모바일 앱 스피드 인덱스 측정 도구 설계 및 구현

Design and Implementation of Mobile Application Speed Index Measuring Tool

저자
(Authors)

유현식, 문현수, 이영석
Hyunsik Yoo, Hyunsu Mun, Youngseok Lee

출처
(Source)

정보과학회논문지 45(11), 2018.11, 1135-1141(7 pages)
Journal of KIISE 45(11), 2018.11, 1135-1141(7 pages)

발행처
(Publisher)

한국정보과학회
The Korean Institute of Information Scientists and Engineers

URL

http://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE07556787

APA Style

유현식, 문현수, 이영석 (2018). 모바일 앱 스피드 인덱스 측정 도구 설계 및 구현. 정보과학회논문지, 45(11), 1135-1141

ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 45, No. 11, pp. 1135-1141, 2018. 11
https://doi.org/10.5626/JOK.2018.45.11.1135

모바일 앱 스피드 인덱스 측정 도구 설계 및 구현

(Design and Implementation of Mobile Application Speed Index Measuring Tool)

유 현 식 [†] 문 현 수 [††] 이 영 석 [†††]
(Hyunsik Yoo) (Hyunsu Mun) (Youngseok Lee)

요 약 모바일 앱의 속도는 사용자 경험에 큰 영향을 미치기 때문에 개발자들은 사용자의 만족도를 높이기 위하여 앱의 속도를 고려한다. 하지만 안드로이드 개발 도구가 제공하는 앱 속도 측정 방법은 다운로드 컨텐츠의 렌더링 시간을 측정하지 못 하기 때문에 사용자가 실제로 느끼는 속도를 알려주지 못 한다. 본 연구는 웹의 성능 지표인 스피드 인덱스를 모바일 앱에 적용하여 모바일 앱의 속도를 측정하는 방법을 제안한다. 모바일 앱 스피드 인덱스 계산을 위하여 무작위 사용자 이벤트 생성기와 속도 측정기를 개발하였으며 이미지 유사도 비교를 통하여 렌더링 완료를 파악하였다. Google Play Store 인기 차트 1093개의 앱을 대상으로 스피드 인덱스를 계산하여 8%의 느린 앱을 발견하였다.

키워드: 모바일 앱, 스피드 인덱스, 렌더링, 속도 측정

Abstract Because the speed of mobile applications(apps) has a considerably large impact on user experience, developers consider increasing app speed to improve user satisfaction. However, the measurement method employed in the Android development tool does not indicate the actual speed experienced by the user because it does not measure the rendering time of the download content. This study proposes a method to measure the speed of a mobile app by applying a speed index, which is a web performance index, to a mobile app. In this study, we developed a random user event generator and a mobile speed analyzer. By calculating the speed index for 1093 apps on the Google Play Store Top Charts, we found that 8% of the apps were slow.

Keywords: mobile app, speed index, rendering, speed measurement

· 이 연구는 충남대학교 학술연구비에 의해 지원되었음

† 비 회 원 : 충남대학교 컴퓨터공학과
dbgustlr91@cnu.ac.kr
†† 학생회원 : 충남대학교 컴퓨터공학과
munhyunsu@cnu.ac.kr
††† 정 회 원 : 충남대학교 컴퓨터공학과 교수(Chungnam Nat'l Univ.)
lee@cnu.ac.kr
(Corresponding author임)

논문접수 : 2017년 12월 1일
(Received 1 December 2017)
논문수정 : 2018년 7월 31일
(Revised 31 July 2018)
심사완료 : 2018년 8월 6일
(Accepted 6 August 2018)

정보과학회논문지 제45권 제11호(2018. 11)

1. 서 론

스마트폰은 업무 활동 뿐 아니라 일상 생활에서도 필수적인 요소로 자리잡고 있다. 사용자들은 프로세서와 모바일 네트워크의 발전에 따라서 모바일 앱의 속도도 개선되길 기대한다. 모바일 앱 속도는 사용자 경험에 큰 영향을 미치기 때문에 앱 개발자들은 응답 시간이 길어지지 않도록 한다. [1]에서는 모바일 웹 응답 시간이 4.2초 이상이 걸릴 경우 일탈율이 급격하게 증가함을 제시하고 있다. 따라서 모바일 앱도 모바일 웹과 마찬가지로 사용자의 만족도를 높이기 위하여 앱의 속도를 고려해야 한다.

