To Run or Not to Run: Predicting Resource Usage Pattern in a Smartphone

Arijit Mukherjee[1(✉)], Anupam Basu[2], Swarnava Dey[1], Pubali Datta[1], and Himadri Sekhar Paul[1]

[1] Innovation Labs, Tata Consultancy Services, Kolkata, India
mukherjee.arijit@tcs.com
[2] Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, India

Abstract. Smart mobile phones are vital to the Mobile Cloud Computing (MCC) paradigm where compute jobs can be offloaded to the devices from the Cloud and vice-versa, or the devices can act as peers to collaboratively perform a task. Recent research in IoT context also points to the use of smartphones as sensor gateways highlighting the importance of data processing at the network edge. In either case, when a smart phone is used as a compute resource or a sensor gateway, the corresponding tasks must be executed in addition to the user's normal activities on the device without affecting the user experience. In this paper, we propose a framework that can act as an enabler of such features by classifying the availability of system resources like CPU, memory, network usage based on applications running on an Android phone. We show that, such app-based classifications are user-specific and app usage varies with different handsets, leading to different classifications. We further show that irrespective of such variation in classification, distinct patterns exist for all users with available opportunity to schedule external tasks, without affecting user experience. Based on the next to-be-used applications, we output a predicted set of system resources. The resource levels along with handset architecture may be used to estimate worst case execution time for external jobs.

Keywords: Smart phone · Usage prediction · Resource utilisation · Machine learning · Mobile cloud computing · IoT · Sensor data

1 Introduction

With the advancement of technology more people are using high end mobile phones with increased hardware capabilities. Though such phones are being used for functionalities more than just communication, they remain idle for majority of the time. One of the focus areas of Mobile Cloud Computing (MCC) is to explore possibilities of utilizing mobile phones as compute resources to augment the Cloud infrastructure. In [1], jobs are executed on mobile phones using a map-reduce framework. Authors in [2,3] propose offloading jobs between mobiles where the phones act as compute peers. More recently, in the context of Internet of Things (IoT), the idea of edge devices being used to precess data at

R. Giaffreda et al. (Eds.): IoT360 2014, Part I, LNICST 150, pp. 367–375, 2015.
DOI: 10.1007/978-3-319-19656-5_49

the network edge was introduced in Fog Computing [4] and the present authors have implemented a preliminary prototype which is outlined in [5]. The authors believe that in a *smart city* scenario, with numerous sensors and edge devices emitting highly fluctuating volume of data, an MCC framework augmenting the cloud may indeed be useful. Recent works by McCann *et al.* have also pointed towards the use of personal devices as sensor gateways and data-forwarding entities [6,7]. We are of the belief that during such phases, the devices may also be utilised for small amounts of computation in order to reduce the cost incurred in pushing the data to the cloud and in turn save energy (as apart from incurring a communication cost, any network transfer is energy intense). However, to effectively schedule jobs on available phones and other edge devices, the cloud servers would require an estimation of available cpu, memory and other parameters for those devices as the execution time of an externally assigned computation will vary depending on the foreground(user) and background activities running on the phone. The same is true if the devices are used as sensor gateways. This implies that the user experience must not get affected while executing the assigned job or transfering sensor data to the cloud. In other words, it can be said that the user experience is not affected if such tasks are scheduled at the right time, when the device is relatively idle. Figure 1 explains the context described herein.

Fig. 1. Using mobile phones in an IoT context

A number of studies have attempted to classify android applications based on android package details, power consumption and static code analysis. Commercial and free benchmark tools have also been used to measure relative performance of a handset and profile apps. The diversity of smartphone app usage behavior among users was highlighted in [8,9]. These approaches showed that unique usage patterns exist, albeit on a per user basis. With this idea of *user specific patterns*, we propose to predict system resources influencing the performance of an android mobile, based on the currently running android apps, per user.

