# Tag-Based and QoS-Aware Mobile Application Search and Management

Shang-Pin Ma[1], Jing-Hong Lin[1], Shin-Jie Lee[2], Wen-Tin Lee[3], Jui-Hsaing Lin[1]
Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung, Taiwan[1]
Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan[2]
Department of Software Engineering, National Kaohsiung Normal University, Kaohsiung, Taiwan[3]
albert@ntou.edu.tw, fall1600@gmail.com, jielee@mail.ncku.edu.tw, wtlee@nknu.edu.tw, jason51806@gmail.com

***Abstract*—With the emergence of the huge number of mobile applications (apps) and their increasing growth rate, how to appropriately search, recommend, and manage apps is becoming a critical challenge to enhance the usage of mobile apps for users. Although existing mobile app search engines provide basic search functionalities to let users find relevant mobile apps by issuing query keywords, however, the ranking of search results cannot fully satisfy user's expectations. Besides, personal app management mechanism is required for enhancing the usage of apps, but such mechanisms are lacking. Therefore, we propose an approach, called Tag-based and QoS-aware Mobile Application Search and Management (TQMASM), to address above issues. TQMASM mainly provides two functionalities: (1) QoS-aware app search and tag-based app recommendation; and (2) tag-based app management. QoS-aware app search is an objective ranking method for sorting app search results by considering the proposed QoS factors, popularity and reputation. Tag-based app recommendation is an app retrieval approach to find apps according to tags annotated by all users. Tag-based app management utilizes the hierarchical app tree and tag annotation to ease the management and the usage for interested apps. A prototype that realized TQMASM is also developed to demonstrate the feasibility of the proposed approach.**

***Keywords—App Search; App Recommendation; App Management; Quality of Service; Social Tag***

## I. INTRODUCTION

With the widespread adoption of mobile applications (apps) on mobile devices, the number of mobile applications gains exponential growth. For example, Google Play store has officially announced that there are over 1 million Android apps in 2013 [1]. With the emergence of the huge number of apps and their increasing growth rate, how to appropriately search, recommend, and manage apps is becoming a critical challenge to enhance the usage of mobile apps for users.

Although existing app search engines, such Google Play, iTune Store, or Windows Phone Marketplace, provide basic search functionalities to let users find relevant mobile apps by issuing query keywords, the ranking of search results cannot fully satisfy user's expectations. Namely, in the search results, apps with higher ranks may not be more satisfactory than ones with lower ranks. We argue that current app search engines did not consider QoS (quality of service) for apps. Current common QoS factors include availability, reliability or response time [2], however, for mobile apps, these factors are neither appropriate nor accessible since mobile apps are not pure backend functionalities. Compared with server-side web services, mobile apps are user interactive, necessary to be installed, and inclined to be updated periodically. Thus, in this paper, we propose the notion of popularity and reputation for mobile apps. We define that the popularity as the degree of that an app was utilized by the public to express it long-term performance and the reputation as the degree of that an app can be trusted to emphasize its recent quality. The popularity and the reputation are treated as QoS factors for mobile apps.

Besides, when there are plenty of apps in a user's mobile device, it is also difficult to find required apps, including the apps installed or ones useful but uninstalled. A convenient way to ease personal management and utilization of mobile apps is also required. In this paper, we suggest that the apps can be arranged by users with the directory structure and tag annotation mechanism. Through a customized hierarchical app tree and a set of annotated tags, users can fetch and use the required apps more easily than using the default app list.

Therefore, we propose an approach, called Tag-based and QoS-aware Mobile Application Search and Management (TQMASM), to address above issues. TQMASM mainly provides two mechanisms: (1) QoS-aware app search and tag-based app recommendation and (2) tag-based app management. QoS-aware app search is an objective ranking method for sorting search results. In QoS-aware app search, the user can search for apps through the search form of the TQMASM system. The search results are ranked according to proposed QoS-aware ranking method by aggregating popularity score, reputation score, and Google ranking score. Tag-based app search is an app retrieval approach to find apps according to tags annotated by all users. Tag-based app management utilizes the notion of hierarchical app trees and the tag annotation mechanism to ease the management and the usage for installed apps or apps in which the user is interested but did not installed yet.

