Mobilewalla: A Mobile Application Search Engine

Anindya Datta, Kaushik Dutta, Sangar Kajanan, and Nargin Pervin

College of Computing, National University of Singapore, Singapore
{datta,dutta,skajanan,nargis}@comp.nus.edu.sg

Abstract. With the popularity of mobile apps on mobile devices based on iOS, Android, Blackberry and Windows Phone operating systems, the number of mobile apps in each of the respective native app stores are increasing in leaps and bounds. Currently there are almost 700,000 mobile apps across these four major native app stores. Due to such enormous number of apps, both the constituents in the app ecosytem, consumers and app developers, face problems in terms of 'app discovery'. For consumers, it is a daunting task to discover the apps they like and need among the huge number of available apps. Likewise, for developers, making it possible for users to discover their apps in the large number of available apps is a challenge. To address these issues, Mobilewalla(MW), provides an independent unbiased search engine for mobile apps with semantic search capabilities. It has also developed an objective scoring mechanism based on user and developer involvement with an app. The scoring mechanism enables MW to provide a number of other ways to discover apps - such as dynamically maintained 'hot' lists and 'fast rising' lists. In this paper, we describe the challenges of developing the MW platform and how these challenges have been mitigated. Lastly, we demonstrate some of the key functionalities of MW.

Keywords: Mobile App, Search Engine, Semantic Similarity.

1 Introduction

Consumer software applications that run on smartphones (popularly known as mobile apps, or, simply, apps) represent the fastest growing consumer product segment in the annals of human merchandising [1,2]. The absolute number of apps currently in existence, as well as their rates of growth, are remarkable. At the time of writing this paper, there are 404126, 274555, 30784, 19796 apps available in Apple, Android, Blackberry and Windows platforms respectively. Since December, 2010, the app growth rates for the Apple and Android platforms are nearly 4% and 7% on monthly basis respectively.

This scenario creates a number of problems for the two key constituencies in the app ecosystem, the consumers and the developers. For consumers, there are simply too many apps and far too much fragmentation in these apps (e.g., a large number of categories). The analogy we often use to describe the confusion faced by a mobile app consumer is to imagine a customer walking into a grocery store, needing only a few items, and finding that all aisles and category labels have been eliminated, and every product has been thrown into a pile on the floor. It is a similarly daunting task for a consumer to navigate native app stores [3,4] and discover apps they need and like, as has been widely discussed in media forums in the recent past [5,6].

For application developers, the situation is far worse. There are almost 700,000 mobile apps between Apple and Android alone and most smartphone owners only can only identify a handful - this is a nightmare scenario for developers whose success is contingent upon getting their apps “found” by consumers. “How will my apps be discovered?” is the number one question in the mind of app developers. This issue, known as the “app discovery” problem, has received wide attention in the media as well [7,8].

Clearly, a key requirement to address the above issues is an effective system of discovering and searching mobile applications - in essence, a “search engine” for apps. The reader might point out that each of the native app markets (e.g., the iTunes appstore and the Android market) offer search capabilities. However, as has been widely discussed [9,10], the native app stores are commercially driven and search results are highly influenced by what the store wants the consumer to see, rather than being solely focused on producing relevant output. Moreover, as has also been widely reported, app store designs are confusing along many dimensions, such as having multiple display lists where individual apps are often listed in conflicting orders. For example, at the time of writing this paper ‘mSpot Music’ was the first featured app in the Android market [3]. This app is in category ‘Music & Audio’. Investigating further in the ‘Music & Audio’ category of the Android market, we found ‘mSpot Music’ is not even in top 200 apps in that category.

In response, there is intense interest in creating independent unbiased search systems for mobile apps. One of the earliest entrants in this space is Mobilewalla (MW) (www.mobilewalla.com). MW is a full fledged app search engine employing semantic search capabilities. In addition to supporting basic keyword search, MW also features a number of other ways to discover apps - such as dynamically maintained “hot” lists and “fast rising” lists. One of MW’s major contributions is the creation of an unbiased scoring system for apps based on the Mobilewalla Score (MWS). MWS has rapidly become the defacto standard of rating apps and has been recently featured in a number of “top app” lists, the most prominent being the New York Times Box Scores [11].

In this paper we describe Mobilewalla platform. In Section 2, we describe the architecture and approach. In Section 4, we explain the functionality of Mobilewalla platform with some screen shots. In Section 5, we discuss some of the other app search platforms and compare those with Mobilewalla system. Lastly in Section 6, we conclude the paper.

2 Overview

As articulated previously, Mobilewalla has been developed as a search, discovery and analytics system for mobile apps. The Mobilewalla consists of three independent components.

1. A data collection (DC) component, which collects data from native app stores (iStore [4], Android Market [3], Blackberry App World [12] and Windows Mobile Marketplace [13]).
2. The Computational Backend (CB), which cleans and processes data collected by the DC.
3. A GUI component, which displays data that has been collected in step 1 and computed in step 2.

All of these components were created to address a number of challenges, which we outline below.