Our work focuses on analyzing the CPU, memory etc. resource usage, when apps are running on the devices to detect phases when the system resources are relatively free for external job execution. We classify CPU and RAM based on android apps used on a handset by the user of the handset by analysing the log containing running apps snapshots, overall idle CPU time, available memory etc. using machine learning techniques and decide whether *to run* or *not to run* any externally assigned task. We also show the variation of the classification results with user and handset and present the result of our field-study showing the correlation of our prediction with benchmark scores from a well known benchmark tool, AndEbench [10].

The rest of the paper is organized as follows. We analyze the previous work done in android app classification and android benchmarking in Sect. 2. In Sect. 3, we describe our approach regarding data collection, preprocessing, feature selection classifier selection and field study. We present the results in Sect. 4 and conclude with a summary of the contributions and some pointers for future work.

2 Related Work

Not many systems exist for classifying smart-phone system resources based on app usage. A number of research works have focused on classifying android apps based on android package details, power consumption and static code analysis. The main focus of such analysis is to segregate apps from malwares. Several commercial tools benchmark system resources at both app and handset levels.

Zefferer *et al.* [11] presented a scheme of malware detection by classifying android apps based on power consumption. They found that the power-consumption signature for a given application or phone state could not be determined uniquely and the signature for the same app was analogous to wide pitch and frequency variance of the different speech records from the same person. Sanz *et al.* [12] developed a new app classification scheme using extracted features from said app and the Android Market. They worked with a large set of 820 apps categorizing them into seven categories by using classification techniques and providing a comparative evaluation using the Area Under ROC Curve (AUC). Shabtai *et al.* [13] focussed on app classification using framework methods and classes used by the app, user interface widgets etc. and identified the optimum combination for feature selection method, top features selected and the classifier.

Several commercial tools are available for benchmarking. Notable among these is the Trepn Profiler [14] from Qualcomm which provides system or app specific cpu profiling. AndEBench is another tool that we used extensively in this work and it shows a native and java score for each phone. However, none of the benchmark tools however categorize apps based on the system resource usage or provide a relative scoring for each app. Our work aims to classify system resources based on android apps per user per phone, leveraging the unique usage pattern.

3 Approach

A logging application for android devices was deployed on the mobile phones of several users which was used to gather data over a period of two weeks for each user. This application gathers last app, last service component, data transfer and memory available using android APIs. For system CPU usage and process details, the system parses the `top` command output which outputs processes like `system_server`, `uevent` and several other system activities that are not available using android APIs. As the logger logs data in a very precise form, with only the required values for our analysis, the log file size (at most 2MB in two weeks) is never a concern for the volunteers.

To determine the cpu availability, we used the jiffy values from android `top` output and calculated the percentage of time the cpu was idle. For the two class classification (in this case, *high* and *low*), we applied *k-means clustering technique* [15] on the idle cpu percentage values. For multiprocessor systems, `top` provides a measure of summation over (number of cores x percentage utilized in each core). We scaled the overall value for all cores by dividing it by the number of cores for that phone System On Chip (SoC). We collected available memory information using the `getMemoryInfo` API of android *ActivityManager* and the system memory information from `/proc/meminfo`. An equally weighted average of the two values was used to express the memory free percentage. Similar to the cpu values, k-means clustering was applied on the free memory percentage values, to create two clusters *high* and *low*.

One pertinent note at this point may be that - out of a myriad of available android applications (as per [16] the latest number is 1175286) we are classifying on the basis of only a small subset. To justify our approach, it may be said that as we apply our system on a per user basis, the applications usually running on the phone of the user determines the classification of system resources. The analysis using Principal Component Analysis (PCA) and the ranker algorithm also proves that only a subset of all the apps installed in a phone have any visible effect on the resources as is shown in Table 1, which lists the top-6 ranked features for two different users using two different handsets.