The rest of this paper is organized as follows. Section II describes the details of the proposed approach. The design and implementation are shown in Section III. Related work is presented section IV. Section V summarizes the benefits and features of the proposed approach and outlines the future work.

INISCom 2015, March 02-04, Tokyo, Japan

DOI 10.4108/icst.iniscom.2015.258404

## II. Approach Descriptions

In this section, we describe the proposed TQMASM (QoS-aware Mobile Application Search and Management) approach in depth.

### A. Tag-Based App Tree Management

For the better management of mobile apps, we utilize the mechanisms of tags and categories, which are often applied to the text retrieval systems, to enhance the usability. The user can manage mobile apps via a hierarchical category structure, called "app tree" in this research. The user can attach an app into a given category, no matter the app is installed or not. A category is allowed to attach other categories to form the hierarchical structure. For example, the user can put the apps "Facebook" and "Line" in a category named "Social Network". And the "Social network" category can be put in the "Life" category. Besides, an app can be put into multiple categories if the user thinks it is convenient to find and browse apps.

When the user wants to describe an app via more specific terms, she can add tags to the app. For instance, she can put "personal", "note" or "inspiration" to the app "Evernote". If she wants to find Evernote on her mobile device, she can input keywords "personal", "note" or "inspiration" to find out and use this app.

Furthermore, the app tree is portable across multiple mobile devices and desktops. The user can manage apps on the desktop and can browse her app tree on multiple mobile devices. In other words, the user can edit her app tree in the large-sized screen of her desktop, and browse her app tree and use her apps via the app tree in her smartphone. Besides, the user can also find her apps by inputting annotated tags.

### B. QoS-Aware and Tag-Based App Search

As above mentioned, we devise a method to evaluate the quality of service (QoS) for an app based on its popularity and reputation. Then, we rank the retrieved apps from the Google Play by the calculated QoS score to provide a more reasonable ranking for search results. In the followings, we explain the popularity score, the reputation score and the Google ranking score in detail.

#### 1) Popularity Score (PS)

In general, users prefer to choose apps which were installed many times or have been updated recently [3]. Thus, to integrate these two indicators, we utilize the number of installations and the last update time to determine the popularity score by the fuzzy theory [4]. It is noted that the popularity score of an app can represent both the long term performance as well as the freshness of this app.

For the viewpoint of popularity, our goal is to estimate the QoS score based on two indicators: period from update date (PUD) and times of installation (TI). To obtain reasonable score, we apply the fuzzy theory to calculate the popularity score. In the calculation, the first step is to convert the crisp sets into fuzzy ones, so we define fuzzy sets of PUD and TI separately. We designate five fuzzy sets PUD1, PUD2, PUD3, PUD4 and PUD5 to averagely represent five ranges in 24 months, and five fuzzy sets TI1, TI2, TI3, TI4, and TI5 to fairly allocate the 18 ranges in Google Play.

The operations in fuzzy reasoning include Intersection (see Equation (1)) and Union (see Equation (2)). To produce the popularity score, we first use the Intersection operation to compose the TI sets and PUD sets according to the aggregation rules shown in Figure 1. The fuzzy sets VL (very low), L (low), M (medium), H (high), and VH (very high) indicate different ranks for the popularity. If there are more than one membership functions for any popularity fuzzy set, we use the Union operation to produce a single fuzzy degree.

$$\mu A \cap B(x) = min[\mu A(x), \mu B(x)] \quad (1)$$

$$\mu A \cup B(x) = max[\mu A(x), \mu B(x)] \quad (2)$$

| | TI1 | TI2 | TI3 | TI4 | TI5 |
|---|---|---|---|---|---|
| PUD5 | VL | VL | L | L | M |
| PUD4 | VL | L | M | M | H |
| PUD3 | L | L | H | H | H |
| PUD2 | M | M | H | VH | VH |
| PUD1 | H | H | VH | VH | VH |

VL: very low; L: low; M: medium; H: high; VH: very high

Figure 1. The aggregation rules for TI and PUD

After aggregating the TI and the PUD, we get the fuzzy degrees that represent the popularity of this app. Then, we use the center of gravity (COG) method to perform the defuzzification [4] to get a crisp number to represent the popularity (as shown in Equation (3)).

$$\mathrm{PS(N)} = \mathrm{COG} = \frac{\sum_{x=a}^{b} \mu A(x)x}{\sum_{x=a}^{b} \mu A(x)} \quad (3)$$

We use two apps as examples to illustrate our popularity scoring method. The PUD and TI of the first app are 6 months and 1,000,000 ~ 5,000,000, and the PUD and TI of the second are 12 months and 100,000,000 ~ 500,000,000.

