Future Generation Computer Systems 56 (2016) 766–772

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A novel pre-cache schema for high performance Android system

Hui Zhao[a], Min Chen[c], Meikang Qiu[b,*], Keke Gai[b], Meiqin Liu[d]

[a] *Software School, Henan University, Kaifeng, Henan, 475000, China*
[b] *Department of Computer Science, Pace University, New York City, NY 10038, USA*
[c] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wanhan, China*
[d] *College of Electrical Engineering, Zhejiang University, ZJ 310027, China*

HIGHLIGHTS

- Propose a novel model for high-performance Heterogeneous Android systems.
- Use middleware-based pre-cache technology.
- Use middleware-based approach to save networking traffics.
- Generate Version Flags, which is a proposed new component of web pages.

ARTICLE INFO

Article history:
Received 12 March 2015
Received in revised form
7 April 2015
Accepted 5 May 2015
Available online 21 May 2015

Keywords:
Android systems
Pre-caching
Middleware services
High performance
Distributed computing systems

ABSTRACT

As a mobile operating system framework, Android plays a significant role in supporting mobile apps. However, current Android application model is not efficient by using current two common approaches, including *Activity+XML Layout Files* (AXLF) and *HTML+WebKit* (HWK) models. In this paper, we propose a novel middleware service solution that overcomes the drawbacks with using the pre-cache approach, *PrecAche Technology of Android System* (PATAS). The proposed method uses HTML to design the application interface and separately store the *Page Framework* (PF) and *Page Data* (PD). We create a new middleware of web pages, *Version Flags*, to indicate whether PF and PD are expired. Our experimental results represent that the proposed approach can improve the execution efficiency as well as reduce the networking costs, which can be broadly used in cloud-based distributed systems.

1. Introduction

With the rapid development of the mobile technologies, mobile applications have been broadly used in multiple domains. The implementations of the mobile apps have brought mobile users a variety of benefits, such as entertainments, communications, and personal information management. Mobile devices have become a platform supporting mobile apps and wireless communications [1]. The high-performance of wireless networks are also beneficial for the expansion of the mobile apps, such as distributed computing and mobile cloud computing [2,3]. The openness of Android systems brings a higher-level flexibility for implementing multiple wireless technologies [4]. However, current Android apps are facing a great challenge in efficiency and fast response to the user demands due to the inefficient execution models [5]. This paper addresses this issue and proposes a novel approach for high performance Android systems by using pre-cache technologies.

As a popular framework, Android is a mobile phone operating system that is established on the basis of the Linux kernel. Considering a complete open mobile platform, the emergence of Android has brought a large number of opportunities to smart phones as well as challenges. Android system has become a popular operating system on mobile embedded systems, which has been continuously increasing as the improvements of computation capability, wireless networks, and distributed deployments [6–8]. Cloud computing provides Android systems with a broad service delivery platform with a strengthened communication capability [9]. Content caching is an approach for optimizing performances of Android systems in distributed computing systems [10,11].

Nevertheless, increasing the efficiency of the Android apps is a challenging issue for current mobile developers. There are two traditional app approaches for Android, including *Activity*

* Corresponding author.
E-mail addresses: zhh@henu.edu.cn (H. Zhao), minchen2012@hust.edu.cn (M. Chen), mqiu@pace.edu (M. Qiu), kg71231w@pace.edu (K. Gai), liumeiqin@zju.edu.cn (M. Liu).

http://dx.doi.org/10.1016/j.future.2015.05.005
0167-739X/© 2015 Elsevier B.V. All rights reserved.

+ *XML Layout Files* (AXLF) and *HTML* + *WebKit* (HWK). First, as an official method recommended by Google, AXLF performs a lower-level response to user demands, even though it supports all great features of Android system. The other approach is HWK, which uses HTML technology to develop mobile apps. Despite the efficiency of HWK is better than AXLF, the drawback of using this model is that a lot of unnecessary documents will be generated by repeatedly downloading from remote servers. Therefore, there is an urgent demand for finding out an approach that can take all advantages of prior models as well as avoid disadvantages.