Table 1. Table of top ranked features in phones A110q and Xperia L

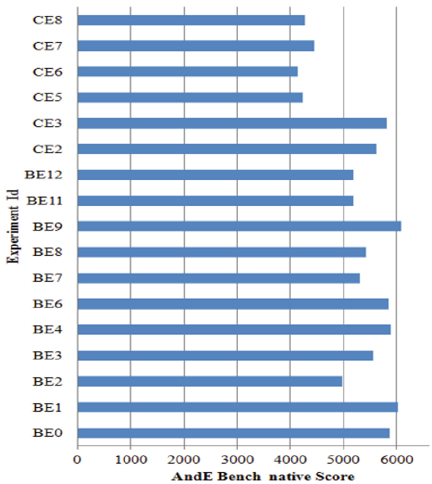
A110 top features	Xperia L top features
surfaceflinger	surfaceflinger
mediaserver	system_server
mediatek.bluetooth	mediaserver
android.chrome	king.candycrushsaga
android.systemui	textinput.uxp
android.youtube	android.systemui

We used *machine learning* concepts to depict the dependency between application running in a phone and the level of the available system resources. However, we haven't yet implemented a full-fledged online app prediction system and rather have taken cues from [17] to create a Naïve Bayes classification of offline data from the mobile phones on which we evaluated the current system. We built a prediction model using the WEKA [18] tool and evaluated using test data for top four apps being used in the system, during a 5 second interval. To evaluate our system on real mobile phones we used the *Weka-for-Android* [19] implementation for the Naïve bayes classifier along with the offline model created for that phone using desktop WEKA. With the next four running app prediction at hand along with all other features required for system resource classification, we used a modification of the *LibSVM-androjni* [20] project to run our Logistic Regression classifier. We chose the Logistic Regression classifier for the field study as an android port was easily available and it performed reasonably well, as detailed in the Sect. 4. We mapped the output of the classifier (system resource level high or low) to our final decision - *to run* or *not to run* the external task.

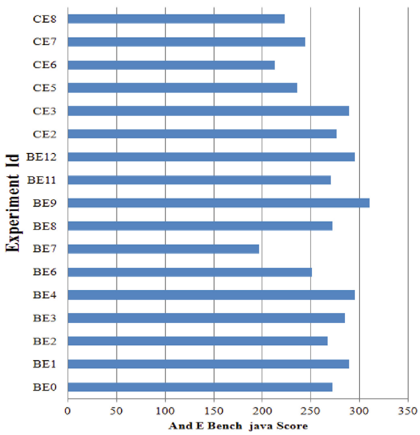
As the MCC frameworks ANGELS [5] is still under development, we decided to use a different innovative measure to evaluate the output. We designed a set of experiments (a snapshot of which is given in Fig. 2a) to obtain a correlation between the AndEBench score and the underlying activities. In each experiment,

ID	Activity List	ID	Activity List
BE2	1. Sanitize 2. Play video with Mx Player	BE4	1. Sanitize 2. Play audio with stock music player
BE11	1. Sanitize 2. Run Sygic Navigator to navigate	CE3	1. Sanitize 2. Connect bluetooth headset
CE2	1. Sanitize 2. Connect bluetooth headset 3. Play audio using PlayerPro	CE1	1. Sanitize 2. Connect bluetooth headset 3. Play video using Mx Player

(a) Test Scenarios



(b) Native score variation



(c) Java score variation

Fig. 2. Experimental scenario and AndEBench scores for native & Java on A110q

the *sanitize* step kills user processes, cleans the cache from task manager, starts AndEBench, performs test scenarios and finally records the scores. We observed this and triggered a run of AndEBench, based on recommendation output from the system resource classifier. As a single AndEBench run takes around one minute, we kept the prediction cycles separated by five minutes for evaluation. Our aim was to correlate the prediction of system resource level for the next cycle to the score from AndEBench in the next cycle. For a *high* level for system resource prediction, if the score of AndEBench is also high, we considered the prediction to be accurate. The apparent correlation between the benchmark score and underlying activities for a snapshot of the experiments is shown in Fig. 2b and c.