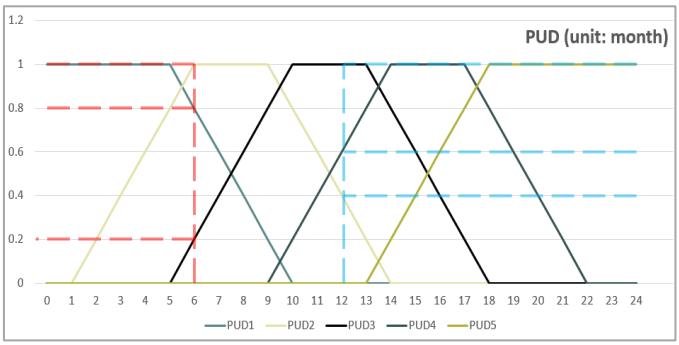


Figure 2. The fuzzy sets of PUD (period from update date)

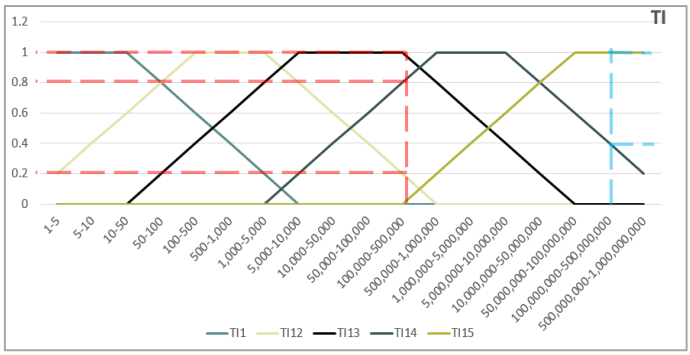


Figure 3. The fuzzy sets of TI (times of installation)

Figure 2 illustrates that the PUD membership functions of the two apps. In this example, the dotted red line is the first app with 0.8 membership degree for PUD1, 1.0 for PUD2, and 0.2 for PUD3. The dotted blue line is the second one with 0.4 membership degree for PUD2, 1.0 for PUD3, and 0.6 for PUD4.

Figure 3 depicts that the TI membership functions of the two apps. In this example, the dotted red line is the first app with 0.2 membership degree for TI2, 1.0 for TI3, and 0.8 for TI4. The dotted blue line is the second one with 0.4 for TI4, and 1.0 for TI5.

In Figure 1, the cells in red color are the parts that the first app is applied, and the cells in blue are the parts that second one is applied. The cells in purple are ones both two apps are applied. For example, in the first app, for calculating the membership degree of popularity set H, the intersections of membership degree pairs are calculated first: "TI3 and PUD3", min[1.0, 0.2] = 0.2; "TI4 and PUD3", min[0.8, 0.2] = 0.2; "TI3 and PUD2", min[1.0, 1.0] = 1.0; "TI4 and PUD2", min[0.8, 1.0] = 0.8; and "TI2 and PUD1", min[0.2, 1.0] = 0.2. Then we apply the union operation: max[0.2, 0.2, 1.0, 0.8, 0.2], to derive that the membership degree of H is 1.0. For the second app, the intersection of membership degree pairs are calculated first: "TI5 and PUD4", min[1.0, 0.6] = 0.6; "TI4 and PUD3", min[0.4, 1.0] = 0.4; "TI5 and PUD3", min[1.0, 1.0] = 1.0. Then we also apply the union operation: max[0.6, 0.4, 1.0], to derive that the membership degree of H for the second app is also 1.0.