Addressing this issue, we propose a novel method named *PrecAche Technology of Android System* (PATAS). The proposed paradigm mainly consists of three algorithms, namely *Server-Side Execution Procedure Algorithm* (2SEPA), *On-Premise Load Page Algorithm* (OPLPA), and *Page Synchronization Algorithm* (PSA). Implementing the proposed schema aims to prevent from unnecessary downloading data base on the high efficiency performance. The main contributions of this paper are twofold:

1. We proposed a novel model for high-performance Android systems using the pre-cache-based technology.
2. Our schema uses middleware-based approach to save networking traffics by generating *Version Flags*, which is a proposed new component of web pages.

The remainder of the paper is organized as the following order: In Section 2, related works in Android domains are reviewed. Section 3 describes the models and concepts used in our proposed schema. A motivational example is provided in Section 4. Next, the details of our proposed algorithm is given in Section 5. The experimental results and the conclusions are represented in Sections 6 and 7.

2. Related works

Prior research has addressed Android research in multiple perspectives. This section reviews the related works concerning Android models by using AXLF and HWK. Being aware of the Android systems, the *Open Handset Alliance* (OHA) was established by Google and its partners in November 2007 when released the first Beta version of the Android operating system, *Software Development Kit* (SDK). More than 80 firms in mobile domains attach to the organization and most mobile apps follow the standard formed by OHA.

Due to the restrictions of the computing capabilities, current mobile apps are required to perform low energy costs [12]. Many previous research has addressed the issue of energy consumption [13]. Leveraging mobile cloud-based solutions is an option for mobile apps to achieve low energy costs, high performance, and optimal task scheduling [14–16]. Using heterogeneous distributed computing has a positive impact on mobile computation offloading and connections across multiple platforms [17]. The benefits of heterogeneity in mobile distributed computing enable many mobile systems to migrate the focuses into interface design and data display with implementing human–computer interactions.

AXLF is one of the classic methods supported by Android systems. Applications use this model can straightly use the components provided by Android as well as maximize the usage of the Android system resources to achieve a dazzling visual effects. Multiple functionality-aimed apps can be executed by this approach, such as games, multimedia, and online communications. Two crucial parts in this model include *Activity Manager* (AM) and *View System* (VS). Using XML layer files to describe user interface layout is able to reduce the complexity of the system [18]. The interconnections between platforms can enable the apps communications based on the cross-platform tools [19].

However, some mobile apps do not need fashionable user interfaces but need complex system structures, such as e-dictionary, *Office Automation* (OA) systems, and E-commerce-related apps. Using AXLF approach can result in heavy workload and interface executions. In addition, the default component provided by Android needs users to have a strong background in computing, which results in the difficulties in workload controls. There are some system which need to be provided traditional computing network access and mobile access both, it will take twice as much time to developed respectively webpage design and mobile applications, and these two works cannot be shared by each other, this also greatly increased the workload of developers.

HWK is the other optional approach because the AXLF model is not efficient in running apps delivering information services. When using this model in Android applications, HTML technology is applied to develop webpage so as to achieve the applications user interface and response functions of each component. Then WebKit components of Android are used to load the page to display complex user interface and useful information in the form of the page [20]. When programmers develop applications with using this model, such as writing a webpage, the pages in the website can be directly used as the mobile user interface. Loading the page for the application user interface can achieve human–computer interactions. Currently, this model has gradually become one of the mainstream models for Information-as-a-Service mobile apps. However, implementing HWK model usually need a lot of extra execution time and can cause heavy networking traffic. Due to the workload of updating data, users have to download many extra contents.