안드로이드 개발 도구는 모바일 앱의 속도를 측정할 수 있는 함수를 제공한다. 앱의 액티비티(화면)와 UI 쓰레드가 처리되는 시간을 측정하는 함수(DisplayedTime())와 렌더링 완료 시간을 측정하는 함수(reportFullyDrawn())가 제공된다. 측정 결과는 Logcat을 이용하여 ms 정밀도로 확인할 수 있다. 그림 1은 관광지를 소개하는 오픈소스 안드로이드 앱 'Travel Mate[2]'의 초기 로딩 화면을 시간 순서대로 나열한 것이다. DisplayedTime()과 reportFullyDrawn()이 1232ms에 호출되며 렌더링이 완료되었음을 개발자에게 알려준다.

하지만 안드로이드 개발 도구는 다운로드 컨텐츠의 렌더링을 측정하지 못하기 때문에 사용자가 실제로 느끼는 앱의 속도를 알려주지 못한다. 상황에 따라서 내용이 업데이트 되는 이미지나 동영상과 같은 리소스는 앱과 함께 스마트폰 내부에 설치되지 않고 외부 서버에서 다운로드하여 렌더링한다. 그림 1의 앱에서는 DisplayedTime()과 reportFullyDrawn()이 호출된 이후에 외부 서버에서 이미지를 다운로드하여 렌더링한다. 필요한 이미지를 모두 다운로드한 후 렌더링을 완료한 시간은 4000ms로 개발 도구가 알려준 1232ms와 3.24배 차이가 난다. 따라서 사용자가 실제로 느끼는 앱의 속도를 측정할 수 있는 도구가 필요하다.

스피드 인덱스는 WebPageTest.org[3]에서는 사용자가 웹 페이지에 접근할 때 렌더링이 얼마나 빨리 되는지 나타내는 지표다. 웹 브라우저의 화면을 동영상 형태로 녹화하고 영상을 프레임으로 나누어 프레임간 픽셀 단위 비교를 통해 렌더링 완료율을 계산한다. 이때 렌더링이 완료된 화면을 탐지하기 위하여 JAVASCRIPT onLoad() 함수를 사용한다. 웹에서는 렌더링이 완료되면 onLoad() 이벤트가 발생하기 때문에 이벤트가 발생한 시점에 녹화된 화면을 기준으로 프레임간 비교를 할 수 있다.

하지만 모바일에서는 웹에서의 onLoad() 이벤트를 활용할 수 없기 때문에 렌더링이 완료된 시점을 탐지할 수 있는 알고리즘이 필요하다. 우리는 렌더링 완료 시점을 탐지하기 위하여 UI automator XML 파일과 이미지 유사도를 활용하였다. 그리고 모바일 스피드 인덱스를 대규모로 측정하기 위해서는 스마트폰을 제어하고 화면을 녹화하기 위한 프로그램이 필요하기 때문에 각각의 프로그램을 개발하였다.

본 논문에서는 모바일 앱의 속도를 나타내는 모바일 앱 스피드 인덱스 측정 도구를 소개한다. 모바일 앱의 속도를 자동으로 측정하기 위하여 스마트폰에 모바일 앱을 설치하고 사용자 이벤트를 발생시킨다. 또한 우리는 이미지 유사도 비교를 사용하여 렌더링 완료 시간을 계산하였다. 이 도구를 이용하여 Google Play Store 인기 차트에 존재하는 앱 1093개의 속도를 분석하였다.

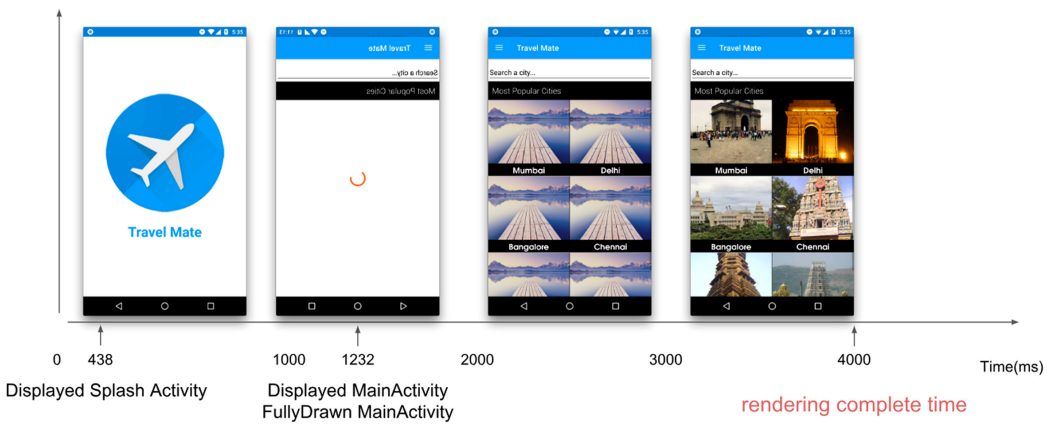

그림 1 앱 Displayed 신호와 렌더링 완료 시간의 차이
Fig. 1 Difference between app displayed signal and rendering completion time