1. Developing automated extraction mechanisms, or *crawlers* that work on dynamic web sites.
2. As soon as we started playing around with the data available in the native app stores, we realized that there were a lot of issues with the data, such as incompatible categorizations across different stores and mischaracterization of app content by developers. A key goal was to develop automated *data cleaning* methods that presented "clean" data to the user.
3. Allowing real-time search of vast app content
4. Computing "similarity" between, and across, mobile apps
5. Finally, a key usability goal was to find effective mechanisms to present a vast amount of multi-media information to the user, which in turn would result in effective searching of apps and the discovery of apps relevant to the user's requirements.

Below, we describe the high level architecture of the complete Mobilewalla system, and elaborate on the solution methodologies employed to overcome the above challenges.

3 Architecture

Figure 1 shows the architecture of the Mobilewalla platform. The Data Collector (DC) component gathers information from the native app stores, extracts data and stores it in the Mobilewalla database (MDB). The Computational Backend (CB) interacts with the MDB to perform computations which enable users to search and discover apps efficiently. Finally, the GUI component interacts with the application server to retrieve the data from MDB and present it to users.

3.1 Data Collector (DC)

The essence of the DC component is an automated extraction system, or a *crawler*, that fetches useful data out of the native app stores and writes it to the MDB. We first describe the crawler and then discuss the MDB.

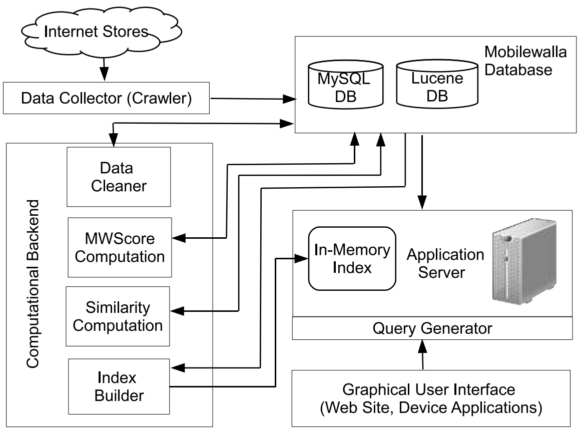


Fig. 1. Architecture of Mobilewalla

Designing crawlers for app stores is not easy, as the app stores are not based on a uniform structure of data representation. Rather, each app store has its own format. Therefore, we have developed four different crawlers, one for each of the four native app stores - (i) iStore [4], (ii) Android Market [3], (iii) Blackberry App World [12] and (iv) Windows Mobile Marketplace [13]. Typical crawlers [14,15,16], that are widely available as open source software components, have been developed to fetch information from generic internet sites. These crawlers work on the assumption that web content is available in static HTML pages and do not work on dynamically generated content created with technology such as AJAX, Silverlight and Flash ActionScript. It turns out that the native app stores are built using such kind of dynamic web technologies. These technologies are based on scripting languages, which create component based web pages on the fly. Let us consider the example of AJAX, arguably the most prevalent dynamic web development framework. AJAX is dependent on a scripting language called Javascript, which allows dynamic update of parts of a web page. This makes it difficult to download and parse the complete web page as is usually done in standard crawlers. Consider the example of the Google search engine. Google can crawl static content such as that can be found in HTML pages, PDF and Word documents. Google is unable to crawl an AJAX based web site such as Kayak.com. For a AJAX based web application, the response from the web server comes in the form of JSON and not in the form of HTML. To crawl and extract information from a AJAX based web site, one needs to parse JSON responses.

Complicating matters further, each app store has its own JSON response format. To handle this, we have developed ways to parse these individual JSON responses to extract app specific details based on the JQuery library. Consequently, we have had to develop individual store specific crawlers that encapsulate the knowledge of the JSON format for each individual store. Understanding the JSON format for each store and developing "personalized" parser routines has proven to be a non-trivial task.

Having described our idea behind our automated extraction mechanism, we now focus on the specific data item that we extract. These are shown in Table 1.

Table 1. Data Captured by Crawler

Static Data	App Name	Dynamic Data	Comment
	App Description		Rating
	Release Date		Version
	Platform		Rank in Store
	Price		Times Downloaded
	Screen Shot		
	Icon		
	Category		
	Seller/Developer Name		
	URL		
	Language		
	Size of an App (MB)		

Our extracted information consists of both static and dynamic data elements as shown in Table 1. This data is extracted and stored in the Mobilewalla database (MDB) described below in Section 3.2.

The static data is downloaded and stored in the database only once. If any changes occurs to this data the updated information is downloaded again and the old data is replaced with this new information. For example, an app might have its description updated. In this case, the old description is overwritten in the database with the new description.

Dynamic data, which typically changes frequently, is fetched by the crawler on a continuous basis. For dynamic information the old data is never deleted or replaced. Rather, all dynamic data is time stamped and retained in the database. As an example, consider the user rating of an application, an important dynamic data item. Whenever there is a change in an application's ratings in its native app store, the new rating is stored along with a date stamp. Similarly, any new comment, if discovered, are appended to the database in the same manner.

