SatScore: Uncovering and Avoiding a Principled Pitfall in Responsiveness Measurements of App Launches

Zhijia Zhao, Mingzhou Zhou
College of William and Mary, USA
{zzhao,mzhou}@cs.wm.edu

Xipeng Shen
North Carolina State University, USA
xshen5@ncsu.edu

ABSTRACT

Important for user experience on mobile devices, app launch responsiveness has received many recent attentions. This paper reveals a principled pitfall in previous studies. Most of these studies have used average reduction of response delays as the metric for responsiveness. Through a systematic user study and statistical analysis, this paper shows that the metric fails to faithfully reflect user experienced responsiveness. To avoid the pitfall, a straight-forward solution is to employ users' direct feedback as the responsiveness metric, which is unfortunately hard to obtain. This paper presents the promise of solving the dilemma through a SatScore model. It further demonstrates some new opportunities for responsiveness enhancement enabled by the SatScore model.

Author Keywords

Smartphone, Launch responsiveness, User study, Measurement Pitfalls

ACM Classification Keywords

H.5.2 User Interfaces: Evaluation/methodology

INTRODUCTION

With the increasing popularity of tablets and smartphones, mobile apps are becoming essential to people's everyday life. Research has long shown that reducing application response time is one of the most important variables for user satisfaction on handheld devices [8,15]. Multiple studies by industry have echoed the finding: Google and Microsoft reported that even a delay of 500ms has a major impact on business metrics, and a 200ms increase in page load latency resulted in "strong negative impacts" [6, 13, 14].

Recent years have seen many efforts in enhancing the responsiveness of mobile apps. The proposed techniques include the exploitation of advanced disk techniques [11, 12], content prefetching and app preloading [18, 24], the leverage of app usage pattern prediction [9,18,19,23,24], and so on. As app launch is a main source of users' frustration in app responsiveness [4], it has been the primary target of most of those studies; it is also what this paper focuses on.

A fundamental factor underlying all such studies is the measurement of responsiveness. Without an appropriate metric, it is difficult to objectively assess the value of a technique for responsiveness improvement, no matter whether the technique is on prefetching, preloading, new device designs, or new scheduling policies in the operating systems.

The metrics used in existing studies are mainly some simple variants of the reduction of *response delay*—defined as the time between the start of an operation (e.g., an app launch) and the time point when the results of the operation become ready for users to interact with. The variants are often either the total of the absolute response time reduction across all tested apps [21, 24], or the average percentage of the time reduction [11, 12] brought by the technique of interest.

Although response delay reduction is informative, what is ultimately relevant is how the change affects user experience. In almost all prior responsiveness studies, response delay reduction has been the only metric reported in assessing the effectiveness of a technique. In our survey of papers in the last decade of MobiCom, MobiSys, Ubicomp, and other refereed conferences, we found 28 papers related with responsiveness studies, 27 of which use delay reduction as the metric for responsiveness. The only exception uses the metric, "alternation", to measure the quality of Skype in supporting conversation quality in the special context [3].

Response delay reduction is a metric easy to use. However, some basic questions on its connection with the ultimate goal—user experienced responsiveness—are left open. Does a 20% average response delay reduction necessarily mean more improvement of user experienced responsiveness than a 10% reduction does? Is response delay reduction a sufficient metric for assessing the responsiveness improvement? More generally, what is the relationship between response delay reduction and user experience? Answers to these open questions will provide some principled understanding on the responsiveness evaluation of mobile apps. They are also essential to a proper assessment of various techniques in responsiveness enhancement.

This work makes two major contributions. First, it conducts a systematic investigation on the relationship be-

UbiComp'14, September 13 - 17 2014, Seattle, WA, USA.

ACM 978-1-4503-2968-2/14/09...$15.00.
http://dx.doi.org/10.1145/2632048.2632080

tween response delay and user experience, and uncovers a fundamental pitfall in current responsiveness measurements. Second, it proposes some practical solutions by introducing an alternative metric, SatScore, along with both offline and online techniques to support its practical usage.

Specifically, the first part of the paper describes the relationship investigation and the pitfall. It centers around a carefully designed user study on Android tablets. In the user study, with the help of a framework named *MagicLauncher*, each of 30 popular apps is launched repeatedly with delay varied randomly. Fifty two participants are asked to rate the responsiveness of each launch as satisfactory or not. A series of statistical analysis shows that significant non-linearity and app-dependence exists in the relationship between launch delay and user experienced responsiveness. A further comparative experiment concludes that the non-linearity and app-dependence are *so prominent* that purely relying on average delay reduction for responsiveness measurement is *largely flawed*: Comparing average delay reductions by two techniques may easily lead to conclusions reversal to their overall effects on user experienced responsiveness. Responsiveness studies on app launches must avoid such an important pitfall.

But how to avoid the pitfall? Collecting user's feedback on responsiveness is a direct solution, but the cost and inconveniences could add tremendous barriers to responsiveness studies—so much that it would be practical only when rigorous assessments are truly indispensable. Can we lower the barrier? In another word, can we have an option that better reflects user experience and at the same time is still convenient to use?

The second part of this paper presents our solution, SatScore modeling. *SatScore* is a new metric for measuring user experience on responsiveness, defined as the probability for a user to be satisfied with the response delay. A SatScore model is a mapping function between SatScore and launch delay. With such a model, one may derive SatScore, a metric more directly connecting with user experience, from launch delay, a metric much easier to measure.

An easy way to build up SatScore models is hence key to practical usage of SatScore. We examine three regression-based methods, including a method that builds app-neutral models, an offline method that builds app-specific models, and an online app-specific method. The results show that the latter two methods both produce practically useful SatScore models, making SatScore a valuable metric for assessing responsiveness of app launches. In addition, being app-specific, they open up opportunities for designing techniques that enhance launch responsiveness in an app-adaptive manner.

Overall this work makes the following contributions:

- **Relationship:** Through a systematic user study and analysis, it uncovers the relationship between response delay and user experience on mobile app launches, and reveals the significance of non-linearity and app-dependence of the relationship.
- **Pitfall:** It points out an important pitfall in existing responsiveness studies of app launches.
- **SatScore:** It proposes SatScore, a new metric that better measures user experience on responsiveness than existing metrics do.
- **Modeling:** It investigates the complexities in building up a SatScore model, and demonstrates the feasibility of solving the problem through offline and online regression-based methods.

