Context Aware Mobile Application Pre-Launching Model using KNN Classifier

Malini Alagarsamy
Department of Computer Science,
Thiagarajar College of Engineering, India
amcse@tce.edu

Ameena Sahubar Sathik
Infosys Pvt Ltd, India
ameenasahubarsathik@gmail.com

Abstract: *Mobile applications are the application software which can be executed in mobile devices. The Performance of the mobile application is major factor to be considered while developing the application software. Usually, the user uses a sequence of applications continuously. So, pre-launching of the mobile application is the best methodology used to increase the launch time of the mobile application. In Android Operating System (OS) they use cache policies to increase the launch time. But whenever a new application enters into the cache it removes the existing application from the cache even it is repeatedly used by the user. So the removed application needs to be re-launched again. To rectify it, we suggest K number of applications for pre-launching by calculating the affinity between the applications. Because, the user may uses the set of applications together for more than one time. We discover those applications from the usage pattern based on Launch Delay (LD), Power Consumption (PC), App Affinity, Spatial and Temporal relations and also, a K-Nearest Neighbour (KNN) classifier machine learning algorithm is used to increase the accuracy of prediction.*

Keywords: *Mobile application, launch time, app affinity, pre-launch, context-aware*.

Received October 10, 2020; accepted December 14, 2021
https://doi.org/10.34028/iajit/19/6/11

1. Introduction

Mobile phones are the best companion for us which can be used to communicate with others, check e-mails, browse various websites, playing games and to take photos, etc., [1]. User experience of mobile usage is mainly depends on the mobile applications. Hence, the performance of these mobile applications plays a major role in enhancing the user experience. Usually users want to work with the application which run faster whenever the user click the icon of the mobile application. The time taken by the application to launch or to arrive at ready state is called as launch delay. If the application takes more time for launching then it affects the performance of the application. Cold launch delay is the time taken by the application to get started when the user clicks the application icon. If the application never launched before or it is not available in the cache then we have to fork the complete application process which contains Create, Start and Resume states.

Hot start is the time taken by the application to resume or resumption of already existing process. The application was already available in the cache. In this case, Start and Resume states are involved in it. Compared to the cold start, hot start produces lower overhead. If all of your application's activities are still resident in memory, then the app can avoid having to repeat object initialization, layout inflation and rendering. However, if some memory has been purged in response to memory trimming events, then those objects will need to be recreated in response to the warm start event.

The user may use many applications and navigates among those applications when necessary. This navigation requires the application to be re-launched. This re-launches affects the performance of the mobile application. In android, caches are used for fast launching. If the cache contains more number of applications then it consumes more energy [2, 3]. By pre-launching the necessary application, we can reduce the launch delay and energy consumption [13, 14, 15]. In android Operating System (OS), Least Recently Used (LRU) cache policy is used for eviction.

A study on the Android app usage profiling of 20 participants was conducted and a similar tendency was observed. It is observed that, for most users, there is a small set of distinctive app usage patterns that are repeatedly appearing. In particular, it was quite common to see that two apps are strongly related each other, often being launched successively. For example, it is observed that one of 20 study participants has launched Amazon and Flipkart together in majority launches. Furthermore, it is observed that, the usage pattern of a single user differs according to the current environmental contexts. For example, when the user in Home, he uses some set of applications repeatedly and he uses different set of applications in college.

From the analysis of the collected usage logs, it is observed some distinct characteristics of app usage patterns, which formed the main motivation of this work. First, it is observed a well-known app usage

tendency that only a small number of favored apps are heavily used. Second, it is also observed that there is a strong affinity on how related apps are used. In particular, it is observed that there are many pairs of apps that are used together in a particular situation. By identifying these distinctive usage patterns among a small set of favored apps during runtime and pre-launching related apps according to context, could improve user experience [16, 17, 18]. The objective of this work is to reduce the application launch delay in android by pre-launching the related apps.

In this paper, Context Aware Mobile application Pre-launching (CAMP) model is proposed which reduces the re-launches and restart count by introducing a new eviction policy called Launch App Correlation Evictor (LACE). This model collects the app usage log of a smartphone user and analyzes that log which contains the app usage history along with the temporal and spatial details of the app used. This proposed model optimizes the launching experience by predicting the user's future app usage tendency and pre-launches the related apps together. LACE policy which will be calculating the affinity among the apps and evict the app which have low affinity value is proposed. This LACE policy reduces restart count when compared to LRU since it gives priority to related apps.

The contributions of this paper are,

- Suggest K number of applications for Pre-launching to accelerate the launch time of an application.
- Reduce the re-launches of repeatedly used application using LACE policy.
- Predict the applications using affinity between the applications, location and time of the application used.
- Improves the accuracy of prediction using KNN classifier machine learning technique.

The remainder of this paper is organized as follows: Section 2 shows existing literature review, section 3 discusses the proposed context aware mobile application pre-launching model and its overall design, section 4 deals with the real time implementation and results and section 5 presents the conclusion.