When a new app is encountered, the crawler downloads all data related to this app. The crawler then runs regularly to track data about this app. How frequently the crawler will revisit a particular app is determined dynamically. Let us assume, T_a denotes the interval at which the crawler will revisit a particular app a. Initially $T_a = 3$ hours. If in the next iteration the crawler does not find any update for the app a, the next revisit will happen at $n \times T_a$ period, i.e. T_a will be updated as $T_a^{new} = n \times T_a^{old}$, where n is a positive number. If even in the subsequent visit, the crawler does not find any new update or new dynamic data, then T_a will be updated again. At maximum value of $T_a = T_a^{max}$, the value of T_a is no more updated. We set $T_a^{max} = 48$ hours, i.e. at every 2 days the crawler will revisit each app irrespective of whether there is any update or not for that particular app. Such an approach ensures two aspects of the crawler. (i) The crawler need not visit all the apps in a store at every iteration, which reduces the total time the crawler takes at every run. In our environment, this enables the crawler to run at every 3 hours. For highly dynamic and popular apps, this in turn guarantees that the crawler will quickly (within 3 hours) identify the new data and put into our system. (ii) For less dynamic apps, the crawler will identify the new data within a maximum 2 days. This system creates a balance between dynamism of the app and the scalability of the crawler system.

In many instances, existing apps get deleted from its native stores. In these circumstances, the crawler will fail to download any data from the store. We keep a note of how many consecutive times the crawler is failing to download the information related to a particular app. If the count crosses a threshold, we assume that the app has been deleted from the store and mark it as inactive in our database.

3.2 The Mobilewalla Database

Having described the crawler system, we now delve into the Mobilewalla database (MDB), which is the store-house of the crawler extracted information. The MDB has a structured relational component created on top of MySQL DBMS [17] and an unstructured data store based on Lucene text search engine [18].

MySQL Database. The relational MySQL database of MDB primarily contains structured data captured by the DC such as Version Number, Release Date and Price along with the unstructured textual data such as title, description and comments. The database schema reflects the information captured in Table 1. Other than few global tables, each native app store has its own set of tables, that contains the data captured by DC from that store only.

One of the challenges encountered in Mobilewalla was how to enable users to browse apps by categories such as Finance, Entertainment or Life Style. Native app stores designate categories for each app; however these store specified categories have a number of issues. (1) Developers tend to wrongly categorize apps based on where it would attract most users rather than the category to which the app naturally belongs based on its content. This often results in gross miscategorization of apps. For instance, the android app 'Men and women of passion video' is in the category 'Health & Fitness' in the Android market, whereas this app is an adult video app, and should have been appropriately categorized under 'Media & Video'. Natives stores themselves often do not perform extensive verification of such errors. One of our goals in Mobilewalla was to remove these miscategorizations as far as possible. (2) Another major issue is the lack of uniform categorization across the stores. For instance, 'Skype' is under 'Communication' category in Android market, whereas it is under 'Social Networking' category in iStore. Our goal in Mobilewalla was to present apps based on a uniform ontology.

To address this issue, we have developed a unique categorization scheme across multiple stores, called global categories. The information about these global categories are kept in separate global tables. We have developed an automatic categorization scheme, which is based on the Wikipedia ontology [19]. At a high level, we use the title and the description of an app to identify the keywords for an app. These keywords are matched with the keywords for Wikipedia categories to categorize the app in a global category defined in global tables.

Lucene. A key requirement in Mobilewalla was to support free form keyword search. We use the Lucene text search engine to fulfill this requirement. In particular we create an inverted index out of the textual content attached to each app (including the description and the title). Subsequently we support search based on this index.

We should point out however that a straight forward implementation on top of Lucene was not possible. The reasons are,

1. The Lucene query processor awards equal weights to every keywords in a document. This does not work for us, as we have developed certain proprietary rules about the importance of different keywords. (e.g., keywords in title should have higher weights than keywords in the description) To handle this we modified Lucene such that variable weights could be attached to keywords.
2. Lucene does not perform stemming, i.e. if a user searches for the word "swimming" the apps related to the keyword "swim" will not be returned by Lucene. We incorporated stemming and lemmatization to transfrom a word to its base form. The stemming uses simple word transformation to modify a word to its base form, e.g. "going" to "go". The lemmatizer uses the word's parts of speech (POS) and a valid dictionary to identify the original form. Our lemmatizer is based on the OpenNLP [20] parser and the Wordnet dictionary. We used both the original word and the transformed base form of the word to index the app in the Lucene database.

3.3 Computational Backend (CB)

The Computational Backend (CB) components operates on the base data present in the MDB to perform a set of secondary computation; it first *cleans* the data and then produces a set of *proprietary metrics.*

Data Cleaning. The raw data captured by the crawler has been primarily entered by the developers who typically make a number of (often intentional) errors. The data cleaning component of the CB attempts to fix these errors by following tasks.

Categorization - We discussed the issues related to store categories before. As discussed, we can't rely on the store and developer provided categories. So we needed to develop our own automatic categorization system. A key task of the CB is to perform the Wikipedia based categorization described before.

Duplicate App Removal - Often app developers submit an app in a store and for the next version the app developers do not update the already existing app, rather they create a new entry in the system. This creates duplicate entry of the same app in the store. In iPhone stores we found that about 10,000 such apps exist. We have developed an automatic identification method of the duplicate apps based on comparing core app attributes such as app title, description and developer name.