2. 관련 연구

스마트폰과 원격 서버와의 통신이 모바일 앱 성능에 얼마나 영향을 주는지 분석하기 위하여 앱에서 발생하는 네트워크 트래픽을 측정하여 비교하는 연구가 진행되었다[4-6]. 모바일 앱의 트래픽을 발생시키기 위하여 APK파일을 디컴파일한 후 원하는 부분만 실행시키거나 직접 사용자 이벤트를 입력하여 트래픽을 수집한다. 하지만 소스코드 난독화가 되어있을 경우 디컴파일러를 활용할 수 없고 직접 사용자 이벤트를 입력할 경우 대규모 실험이 불가능하다는 한계가 존재한다.

스마트폰의 처리 속도가 모바일 앱 성능에 얼마나 영향을 주는지 분석하기 위하여 UI, 리소스 사용량을 분석하는 연구가 진행되었다[7-9]. 안드로이드 도구 중 하나인 UI automator를 사용하면 스마트폰 화면의 UI 컴포넌트(Layout, Button)를 XML형태로 추출할 수 있다. 부모 Main Frame부터 자식 Text/Image Frame까지 계층구조로 나와있는 UI XML파일을 이용하여 각 컴포넌트가 모바일 앱 성능에 주는 영향을 분석할 수 있다. 다만, 스마트폰 내부의 UI만을 분석하기 때문에 동적 광고나 원격 서버와의 통신은 고려하지 못한다.

모바일 앱의 성능을 개선하기 위하여 불필요한 렌더링을 예측하여 줄이거나 사용자의 컨텍스트를 미리 파악하여 리소스를 미리 불러오는 연구가 진행되었다[7,10,11]. 사용자 이벤트 데이터를 활용하여 사전에 준비하는 방법이다. 또한 모바일 앱이 사용하는 리소스와 네트워크 트래픽을 모두 고려하여 모바일 앱의 성능 개선을 유도하는 연구도 진행되었다[12,13]. 하지만 APK 디컴파일의 한계를 가지고 있거나 사용자 이벤트 발생에 대한 문제를 가지고 있다.

디컴파일이나 사용자 이벤트를 수동으로 입력하지 않고 안드로이드 Monkey를 활용하여 사용자 이벤트 생성을 자동화한 연구가 진행되었다[14]. 자체 스크립트 언어를 개발하여 UI 자동화를 이루었다. 한편 앱 분석에 초점을 두고 사용자 이벤트 자동화를 연구한 결과도 있다[15]. 스마트폰 화면내의 UI 컴포넌트를 트리 형태로 분석해내어 모바일 앱 테스트를 진행하였다. 우리는 사용자가 무작위로 UI 컴포넌트를 선택하는 것을 간주하였다. 따라서 Monkey를 활용하여 무작위로 사용자 이벤트를 발생시킨다.

3. 모바일 스피드 인덱스 측정 도구

모바일에서는 웹에서의 onLoad() 이벤트를 활용할 수 없기 때문에 렌더링이 완료된 시점을 탐지할 수 있는 알고리즘이 필요하다. 우리는 렌더링 완료 시점을 탐지하기 위하여 UI automator XML 파일과 이미지 유사도를 활용하였다. 그림 2는 Google Play Store 인기 앱 중 100개를 무작위로 선택하여 초기 로딩시간을 측정한 결과이다. 100%의 앱이 10초 이내로 초기 로딩이 완료되었다. 스마트폰 화면의 Layout XML 파일을 추출하는데 2~3초가 걸리기 때문에 우리는 Layout XML이 5번 연속 변화가 없으면 렌더링이 완료된 것으로 간주하였다.