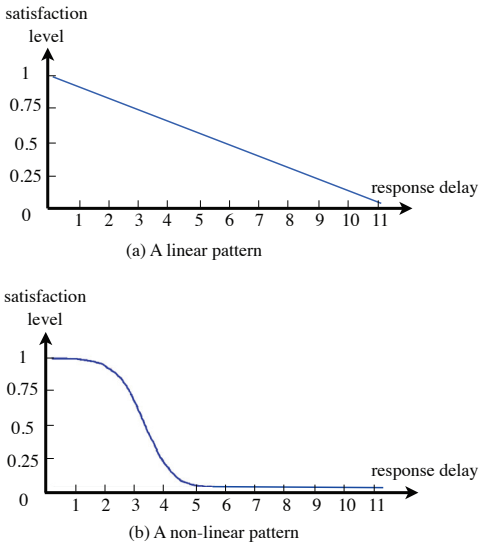

Figure 1. Two hypothetic relationships between response delay and satisfaction levels.

PROBLEM DEFINITION AND MOTIVATION

Response delay is relevant to not only user experienced responsiveness of the app, but also other aspects—such as energy consumption. Our focus is on responsiveness only, trying to examine whether, by itself, response delay is sufficient to serve as a metric for responsiveness. In particular, we concentrate on app launch delay, one of the primary sources of users' frustration in smartphone app responsiveness [4].

Responsiveness is tied with user experience. The ultimate criterion for being responsive is often whether the response delay is short enough to satisfy the user's needs. Most prior studies in enhancing responsiveness of mobile apps [11, 12, 18, 21, 24] have taken as their motivation that responsiveness critically affects users' experience. It is therefore natural to expect that these studies have used some metrics to measure how much their proposed techniques enhance users' experience. However, as aforementioned, in our survey, 27 out of 28 previous papers on mobile app responsiveness have used response delay reduction as the only metric on responsiveness.

Table 1. Example of the effects of two responsiveness enhancement techniques
(*sat*: the satisfaction levels of users on the responsiveness, derived from the delay-sat relationship in Figure 1 (b))

	app$_1$		app$_2$		delay reduction		avg.
	delay	sat	delay	sat	time	percent	sat level
default	11s	0.04	5s	0.05	0	0	0.045
w/ tech A	5.5s	0.048	4.5s	0.1	3s	30%	0.074
w/ tech B	9s	0.041	3s	0.75	2s	29%	0.40

Intuitively, this metric is more or less related with users' experience on mobile apps: A much longer delay is intuitively less likely for a user to get satisfied. But intuition is not enough for a proper usage of metrics. For instance, the intuition may lead one to believe that technique B is more effective than A in enhancing users' experience on responsiveness if it reduces more delay than A does. We believe that this kind of beliefs are some plausible reasons why prior studies have assessed their techniques only on response delay reduction, even though their motivation is to enhance users' experience.

Whether such beliefs are valid depends on what relationship exists between response delay and user experienced responsiveness. If, for example, it is a linear relationship as Figure 1 (a) illustrates for all apps, the aforementioned belief would hold. But if it is as shown in Figure 1 (b), the belief would not necessarily hold. As the rightmost three columns in Table 1 show, under that relationship, technique B improves the overall satisfaction level much more than technique A does, even though it lags behind in both types of response delay reduction.

For the importance of the relationship between launch delay and user experienced responsiveness, we name it *delay-sat relationship*, and call a function that models the relationship a *delay-sat function*. Despite the importance of delay-sat relationship, we have found no systematic explorations on it for mobile computing.

EXAMINE DELAY-SAT RELATIONSHIP

To study the relationship between launch delay and user experienced responsiveness, we design a user study, in which, participants are asked to rate the responsiveness of the launches of some Android tablet apps. The survey ideally shall happen in the field. But it adds tremendous difficulty of control. We instead develop *MagicLauncher*, a controllable environment for simulating app launches of various delays. We try to make the simulation as close to the real situations as possible through some careful designs. For instance, we ask the participants to play with the apps for a period of time before the experiment starts, prepare them with a typical usage scenario of each app (listed in the technical report [25]), and make MagicLauncher emulate the launching interface of the apps as closely as possible. Please refer to our technical report for details [25].

The study includes 30 Android apps on ten Google Nexus 7 tablets with Android 4.2 installed. These real apps are selected to cover a variety of domains with a spectrum of size and complexity, as shown in Table 2. Fifty two participants are involved. Most of them are college students between ages 20 and 30 with some experience in using smartphones or tablets. Each app is launched multiple times by the participants in the MagicLauncher; the launch delay is set by MagicLauncher to a random value between 0 and 18 seconds. At each launch, the participant is asked to rate the responsiveness of the launch as satisfactory or not, in the context of a representative usage scenario of the app. Each participant experiments with 8 apps and launches each of the apps 12 times. On average, 14 participants worked on one app, producing 126 samples per app—here, each sample means one instance of the (delay, satisfaction) tuple for that app. As the apps are randomly chosen for each participant, the numbers of participants and samples vary slightly (21% and 23% standard variations) across apps.

Delay-Sat Graph

To analyze the delay-sat relationships of the apps, we build a delay-sat graph for each app based on the data produced in the user study.

A delay-sat graph is a graph in a 2-D space with response delay and satisfaction score as the horizontal and vertical axes respectively. *Satisfaction score* is defined as the probability for an arbitrary user to be satisfied with the response delay. It is also called *SatScore* in short. The value of SatScore ranges from 0 to 1; the higher the more satisfying. It is derived from the collected data in the following way. For a given launch delay of an app, the satisfaction score of the launch is m/n, where, m is the number of participants who have indicated that they are satisfied with the current launch or a launch longer than the current delay, and n is the total number of participants who have ever seen launches of this app that have a delay equal or greater than the current delay.

Figure 2 shows the delay-sat graphs of all the thirty apps. The results seem to agree with the intuition on a monotonic relationship between launch delay and satisfaction score; a longer delay results in a lower satisfaction score, and vice versa. More importantly, though certain linearity exits (typically in the middle of the curves) for a few apps (e.g.*candycrushsaga*), most graphs show certain non-linearity. For example, the samples in the *RealCalc* at the bottom corner in Figure 2 shows that the satisfaction score changes sharply when the delay is smaller than 8 seconds but slowly otherwise. In addition, the graphs show certain app-dependence in the relationship with some clear differences across the curves of different apps. For instance, let us compare *mathway* and *translate*, the two apps at the top corner in Figure 2. At a delay of 5 seconds, the satisfaction score on *mathway* is about 0.5, while the score on *translate* is only about 0.2. The score changes much more dramatically on *mathway* than on *translate* when the delay is small.