Deleted or Inactive App - Many apps in native stores turn out to be orphaned or defunct. These apps, after release, have not had any version upgrades or platform upgrades, likely because the developer abandoned the app. We have developed a heuristic based solution to identify these defunct apps based on inactivity periods and platform compatabilities.

Mobilewalla Score (MWS). One of the values Mobilewalla provides to the end user is providing a unique and uniform scoring mechanism of apps across categories and platforms. Unlike existing app search engines, where app rankings are based on highly subjective and often commercial considerations, in Mobilewalla we wanted to rate and rank apps based on uniform unbiased criterion. This is achieved through the Mobilewalla Score (MWS). The MWS of an app is based on several objective parameters that denote how users and developers are engaged with the app. Some of the factors used to compute the MWS are (1) the current and historical ratings of the app in its native store, (2) the frequency of releases, (3) the duration that the app is active in the store since its first release, (4) number of users who rated the app, (5) number of users who provided comments on this app, (6) the current and historical rank of the app in its native store, (7) the number of apps in the category the app belongs to, and (8) the past history of the developer in producing apps. The system computes the MWS every night and posts it in the database. We keep track of the historical value of MWSs for each app.

The current and historical value of MWS are used to create a number of popular lists of Mobilewalla.

Hot Apps - This is the list of apps with the highest MWS at current time.

Fast Movers - This is the list of the apps for which the change of the MWS is the highest. This measures the velocity in the MWS for each app and report top apps with the rate of change.

All Time Greats - This is the list of the apps which have the highest average MWS over last 3 months.

Similarity Computation. Mobilewalla enables users to find apps similar to a particular app and compare the similar apps on characteristics such as MWS and price (much like the "if you like this you might also like" feature in Amazon). We compute a measure of similarity between two apps based on the semantic similarity across several parameters, such as description, title and comments received from users. The semantic similarity computation of these features are done using hypernym, hyponym and synonym relationships in Wordnet [21]. Based on the importance in describing features of an app, the similarity measurement across each of these parameters is given different weight in the final similarity measurement. For example, the similarity on title is given more weight, than the similarity on description. The similarity on comment is given the least weight.

For each app, we identify the similar apps that cross a threshold on similarity measurement with respect to that app.

In-memory Index and Index Builder. One of the objectives of the Mobilewalla application is to enable complex search in the database in real time. For this, the in-memory index contains materialized views of the database containing following items - (1) The inverted index of the Lucene data and (2) Index of the app on the basis of app parameters, such as price, release date, and developer name. These indexes are pre-built nightly, and reduce the complex join operations that require querying the database against some of the user queries.

3.4 Query Generator

The query generator module receives user provided query keywords for searching the app database. The query generator transforms this user query across several dimensions to generate more than one query. The result of the query generator is a series of queries ordered in priority. These queries are executed in Lucene's in-memory index to identify the app. If the prior queries result in a threshold number of relevant apps from the Lucene in-memory index, the later queries are not executed.

The query generator expands the query in following different ways.

Stemming and Lemmatization - The words in the user query will be stemmed and lemmatized to create original base form of the query words. Additional queries will be generated using these base forms. For example, a user query "swimming" will create a second query "swim".

Formulation of AND and OR Query - First we do "AND" query for the user given keywords. If the "AND" query does not return sufficient result, we expand the query using "OR" across the user given keywords. For example, if user given query is "Racing Bicycle", then the two queries "Racing AND Bicycle" and "Racing OR Bicycle" are formed. If the query "Racing AND Bicycle" returns less than a threshold number of results (in Mobilewalla implementation that threshold value is 20), the OR query "Racing OR Bicycle" is executed. This "AND" and "OR" combination is also applied on the original base form of the user give query keywords.

Query Expansion - If the user given query does not return a threshold number of results, we expand the query using synonyms and hyponyms of the query keywords. For example, a query "Bollywood Movie" will be expanded into "Bollywood Cinema" and "Bollywood Picture", because "Cinema" and "Picture" are the synonym and hyponym of the word "Movie".

4 Screen Shots and Descriptions

The Mobilewalla architecture is flexibly implemented using a JSON interface. The application server provides a set of JSON APIs that can be invoked by any client over HTTP. Currently, the clients supported include iOS devices (i.e., iPhone/iPod/IPad), Android devices and desktop web applications. All these clients communicate with the Mobilewalla server application using the same JSON API set, but differ in the user interface offered to the end user. We will now proceed to demonstrate some of the important functionalities of the Mobilewalla application by using desktop web application as an example (the user may interact with this application at www.mobilewalla.com).

