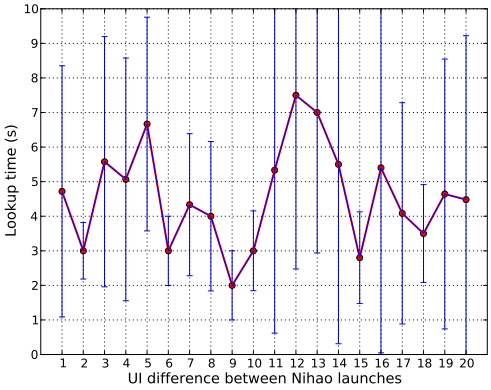


Fig. 18. Correlation between UI difference and app lookup time

6 Conclusion and the Future Work

To find apps quickly on a smartphone with the overwhelmingly large app base, a more efficient and intelligent app launcher is necessary. Nihao, as a time-aware, location-aware, app correlation-aware and personalized app launcher, demonstrates it effectiveness in term of app lookup time and management time. Data analysis result shows that looking up downloaded apps using Nihao yields less time than using the default launcher on average. Then through the user survey, we found that majority of the users thought Nihao could help save the app management time by ranking the apps automatically.

Then we validated that users' app usage pattern are indeed context dependent by showing that Nihao predicts better than the frequency-based predictor. Thus, exploring the correlation between app usage and the richer context is one of our future work.

Finally, we found that the larger UI change does not necessarily yield longer app lookup time as the app lookup time highly depends on the app icon position on screen, which suggests the prediction accuracy is the most important factor in designing such a system. Thus, in the future, we plan to focus on improving the prediction accuracy by experimenting with different models leveraging richer context and testing the system with more users of different backgrounds.

Acknowledgment. This material is based upon work supported partly by the National Science Foundation under Grant No. 1040725 and No. 0917112. Any opinions, ndings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reect the views of the National Science Foundation.

References

1. androlib, http://www.androlib.com
2. Yelp, http://www.yelp.com
3. Hartmann, M.: Challenges in developing user-adaptive intelligent user interfaces. In: Proc. LWA 2009 (2009)
4. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in smartphone usage. In: Proc. ACM MobiSys 2010, San Francisco, CA (June 2010)
5. Bohmer, M., et al.: Falling asleep with angry birds, facebook and kindle - a large scale study on mobile application usage, Stockholm, Sweden (August 2011)
6. Do, T.M.T., Blom, J., Gatica-Perez, D.: Smartphone usage in the wild: a large-scale analysis of applications and context, Alicante, Spain (November 2011)
7. Huang, K., Ma, X., Zhang, C., Chen, G.: Predicting mobile application usage using contextual information. In: Proc. ACM Sagaware 2012 (2012)
8. Xu, Q., et al.: Identifying diverse usage behaviors of smartphone apps, Berlin, Germany (November 2011)
9. Sears, A., Shneiderman, B.: Split menus: effectively using selection frequency to organize menus. ACM Trans. Comput. Hum. Interact. (1994)
10. Findlater, L., McGrenere, J.: Impact of screen size on performance, awareness, and user satisfaction with adaptive graphical user interfaces. In: Proc. SIGCHI 2008 (2008)
11. Phithakkitnukoon, S., Dantu, R., Claxton, R., Eagle, N.: Behavior-based adaptive call predictor. ACM Transactions on Autonomous and Adaptive Systems 6(3) (September 2011)
12. Yan, B., Chen, G.: Appjoy: Personalized mobile application discovery. In: Proc. ACM MobiSys 2011 (2011)
13. Bohmer, M., et al.: Exploring the design space of context-aware recommender systems that suggest mobile applications. In: Proc. CARS 2010 (2010)
14. Yan, T., et al.: Fast app launching for mobile devices using predictive user context. In: Proc. ACM MobiSys 2012 (2012)
15. Google's data interchange format, http://code.google.com/p/protobuf/
16. Play framework, http://www.playframework.org/