2. Related Work

Lu and Yang [10] presented a Spatial and Temporal App Recommender (STAR) framework which predicts the applications using spatial and temporal relation. In this paper, the authors proposed Spatial and Temporal App Usage Pattern (STAUP) Mine algorithm to discover time and location details of the applications used by the user from the geographic Global Positioning System (GPS) trajectory. The spatial information related with the applications is continuously tracked using GPS. GPS raw information's are converted into locations using the STAUP mine algorithm. By analyzing the usage pattern they discover the temporal details using timestamps. Depending on these spatial and temporal details, one application is suggested for pre-launching. The STAR framework proposed in this paper suggests only one application for pre-launching. It does not consider power consumption and launch delay of an application during prediction.

Song *et al*. [18] developed Application Usage Model (AUM) and AUM based optimization model. In AUM module, the app usage behavior of the user is continuously monitored and the app usage pattern is constructed. Using this observed usage pattern, the app radius is calculated. Affinity among the applications is calculated using the app radius. In optimization model clustering method is used to predict the application to be pre-launched. The framework suggests the application which is used only one time by the user for pre-launching. The suggested application pre-launched into the cache even it does not used by the user.

Li *et al*. [8] implemented a low memory killer using reinforcement learning technique. Through the trial and error exploration the killer interacts with environment and predicts the app launch latency. The predicted app is killed by the killer. The automatic decision makers continuously observe the various indicators from the environment and make the decision based on the environment factors. Whenever the memory is not enough to store the reused application it randomly selects the application and kill that application. If the killed application used by the user in near future it re-launched again.

Zhao *et al*. [20] implemented a client centric technique Program Analysis for Latency Optimization of Mobile Apps (PALOMA) which pre-fetches the Hypertext Transfer Protocol (HTTP) requests and reduces the network latency. String analysis and call back control flow analysis are used here to pre-fetch the request. String analysis is used to analyze the request sent by the user. And call back control flow analysis is used to pre-fetch the results and stored it in temporary buffer. Pre-fetching phase of PALOMA technique requires a large amount of memory to store the received responses. Because it stores each and every responses related with the words in the given URL.

Several techniques were implemented using non-volatile memory [6, 16], volatile memory [9, 13] and memory reclamation techniques [7] to accelerate launch delay. But these techniques did not consider usage behavior and energy.

The proposed model uses app affinity, temporal and spatial relation for predicting and suggests K number of applications for pre-launching as shown in Figure 1. And also a machine learning mechanism is used to increase the accuracy of prediction. Whenever cache try to insert new application it evicts the application based on LACE policy.

3. Proposed Methodology

CAMP Model minimizes the user-perceived delays, when a user launches apps. App usage pattern and information about the applications which are used previously by the user is tracked using the Android-App-Tracker application. The tracked information are given as an input for calculating Launch Delay, Power Consumption, App affinity, Spatial and Temporal segmentation. By applying KNN classifier machine learning algorithm on that information the application to be pre-launched is predicted.

3.1. Android-App-Tracker

In order to understand better how smartphone apps are used by different users, detailed logs of smartphone usage from 20 participants were collected. All the participants of this usage study were typical smartphone users, almost always carrying their smartphones with them. For this usage study, a special Android app namely Android-App-Tracker shown in Figure 5 was developed which collects various usage information while users interact with their favored apps [11]. This app automatically collects information on smartphone use, including information about apps used based on context (location, time), the start and end of each app use. It also finds out the cold launch and hot launch delay of the every application. Cold launch or launch delay is the time taken by an application to come to foreground when the application is clicked. Hot launch delay or re-launch delay is the resumption of existing application process. The hot and cold launch delays are measured in milliseconds for better accuracy. These details are sent to the Decision Engine for further processing.

3.2. Computation of Launch Delay (LD) and Power Consumption (PC)

Using the data given by the application tracker, launch probability of the application is identified. Launch probability defines the probability of using the application in near future. It is computed using Equation (1),

$$\text{Launch Probability}(\text{app}_x) = \frac{\text{usage\_count}(\text{app}_{x,\text{time,location}})}{\text{Total\_usage\_count}_{\text{time,location}}} \quad (1)$$

Usage-count ($\text{app}_{x,\text{time,location}}$) indicates how many times app '*x*' is used by the user at particular time and location.

LD is computed using Equation (2) for every application [19].

$$\text{Launch Delay } \text{LD}_A = \text{P}_A \times (\text{Cold}_A - \text{Hot}_A) \quad (2)$$

Where, P_A is the launch probability of an application *A*, $Cold_A$ is launch delay in milliseconds and Hot_A is re-launch delay in milliseconds.