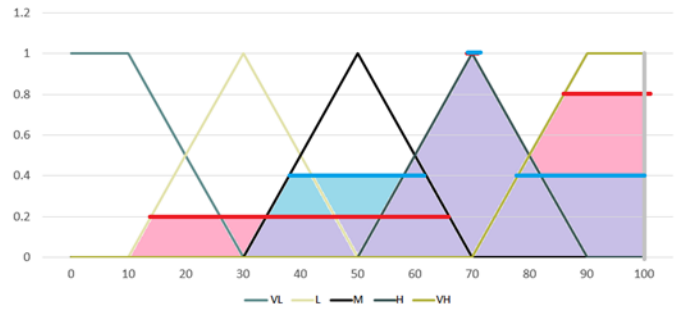


Figure 4. The popularity score

Figure 4 represents the fuzzy popularity scores of the two apps. The area with pink and the area with purple represent the score of the first app with 0.2 L, 0.2 M, 1.0 H, and 0.8 VH. The area with blue and the area with purple represent the score of the second one with 0.4 M, 1.0 H, and 0.4 VH.

By applying the center of gravity (COG) defuzzification method, we can calculate the popularity score of the first app is 69.91 (pink and purple parts), and the second one is 67.80 (blue and purple parts). Although the TI score of the second one is about 100 times than the first one, it has not been updated one year ago. It is the reason why the second one gets worse popularity score than the first one. Notably, if an app does not keep the freshness, it is very possible that it does not meet users' current requirements or does not fix some defects.

*2) Reputation Score (RS)*

The reputation of an app is also an important concern when users search for apps in the app repository. In this research, we emphasize recent ratings of an app more than it past performance to evaluate the reputation of an app, Thus, The reputation score can avoid finding apps with averagely good but recently bad performance.

In this research, we apply the REAL (Risk-Enabled Reputation Model) method [5] to evaluate a mobile app's reputation. There are three steps of applying REAL: Step 1 is to record ratings of an app, as the basic ingredient of the reputation model. Step 2 is to discover active ratings that can be used to calculate the reputation of the app by the next step. And step 3 is to calculate reputation of the app.

By applying REAL, the recent bad ratings of an app can be detected as the active ratings. Thus, we can derive a reasonable score to represent the reputation of an app, not only the average rating score. For example, the rating of the Facebook app on Google play was 3.8 on June 10, 2014, and its reputation score was 3.31. The reason why Facebook gets worse reputation score than it average rating is because that it gets bad ratings since June 2014 due to its inconvenient user interface of the new published version. In summary, the reputation score can avoid that past high ratings shadow the recent bad performance and can produce a more reasonable score to represent the app's real reputation.

*3) Google Ranking Score (GRS)*

Although the popularity and the reputation score can determine the quality of an app, the relevance between the query term and the retrieved apps is also important in app search. Because Google Play has achieved a better precision of retrieving apps based on the query term but provided inappropriate rankings from the view of popularity and reputation, we apply the Google ranking as the basic score and integrate the proposed popularity score and reputation score with the basic score to enhance the satisfaction of search results.

We define Google Ranking Score (GRS) as below. First, we set two constants (MAX_SCORE: 90 and RANK_GROUP: 3). Then we set a variable called *reduction*. The calculation of *reduction* is as Equation (4). Note that the value of (N / RANK_GROUP) is rounded down to the integer. The GRS is calculated according to Equation (5).

$$\text{reduction}(N) = 0.3 \times \left(1 + \frac{N}{\text{RANK\_GROUP}}\right) \quad (4)$$

$$\text{GRS}(N) = \text{GRS}(N-1) - \text{reduction}(N) \quad (5)$$

The rationale of integrating the concept of reduction is because we observe that apps with high ranks (such as rank 1~6) are usually highly related to the query keyword whereas apps with low ranks (such as rank 25~30) are probably totally unrelated to the keyword. Thus, we define the *reduction* score as the score which should be subtracted. Basically, the lower the rank, the higher the *reduction* score. According to the definition, the first ranked app in Google Play has no reduction, so it will get score 90 (MAX_SCORE). The reduction of $N^{th}$ app will be 0.3 * (1 + (N / 3)), and its score is the score of the previous score subtracting this reduction score. For example, the reduction of the second one is 0.3, so the score of this app is 89.7. Regarding another example, the score of the 29th app is 43.8, and the reduction of 30th app is 3, so the score of 30th app is 40.8.