Applications can store the data on-premise when the information is not needed to be changed, such as e-dictionaries, which implies that the wireless networks are not required if the source is stored locally. A middleware who can temporarily or permanently store the resource is an optional method when the resources are migrated to the clouds [17,21,22]. Using a middleware-based model has a potential to solve the problems we mentioned before. However, apps implementing HWK model must download all pages when apps needs to update the details of the information, even though most page information is already stored in mobile devices. This results in a longer execution time and unnecessary network traffics. Therefore, implementing HWK model usually needs extra execution time and can cause heavy networking traffics. Users have to download extra contents due to the workload of data updates.

3. Models and concepts

In this section, we introduce the basic models and concepts of our proposed schema, PATAS. Two crucial aspects of PATAS are covered in this section, including HTML page separation and execution procedure re-design. The whole HTML page is separated into two components, including *Page Framework* (PF) and *Page Data* (PD). A new component of web pages is introduced in our proposed schema, which is set of *Version Flags*. A version flag is a middleware that indicates whether PFs or PD are expired. Fig. 1 shows the architecture of PATAS.

Page Framework and Page Data: A PF refers to the framework of web pages, such as page layout, configuration information, and components and executions. A PD represents the real-time data that need load into the page, such as prices and images in e-commerce. Using pre-caching technology can allow applications to avoid downloading the re-stored PF and PD. Workload can be reduced by using this model without increasing networking traffics.

Page Separation: we separate the page contents into two parts, PF and PD. The purpose of page separation is that the mobile apps can respectively download and cache the two parts to the local device when running apps. Two parts are combed into a complete

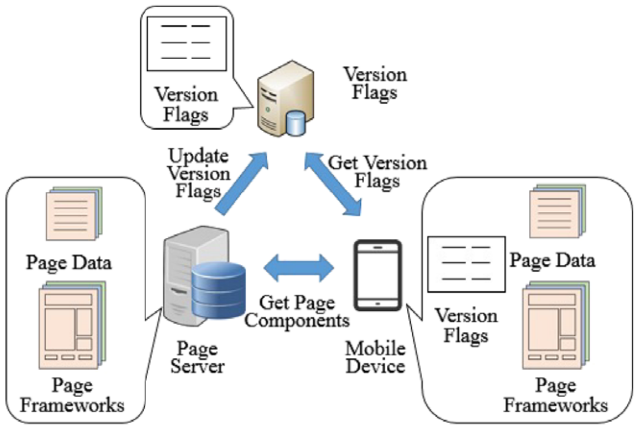

Fig. 1. Architecture of PrecAche Technology of Android System (PATAS).

page and the apps use WebKit to load and display the page. This approach enables separate downloads that can lead to reducing network traffics.

Version Flags: A set of version flags is a middleware service that can determine whether PF and PD stored in local mobile devices match that of in remote servers. Both servers and mobile devices maintain a set of version flags. Mobile apps download this set of version flags from server while mobile apps start. Comparing version flags local cached with version flags downloaded, the apps can select to just download the contents that need updating from servers.

Data Format: We select *JavaScript Object Notation* (JSON) as the data interchange format. JSON is a text format that is completely language independent. Version flags can define flexibly according to the needs of applications. For example, we define version flag format of a page in our system coarse-grained like this:

```
{
“frameVersion” : “12”,
“dataVersion” : “123”.
}
```

The requests and responses of data is also encapsulated to JSON string while they are uploaded and downloaded. The data format can define flexibly according to the needs of apps.

Compression: The servers compress the page components that need to be downloaded to further reduce network traffics. As the data which is transported between server and mobile terminal is basically character string, there is enough compression space. The compression can reduce the networking traffic, even though it increases the workload of compression and decompression, Compressing data increases the performance in wireless communications.

4. Motivational example

A motivational example is given in this section to describe the implementation of PATAS. We use a map $Map\langle pName, vMapCached\rangle$ *flags* to store version flags, which are information pairs *pName* and *vMapCached*. Here *pName* refers to a *Page Name*. *vMapCached* refers to another map $Map\langle vName, vValue\rangle$. In *vMapCached*, *vName* refers to a version flag name such as “framVersion” and “dataVersion”. *vValue* refers to a version flag value that is an integer. We assume that there is an app using PATAS algorithm running on a mobile device. The server is ready and the app has been installed into a mobile device.