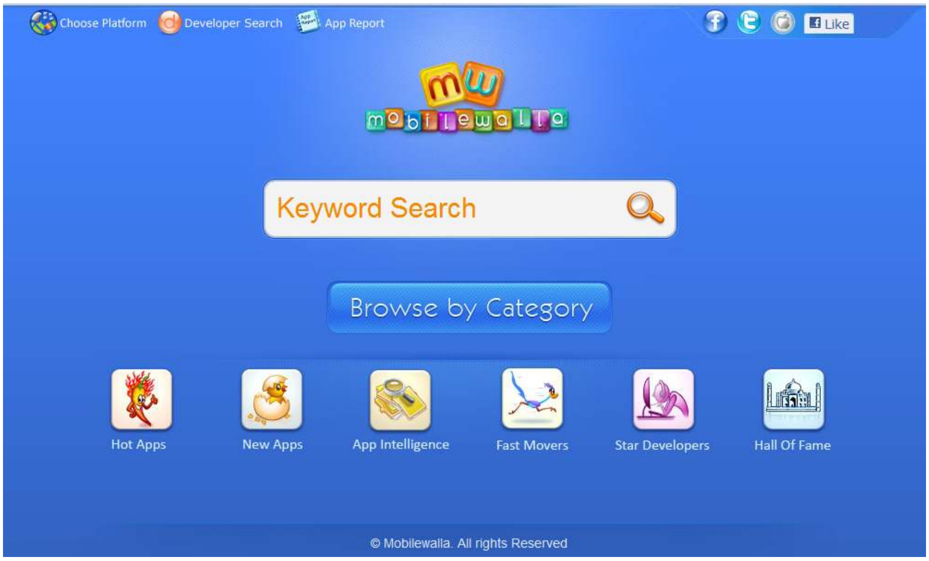


Fig. 2. Main Menu Page

When the user arrives at the Mobilewalla application, the first step is to choose a platform of interest, i.e., the user must specify which smartphone platform is of interest to the user – iPhone/iPod, iPad, Android, Blackberry or Microsoft (the user may also choose a "don't care" option, marked as "All" in Mobilewalla). Once a platform is chosen the user will be directed to the main "splash page" shown in Fig 2. In the screenshot shown in Fig 2, the platform chosen appears on the extreme right of the top menu bar (iPhone/iPod in this case). This means that all apps presented to the user will correspond to the iPhone/iPod platform until this is explicitly changed, by selecting the "Choose Platform" widget present on the extreme left of the top menu bar.

From this screen, the user may choose to navigate the app world in a number of ways. The first, and the most common method of interaction is by entering a search query in the keyword input box. Let's assume the user enters the search term "`earth photo`". Mobilewalla returns a set of apps that fit the user's interest as shown in Fig 3 – in this view Mobilewalla provides not only the app name, but also a number of other core features such as author and price. One notable feature of this view are the *relevance* and *Mobilewalla meter* (*MW Meter*) indicators present in each app box. Relevance indicates the quality of "fit" of that app with respect to the input seach query, whereas *MW Meter* is an encapsulation of the "goodness" of the app as measured by Mobilewalla (this is based on the Mobilewalla Score metric described earlier). Also, while not shown in the screenshot, we also segment the apps by Free and Paid and allow a number of options to sort the result set (the user may view these by visiting mobilewalla.com).

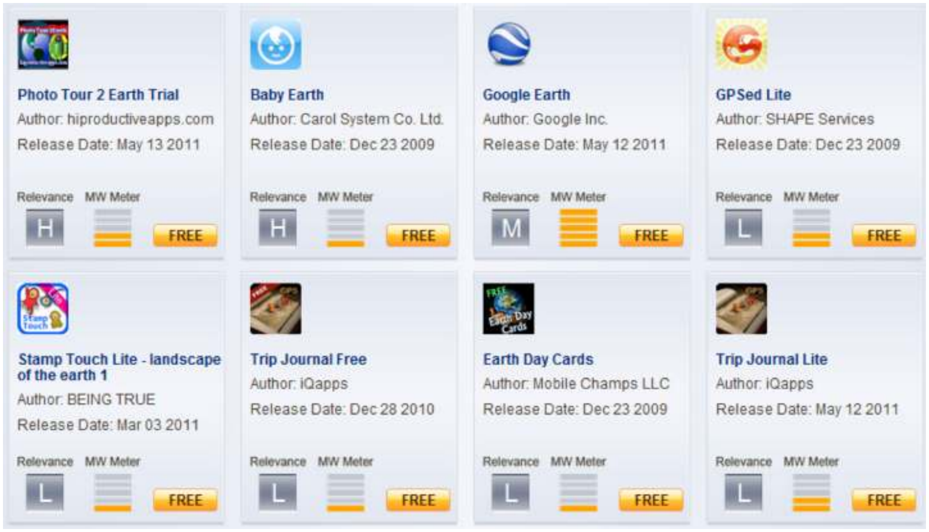


Photo Tour 2 Earth Trial
Author: hiproductiveapps.com
Release Date: May 13 2011
Relevance MW Meter
H FREE

Baby Earth
Author: Carol System Co. Ltd.
Release Date: Dec 23 2009
Relevance MW Meter
H FREE

Google Earth
Author: Google Inc.
Release Date: May 12 2011
Relevance MW Meter
M FREE

GPSed Lite
Author: SHAPE Services
Release Date: Dec 23 2009
Relevance MW Meter
L FREE

Stamp Touch Lite - landscape of the earth 1
Author: BEING TRUE
Release Date: Mar 03 2011
Relevance MW Meter
L FREE

Trip Journal Free
Author: iQapps
Release Date: Dec 28 2010
Relevance MW Meter
L FREE