*4) Aggregating Popularity Score and Reputation Score into the QoS Score*

By examining both the long term and recent performance, TQMASM provide a reasonable ranking of the search results for a given query. The QoS score of a mobile app *a* is calculated by aggregating popularity score, reputation score and Google ranking score based on weights *w1*, *w2 and w3* (shown in Equation (6)).

$$\text{QoS}(a) = w1 * \text{PS}(a) + w2 * \text{RS}(a) + w3 * \text{GPS}(a) \quad (6)$$

### C. Tag-Based App Recommendation

In addition to the proposed QoS-aware app search, we also provide an app recommendation mechanism based on annotated tags from all users. The basic idea is that if the user is willing to put an app into her app tree, then the app is worth to recommend.

To realize the recommendation functionality, we use the commonly used information retrieval technique, TF-IDF (term frequency-inverse document frequency), to rank apps based on extracted indices. In order to build tag indices, TQMASM gathers all tags and category names from the app trees of all users and store them as tag indices. TF (term frequency) means the importance of a tag index for an app. TF is obtained by calculating the proportion of the amount of a tag index for an app to the total amount of all tag indices for an app. IDF (inverse document frequency) indicates the distinguishability of a tag for all apps. IDF is obtained by calculating the proportion of the total number of apps to the number of apps annotated with this tag, and taking the logarithm of the proportion. The TF-IDF value is calculated by multiplying TF and IDF.

Figure 5 is the table that displays the TF-IDF information of two illustrative apps. There are 205 apps in the app repository. The app #1 is "Real Raohe Night Market" ("正港饒河夜市" in Chinese), and app #2 is "Night Market Plan" ("夜市通" in Chinese). The first candidate app was tagged "Raohe" ("饒河" in Chinese, the name of the Night Market) 3 times, "Night Market" 11 times, "Songshan" ("松山" in Chinese, a region name) 6 times, and "Wu Fen Pu" ("五分埔" in Chinese, a place name) 6 times, and the second one "Night Market Plan" was tagged "Night Market" 4 times and "Taiwan" 1 times. There are 2 apps annotated with "Raohe", 9 apps with "Night Market", 6 apps with "Songshan", 1 apps with " Wu Fen Pu", and 6 apps with "Taiwan".

| 1 | Raohe | Night Market | Songshan | Wu Fen Pu |
|---|---|---|---|---|
| TF | 3/22 | 11/22 | 6/22 | 6/22 |
| IDF | log(205/2) | log(205/9) | log(205/6) | log(205/1) |

| 2 | Night Market | Taiwan |
|---|---|---|
| TF | 4/5 | 1/5 |
| IDF | log(205/9) | log(205/5) |

Figure 5. The TF-IDF score of candidate apps

Issued query terms are "Raohe", "Night Market", "Wu Fen Pu", "Songshan", and "Taiwan". The detailed data of TF and IDF are shown in Figure 5. The final TF-IDF scores of these two candidate apps are 1.47 and 1.41 separately.

$$Score1 = \log(205/2) * (3/30) + \log(205/9) * (11/30) + \log(205/6) * (6/30) + \log(205/1) * (6/30) = 1.47$$

$$Score2 = \log(205/9) * (4/5) + \log(205/5) * (1/5) = 1.41$$

## III. Design and Implementation

The architecture of the prototype system that realizes TQMASM is shown in Figure 6. The prototype is developed using Android (mobile client-side), HTML and jQuery (web client-side), and Java Servlet technology (server-side). The app data are extracted from Google Play.