The app cache is empty at the initial stage. When the app tries to display a page as user interface and searches the version flags *vMapCached* in version flags cache *flage*. It cannot find version flags corresponding the page now at the initial status. Thus, the app sends the request to the server and downloads all components of the page, including PF, PD, and version flags as *vMap*, which has the same data type as *vMapCached*. Mobile apps store all these components into the local caches. This is the process of initializing page. After this, the app combines PF and PD into a HTML file and cache it as well. Finally, the app calls Webkit to load and show the page to users.

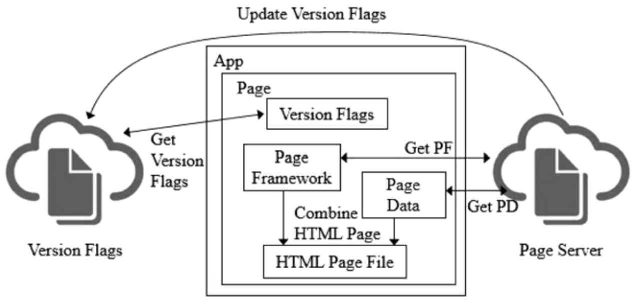

Fig. 2. Framework of the motivational example.

Fig. 2 exhibits a framework of the motivational example. As shown in the figure, *Compress* refers to reducing the data storage requirement for saving the networking traffics. *Uncompressed* is an opposite process to the *Compress*. *Encapsulate* means a process converting the version flags into JSON strings, and *Decapsulate* is an opposite process to the *Encapasulate*. Version flags play a middleware role who check the status of PF and PD. The app tries to show the page that is initialized again after a short period. It searches and gets the version flags *vMapCached* in local cache *flags*. Then, the app sends the request for the newest version flags *vMap* of the page from the server. After downloading the newest version flags, the app compares the version flags from server *vMap* and the version flags stored in local cache *vMapCached*. We assume they are all same at this time. This means all the page components are ready. Thus, the app calls Webkit to load HTML file in the cache and shows it to users.

The app tries to show this page once again after a longer period. First, the app searches and gets the version flags *vMapCached* in local cache. After getting the version flags, the app sends the request for the newest version flags of the page *vMap* from the server. Then the app compares the version flags from server *vMap* and the version flags stored in local cache *vMapCached*. We assume there are some different version flags at this stage, which means some page components are expired.

In this case, the app download expired page components that the version flags indicated and update this part components in local cache. Then, the app update the *vMapCached*. After doing this, the app combines PF and PD into a HTML file and update it in local cache. Thus, all components of the page store in local cache are up to date now. Finally, the app calls Webkit to load and show the page to users. The above is the process of using PATAS algorithm, which can reduce network traffics effectively.

5. Algorithms

We detail our proposed algorithms that are designed to save total network traffic costs and power costs with using a pre-cache technique. The algorithms support activities on the server-end and mobile devices respectively. Notations used in this paper are listed in Table 1.

The first proposed algorithm is 2SEPA that is shown in Algorithm 1. The algorithm runs on the server end, which is designed to maintain all materials of the pages, such as version flags, PF, and PD. Applying Algorithm 1 can provide all materials of the page for mobile devices. As shown in the pseudo algorithm,

Table 1
Main notations and definitions.

Notations	Definitions
©	Operator for accomplishing a "compress" activity
$\bigcirc$	Operator for accomplishing an "uncompress" activity
$\odot$	Operator for accomplishing a "decapsulate" activity
$\oplus$	Operator for accomplishing an "encapsulate" activity
$\otimes$	Operator for accomplishing an "update" activity
J^a_{string}	Represents a JSON string before compressed
J^b_{string}	Represents a JSON string after compressed
PF^c	Compressed Page Framework
DC	Data Cache
FC	Framework Cache
web	A new HTML page file
vValue	A version flag value that is an integer
$vValue_{Data}$	The corresponding data version *vValue*
$vValue_{Framework}$	The corresponding framework version *vValue*
vName	Version flag name
vMap	A map including a pair of information $\langle vName, vValue\rangle$
vMapChached	A local version flag map of a page with the same data type of *vMap*
pName	A page name
flags	A map including a pair of information $\langle pName, vMap\rangle$
flagsChached	A local version flag map of some pages with the same data type of *flags*
changed	A Boolean indicating whether the page needs an update