PC [12] is computed using Equation (3) for every application,

$$\text{Power Consumption } \text{PC}_A = (\text{AvgPC}_A) \times \text{D}_A \quad (3)$$

Where, $AvgPC_A$ is average power consumption of an application in joule and is duration of application used in seconds. Power tutor is used for power profiling. It measures the energy consumption in joule. So, convert the energy into watts/second using Equation(4)

$$\text{Power Consumption (watt)} = \text{Energy (joule)} / \text{Time (second)} \quad (4)$$

3.3. Computation of Spatial and Temporal Segmentations

To predict the Spatial and Temporal segmentations, we use the different timestamp values of the used application which can be extracted from the app usage pattern.

Si={*t1, t2, t3,…..tn*} S_i=sequence of timestamps having highest frequency.

Compute the threshold value α using the Equation (5),

$$\alpha = \sum_{i=2}^{n} \frac{ti - t(i-1)}{n} - (n-1) \quad (5)$$

dij=difference between the two timestamps *ti* and *tj*.

If dij> α then add the two timestamps ti and tj into the ntsp.

Finally divide S_i into ntsp+1 time intervals. Depending on these time intervals, app usage information and user past history we computed the spatial information.

3.4. Computation of App Affinity

App affinity Computation Engine identifies strongly related apps based on a metric that characterizes the launch affinity among apps [18]. It computes app affinity between apps from usage sequence collected by Application Tracker. Let 'S' be the app sequence used by the user for a particular period,

$$S = < a_1, a_2, a_3, \ldots\ldots\ldots\ldots, a_n >$$

Algorithm (1) represents the sequence of steps used to compute the affinity between the application in the given application sequence. First, Algorithm (1) calculates the usage count and app usage position values of all the application. And then it calculates the distance between each and every application in the given application sequence. It calculates the average distance between the applications which are repeatedly used by the user or the value of AUC>1. Affinity between the two applications is calculated by adding the average distances between the applications.

For every app 'A' in S, following data can be computed,

a) Launch Distance D_i (A) of app 'A' relative to B_i in S defined as 'd', where,

1. A = $a_j$in S
2. d = | i - j | such that, there is no a_k = A for | i - k | <d

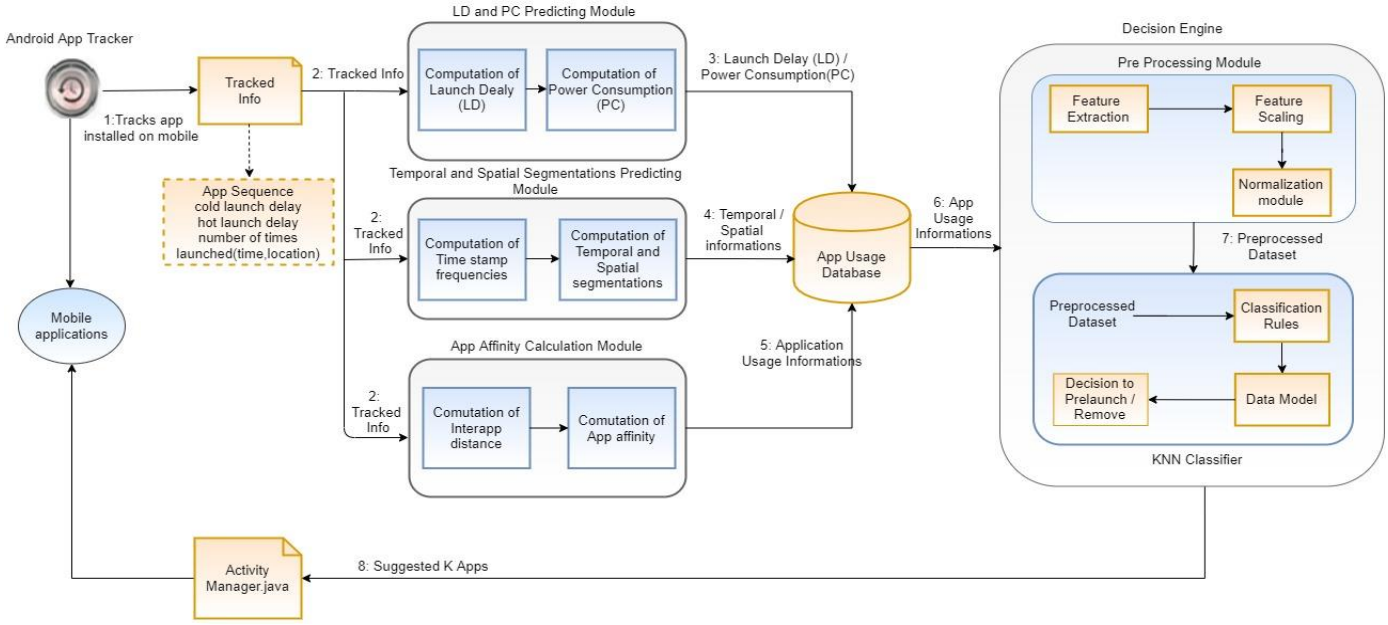

Figure 1. Architectural design of CAMP model.