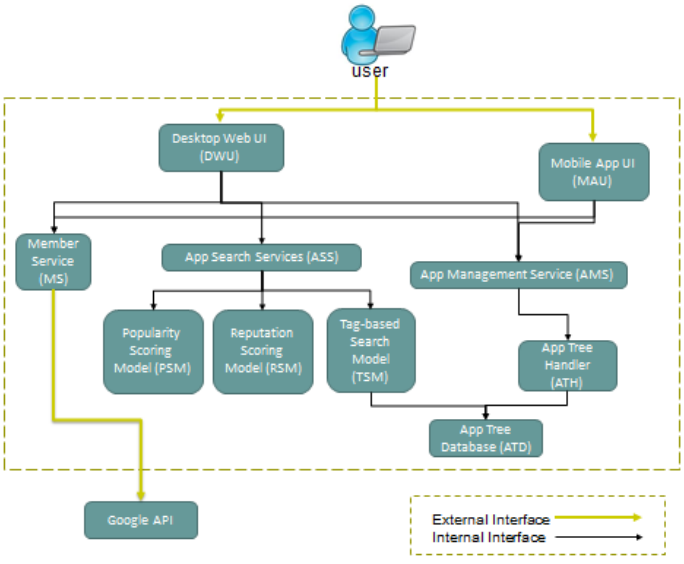


Figure 6. System architecture of the implemented prototype

In the architecture, Mobile App UI (MAU) is the client-side module that provides all required user interfaces in the

mobile device. Users can obtain their own app tree and utilize apps on the MAU. App Manage Service (AMS) is the controller to invoke App Tree Handler. App Tree Handler creates and modifies the app tree and tags. App Tree Database contains the category and tag data of all users. Desktop Web UI (DAU) is the user interface that provides complete functionalities to allow search or manage apps. App Search Services (ASS) are the controllers to coordinate Popularity Scoring Model (PSM), Reputation Scoring Model (RSM), or Tag-based Search Model (TSM). RSM generates a score of an app from its recent ratings. PSM produces a score of an app from its TI (times of installation) and PUD (period from update date). TSM builds tag indices and calculate TF-IDF scores to recommend apps. Member Service (MS) lets user to log in our system.

The implementation is realized as a Web application and hosted in a desktop computer with the following configuration: Intel Core 3.07GHz with 8G RAM, 500G hard disk, and Windows 7 (64bit).

Figure 7. The system prototype: QoS-aware search

For the usage of app search, we prepare a simple case to illustrate the search functionality. In this case (see Figure 7), the user wants to search for apps about building muscle, she may enter "workout" to find suitable apps. According to the ranked search results, we focus on two apps with ranks changing largely: (1)"Workout Trainer" ("健身教練" in Chinese) has a higher rank (from rank 13 to rank 5), and (2) "Workout Expert" ("健身專家" in Chinese) has a lower rank (from rank 1 to rank 13). Comparing these two apps, the first app is fresher and is with more times of installation than the second one. For the actual functionalities of both apps, the first one can arrange workout classes and provide different videos to meet users' needs whereas the second one supplies rich static data about workout but has less interactive functions. The new ranks for these two apps are more reasonable than the original ranks.

## IV. RELATED WORK

In this section, we review several studies related to our approach and compare our approach with these efforts.

### A. App Search Methods Based on App Usage

Shi and Kamal [6] and Yan et al. [7] presented app search approaches based on the app usage. They believe that only using the public app information on the App Marketplace to retrieve apps may is inefficient. For example, (1) it is common that only a few users keep rating records to apps; (2) The ratings may be unfair due to fake records; and (3) The ratings will be obsolete in a short period.

Yin et al. [8] recorded a large number of contest information between the apps in user's device and the new app. And the authors devised an AT (actual value and tempting value) model to predict the possibility about whether the user would install the new app or not. Karatzoglou at al. [9] proposed a Djinn model based on the context parameters (such as time and location) to perform collaborative filtering method for implicit data based on tensor factorization.

Although app-usage-based methods can retrieve apps according to private personal data, in our research, we aim to build an app search engine without violating users' privacy. Instead of using the private data like history of app usage or context, TQMASM is based on public app data, such as app meta data and social network, or the public tags from all users. Besides, to avoid the bias of app data, we focus on both long-term quality (popularity) and recent performance (reputation) to enhance the data validity.

### B. App Search Methods Based on App Information

Datta [3] utilized the public app information to search apps. The author distinguished the information into static information and dynamic information. The former are app name, description, date of update, price, type, developer name, language and the app size. The latter are comments, ratings, version number, ranking and times of installation. The score for sorting apps is calculated by the ranking of the app in its main category and the average scores of the same developer's other apps.