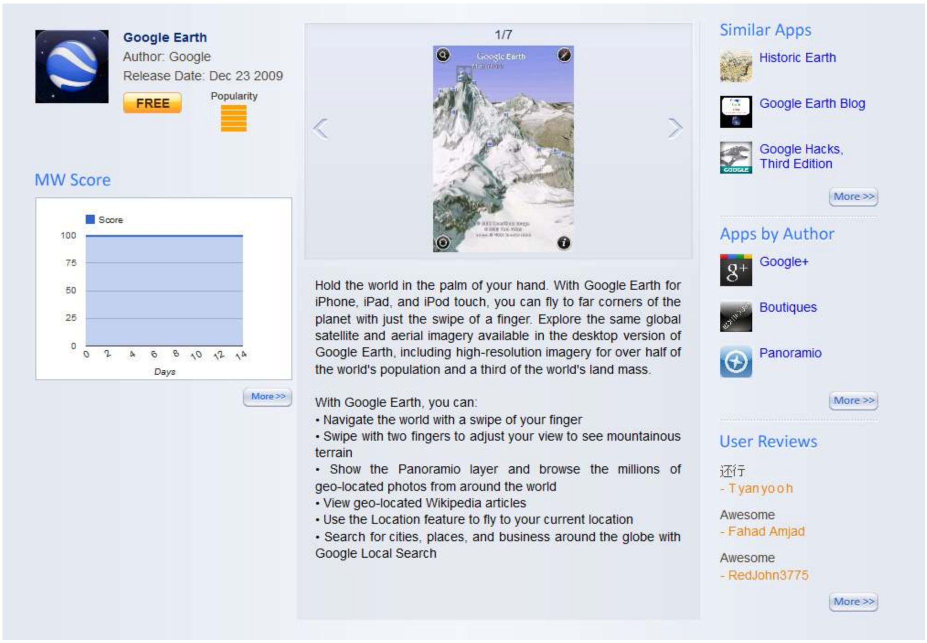
Earth Day Cards
Author: Mobile Champs LLC
Release Date: Dec 23 2009
Relevance MW Meter
L FREE

Trip Journal Lite
Author: iQapps
Release Date: May 12 2011
Relevance MW Meter
L FREE

Fig. 3. Keyword Search Results Page

The user may choose any app from the app-list view just described and delve into its details. Let us assume the user chooses the Google Earth app. In this case she will be presented with the detail view of this app, shown in Fig 4. In this view, Mobilewalla displays details such as the app description and screenshots and also allows the user to view a number of other interesting artifacts related to this app, such as "Apps by Author" (other apps created by the author of the app detail being viewed), "Mobilewalla Score"(the Mobilewalla score history related to this app over the past 14 days), "Comments", and "Similar Apps" (similar to the "if you like this, you might also like" feature in Amazon).

The above two paragraphs describes how a user might interact with Mobilewalla by performing a keyword search and then drilling down on the results. However, keyword search is just one of many ways that the user can navigate Mobilewalla. He might also choose to view apps by categories, or choose one of the many "pre-defined" list options such as "Hot Apps", "Fast Movers" and "New Apps". Choosing the "Browse my category" option reveals a number of category icons from which the use may navigate the app world – Fig 5 shows the results of choosing the "Maps & Navigation" category.



Hold the world in the palm of your hand. With Google Earth for iPhone, iPad, and iPod touch, you can fly to far corners of the planet with just the swipe of a finger. Explore the same global satellite and aerial imagery available in the desktop version of Google Earth, including high-resolution imagery for over half of the world's population and a third of the world's land mass.

With Google Earth, you can:
• Navigate the world with a swipe of your finger
• Swipe with two fingers to adjust your view to see mountainous terrain
• Show the Panoramio layer and browse the millions of geo-located photos from around the world
• View geo-located Wikipedia articles
• Use the Location feature to fly to your current location
• Search for cities, places, and business around the globe with Google Local Search

还行
- Tyanyooh

Awesome
- Fahad Amjad

Awesome
- RedJohn3775

Fig. 4. App Details Page

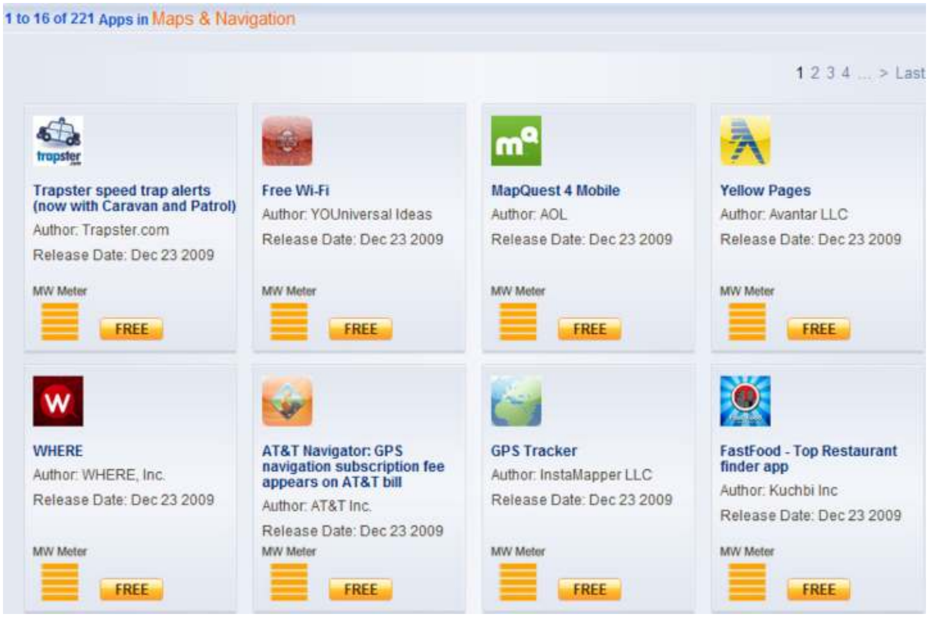
Similarly choosing “Hot Apps” displays the list of the top 1000 apps ordered by their Mobilewalla Scores, while “Fast Rising” apps are those whose Mobilewalls scores have demonstrated the steepest ascent, i.e., apps getting hot the fastest. “New Apps” are those that are less than a month old. In every case a number of sort options are available that allow users to manipulate the result set along various dimensions.