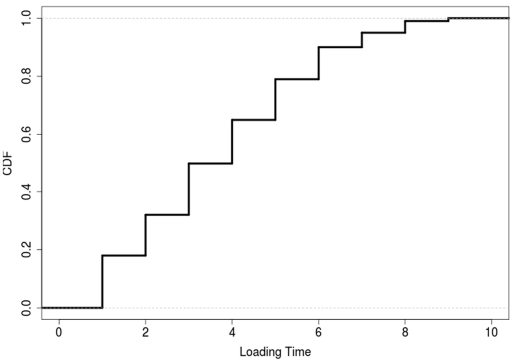

그림 2 100개 앱 초기 로딩 시간 분포.
Fig. 2 Initial loading time distribution of 100 apps

$$MobileSpeedIndex = \int_0^{end} 1 - VC$$

UI automator XML 파일을 이용한 대략적인 렌더링 완료 시점을 탐지한 후 정확한 렌더링 완료 시점을 알아내기 위하여 녹화 영상의 프레임간 이미지 유사도 비교를 하였다. 이미지 유사도 비교를 통해 렌더링 완료 시점을 탐지한 후 세션(사용자 입력이 가해진 후 렌더링이 완료된 시점까지의 시간)의 스피드 인덱스를 구할 수 있다. 위 수식은 모바일 스피드 인덱스를 구하는 공식으로 end는 렌더링이 완료된 시간을 나타내며 VC (Visually Complete)는 렌더링 완료 비율을 나타낸다. 그림 3은 아마존 앱 스피드 인덱스를 구하는 방법이다. 렌더링 완료 화면을 기준으로 이미지 유사도를 통하여 시각적 완료 비율(VC)를 구한 후 모바일 스피드 인덱스를 계산할 수 있다.

그림 4는 무작위 사용자 이벤트 생성기(Random User Event Generator)와 속도 측정기(Speed Analyzer)로 구성된 모바일 스피드 인덱스 측정 도구다. 무작위 사용자 이벤트 생성기는 APK 설치/삭제, 화면 녹화 등 스마트폰 제어를 담당한다. 속도 측정기는 녹화 영상을 기반으로 정확한 렌더링 완료 시간을 찾고 모바일 앱 스피드 인덱스를 계산한다. 표 1은 구현에 활용한 도구 목록이다. 각 프로그램은 [16]에 공개되어있다.

무작위 사용자 이벤트 생성기는 ADB(Android Debug Bridge)를 활용하여 스마트폰에 모바일 앱을 설치/삭제

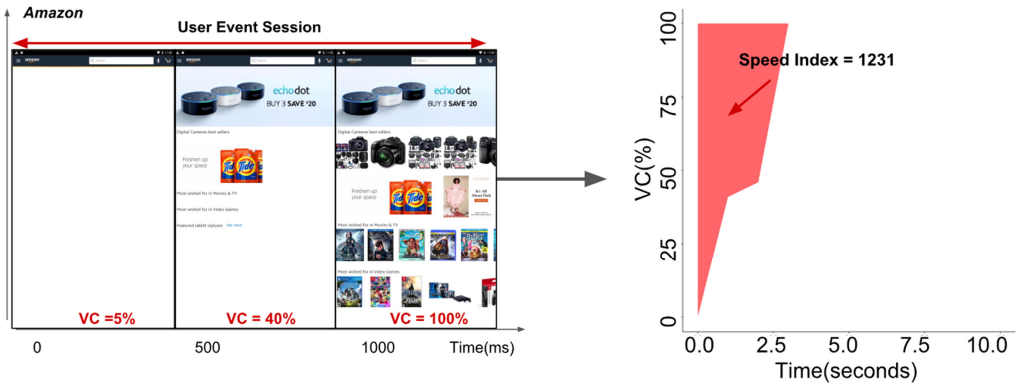

그림 3 아마존 앱 스피드 인덱스(VC: 시각적 완료 비율

Fig. 3 Amazon app speed index

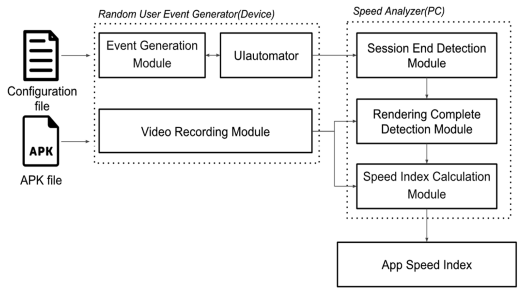

그림 4 시스템 구조

Fig. 4 System architecture

표 1 모듈과 구현방법

Table 1 Modules and Implementation

Category	Mobule name	Implementation
Random User Event Generator	Event Generation Module	ADB, Monkey
	UI Automator	UI Automator
	Video Recording Module	ADB
Speed Analyzer	Session Finish Detection Module	Python3
	Rendering Finish Detection Module	Python3, OpenCV
	Speed Index Calculation Module	Python3