The monotonicity is intuitive; existence of some non-linearity and app-dependence may not be much a surprise either. However, what is unclear is how promi-

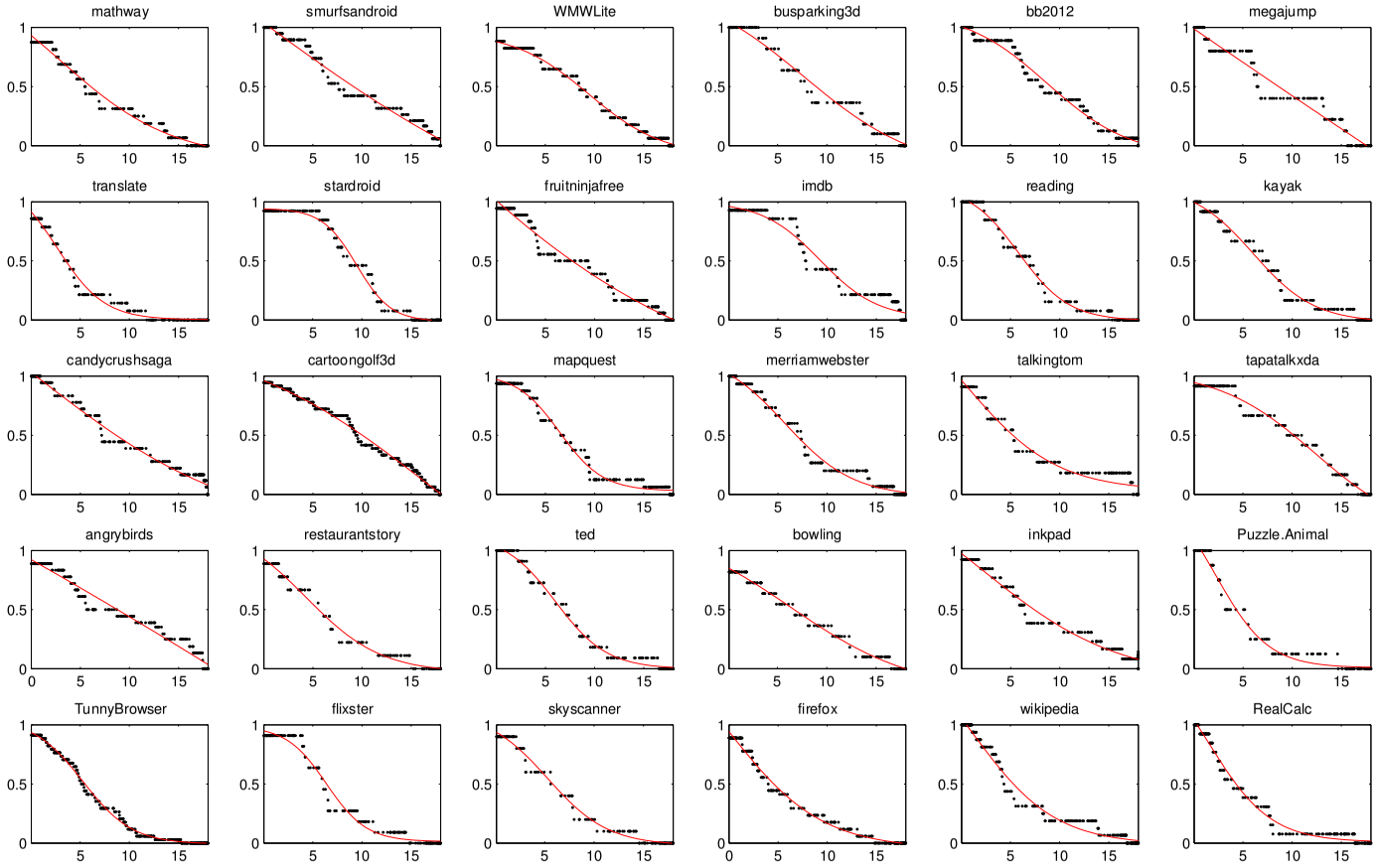

Figure 2. Delay-sat graphs of each app: X-axis is launch delay in seconds, and Y-axis is satisfaction score.

Table 2. Apps used in this study.
Sat. score is the satisfaction ratio on the default delay in terms of users

Name+	Description	Category*	Size	Footprint	Rating*	Default delay	Sat. score
megajump	Action game	Arcade/Action	31.00 MB	23 MB	4.5	4712 ms	0.66
fruitninjafree	Action game	Arcade/Action	47.79 MB	47 MB	4.5	8641 ms	0.34
angrybirds	Shooting game	Arcade/Action	41.00 MB	48 MB	4.6	9300 ms	0.32
stardroid	Viewing stars	Books/Reference	2.20 MB	34 MB	4.7	5572 ms	0.59
merriamwebster	Dictionary	Books/Reference	20.00 MB	29 MB	4.6	2425 ms	0.83
wikipedia	Wikipedia	Books/Reference	1.50 MB	26 MB	4.5	3478 ms	0.76
WMWLite	Puzzle game	Brain/Puzzle	49.00 MB	50 MB	4.6	5793 ms	0.57
busparking3d	Car parking game	Brain/Puzzle	16.38 MB	38 MB	4.1	3618 ms	0.75
Puzzle.Animal	Puzzle game	Brain/Puzzle	9.20 MB	5 MB	4.3	1264 ms	0.92
smurfsandroid	Village building game	Casual	92.00 MB	38 MB	4.6	6278 ms	0.53
candycrushsaga	Puzzle game	Casual	40.00 MB	42 MB	4.4	9438 ms	0.31
restaurantstory	Restaurant game	Casual	18.71 MB	29 MB	4.4	6302 ms	0.52
tapatalkxda	Forum	Communication	3.10 MB	22 MB	4.4	7924 ms	0.38
TunnyBrowser	Dolphin web browser	Communication	7.50 MB	18 MB	4.7	2415 ms	0.84
firefox	Firefox web browser	Communication	24.00 MB	48 MB	4.4	2477 ms	0.83
mathway	Math education	Education	12.58 MB	16 MB	4.5	4051 ms	0.72
reading	Preschool learning	Education	23.03 MB	13 MB	4.5	6003 ms	0.55
ted	TED's official app	Education	4.80 MB	38 MB	4.6	3944 ms	0.73
imdb	Movie information	Entertainment	4.64 MB	29 MB	4.4	4621 ms	0.66
talkingtom	Talking game	Entertainment	23.00 MB	74 MB	4.5	5808 ms	0.57
flixster	Movie information	Entertainment	6.79 MB	8 MB	4.6	3498 ms	0.76
bb2012	Baseball game	Sports Games	37.00 MB	55 MB	4.6	16165 ms	0.70
cartoongolf3d	3D golf game	Sports Games	18.61 MB	39 MB	3.4	5470 ms	0.60
bowling	3D bowling game	Sports Games	10.59 MB	29 MB	4.4	6002 ms	0.55
translate	Google translate	Tools	4.29 MB	12 MB	4.6	1610 ms	0.90
inkpad	Notepad	Tools	0.81 MB	17 MB	4.4	1483 ms	0.91
RealCalc	Scientific calculator	Tools	0.30 MB	10 MB	4.7	1687 ms	0.89
kayak	Hotel and Flights	Travel/Local	14.71 MB	37 MB	4.5	3682 ms	0.75
mapquest	Map	Travel/Local	2.90 MB	38 MB	4.3	4758 ms	0.66
skyscanner	Flight information	Travel/Local	7.49 MB	15 MB	4.6	1423 ms	0.91