Algorithm 1: Computing the affinity correlation between two applications.
INPUT: Array contain sequence of apps, s=<a_i, a_{i+1}, a_{i+2}, …..a_n > where i=0.
OUTPUT: Apps having high affinity.
STEPS:
1.Compute usage count (UC) for each app a_i in the given sequence.
2. Compute the distance between the apps,
D[i][j]=|i-j|
Where D[i][j] = distance between the apps in i^{th} and j^{th} position.
i,j = position values of an apps.
3. For each app a_i ,
If UC[i] >1 then
Compute the average distance,
Avg_dis[i] = (n-D[i][j])/UC[i];
Avg_dist = Average distance value of an app which is in i^{th} position.
n= sequence length.
UC[i] = App Usage Count value of an app in the i^{th} position.
4. Compute the Affinity between the apps,
Aff[i][j] = Avg_dis[i] + Avg_dis[j];
Aff[i][j] = Affinity between the apps in the i^{th} and j^{th} position.
5. Select the apps in ith and jth position which having maximum value in Aff[i][j].
Apps in the i^{th} and j^{th} position are strongly related

Since the same app '*B*' can appear in multiple locations, the average app launch distance of app A relative to B is computed using Equation (6),

$$\text{Davg}(A|B) = \frac{\sum_{i \in S_B} n - D_i(A)}{|S_B|} \quad (6)$$

n -Length of app sequence 'S'
A -App in app Sequence 'S'
$D_i(A)$-Distance of app A relative to B_i in S
S_B-{ j∈ {1,…., n } | a_j = B in S}

b) App affinity value can be computed using Equation (7),

$$AA(A, B) = D_{avg}(A|B) + D_{avg}(B|A) \quad (7)$$

The higher value of app affinity indicates, the apps are strongly related.

3.5. KNN Classifier Machine Learning Algorithm

K-nearest-neighbor is a data classification algorithm that attempts to determine the group of a data point by looking at the data points around it. An algorithm, looking at one point on a grid, trying to determine if a point is in group A or B, looks at the states of the points that are near to it. The range is arbitrarily determined, and then it took the sample of data. If the majority of the points are in group A, then it is likely that the data point in question will be A rather than B, and vice versa.

Here, five inputs are supplied to decision engine- App Affinity, Launch Delay, Power Consumption, Location and Time. The app usage information is preprocessed. The various levels of inputs are low, medium and high. The output Victim selection has two levels:

1. Pre-Launch.
2. Remove.

Based on the rules in the rule set, the decision engine will decide whether to pre-launch or remove the application. The rules defined in rule set are listed below,

- *Rule* 1: if app affinity is high AND Launch Delay is high AND Power Consumption is low AND Location is home AND Time is day THEN pre-launch.
- *Rule* 2: if app affinity is low AND Launch Delay is low AND Power Consumption is high AND Location is home AND Time is day THEN remove.

- *Rule* 3: if app affinity is low AND Launch Delay is high AND Power Consumption is high AND Location is home AND Time is day THEN remove.
- *Rule* 4: if app affinity is high AND Launch Delay is high AND Power Consumption is low AND Location is college AND Time is day THEN pre-launch.
- *Rule* 5: if app affinity is high AND Launch Delay is high AND Power Consumption is low AND Location is home AND Time is night THEN pre-launch.
- *Rule* 6: if app affinity is high AND Launch Delay is low AND Power Consumption is high AND Location is home AND Time is day THEN remove.
- *Rule* 7: if app affinity is low AND Launch Delay is high AND Power Consumption is low AND Location is college AND Time is night THEN remove.
- *Rule* 8: if app affinity is high AND Launch Delay is high AND Power Consumption is low AND Location is Home AND Time is day THEN pre-launch.

3.6. Launch App Correlation Eviction

LACE uses app affinity computes as explained in section 3.4. Let consider the user pattern (most used) observed for a user as shown in Figure2. For simplicity, consider stack contains only 4 apps, since there are six different apps used by the users in a particular pattern.

Under the Android's default LRU policy whenever cache is filled, the new arrived app replace the least recently used app. According to this scenario, for the identified pattern shown in Figure 2, Facebook enters into cache at step 4 and 9 and it replaces the least recently used app WhatApp from the cache even it is used right after Facebook. Similarly Flipkart enters into cache at step 6 replaces the least recently used app Amazon from the cache even it is used right after Flipkart. Hence we have to restart those applications which are recently evicted from the cache.

This could be avoided by the Launch App Correlation Eviction policy. Since LACE uses affinity values between apps for eviction, it could identify strongly related apps. Whenever new app enters into cache the lace policy finds the app having lower affinity with the new app. It evicts that app and inserts the new app into cache. If the app is having high affinity with new app, then it will be reused after sometime. By using lace policy the restart count reduced from 6 to 2 in Figures 3 and 4 respectively.

Figure 2. App usage pattern.

Figure 3. LRU eviction policy-restart count 6.

Figure 4. LACE policy-restart count 2.