하고 무작위로 입력 이벤트(터치, 스와이프, 줌인, 줌아웃)를 발생시킨다. 그림 5는 사용자 이벤트를 무작위로 발생시키는 흐름도다. UI automator가 추출한 스마트폰 화면의 Layout XML 파일을 분석하여 렌더링이 완료되었다면 ADB, Monkey를 활용하여 무작위 사용자 이벤트를 입력한다. 무작위 사용자 이벤트의 적중률을 높이기 위하여 Layout XML 파일을 분석한다. 그림 6은 앱 Layout XML의 일부로 node의 clickable, bounds가 정

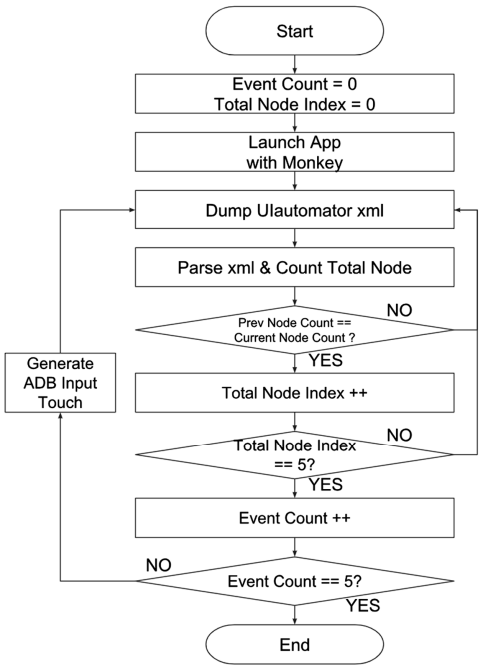

그림 5 사용자 이벤트 발생 흐름도

Fig. 5 User event flow chart

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
    <hierarchy rotation="0">
    <node index="0" text="" resource-id="" class="android.widget.FrameLayout"
    package="com.google.android.packageinstaller" content-desc=""
    checkable="false" clickable="false" enabled="true" focusable="false"
    focused="false" scrollable="false" long-clickable="false" password="false"
    selected ="false" bounds="[39,0][1041,1920]">
    <node .... >
```

그림 6 UI automator XML 파일

Fig. 6 UI automator XML File

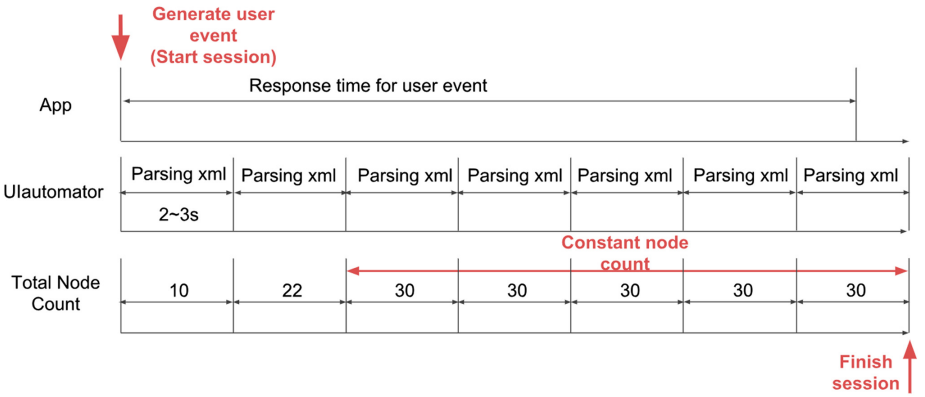

그림 7 렌더링 완료 시간 탐지
Fig. 7 Rendering finish time detection

의되어 있다. 우리는 clickable이 true인 node를 찾아 bounds 내부에서 이벤트를 발생시킴으로써 무작위 사용자 이벤트의 적중률을 높였다.

무작위 사용자 이벤트 생성기에서는 사용자 이벤트로 인한 렌더링이 완료되었는지 확인할 수 있어야 한다. 모바일 앱에는 웹 환경에서의 onLoad() 함수와 같은 도구가 없기 때문에 UI automator Layout XML 파일을 이용하였다. 그림 7은 렌더링이 완료되었는지 탐지하는 방법을 나타낸다. 사전 실험결과 100개 앱 모두가 10초 이내로 렌더링이 완료되었으며 XML 파일이 2~3초 마다 추출되기 때문에 5번 연속 XML 파일에서 나타나는 UI 컴포넌트 개수가 일정하면 렌더링이 완료되었다고 판단할 수 있다. 렌더링이 완료되었을 때 무작위 사용자 이벤트를 발생시켜 사용자 이벤트에 따른 모바일 앱의 속도를 측정할 수 있다.

무작위 사용자 이벤트를 발생시킴과 동시에 스마트폰의 화면을 녹화한다. 이 녹화 영상은 mp4 형태로 저장되며 모바일 앱 스피드 인덱스 계산에 활용된다.

속도 측정기에서는 모바일 앱 스피드 인덱스를 계산하기 위하여 무작위 사용자 이벤트 발생기에서 저장한 Layout XML과 mp4 영상 파일을 사용한다. Layout XML을 통해 구해낸 렌더링이 완료된 시점을 시작으로 0.1초 이전의 영상 스냅샷과 이미지 유사도를 비교한다. 시간의 역순으로 0.1초 이전 스냅샷과 비교를 진행하다 이미지 유사도가 임계치보다 낮아졌을 때 렌더링이 완료된 시점으로 판단한다. 우리는 렌더링이 완료된 것으로 측정하기 위한 임계치를 계산하기 위하여 Google Play Store 인기 차트 앱 100개를 무작위로 선택하여 렌더링 완료 적중률을 측정하였다. 그림 8과 같이 이미지 유사도 임계치를 88%로 두었을 때 정확도 90.7%로 오차율이 가장 낮았다.

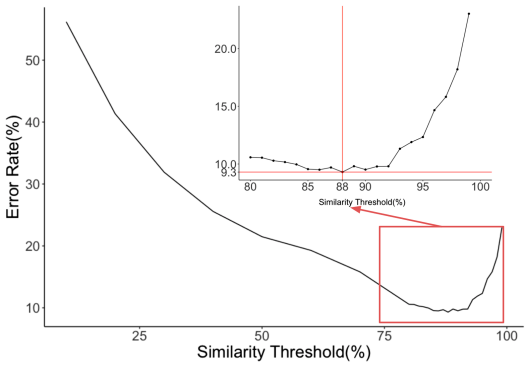