Algorithm 1 Server-Side Execution Procedure Algorithm (2SEPA)

```
Require: flags, PD, PF
Ensure: New Page Components, $J^a_{string}$
1:  start server, listen to a specific port
2:  IF ∀ received the request of the version flags THEN /*When
    receiving a request of flag version*/
3:      DO $\oplus$ $Map\langle vPName, vVMap\rangle flags$ to $J^a_{string}$;
4:      © $J^a_{string}$
5:  RETURN $J^b_{string}$
6:  ENDIF
7:  IF ∀ received PD requests THEN /*When receiving a request of
    PD*/
8:      $PD \oplus J^a_{string}$
9:      © $J^a_{string}$
10:     RETURN $J^b_{string}$
11: ENDIF
12: IF ∀received PF requests THEN /*When receiving a request of
    PF*/
13:     © PF
14:     RETURN $PF^c$
15: ENDIF
16: IF ∀received PD update requests THEN /*When receiving a
    request of PD update*/
17:     $\otimes$ the PD
18:     adds 1 to $vValue_{Data}$ in $Map\langle vPName, vVMap\rangle flags$;
19: ENDIF
20: IF ∀ received PF update requests THEN /*When receiving a
    request of PF update*/
21:     $\otimes$ the PF
22:     adds 1 to $vValue_{Framework}$ in $Map\langle vPName, vVMap\rangle flags$;
23: ENDIF
```

Algorithm 1, there are a number of phrases that are given as follows.

1. On the server side, we start server program. Let server program listen to a specific port and wait for the access request of the mobile devices.
2. The server encapsulates the version flags to JSON string, compresses and sends it back to mobile device after receiving the request of the version flags from the mobile device.
3. According to the data flags provided by the mobile device, the server encapsulates the recorded data to JSON string, compress and send it back after receiving the request of the data from the mobile terminal.
4. The server compresses the stored PF contents, compress them and then send them back after receiving the request of PF from the mobile device.
5. The server updates the PD of a page, and then let middleware add 1 to the PD version flag of the page after receiving the request of the page's PD update.
6. The server updates the PF of a page stored by the server, and then let middleware add 1 to the PF version flag of the page after receiving the request of the page's PF update.

Mobile device also keeps a set of version flags *flagsChached* on-premise. Mobile apps use these version flags to decide whether it needs to download page components from server when the apps try to load a page. Algorithm 2 describe the algorithm of loading pages.

Algorithm 2 On-Premise Load Page Algorithm (OPLPA)

```
Require: flagsCached, pName
Ensure: web
1:  IF flagsCahched.get(pName) == NULL THEN
2:      download $J^a_{string}$ of version flags of the page from server
3:      $\bigcirc J^a_{string}$
4:      vMapCached.add($\odot J^b_{string}$)
5:      versionFlags.put(pName, vVMapCached);
6:      download PF from server
7:      download PD from server
8:      combine DC and FC
9:      generate a new HTML page file web;
10:     cache the HTML page
11: ELSE
12:     synchronize the page
13: ENDIF
14: call WebKit to load the HTML page
```

On mobile devices, a mobile app searches the version flags of a page when the app tries to display the page. The page will be initialized if the page's version flags cannot be found. When a page is initialized, all materials of this page are downloaded and cached to mobile devices, such as version flags, PF, and PD. Then the mobile app combines *Data Cache* (DC) and *Framework Cache* (FC) to generate a new HTML page file and store the HTML page in the local cache. If the version flags of this page are found, the mobile app synchronizes the page using version flags. Finally, the app loads the page when the page is ready. For each time of initializing page, it must be synchronized with server before mobile app loads it. Algorithm 3 describes the algorithm of synchronizing a page.