While Mobilewalla has a number of other interesting features, it is infeasible to describe them in this paper due to length restrictions. We invite the user to visit the site.

5 Related Work

Mobilewalla is one of the earliest entrants in the app search and discovery space. In this section, we describe few alternatives available for app search.

Appolicious Inc, associated with Yahoo Inc [22], is a web application to help users easily find mobile applications that are useful and interesting to them. The Appolicious recommendation engine determines what apps you have, what apps your friends and the rest of the community have, and uses individual app ratings and reviews to suggest new apps for your use. Unlike Appolicious, Mobilewalla depends on its own developed scoring mechanism. Users can search for an app in the Mobilewalla system using keywords, which is based on its own index mechanism. Mobilewalla also identifies similar apps based on content than the usage,


1 to 16 of 221 Apps in Maps & Navigation

1 2 3 4 ... > Last

Trapster speed trap alerts (now with Caravan and Patrol)
Author: Trapster.com
Release Date: Dec 23 2009
MW Meter
FREE

Free Wi-Fi
Author: YOUniversal Ideas
Release Date: Dec 23 2009
MW Meter
FREE

MapQuest 4 Mobile
Author: AOL
Release Date: Dec 23 2009
MW Meter
FREE

Yellow Pages
Author: Avantar LLC
Release Date: Dec 23 2009
MW Meter
FREE

WHERE
Author: WHERE, Inc.
Release Date: Dec 23 2009
MW Meter
FREE

AT&T Navigator: GPS navigation subscription fee appears on AT&T bill
Author: AT&T Inc.
Release Date: Dec 23 2009
MW Meter
FREE

GPS Tracker
Author: InstaMapper LLC
Release Date: Dec 23 2009
MW Meter
FREE

FastFood - Top Restaurant finder app
Author: Kuchbi Inc
Release Date: Dec 23 2009
MW Meter
FREE

Fig. 5. Category Search Results Page

as is the case in Appolicious. The most important attraction of Mobilewalla [23] is the availability of different kind of Search functions which are ‘keyword based search’, ‘category search’ and ‘developer search’ based on the semantics and topics. Because of this uniqueness, through Mobilewalla one can find a much larger variety of apps.

Chomp is an app search engine [24], where the search primarily occurs based on string search. Mobilewalla, on the other hand, searches apps based on their semantic content. For example, Mobilewalla includes semantically related words in search terms. It also identifies words in app title and description that are semantically more meaningful than others. As a result, the search outputs are vastly different. For example, the search term “ESPN Cricket” in Android platform returned only one app “Sports Eye - Lite” in Chomp, whereas in Mobilewalla it returned total 486 apps with “zeendaTV Cricket”, “Cricbuzz Cricket Scores and News”, “ESPN Mobile” and “Cricket Mania” as the first few apps in the result list. Also, unlike Mobilewalla, which is applicable in iPhone/iPod/iPad, Blackberry, Android and Windows platform, Chomp’s search is limited only to Apple and Android app stores. Chomp’s trending search option is based on existing popularity of the app. Whereas Mobilewalla’s trending search results are based on Mobilewalla’s unique scoring mechanism, which uses other parameters indicating both the developer’s and user’s engagement.

AppBrain is a website [25] to search Android apps available specifically in the Google Android market, whereas MobileWalla [23] covers all the major smart phone platforms (Windows, Blackberry, Android and Apple). The main core feature of Mobilewalla system is its search functionality. It has semantic enabled keyword, category and developer based search functionalities. Keyword search in Mobilewalla is implemented in a way to find the semantically meaningful apps but not purely based on exact string match like most of the currently available app search engines like AppBrain do.

uQuery.com [26] is a mobile app search platform based on social media Facebook.com. Users can search and find applications on the iTunes App store based on what apps friends in Facebook are using. The key difference between Mobilewalla and uQuery is in the search mechanism. Mobilewalla relies on its own metrics which is combination of both the usage and the developer's engagement with an app, whereas uQuery relies on usage by friends in Facebook. The Mobilewalla's keyword based search engine is much more extensive than uQuery. Mobilewalla's keyword search can handle semantically related keywords, keywords appearing in title, description and user comments. Mobilewalla's similar app search mechanism is based on semantic similarity of apps, rather than the usage.