4. Results and Discussion

The evaluation of the CAMP model is performed on the set of android applications selected from the Google play store. The evaluation is done on Moto G4 device with 2 GB RAM and Android 6.0.1 OS. Various categories of applications are taken for evaluating the proposed model. The major categories are online and offline applications. Both are further classified into multimedia applications like Facebook, YouTube etc. and non-multimedia applications like contacts, phone etc. The sample set of applications used by a user, its cold and hot launch delay calculated by application tracker is listed in the Table 1. Here the Android-Ap-Tracker application is installed on the device and the users are asked to use the device normally for a period of 4 weeks to identify the usage pattern. It is evident that the usage of smart phones/tablets varies from user to user. So the averages of cold launch and hot launch are taken for the applications. From Figure 6, it is obvious that the cold launch delay is greater than the hot-launch or re-launches delay, since resuming the applications takes lesser time. The app sequence identified by the application tracker for the applications listed in Table 1, are given in Figures 7, 8, 9, and 10. Length of app sequence taken is 20, since only a small number of different apps (approximately 10 to 15 apps) are used by each user. Two locations as Home and Office and time as Day and Night were taken.

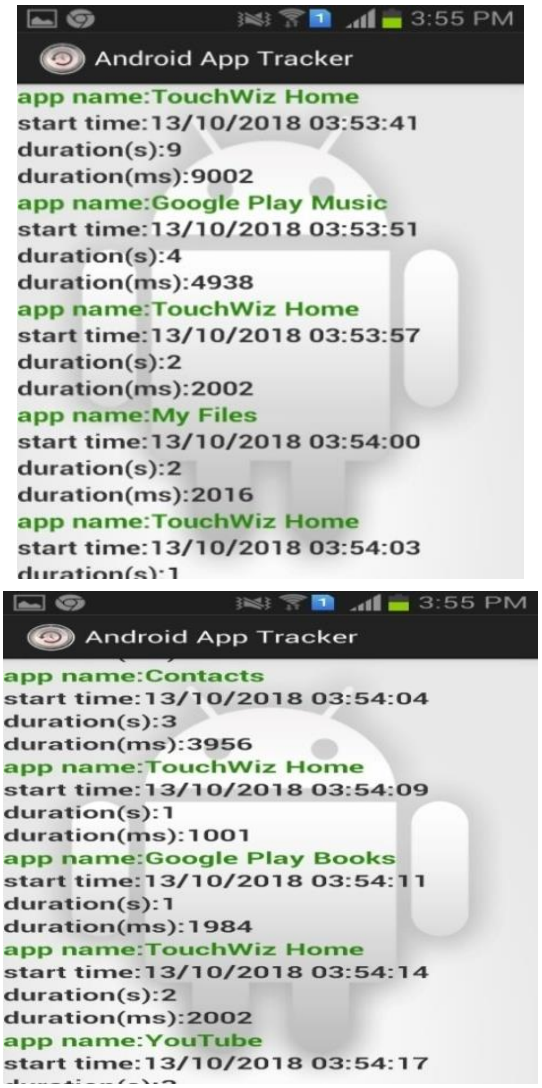

Figure 5. Android app tracker-app sequence.

Table 1. Average cold and hot launch delay of the applications.

Application Name	Cold-launch Delay (ms)	Hot-launch Delay (ms)
Adobe reader	1281	890
Amazon	1948	1230
Angry birds	3564	2623
Browser	1598	829
Calculator	945	753
Contacts	632	387
Facebook	2450	1542
File manager	895	654
Flipkart	1831	1132
Gallery	991	683
Gmail	1040	703
Google Drive	1002	678
Music	664	402
Olive office premium	978	870
Phone	656	358
PhonePe	2067	1432
Temple Run	3895	2550
WhatsApp	1802	1024
Youtube	1654	897

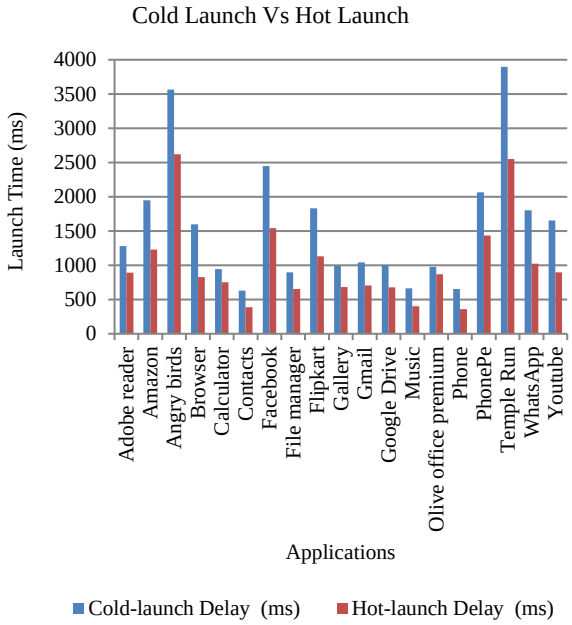

Figure 6. Cold launch Vs hot launch.