+see [25] for full names. *Category and Rating come from Google Play.

nent the non-linearity and app-dependence are, and how much disparity they cause between average delays and user experienced responsiveness. The rest of this section answers both questions.

Statistical Analysis

We conduct a statistical analysis to find out the significance of the non-linearity and app-dependence.

For non-linearity, we apply a T-test on linear regression models for the data in each of the delay-sat graphs. T-test is a standard way to test for a significant linear regression relationship between the response variable and the predictor variables. It reports a p-value for each coefficient in the regression model, which indicates the significance of having that corresponding term in the regression model. The smaller the p-value is, the more likely that the term exists in the actual delay-sat relation. A commonly used significance threshold is 0.05. In our regression results, only on four apps, *WMWLite, bb2012, megajump, angrybirds*, the p-values of the quadratic coefficient are greater than 0.05. On most other apps, the p-value is even smaller than 0.00001, suggesting that it is statistically impossible for the actual relationship between delay and SatScore to be linear.

For app-dependence, we conduct two quantitative analyses. The first is to examine how significantly different the apps are at three anchor points, which are the points when the satisfaction score equals 0.25, 0.5, 0.75. The idea is that if the apps differ much at these anchor points, then their relationship between delay and satisfaction must be different. Table 3 displays the launch delays of those apps at those anchor points, obtained through a linear interpolation on the data collected in the user study. It shows large differences across apps, reflecting the different degrees of delay tolerance to different apps. On the translation tool, *translate*, for instance, most people expect a prompt response; when its delay is around 4.5 seconds, only 25% of people feels satisfied with its responsiveness. On the other hand, on the village building game, *smurfsandroid*, 75% of people feels satisfied even when it takes 5 seconds to launch.

To further confirm the app dependence, we conduct a focused study on two apps, *TunnyBrowser* and *cartoongolf3d*. We collect about 700 extra data samples from more participants on these two apps. We consider two anchor points where satisfaction scores equal 0.25 and 0.5 respectively.

For each of the two apps, we partition its data into 12 sets, with each set containing the samples from 10 participants. Based on each set, we obtain the delay corresponding to the two anchor points. In total, we get 12 pairs of delays at the anchor points for each of two apps. We then put those points into a 2-D graph, with the X-axis for the delay at anchor point (sat=0.25) and the Y-axis for the delay at anchor point (sat=0.5). Figure 3 shows the graph. The data on the two apps show

Table 3. Launch delay (in seconds) at three anchor points when the satisfaction score equals 0.25, 0.5, and 0.75.

app	0.25	0.5	0.75
mathway	10.4	5.5	2.6
smurfsandroid	14.1	6.6	5.0
WMWLite	11.6	8.4	3.9
busparking3d	13.4	7.4	5.3
bb2012	12.9	8.0	5.5
megajump	13.3	6.4	1.6
translate	4.5	3.4	1.1
stardroid	11.2	8.1	6.6
fruitninjafree	11.4	6.0	3.6
imdb	11.2	7.8	6.9
reading	8.5	5.7	3.8
kayak	8.8	7.0	3.0
candycrushsaga	12.3	7.2	4.2
cartoongolf3d	13.9	9.4	5.0
mapquest	9.5	6.6	4.2
merriamwebster	8.3	7.0	3.7
talkingtom	7.7	4.5	2.5
tapatalkxda	13.7	9.5	4.7
angrybirds	13.6	5.6	3.4
restaurantstory	7.7	5.5	1.6
ted	8.5	6.0	3.5
bowling	10.6	4.9	1.9
inkpad	13.3	5.9	3.1
Puzzle.Animal	5.7	3.2	2.5
TunnyBrowser	8.5	5.1	2.3
flixster	6.5	5.9	4.2
skyscanner	7.8	5.6	2.1
firefox	8.0	4.0	1.4
wikipedia	8.3	4.3	3.1
RealCalc	7.6	3.1	1.9
Max	14.08	9.48	6.95
Min	4.49	3.13	1.11
Mean	10.09	6.12	3.48
Std	2.71	1.68	1.50

a clear disparity in distribution. A two-sample T-test confirms that it is virtually impossible for the two apps to have the same SatScore distribution (p-value equals 0.0000048).

Overall, these statistical analyses confirm that *non-linearity* and *app-dependence* are statistically significant in the relationship between launch delay and user experienced responsiveness. But it still remains unclear whether the two properties are prominent enough to cause significant disparity between average delay reduction and user experienced responsiveness. We next answer this ultimate question through a comparative study.