그림 8 유사도에 따른 에러 비율 그래프
Fig. 8 Error rate graph based on similarity

사용자 이벤트와 렌더링 완료 시점을 구해낸 것을 기반으로 모바일 앱 스피드 인덱스를 계산할 수 있다. 렌더링 완료 시점의 스냅샷과 사용자 이벤트 시작부터 0.1초 단위의 스냅샷사이의 이미지 유사도 비교를 통하여 시각적 완료 비율(VC)을 구할 수 있다. 각 사용자 이벤트 세션별로 렌더링 완료 시점을 계산하여 모바일 앱 스피드 인덱스를 계산할 수 있다. 이때 모바일 앱 실행부터 렌더링 완료까지의 모바일 앱 스피드 인덱스를 초기 인덱스라고 하며 유저 이벤트가 발생한 이후부터 렌더링 완료까지의 모바일 앱 스피드 인덱스를 런타임 인덱스라고 한다.

4. 실험 및 평가

우리는 Google Play Store 인기 차트 앱 1093개를 대상으로 모바일 앱 스피드 인덱스를 계산하였다. 실험 단말은 Galaxy A5(Android 6.0.1)를 사용하였으며 모바일 앱 별 사용자 이벤트는 5개씩 발생시켰다. 또한 스

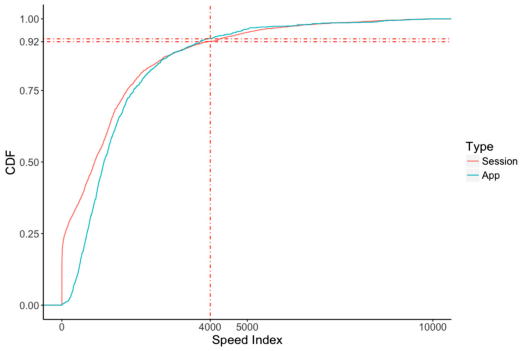

그림 9 세션, 앱별 스피드 인덱스
Fig. 9 Speed index by session and app

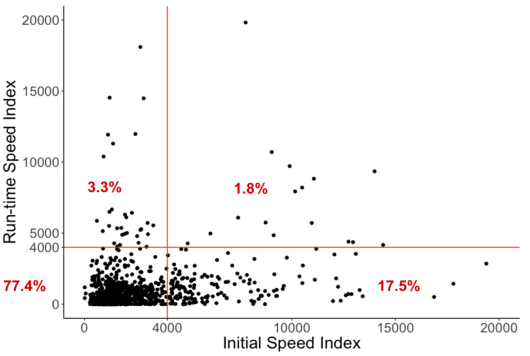

그림 10 초기 vs 런타임 스피드 인덱스
Fig. 10 Initial vs run-time speed index

마트폰은 WiFi에 연결하여 실험하였다.

하나의 앱마다 사용자 이벤트를 5개씩 발생시켰기 때문에 사용자 이벤트가 모두 적중할 경우 세션은 각각의 앱마다 최대 5개씩 생성될 수 있다. 우리는 모바일 앱 스피드 인덱스(최대 세션 5개의 평균)와 세션 스피드 인덱스의 분포를 살펴보았다. 웹에서 사용자가 불편함을 느끼는 렌더링 시간 4.2초는 스피드 인덱스 4000 앱의 렌더링 시간과 같다. 그림 9는 앱 스피드 인덱스와 세션 스피드 인덱스 분포를 나타낸다. 사용자가 느리다고 체감하는 스피드 인덱스 4000보다 큰 앱과 세션이 실험군에서 8%를 차지한다. 이는 스마트폰 성능이 발전했음에도 불구하고 사용자가 불편을 느낄 수 있을 만큼 느린 앱이 존재함을 의미한다.

한편 스피드 인덱스가 0에 가까운 세션이 25%나 발견되었다. 이것은 사용자가 이벤트를 입력한 즉시 렌더링이 완료된 것이다. 이러한 세션은 앱 실행시에 리소스를 모두 불러온 후 사용자 이벤트에 따라서 출력한다. 같은 원리로 5개의 세션 중에서 앱 실행 로딩을 의미하는 첫 세션의 스피드 인덱스는 다른 세션 스피드 인덱스보다 컸다(느리다.).

첫번째 세션이 그 이후의 세션들보다 얼마나 느리게 렌더링이 완료되는지 확인하기 위하여 초기 인덱스와 런타임 인덱스를 비교하였다. 초기 인덱스는 모바일 앱 세션 중 첫번째 세션의 스피드 인덱스를 의미한다. 런타임 인덱스는 첫번째 세션을 제외한 나머지 세션의 평균을 의미한다. 비교 기준은 웹에서의 기준을 모바일 스피드 인덱스와 맞춘 4000으로 정하였다.

그림 10은 실험 대상 앱의 초기 인덱스와 런타임 인덱스의 비교 분포를 나타낸다. 실험 앱의 77.4%는 초기 렌더링과 사용자 이벤트로 인한 렌더링이 모두 빨랐다. 하지만 초기에 많은 리소스를 불러오느라 느리게 반응한 후 사용자 이벤트에는 빠르게 렌더링이 되는 앱은 17.5%를 차지했다. 이러한 앱은 광고를 포함한 다량의

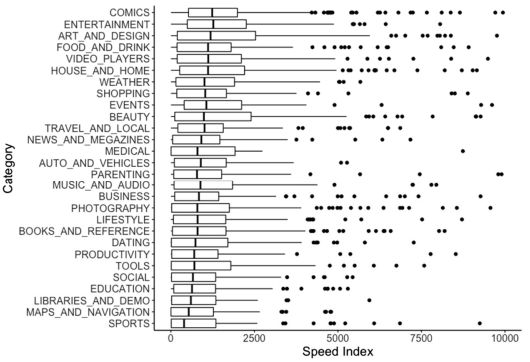

그림 11 카테고리별 스피드 인덱스
Fig. 11 Speed index by category

이미지를 원격 서버에서 받아오거나 큰 데이터를 로딩 화면에서 다운로드 받기 때문에 느린 것으로 분석되었다. 실험 대상 앱의 3.3%는 초기 스피드 인덱스는 낮으나(빠르나) 런타임 인덱스가 크다. 이러한 앱들은 사용 중 광고가 생성되거나 공지사항, 이벤트 알림 등으로 인하여 팝업 뷰가 생성되어 느린 것으로 분석된다. 1.8%의 앱은 초기에도 런타임에도 느린 것으로 분포된다. 느린 앱(세션)은 광고 수를 줄이거나 팝업 컨텐츠 크기를 줄이는 것으로 속도를 높일 수 있을 것이다.