For app sequence given in Figure 7, during day time in Home location, Flipkart appeared in 1, 10 positions. Then the average launch distance of Amazon relative to Flipkart is computed using Equation (8). Let A= Amazon and B= Flipkart. Total number of occurrences for Flipkart is 2, So $|S_{Flipkart}|=|S_B|= 2$. The app sequence length is 20, i.e., n= 20. Amazon is appeared in 2, 11 positions i.e., immediately after Flipkart. Hence the launch distance of Amazon relative to Flipkart is 1 in both cases, i.e., D_1(Amazon)=1 and D_{10}(Amazon)=1.

$$D_{avg}(A|B)=\frac{\sum_{i\in S_B} n- D_i(A)}{|S_B|} \quad (8)$$

D_{avg}(Amazon|Flipkart) =(20-D_1(Amazon))+(20-D_{10}(Amazon))/$|S_{Flipkart}|$
= (19 + 19) / 2 = 19

Similarly,

D_{avg}(Flipkart |Amazon) = (20- D_2(Flipkart))+(20- D_{11}(Flipkart))/$|S_{Amazon}|$
= (19 + 19) / 2 = 19

Similarly for all pair (X, Y) in D_s,app affinity is computed, where D_s= set of distinct apps in app usage pattern shown in Figure 7. The results are tabulated in Table 2. The high affinity value indicates that the apps are strongly related apps and they are pre-launched together. In this case, the pair of apps (Flipkart, Amazon) and (Browser, Music) has high affinity values, i.e. the apps Flipkart and Amazon are strongly related apps. They are almost launched together by the user. Similarly the user launches Browser and Music almost together of his/her usage. So these applications have to pre-launch.

Table 2. Launch time improvement.

Time / Location	Launch Time improvement (%)
Day / Home	40.65
Night / Home	38.56
Day / Office	36.89
Night / Office	41.95

App usage pattern identified for context (Night, Home), (Day, Office) and (Night, Office) are shown in Figures 8, 9, and 10 respectively. The app affinity values for context (Night, Home), (Day, Office) and (Night, Office) are computed as explained in section 3.4. In the context (Night, Home), the pair of apps (Angry Birds, Temple Run), (Browser, Music) and (Facebook, Youtube) are strongly related apps, since they have high affinity values. They are almost launched together by the user. So these applications have to pre-launch during (Night, Home).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Day / Home	Flipkart	Amazon	WhatsApp	Browser	Music	Phone	Gallery	WhatsApp	Contacts	Flipkart	Amazon	PhonePe	WhatsApp	Youtube	Gallery	WhatsApp	Gmail	Browser	Music	File Manager

Figure 7. App usage pattern (Day, Home).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Night / Home	WhatsApp	File Manager	Browser	Music	Phone	Angry Birds	Temple Run	WhatsApp	Gmail	Facebook	Youtube	Gmail	Adobe reader	Angry Birds	Temple Run	Phone	Facebook	Youtube	Browser	Music

Figure 8. App usage pattern (Night, Home).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Day / Office	Browser	Gmail	Google Drive	WhatsApp	Olive Office premium	Facebook	Youtube	File Manager	Gmail	Phone	Google Drive	File Manager	Adobe reader	WhatsApp	Gallery	Browser	Calculator	Facebook	Phone	Youtube

Figure 9. App usage pattern (Day, Office).

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Night / Office	Gmail	WhatsApp	File Manager	Browser	Music	Facebook	WhatsApp	Amazon	Music	Browser	Phone	WhatsApp	Facebook	Temple Run	Music	Google Drive	Browser	Music	Gmail	Contacts

Figure 10. App usage pattern (Night, Office).

During Day time in Office, the apps (Facebook, Youtube) and (Gmail, Google Drive) are identified as strongly related apps through affinity value. During (Night, Office) context, the apps (Facebook, WhatsApp) and (Browser, Music) are identified as strongly related apps through affinity value. So these applications have to pre-launch.

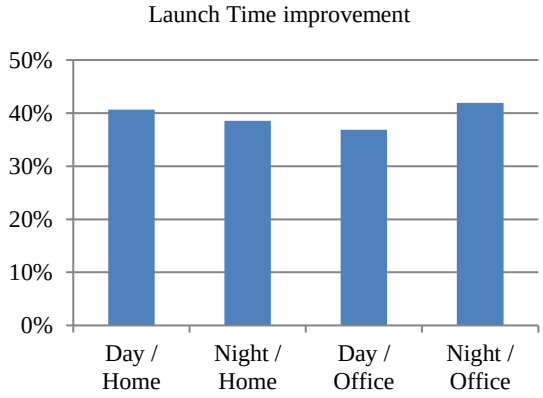

Figure 11. Launch time improvements (Context wise).