Different from the existing methods, TQMASM emphasizes the popularity and the reputation of an app to re-rank search results based on the proposed objective QoS score.

### C. Tag and Category

Social bookmarking is a mechanism to enable users to add, annotate, edit, and share bookmarks of web documents. The folksonomy [10] indicates the mechanism to classify tags built up by the action of user tagging to produce a user generated taxonomy as opposed to an authoritative hierarchical taxonomy. Noll et al. [11] uses techniques of social bookmarking and tagging to re-rank web search results. Karlson et al. [12] provided a method to solve the problem of the navigating search results in the mobile device. The authors devised a hybrid model to perform iterative data filtering by the navigation and selection of hierarchical category (facet navigation) with incremental text entry to further narrow the search results.

In TQMASM, motivated by the social tag mechanism, we furnish a tag-based app management method for users to classify and utilize their apps based on hierarchical categories and tags.

## V. CONCLUSION

In this paper, we propose an approach, called Tag-based and QoS-aware Mobile Application Search and Management (TQMASM). There are three main features of TQMASM:

- An objective ranking method is devised to consider popularity and reputation in mobile app search.
- A tag-based app retrieval mechanism is also provided to find mobile apps according to tags annotated by all users.
- A category-based app management mechanism is built to ease the management and the usage of mobile apps.

Our future research plans include (1) devising an automatic or semi-automatic tag annotation method to lower the users' effort; (2) analyzing the app tree periodically to recommend appropriate newly-published mobile apps to users actively; and (3) integrating the QoS-aware app search approach with the tag-based recommendation method to further improve the precision and satisfaction degree of app search.

## Acknowledgements

This research was sponsored by Ministry of Science and Technology in Taiwan under the grant MOST 103-2221-E-019-039 and 103-2221-E-006-218.

## REFERENCES

[1] *Android's Google Play beats App Store with over 1 million apps, now officially largest*. 2013; Available from: http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

[2] S.-P. Ma, C.-L. Yeh, and P.-C. Chen, *Service Composition Management: A Risk-Driven Approach.* Journal of Universal Computer Science, 2014. **20**(3): p. 302-328.

[3] A. Datta, S. Kajanan, and N. Pervin, *A Mobile App Search Engine.* Mob. Netw. Appl., 2013. **18**(1): p. 42-59.

[4] X. Liu, *Parameterized defuzzification with maximum entropy weighting function-Another view of the weighting function expectation method.* Math. Comput. Model., 2007. **45**(1-2): p. 177-188.

[5] J. Lee, S.-J. Lee, H.-M. Chen, and C.-L. Wu, *Composing web services enacted by autonomous agents through agent-centric contract net protocol.* Information and Software Technology, 2012. **54**(9): p. 951-967.

[6] K. Shi and K. Ali, *GetJar mobile application recommendations with very sparse datasets*, in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, ACM: Beijing, China. p. 204-212.

[7] B. Yan and G. Chen, *AppJoy: personalized mobile application discovery*, in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. 2011, ACM: Bethesda, Maryland, USA. p. 113-126.

[8] P. Yin, P. Luo, W.-C. Lee, and M. Wang, *App recommendation: a contest between satisfaction and temptation*, in *Proceedings of the sixth ACM international conference on Web search and data mining*. 2013, ACM: Rome, Italy. p. 395-404.

[9] A. Karatzoglou, et al., *Climbing the app wall: enabling mobile app discovery through context-aware recommendations*, in *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2012, ACM: Maui, Hawaii, USA. p. 2527-2530.

[10] S. Hayman. *Folksonomies and tagging: New developments in social bookmarking*. in *Ark Group Conference: Developing and Improving Classification Schemes*. 2007.

[11] M.G. Noll and C. Meinel, *Web search personalization via social bookmarking and tagging*, in *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*. 2007, Springer-Verlag: Busan, Korea. p. 367-380.

[12] A.K. Karlson, G.G. Robertson, D.C. Robbins, M.P. Czerwinski, and G.R. Smith. *FaThumb: a facet-based interface for mobile search*. in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006: ACM.