우리는 모바일 앱 스피드 인덱스를 활용하여 앱 카테고리 별 성능 차이를 분석하였다. 그림 11은 실험 대상 앱 1093개의 카테고리와 스피드 인덱스를 비교한 결과다. 카테고리는 Google Play Store에 등록되어 있는 앱 정보를 사용하였다. 가장 느린 카테고리는 스피드 인덱스 평균 2278의 COMICS다. 가장 빠른 카테고리는 스피드 인덱스 평균 834의 SPORTS로 나타났다. COMICS 카테고리는 대부분 이미지로 구성되어있고 이미지들을 모두 서버에서 받아오기 때문에 다른 카테고리들에 비해 속도가 느리다. SPORTS 카테고리에서도 이미지가 주를

이루는 몇몇 앱이 존재하나 주로 경기 결과를 텍스트로 알려주기 때문에 가장 성능이 좋은 것으로 분석되었다.

우리가 소개한 실험 결과는 네트워크 환경(3G/4G/WiFi) 및 디바이스(저사양/고사양)에 따라서 변동될 수 있다.

5. 결론 및 향후 연구

우리는 모바일 앱의 성능을 평가할 기준을 마련할 수 있도록 웹에서 활용하는 스피드 인덱스를 모바일 앱에 적용시켰다. 스피드 인덱스를 계산하기 위하여 무작위 사용자 이벤트 생성기와 속도 측정기를 개발하였으며 Google Play Store 인기 차트 앱 중 1093개를 대상으로 스피드 인덱스를 계산하였다. 실험 대상 앱의 8%는 사용자가 느리다고 느낄 수 있음을 발견하였으며 초기 스피드 인덱스와 런타임 스피드 인덱스가 다름을 보였다. 향후 속도가 느린 앱의 원인을 분석하고 개선 방안을 제안할 수 있다.

References

[1] Google Impact on conversions and interaction, [Online]. Available: https://support.google.com/partners/answer/7336278

[2] Travel-Mate, [Online]. Available: https://github.com/Swati4star/Travel-Mate

[3] SpeedIndex(WebpageTest), [Online]. Available: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index

[4] Gabale, Vijay, and Dilip Krishnaswamy, "Mobinsight: On improving the performance of mobile apps in cellular networks," *Proc. of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee*, 2015.

[5] Li, Weichao, et al., "On the accuracy of smartphone-based mobile network measurement," *Computer Communications (INFOCOM), 2015 IEEE Conference on. IEEE*, 2015.

[6] Zhang, Lide, et al., "Panappticon: event-based tracing to measure mobile application and platform performance," *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on. IEEE*, 2013.

[7] Gao, Yi, et al., "Every pixel counts: Fine-grained UI rendering analysis for mobile applications," *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE. IEEE*, 2017.