From the Table 2 and Figure 11, it is evident that CAMP model improves launch time in four contexts. As an average, CAMP model improves launch time by 40%. In this paper the statistical t-test analysis of CAMP model was done [4]. The t-value and p-value are 2.0959 and 0.021592 respectively. The result is significant at $p < .05$.

4.1. Comparison with Existing Methods

Chung *et al.* [3] proposed Energy Aware Stack Regulator using Poisson predictor (EASR) to reduce the application launch delay for multimedia applications. EASR improves launch time by 18%. The proposed strategies are best suitable for multimedia applications. Zhong *et al.* [21] proposed NVM-Swap to build high-performance smart phones which replace the part of the DRAM with NVM, and use it as a swap area. NVM-Swap is used to reduce launch time and maintain good user experience. NVM-Swap improves launch time by 21%. The CAMP model proposed in this paper improves launch time by 40%. From Figure 12, it is evident that the proposed CAMP model performs better than existing techniques.

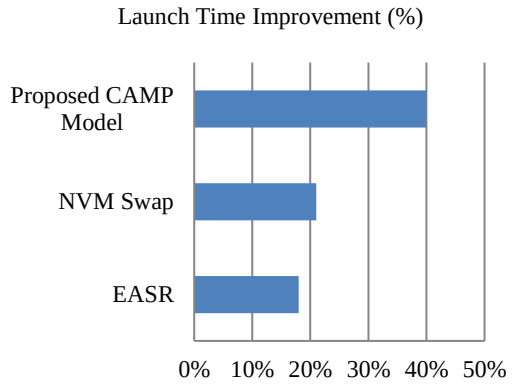

Figure 12. Comparison with existing methods.

4.2. Applying KNN Classifier Algorithm for Decision Making

The collected information are stored in the database and given as an input to the Decision Engine which makes a decision to pre-launch or remove application from cache. We extract features from the collected information. So the raw values are replaced for all attributes in the data set with the modes and means from the training data. We extracted five features Launch Delay, Power Consumption, Time, Location and App affinity.

Feature scaling is different from feature extraction. It creates new features by combining the original feature. It eliminates the irrelevant and redundant features. So, the learning procedure takes less time to learn the features. Learning algorithms focus only on

the essential features because it operates without intervention of irrelevant and noisy features.

Various machine learning algorithms are applied on the pre-processed information. Based on the rules in the rule set described in section 4 the decision engine makes decision whether to pre-launch or remove the application to or from cache. Machine learning algorithms such as KNN, Decision Tree, Bayes Net, Naïve Bayes and Random Tree applied on the dataset [5]. Among the all applied algorithms KNN produces better results such as 92% accuracy. Because KNN algorithm is automatically non-linear, and it can handle large number of data points. The tree algorithms like Decision tree and Random tree performs well in very low dimensional space. In our framework we have high dimensional space and large number of data points which are heterogeneously distributed. Hence, we are proceeding with KNN classifier algorithm in our framework for making decisions.

5. Conclusions and Future Work

In this paper, a CAMP Model with a new eviction policy called LACE is proposed for minimizing the application launch delay. First an application tracker is introduced and this identifies app usage pattern by collecting usage information about apps including apps used based on context (location, time), the start and end of each app use, cold launch and hot launch delay of the every application. From the data set provided by the application tracker, App Affinity computation Engine identifies related apps i.e., the apps that are launched together at most by the user and pre-launch them to improve launch time. The LACE policy reduces restart count when compared to LRU since it gives priority to related apps. The CAMP model proposed in this paper improves the launch time by 40%. And various machine learning algorithms are applied on that information to increase the prediction accuracy. The experimental results shows that KNN classifier machine learning algorithm gives the better accuracy (92.3) when compared with the other machine learning algorithms as shown in Figure 13.

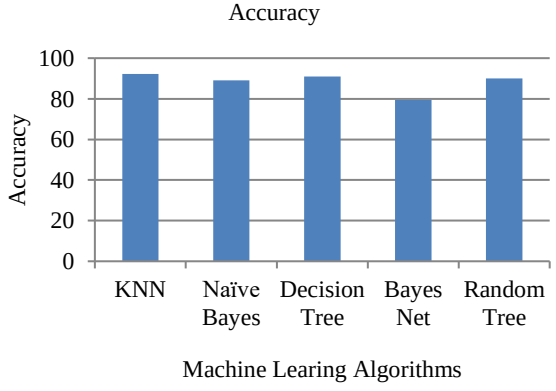

Figure 13. Comparison of Machine learning algorithms accuracy.

Currently, we designed our frame work based on the android architecture and it works only for android devices. In future we will be extended to iOS and black berry smart phones and hardware optimization, Power management can also be done to improve the launch time.

References

[1] Albazaz D., "Design A Mini-Operating System for Mobile Phone," *The International Arab Journal of Information Technology*, vol. 9, no. 1, pp. 56-65, 2012.

[2] Balasubramanian N., Balasubramanian A., and Venkataramani A., "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," *in Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, Chicago, pp. 280-293, 2009.