Implications to Responsiveness Metrics

To examine whether average delay is a satisfactory responsiveness metric, it would be informative to see whether a technique actually enhances user's satisfaction scores more if it reduces more average launch delay of the 30 real apps. To keep the examination general, we try to avoid limiting the investigation to a particular enhancement technique. Instead, we assume that there are some techniques that, through whatever approach, are able to reduce the average response delay of the 30 apps by $x\%$. We compare the enhancement of satisfaction scores under two cases:

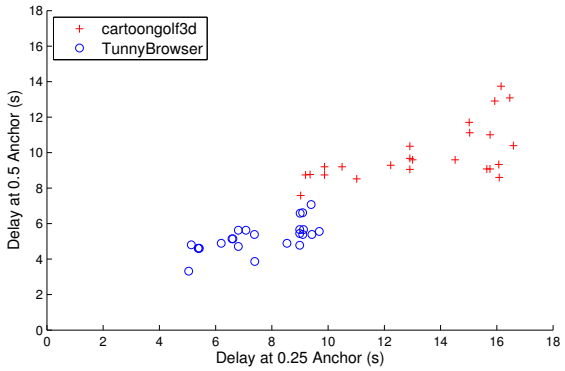

Figure 3. Delays at anchor points (sat score=0.25 and 0.5) on apps *cartoongolf3d* and *TunnyBrowser*.

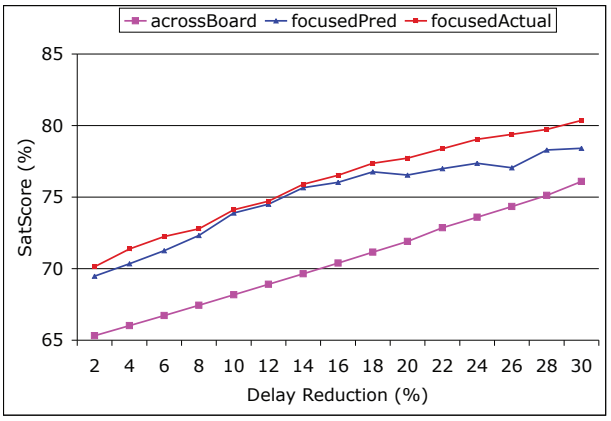

Figure 4. Comparison of the influence on SatScore by different schemes of delay reduction.

1. *AcrossBoard:* The delay reduction is oblivious to app differences; every app gets $x\%$ delay reduction.

2. *FocusedActual:* This case focuses delay reduction according to the potential of an app for SatScore improvement. Here, potential is defined as the slope of the delay-sat curve at the current delay. If the slope is large, a minor delay reduction would lead to much enhancement of SatScore, and the app is considered to have a high potential. This policy works iteratively. In each iteration, it reduces $(0.1 * x\%)$ of the delay of the app that has the largest potential, and then updates the potential of the app. It stops when all the reduction budget $(30 * x\%)$ has been used up. The slope at a point is computed as follows. We first obtain the SatScores at 18 anchor points through linear interpolation on the delay-sat graph of the app. The 18 anchors have delay values evenly distributed between 0 and 18 seconds. The slope at an arbitrary point is computed from its two surrounding anchors.

The bottom and top curves in Figure 4 report the average SatScore values of the 30 apps after a spectrum of reduction of the average delay by each of the two methods respectively. (Later sections will discuss the middle curve.) The first point on the bottom curve, for instance,

shows that the average SatScore of the 30 apps becomes 65.3% when there is a 2% across-board delay reduction.

The curves demonstrate that comparisons of response delay reductions can easily lead to wrong conclusions on the effectiveness of the two schemes. Looking horizontally, one can see that to reach a particular SatScore, the acrossBoard scheme needs to cut the average response delay by about twice as much as what the focusedActual scheme needs. For instance, to enhance the average SatScore to 72%, the focusedActual scheme needs to cut the average launch delay by 8%, while the acrossBoard scheme needs to cut it by 22%. Now consider two techniques, techA and techB. Suppose that techA cuts the average delay of every app by 22%, while techB cuts the average delay by x ($8\% < x < 22\%$) in the focusedActual manner. From Figure 4, we know that techB gives a higher SatScore than techA does and hence is more effective in enhancing user experienced responsiveness. However, if we use the average delay reduction as the metric, we would reach the opposite conclusion. The range of x (8-22%) in which such a disparity could happen is quite substantial, underscoring the unreliability of average delay reduction as a responsiveness metric.

Overall, the study shows that although delay and SatScore exhibit certain linearity when the delay is in a medium range, in general, the non-linearity and app-dependence of the delay-sat relationship are so prominent that average delay reduction is, in many cases, a misleading metric for assessing techniques in enhancing user experienced responsiveness.

SatScore is a better alternative as it measures user experience directly. However, directly using it requires some user studies, doing which is time consuming and often impractical on every launch delay for every app. We propose several solutions to circumvent the problem, as presented next.

SATSCORE MODELING FOR PRACTICAL USAGE

Our solutions for making SatScore a metric easy to use centers around the construction of the delay-sat functions of mobile apps. The basic idea is that if we can find out the delay-sat relationship and characterize it with a function, it would be possible to automatically convert the easy-to-measure metric, response delay, into SatScore, the metric that more directly connects with user experience. We call attainment of such delay-sat functions as *SatScore modeling*. This section discusses whether SatScore modeling can be built or predicted easily. We explored three methods, with respective cons and pros, suiting different usage scenarios.

Offline App-Specific Method

This method is designed to use when user studies are affordable for every app of interest (but not for every possible launch delay). It employs statistical curve fitting on a set of (delay, SatScore) tuples on the target app. The tuples are collected through a user study similar to the one described in the previous section. Through MagicLauncher, launches of a spectrum of delays of each app are conducted, and the SatScores at those launches are collected from users responses. After getting all the tuples, this method applies curve fitting on the data set to build up the delay-sat function for that app. We have explored both polynomial functions and sigmoid functions on the thirty apps included in this study, and found that sigmoid functions provide a better fit in general. Results will be reported later in this section.

This method requires a user study for every app, and builds the delay-sat functions offline. These constraints do not prevent it from being used for rigorous assessment, in which, there is usually a fixed set of apps for testing and offline function construction is typically not a concern. The user study does require some man-power, but once it is done, the fitted curves can be used for many assessments—during which, only launch delays are needed to measure, as the SatScores can be derived from the fitted curves accordingly.

However, the outcome of this method cannot be used for a new app—which is sometimes necessary, especially for guiding runtime responsiveness enhancement. For instance, if one wants to modify the app scheduler in Android to focus response delay reduction on some apps whose SatScore is most sensitive to launch delay reduction, the scheduler would need to tell, at runtime, the SatScore of an arbitrary app at a given delay. The set of apps in a system can be large and continuously evolve. In addition, during the development stage of a responsiveness optimization technique, some quick, even though maybe slightly off, assessment of the responsiveness on some new apps could be desirable to get some sense of the effectiveness of the technique. The next method tries to meet these requirements.