[8] Liu, Yepang, Chang Xu, and Shing-Chi Cheung. "Characterizing and detecting performance bugs for smartphone applications," *Proc. of the 36th International Conference on Software Engineering. ACM*, 2014.

[9] Qian, Feng, et al., "Profiling resource usage for mobile applications: a cross-layer approach," *Proc. of the 9th international conference on Mobile systems, applications, and services. ACM*, 2011.

[10] Parate, Abhinav, et al., "Practical prediction and prefetch for faster access to applications on mobile phones," *Proc. of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, ACM*, 2013.

[11] Yan, Tingxin, et al., "Fast app launching for mobile devices using predictive user context," *Proc. of the 10th international conference on Mobile systems, applications, and services. ACM*, 2012.

[12] Chen, Qi Alfred, et al., "Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis." *Proc. of the 2014 Conference on Internet Measurement Conference. ACM*, 2014.

[13] Ravindranath, Lenin, et al., "AppInsight: Mobile App Performance Monitoring in the Wild," *OSDI*, Vol. 12. 2012.

[14] Hao, Shuai, et al., "PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps," *Proc. of the 12th annual international conference on Mobile systems, applications, and services. ACM*, 2014.

[15] Ravindranath, Lenin, et al., "Automatic and scalable fault detection for mobile applications," *Proc. of the 12th annual international conference on Mobile systems, applications, and services. ACM*, 2014.

[16] Hyunsu Mun Github(ApplicationPerformance), [Online]. Available: https://github.com/munhyunsu/ApplicationPerformance

유 현 식

2016년 충남대학교 컴퓨터공학과 졸업(학사). 2018년 충남대학교 컴퓨터공학과(석사). 관심분야는 모바일 앱 분석, 모바일 트래픽 분석

문 현 수

2015년 충남대학교 컴퓨터공학과 졸업(학사). 2015년~현재 충남대학교 컴퓨터공학과 석박사통합과정. 관심분야는 스파크, 하둡, 웹 데이터 분석, 네트워크 트래픽 분석

이 영 석

1995년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
2002년 서울대학교 전기·컴퓨터(박사)
2003년~현재 충남대학교 컴퓨터공학과 교수. 관심분야는 인터넷트래픽측정, 스파크, 하둡, 미래인터넷