[3] Chung Y., Lo Y., and King C., "Enhancing User Experiences by Exploiting Energy and Launch Delay Trade-off of Mobile Multimedia Applications," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 1, pp. 1-19, 2013.

[4] Dodge Y., Cox D., and Commenges D., *The Oxford Dictionary of Statistical Terms*, Oxford University Press on Demand, 2006.

[5] Hssina B., Merbouha A., Ezzikouri H., and Erritali M., "A Comparative Study of Decision Tree ID3 and C4. 5," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 2, pp. 13-19, 2014.

[6] Kim H., Lim H., Manatunga D., Kim H., and Park G., "Accelerating Application Start-Up with Nonvolatile Memory in Android Systems," *IEEE Micro*, vol. 35, no. 1, pp. 15-25, 2015.

[7] Kim S., Jeong J., Kim J., and Maeng S., "Smartlmk: A Memory Reclamation Scheme for Improving User-Perceived App Launch Time," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 3, pp. 1-25, 2016.

[8] Li C., Bao J., and Wang H., "Optimizing Low Memory Killers for Mobile Devices Using Reinforcement Learning," *in Proceeding of the 13th International Wireless Communications and Mobile Computing Conference*, Valencia, pp. 2169-2174, 2017.

[9] Lin Y., Yang C., Li H., and Wang C., "A Hybrid DRAM/PCM Buffer Cache Architecture for Smartphones with Qos Consideration," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 2, pp. 1-22, 2016.

[10] Lu E. and Yang Y., "Mining Mobile Application Usage Pattern for Demand Prediction By Considering Spatial and Temporal Relations,"

GeoInformatica, vol. 22, no. 4, pp. 693-721, 2018.

[11] Malini A., Sundarakantham K., Prathibhan C., and Bhavithrachelvi A., "Fuzzy-based Automated Interruption Testing Model for Mobile Applications," *International Journal of Business Intelligence and Data Mining*, vol. 15, no. 2, pp. 228-253, 2019.

[12] Mittal R., Kansal A., and Chandra R., "Empowering Developers to Estimate App Energy Consumption," *in Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Istanbul, pp. 317-328, 2012.

[13] Nguyen P. and Garg A., "Application Pre-Launch to Reduce User Interface Latency," *U.S. Patent No. 7,076,616*. Washington, DC: U.S. Patent and Trademark Office, 2006.

[14] Parate A., Böhmer M., Chu D., Ganesan D., and Marlin B., "Practical Prediction and Prefetch for Faster Access to Applications on Mobile Phones," *in Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing*, Zurich, pp. 275-284, 2013.

[15] Schwartz C., Hoßfeld T., Lehrieder F., and Tran-Gia P., "Angry Apps: The Impact of Network Timer Selection on Power Consumption, Signalling Load, and Web Qoe," *Journal of Computer Networks and Communications*, 2013.

[16] Shi L., Li J., Jason Xue C., and Zhou X., "Hybrid Nonvolatile Disk Cache for Energy-Efficient and High-Performance Systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 1, pp. 1-23, 2013.

[17] Shin C., Hong J., and Dey A., "Understanding and Prediction of Mobile Application Usage for Smart Phones," *in Proceedings of the ACM Conference on Ubiquitous Computing*, Pittsburgh, pp. 173-182, 2012.

[18] Song W., Kim Y., Kim H., Lim J., and Kim J., "Personalized Optimization for Android Smartphones," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 2, pp. 1-25, 2014.

[19] Yan T., Chu D., Ganesan D., Kansal A., and Liu J., "Fast App Launching for Mobile Devices Using Predictive User Context," *in Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, Low Wood Bay, pp. 113-126, 2012.

[20] Zhao Y., Laser M., Lyu Y., and Medvidovic N., "Leveraging Program Analysis to Reduce User-perceived Latency in Mobile Applications," *in Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, pp. 176-186, 2018.

[21] Zhong K., Wang T., Zhu X., Long L., Liu D., Liu W., Shao Z., and Sha E., "Building High-performance Smartphones Via Non-volatile Memory: The Swap Approach," *in Proceeding of the International Conference on Embedded Software*, Uttar Pradesh, pp. 1-10, 2014.

Malini Alagarsamy obtained her PhD in Information and Communication Engineering from Anna University, Chennai. She is currently working as an assistant professor at Thiagarajar College of Engineering, Madurai, India. She has published several research papers in journals and international/national conferences. Her research interest includes software Engineering, Testing, Mobile Application development, Green Computing, Internet of Things, Block chain and Machine Learning.

Ameena Sahubar Sathik obtained her B.E in Computer Science and Engineering from Anna University (University College of Engineering, Ramanathapuram) in 2017 and completed her Master degree (M.E) in Computer Science and Engineering in Anna University in 2019. She is currently working as a Senior Systems Engineer at Infosys Pvt Ltd. Her research interest includes software testing, software design and mobile application development.