Online App-Specific Method

This method is based on the observations from Figure 2 that app features may give some hints on their delay-sat relations. For instance, apps (e.g., Puzzle and RealCalc) with smaller package sizes and footprints tend to be more sensitive when their launch delays are less than 5 seconds. Default delay shows a similar impact. In addition, apps of some categories (e.g., entertainment) tend to be more tolerable to delays than some others, plausibly because they are used in less urgent contexts.

The observations lead to the following idea: If we can build up a cross-app predictive model to capture the influence of app properties on delay-sat functions, we may be able to predict the delay-sat function of a new app on the fly based on its features.

To test this idea, we develop a machine learning framework. We first explain our prediction target. Our objective is the delay-sat function of a given app. Given the good fit of sigmoid functions with the sampled data, a straightforward thinking is to predict the parameters of the sigmoid function and then use the resulting sig-

feature	type	description
pkgSz	numer.	package size
memSz	numer.	memory footprint size
rating	numer.	Google Play rating
delay	numer.	default launch delay
domain	categ.	category in Google Play

(a) Feature list

```
% F: feature matrix with each row for one launch, each column for one feature;
% Y: target matrix with each row for one launch, each column for one target parameter;

function modelBuildTool()
    % feature normalization and categorical feature encoding
    dataPreprocessing();

    % loop through columns of Y to build model for each
    for (i=0; i< columns(Y); i++){
        y = Y(:,i); % current column of Y
        rst(:,i).models = regressionWithCrossValidation(y, F);
    end
end
```

(b) High-level function in the machine learning framework

Figure 5. Outline of the machine learning framework and the feature set.

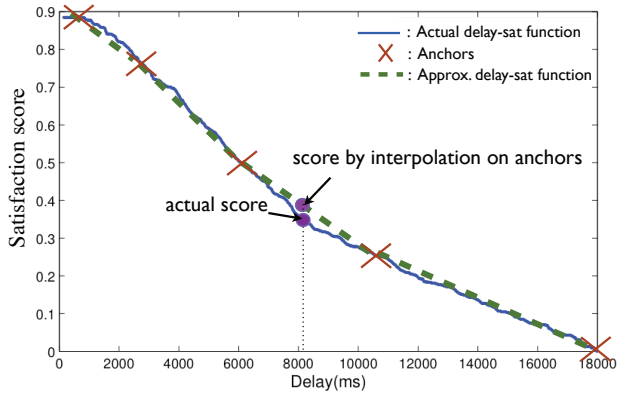

Figure 6. Illustration of anchor-based method for approximating delay-sat functions.

moid function as the predicted delay-sat function. Our exploration, however, shows that this method often leads to large prediction errors because even though the whole function has some strong correlations with the app properties, the correlations between each parameter and the app properties are quite weak.

We instead found an anchor-based method more effective. In this method, the machine learning framework tries to build models that can predict the SatScores at a handful of launch delays for an arbitrary app. These handful of predicted tuples (delay, SatScore) form the set of anchors in the predicted delay-sat function. For an arbitrary response delay of the app, the corresponding satisfaction score can be easily obtained through linear interpolation on the nearby anchors; this anchor-based interpolation is also called piecewise regression. Figure 6 gives a simple illustration. There are five anchors whose values are predicted based on the app features. For a given delay (e.g., 8100ms), the piecewise regression gets its corresponding SatScore by using the two anchors surrounding the delay to do a linear interpolation. For the example shown in Figure 6, the predicted SatScore is about 0.02 higher than the actual.

We develop the machine learning framework in Matlab by putting together some standard statistical methods. Figure 5 provides the set of app features the framework uses and the outline of the top-level subroutine of the framework. The app features are selected for their ease to access, and their potential relevance to user experience observed in our user study.

The first step in the framework involves two preprocessing tasks: It normalizes the values of the four numerical features to the range between 0 and 1, and encodes the categorical feature "domain" with a boolean matrix (each column for one category). The second step builds up the predictive models through regression. Every iteration builds up the models that map from the app feature values to one of the anchors. An invocation of the function *regressionWithCrossValidation* produces a polynomial regression model. We employ the standard step-wise feature selection in the linear regression, and include both linear and quadratic models in the set of candidate models. The tool finds the best subset of features and the best model automatically such that the sum of squared regression errors can be minimized.

During the construction of the models, function *regressionWithCrossValidation* uses ten-fold cross validation to examine the quality of the predictions by the model. Ten-fold cross validation is a standard machine learning method for model assessment. It partitions the data into 10 chunks, builds and tests the regression models 10 times, with each time using a different chunk of data for testing and the other nine chunks for training. The results enable the selection among different orders of polynomial models for each anchor point. The measure used for quality assessment is mean square error as to be explained in the next Evaluation section.

This model construction step only needs to be done once on some training apps. After that, the predictive models can be easily used for other apps; it takes the properties of the app as input and outputs the predicted values for the anchor points. The prediction only involves the computation of a handful of quadratic or linear functions. The overhead is negligible (less than 1% of any app launch in the study), making the method suitable for runtime exploitations of the predicted delay-sat functions.

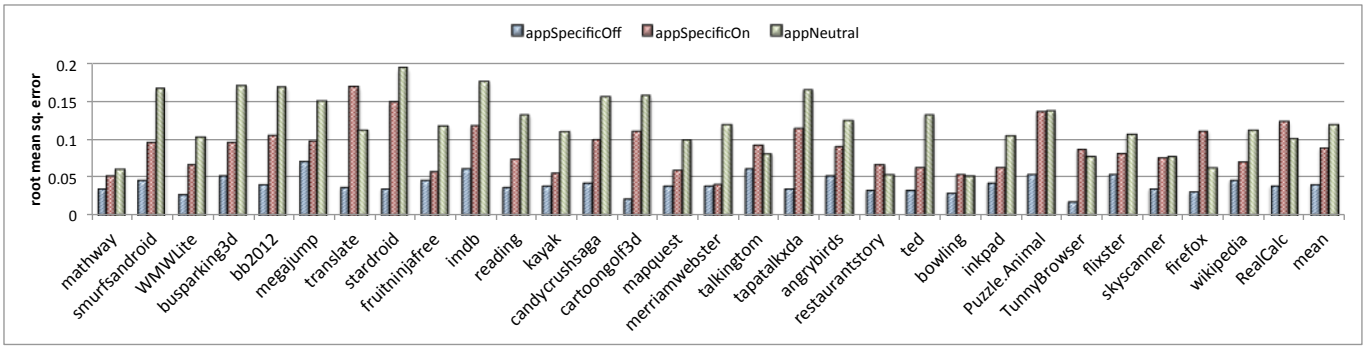

Figure 7. The root mean square errors of the delay-sat functions produced by the three methods (18 anchors are used in the method *appSpecificOn*).