4 Results

We used two sets of comparisons to differentiate the classifiers the area under the ROC curve (AUC), as recommended in several literatures including [21] and traditional error rate based measure as suggested in [22]. We included the latter keeping in mind the drawbacks of AUC, highlighted in [23]. The results are shown in Tables 2a and b[1] from where it can be seen that as per the AUC measure Logistic Regression and Random Forest performed best for both the

Table 2. Result from classification algorithms

Classifier	AUC A110	AUC Xperia L
Bayesian networks (K2)	0.921	0.948
J48	0.869	0.944
Naïve Bayes	0.911	0.944
Random Forest	0.937	0.955
SVM	0.838	0.79
Logistic	0.937	0.957

(a) Comparison of AUC measures on different phone data

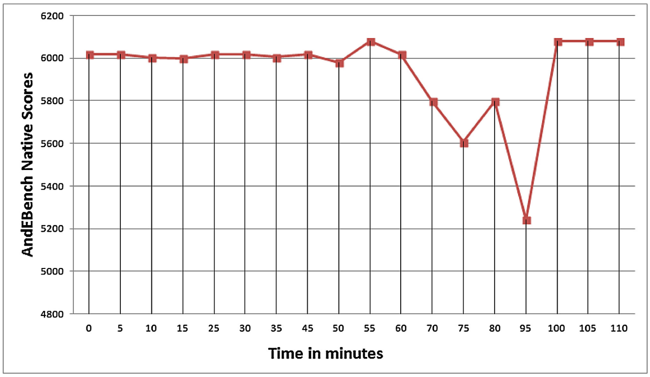
Error	A110Q				Xperia L			
Measure	RMSE	MAE	RRSE	RAE	RMSE	MAE	RRSE	RAE
BN	0.2825	0.1262	71.9829	40.6613	0.3316	0.1394	85.9909	46.8816
J48	0.2639	0.1359	67.2456	44.1284	0.2485	0.1207	64.4535	40.5967
NB	0.2981	0.1279	75.9787	41.5427	0.4303	0.3209	86.1084	64.2394
RF	0.2526	0.1147	63.8559	37.2491	0.2467	0.1147	63.9711	38.5694
SVM	0.2506	0.1147	63.855	37.2491	0.3155	0.0995	81.825	33.4757
L	0.2534	0.1252	63.9517	64.5846	0.2466	0.1195	63.951	40.1906

(b) Comparison of error rates of different classifiers on A110Q and Xperia L

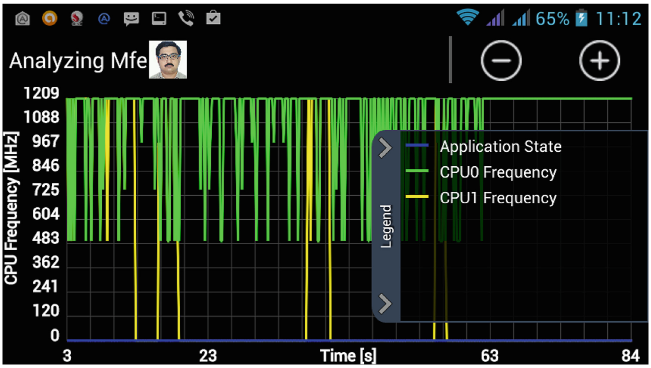
[1] RMSE: Root mean-squared error, MAE: Mean absolute error, RRSE: Root relative squared error, RAE: Relative absolute error.

phones. On the other hand, based on the error rate measure Random Forest and SVM gave better results than all other classifiers for A110q phone. For the Xperia L phone Random Forest, SVM and Logistic Regression performed well. During the classification effort, we also observed from the training data that the system resource availability level is *high* 81 % of time in the A110q phone and 82 % of the time in Xperia L phone. Thus we are able to observe distinct patterns (from classification accuracy) for both the users with available opportunity to schedule external jobs, without affecting user experience.

As stated before in Sect. 3, we triggered a run of AndEBench after a 5 min interval based on the recommendation from our system resource prediction system. The actual run happens only when the system resources are classified as high. We present a snapshot run in Fig. 3a to depict the correctness of recommendations. We also profiled our own app using the Trepn profiler [14] and the result is given in Fig. 3b which shows a fairly good performance measure, although we will consider the optimization of this prediction app as a future work.