In summary, the key difference between the Mobilewalla and existing app search platforms is in two fronts. First, Mobilewalla depends on the semantic description of apps to enable search based on keywords and similar apps. Second, Mobilewalla ranks the app based on a scoring mechanism that incorporate both the user and the developer's involvement with the app. As discussed above, the existing search platforms consider only the user's aspect; like Mobilewalla developer's involvement with the app and the developer's history is not considered.

6 Conclusion

With the skyrocketing popularity of mobile apps on mobile devices based on iOS, Android, Blackberry and Windows Phone operating systems, the number of mobile apps is increasing in leaps and bounds. Currently there are over 700,000 mobile apps across these four major native app stores. Due to such enormous number of apps, both the constituents in the app ecosytem, consumers and app developers, face problems in terms of 'app discovery'. For consumers, it is a daunting task to discover the apps they like and need among the huge number of available apps. Likewise, for developers, getting their apps discovered in the pool of an enormous number of apps is a challenge. To address these issues, Mobilewalla provides an independent unbiased search engine for mobile apps with semantic search capabilities. It has also developed an objective app rating and scoring mechanism based on user and developer involvement with an app. Such scoring mechanism enables MW to provide a number of other ways to discover apps - such as dynamically maintained 'hot' lists and 'fast rising' lists. In this paper, we describe the challenges of developing the MW platform and how these challenges have been mitigated. Lastly, we demonstrate some of the key functionalities of MW.

Acknowledgment. We acknowledge the engineering work done by Ji Cheng and Sameer Samarth for Mobilewalla.com web site, iPhone app and Android app.

References

1. Android market grows a staggering 861.5 per cent
2. Mobile apps market to reach $38 billion revenue by 2015, http://news.brothersoft.com/mobile-apps-market-to-reach-38-billion-revenue-by-2015-6089.html (visited on May 13, 2011)
3. Android market, https://market.android.com/ (visited on May 13, 2011)
4. Apple app store, http://www.istoreus.com/home.html (visited on May 13, 2011)
5. Why native app stores like itunes and android marketplace are bad for mobile developers, http://www.businessinsider.com/why-native-app-stores-like-itunes-and-andoid-marketplace_are-bad-business-for-mobile-developers-2011-5 (visited on May 13, 2011)
6. Would you like to try android apps before buying?, http://www.labnol.org/internet/android-apps-try-before-buying/19422/ (visited on May 13, 2011)
7. Appsfire scores $3.6m as app discovery demands grow, http://gigaom.com/2011/05/30/appsfire-scores-3-6m-as-app-discovery-demands-grow/ (visited on May 13, 2011)
8. The mobile app discovery problem, http://news.cnet.com/8301-30684_3-20011241-265.html (visited on May 13, 2011)
9. Google finally applies its own search technology to apps, http://www.zdnet.com/blog/mobile-gadgeteer/google-finally-applies-its-own-search-technology-to-apps/3332?tag=mantle_skin;content (visited on May 13, 2011)
10. App search engine competition heating up, http://www.zdnet.com/blog/mobile-gadgeteer/app-search-engine-competition-heating-up/3785 (visited on May 13, 2011)
11. Popular demand: App wars, http://www.nytimes.com/interactive/2011/05/16/business/media/16most.html?ref=media (visited on May 13, 2011)
12. Blackberry app world, http://us.blackberry.com/apps-software/appworld/ (visited on May 13, 2011)
13. Windows, http://marketplace.windowsphone.com/Default.aspx (visited on May 13, 2011)
14. Heydon, A., Najork, M.: Mercator: A scalable, extensible web crawler. World Wide Web 2, 219–229 (1999), http://dx.doi.org/10.1023/A:1019213109274, doi:10.1023/A:1019213109274
15. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: Ubicrawler: a scalable fully distributed web crawler. Software: Practice and Experience 34(8), 711–726 (2004), http://dx.doi.org/10.1002/spe.587
16. Hsieh, J.M., Gribble, S.D., Levy, H.M.: The architecture and implementation of an extensible web crawler. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, Ser. NSDI 2010, p. 22. USENIX Association, Berkeley (2010), http://portal.acm.org/citation.cfm?id=1855711.1855733
17. Mysql, http://www.mysql.com/ (visited on May 13, 2011)
18. Lucene, http://lucene.apache.org/ (visited on May 13, 2011)
19. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A large ontology from wikipedia and wordnet. Web Semantics: Science, Services and Agents on the World Wide Web 6(3), 203–217 (2008), World Wide Web Conference 2007 Semantic Web Track, http://www.sciencedirect.com/science/article/pii/S1570826808000437

20. Opennlp, `http://incubator.apache.org/opennlp/` (visited on May 13, 2011)
21. Princeton University, About wordnet, `http://wordnet.princeton.edu` (visited on May 17, 2011)
22. Find mobile apps you will love, `http://www.appolicious.com/` (visited on May 17, 2011)
23. Helping you navigate the app world, `http://www.mobilewalla.com/` (visited on May 13, 2011)
24. The appstore search engine, `http://www.chomp.com/` (visited on May 17, 2011)
25. Find the best android apps, `http://www.appbrain.com/` (visited on May 17, 2011)
26. The appstore search engine, `http://www.uquery.com/` (visited on May 17, 2011)