It is worth noting that the linear models used in this method do not conflict with the non-linearity mentioned in previous sections. That non-linearity is about the relationship between delay and satisfaction scores, while here, the linear models are about the relationship between app features and those anchor values.

Offline App-Neutral Method

For comparison purpose, we implement an app-neutral method, which builds up a single delay-sat function and uses it for an arbitrary app. It includes two steps. The first step derives the average (across all training apps) delay at a series of SatScore values between 0 and 1. For each SatScore, it gets the corresponding delay of every training app by looking for the SatScore value in its collected data. If that score cannot be found in the samples, it uses the linearly interpolated value of the two nearest scores. After getting the delays for all training apps at a satisfaction score, it computes the mean of them. This step finally yields a set of (satisfaction score, average delay) tuples. The second step applies sigmoid curve fitting to the tuple set to find a sigmoid function that matches with the data well. That function is the final app-neutral delay-sat function.

Evaluation

In this part, we report the effectiveness of the three methods in deriving SatScores from the delays collected on the 30 apps in Table 2. The evaluation uses *root mean squared error (rmse)*, a standard measure in statistics, defined as follows:

$$rmse = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(y_i - y_i')^2},$$

where, y_i and y_i' are the actual and predicted satisfaction scores at the ith sample, and N is the total number of samples. The rmse is in the same unit as the satisfaction score. In other words, rmse indicates the average difference between the actual satisfaction score at a response delay and the predicted score at that delay.

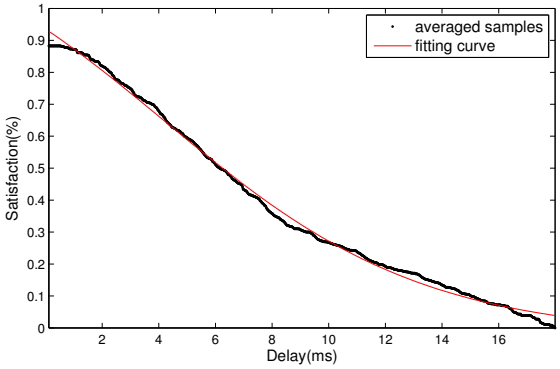

Figure 8. Average points across apps and the sigmoid curve fitted by the app-neutral method.

We use 10-fold cross validation (explained in the previous subsection) for every method to ensure the separation of training data and testing data.

Results

The smooth curves in Figure 2 show the sigmoid curves the first method finds for each of the apps. The first group of bars in Figure 7 report that the rmse of the fitted curves on the samples collected in our user study are all less than 0.05 except *talkingtom*. From its graph in Figure 2, we can see that many points of this app have similar low SatScores in a long range of delays, causing disparity from the fitted curve. The overall result is quite accurate, with an average 0.04 rmse.

Figure 8 shows the average points across apps and the fitted app-neutral sigmoid curve by the third method. The fitting quality is good, but because the curve reflects the average delay-sat relationship across apps, it differs much from the data of each individual app. As the third groups of bars in Figure 7 show, the curve has over 0.15 rmse on nine apps and an average 0.12 rmse.

As the second group of bars in Figure 7 show, the second method (on 18 anchors per app) yields smaller rmse than the app-neutral method does on 24 out of the 30 apps. On average, its rmse is 0.08, about two thirds of the rmse of the third method, indicating the benefits of being

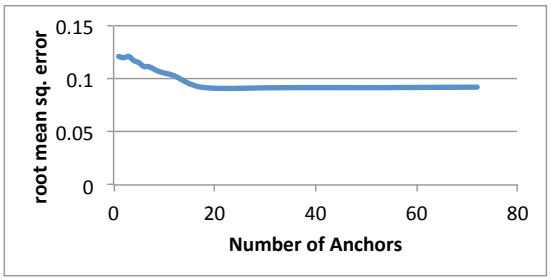

Figure 9. Sensitivity of online app-specific method on number of anchors.

app-specific. Its rmse is 0.04 larger than that of the first method, due to the errors in the anchor prediction and anchor-based linear interpolation. Figure 9 reports the results when different numbers of anchors are used. More anchors lead to smaller rmse; however, the return diminishes when the number gets greater than 18. So 18 anchors are enough to capture the main trends in the delay-sat functions.

Overall, the first method is the most accurate and is preferred for rigorous responsiveness assessment when user studies are feasible for each test app; the other two methods are both applicable for quick assessment or runtime exploitation of delay-sat relationship, with the second method preferred for its higher accuracy brought by its adaptivity to app properties.

Influence of the Errors

One of the advantages of having delay-sat functions is that a responsiveness enhancement can be applied in a more focused manner. In this part, we examine how the approximation errors affect the capitalization of such an advantage. We use a delay reduction scheme similar to the *FocusedActual* scheme described earlier, but instead of using the actual delay-sat relations, we use the predicted ones. More specifically, when computing the slope of the delay-sat curve, we use the delay-sat function predicted with the *online app-specific method*. We call this scheme *FocusedPred*. The average SatScores of the apps by this method is shown as the middle curve in Figure 4. The clear gap from the "acrossBoard" curve indicates that the predicted delay-sat functions are promising for guiding the design of responsiveness enhancement by letting them focus on apps whose SatScores are more sensitive to their delays. The prediction errors throttle some benefits. However, in most cases, the difference from what the *FocusedActual* curve provides (i.e. what the actual delay-sat curves offer) is less than 1.5%; the largest is 2.3%. So, overall, the online app-specific method offers an easy way to derive SatScore models that give a reasonable accuracy for quick responsiveness assessment, as well as good promise for guiding online app-adaptive enhancement of launch responsiveness.

RELATED WORK

Modeling user experience is a topic spanning across multiple disciplines. In marketing, customers' satisfaction directly influences business gains [5, 10, 20]. Modeling in this area has been mainly focused on analyzing various potential factors and how much the delay would affect the service evaluation [10,20], rather than building quantitative models to predict customers' satisfaction.