(a) A sample run of the AndEBench tool for our evaluation scheme



(b) An execution profile of our system using Trepn profiler

Fig. 3. Results of the prediction system

5 Conclusion

In this work we have addressed the issue of predicting the available system resources in the face of a set of apps to be executed. We have applied Logistic regression to classify the availability of resources. We have further showed that the resultant classification correlates with the scores from a well known benchmark tool. The major contribution of the work is the demonstration of the efficacy of the classification approach to predict the resource availability. This can be fruitfully employed while selecting mobile phones where MCC based jobs can be executed in a *smart city* context. As a sidebar of this research we have also found that android system and background tasks are particularly useful in predicting the resource availability and those can be predicted using data from android phones and past execution history of such tasks.

References

1. Marinelli, E.E.: Hyrax: Cloud Computing on Mobile Devices using MapReduce (2009)
2. Shi, C., Ammar, M.H. Zegura, E.W., Naik, M.: Computing in cirrus clouds: the challenge of intermittent connectivity. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, New York, NY, USA, pp. 23–28. ACM (2012)
3. Shi, C., Lakafosis, V., Ammar, M.H., Zegura, E.W.: Serendipity: enabling remote computing among intermittently connected mobile devices. In: Proceedings of the 13th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2012, New York, NY, USA, pp. 145–154. ACM (2012)
4. Bonomi, F., Milito, R., Zhu, J., Addepalli, S:. Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, New York, NY, USA, pp. 13–16. ACM (2012)
5. Mukherjee, A., Paul, H.S., Dey, S., Banerjee, A.: Angels for distributed analytics in IoT. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 565–570. IEEE (2014)
6. Adeel, U., Yang, S., McCann, J.A.: Self-optimizing citizen-centric mobile urban sensing systems. In: 11th International Conference on Autonomic Computing (ICAC 2014), Philadelphia, PA, pp. 16–1167. USENIX Association, June 2014
7. Yang, S., Adeel, U., McCann, J.: Selfish mules: social profit maximization in sparse sensornets using rationally-selfish human relays. IEEE J. Sele. Areas Commun. **31**, 1124–1134 (2013)
8. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in smartphone usage. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys 2010, New York, NY, USA, pp. 179–194. ACM (2010)
9. Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., Venkataraman, S.: Identifying diverse usage behaviors of smartphone apps. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, IMC 2011, New York, NY, USA, pp. 329–344. ACM (2011)
10. An EEMBC Benchmark for Android Devices. http://www.eembc.org/andebench/

11. Zefferer, T., Teufl, P., Derler, D., Potzmader, K., Oprisnik, A., Gasparitz, H., Hoeller, A.: Power Consumption-based Application Classification and Malware Detection on Android Using Machine-Learning Techniques (2009)
12. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G.: On the automatic categorisation of android applications. In: CCNC, pp. 149–153. IEEE (2012)
13. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: 2010 International Conference on Computational Intelligence and Security (CIS), pp. 329–333, December 2010
14. Trepn Profiler. https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler
15. MacQueen, J.: Some methods for classification and analysis of multivariate observations (1967)
16. Number of Android applications. http://www.appbrain.com/stats/number-of-android-apps
17. Shin, C., Hong, J.-H., Dey, A.K.: Understanding and prediction of mobile application usage for smart phones. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp 2012, New York, NY, USA, pp. 173–182. ACM (2012)
18. Garner, S.R.: Weka: The waikato environment for knowledge analysis. In: Proceedings of the New Zealand Computer Science Research Students Conference, pp. 57–64 (1995)
19. Weka-for-Android. https://github.com/rjmarsan/Weka-for-Android
20. Libsvm-androidjni. https://github.com/cnbuff410/Libsvm-androidjni
21. Ling, C.X., Huang, J., Zhang, H.: AUC: a better measure than accuracy in comparing learning algorithms. In: Xiang, Y., Chaib-draa, B. (eds.) Canadian AI 2003. LNCS (LNAI), vol. 2671, pp. 329–341. Springer, Heidelberg (2003)
22. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Series in Data Management Systems, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2005)
23. Hand, D.J.: Measuring classifier performance: A coherent alternative to the area under the roc curve. Mach. Learn. **77**, 103–123 (2009)