There are some studies focused on web responsiveness [2, 8, 17]. Olshefski and others have proposed an approach to measuring the client perceived response time more accurately [17], but have not explored how the perceived response time affects end users' satisfaction. Hoxmeier and Dicesare [8] have found that slow system response time leads to dissatisfaction, and tried to identify the point at which users turn dissatisfied. In their study, they did not explore the design or (either online or offline) construction of predictive models for delay-satisfaction relationship.

There are some studies on finding out the satisfaction thresholds for responsiveness of some services [1,7,15,16]. Some of them have mentioned the differences between requests [1] but given no systematic, quantitative study.

There have been a number of recent studies trying to leverage app usage patterns in app prelaunching or content prefetching for improving responsiveness [9, 18, 19, 24]. Another direction to improve the responsiveness is through disk optimizations [11, 12, 21]. They have used launch delay reduction as the responsiveness metric. With the findings in this paper, one may better assess the user perceived effects of responsiveness enhancement.

In program analysis, recent years have seen some studies on predicting sequences of program behaviors [**?**, 22]. The techniques could potentially get combined with app launch sequence prediction.

CONCLUSIONS

We draw the following conclusions from the study. First, using delay as a responsiveness metric is sometimes acceptable, especially when the delay is medium. However, it is risky in general. Due to the significance of non-linearity and app-dependence of the relationship between delay and user experience, using only that metric could get misleading conclusions on the perceived effects of responsiveness enhancement. Second, SatScore offers a valuable alternative metric. To use it, a SatScore model needs to be built or predicted for each target app. The paper provides ways to build the model either online or offline. The former is easy to use while the latter is more rigorous. Both methods can provide SatScores that better reflect user experience than average delay reduction does. Finally, the online method, by creating the SatScore model on the fly, opens opportunities for designing techniques that leverage the different SatScore models of different apps to enhance launch responsiveness in an adaptive and selective manner.

Acknowledgment

We thank David Chu for his help with the final version of the paper. We thank the anonymous reviewers and Qun Li for all the valuable comments. Jianhua Sun helped

the implementation of MagicLauncher in the early stage; Liang Chen, Lifen Zhang, and Jacqulyn Johnson helped organize the user study. This material is based upon work supported by the National Science Foundation under Grant No. 1320796 and CAREER Award and DOE Early Career Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the agencies.

REFERENCES

1. Department of defense design criteria standard: Human engineering. page 196, 1999.
2. Daniel Andresen and Tao Yang. Multiprocessor scheduling with client resources to improve the response time of www applications. In *Proceedings of the 11th international conference on Supercomputing*, ICS'97, pages 92–99, New York, NY, USA, 1997. ACM.
3. Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. Quantifying skype user satisfaction. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'06, pages 399–410, New York, NY, USA, 2006. ACM.
4. Compuware. Mobile apps: What consumers really need and want. *Survey Report*, 2012.
5. AgneAs Durrande-Moreau. Waiting for service: ten years of empirical research. *International Journal of Service*, 10(2):171–189, 1999.
6. Jake Brutlag Eric Schurman. Performance related changes and their user impact. http://velocityconf.com/velocity2009.
7. D. Galleta, R. Henry, S. Mccoy, and P. Polak. Web site delays: How slow can you go? *Journal of Association for Information Systems*, page 28, 2002.
8. John A. Hoxmeier and Chris Dicesare. System response time and user satisfaction: An experimental study of browser-based applications. In *Proceedings of the Association of Information Systems Americas Conference*, pages 10–13, 2000.
9. Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen. Predicting mobile application usage using contextual information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp'12, pages 1059–1065, New York, NY, USA, 2012. ACM.
10. Michael K. Hui and David K. Tse. What to tell consumers in waits of different lengths: An integrative model of service evaluation. *Journal of Marketing*, 60(2):81–90, April 1996.
11. Yongsoo Joo, Youngjin Cho, Kyungsoo Lee, and Naehyuck Chang. Improving application launch times with hybrid disks. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS'09, pages 373–382, New York, NY, USA, 2009. ACM.
12. Yongsoo Joo, Junhee Ryu, Sangsoo Park, and Kang G. Shin. Fast: quick application launch on solid-state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 19–19, Berkeley, CA, USA, 2011. USENIX Association.
13. Ron Kohavi and Roger Longbotham. Online experiments: Lessons learned. *IEEE Computer*, 40(9):103–105, 2007.
14. Haohui Mai, Shuo Tang, Samuel T. King, Calin Cascaval, and Pablo Montesinos. A case for parallelizing web pages. In *Proceedings of the 4th USENIX conference on Hot Topics in Parallelism*, HotPar'12, pages 2–2, 2012.
15. Fiona Fui-Hoon Nah. A study on tolerable waiting time: How long are web users willing to wait? In *AMCIS*, page 285. Association for Information Systems, 2003.
16. Jakob Nielsen. Response times: The 3 important limits. *Usability Engineering*, 1993.
17. David Olshefski and Jason Nieh. Understanding the management of client perceived response time. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS'06/Performance'06, pages 240–251, New York, NY, USA, 2006. ACM.
18. Abhinav Parate, Matthias Bohmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp'13, 2013.
19. Choonsung Shin, Jin-Hyuk Hong, and Anind K. Dey. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp'12, pages 173–182, New York, NY, USA, 2012. ACM.
20. Shirley Taylor. Waiting for service: The relationship between delays and evaluations of service. *Journal of Marketing*, 58(2):56–69, April 1994.
21. Paolo Valente and Mauro Andreolini. Improving application responsiveness with the bfq disk i/o scheduler. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR'12, pages 6:1–6:12, New York, NY, USA, 2012. ACM.

22. B. Wu, Z. Zhao, X. Shen, Y. Jiang, Y. Gao, and R. Silvera. Exploiting inter-sequence correlations for program behavior prediction. In *Proceedings of the annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2012.

23. Ye Xu, Mu Lin, Hong Lu, Giuseppe Cardone, Nicholas D. Lane, Zhenyu Chen, Andrew T. Campbell, and Tanzeem Choudhury. Preference, context and communities: a multi-faceted approach to predicting smartphone app usage patterns. In *ISWC*, pages 69–76. ACM, 2013.

24. Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys'12, pages 113–126, New York, NY, USA, 2012. ACM.

25. Z. Zhao, M. Zhou, and X. Shen. Satscore: Uncovering and avoiding a principled pitfall in responsiveness measurements of app launches. Technical Report WM-CS-2014-04, College of William and Mary, 2014.