Considering the combination, we generate an algorithm to synchronize pages. As shown in Algorithm 3, the main phrases include:

1. Download and uncompress the page's version flags.
2. Compare downloaded version flags with the corresponding version flags stored in on-premise cache.
3. If downloaded version flags are not as same as the local cached version flags, the app download the components that version flags indicate and update page component cache and version flags.
4. The mobile app combines DC and FC to generate a new HTML page file, and store the HTML page in local cache if any page component is updated.
5. The page is ready for apps.

Algorithm 3 Page Synchronization Algorithm (PSA)

```
Require: flagsCached, pName
Ensure: web
1:  changed ← false
2:  download $J^b_{string}$ of version flags of the page from server;
3:  $\bigcirc J^b_{string}$
4:      $\bigcirc J^a_{string}$
5:      vMap.add($\bigodot J^b_{string}$)
6:  Map⟨vName, vValue⟩ vMapCached ← flags.get(pName)
7:  For EACH vName IN vMap
8:      IF vValue! = vMapCached.get⟨vName⟩ THEN
9:          download compressed page component represent by vName
10:         $\bigcirc$ downloaded component and update cache
11:         vMapCached.put⟨vName, vValue⟩
12:         changed ← true
13:     ENDIF
14: ENDFOR
15: IF changed == true THEN
16:     combine DC and FC
17:     generate a new HTML page file web
18:     cache the HTML page
19: ENDIF
```

6. Experiment and results

The PATAS approach is the improvement of the HWK model. Using our algorithm can assist mobile apps to avoid downloading lots of components that are already stored in the local cache, which can reduce networking traffics. The energy consumption can be reduced when the networking traffics are lowered down. The latency of loading pages can be reduced at the same time. This can improve the user experience and increase the durations of mobile devices.

Our experiment accomplished comparisons between HWK model and our proposed schemas. The comparisons focused on comparing the energy consumptions and execution latency for each app. The outcomes of the experiment indicated that our algorithm has a better performance than HWK model. The examined apps are listed in Table 2. We compare a variety of parameters, including functions, *Data Update Frequency* (DUF), PF, and PD. Three devices used in our experiments are as shown in Table 3.

The first experimental device is HUAWEI U8860 Smartphone with Android OS v2.3 (Gingerbread), Qualcomm MSM8255T Snapdragon Chipset, 1.4 GHz Scorpion (one core) CPU, Adreno 205 GPU, 512 MB RAM, 1 GB Storage. The second experimental device is Samsung Galaxy S5 Smartphone with Android OS v4.4.2 (KitKat), Qualcomm MSM8974AC Snapdragon 801 Chipset, Quad-core 2.5 GHz Krait 400 CPU, Adreno 330 GPU, 2 GB RAM, 32 GB Storage. The Third experimental device is *Android Virtual Device* (AVD) in Android SDK with Android OS v5.0.1 (Lollipop), armeabi-v7a CPU, 343 MB RAM, 512 MB Storage. The AVD was running on a Host with Windows 8 64 bit OS, Intel Core i5-4210U CPU, 8.0 GB RAM, Intel HD Graphics Family GPU.

6.1. Energy consumption

Energy consumption is related to many factors, such as execution time, signal frequency of channel, compressing strategy, data amount, and network traffic. Network traffics have a positive relationship with energy consumptions. In our experimental environment, most parameters are constants except the network traffics. Thus, we use the network traffic to analyze the energy consumptions.

Table 2
The attributes of apps.

App name	Function	DUF	PF	PD	Total
App1	Email	Medium	1.25 MB	42 kB	1.29 MB
App2	MSG	Medium	832 kB	30 kB	862 kB
App3	Notification	Low	1.3 MB	1 kB	1.3 MB
App4	Social Networks	High	523 kB	1.6 MB	2.11 MB
App5	Real-Time Price	High	3.4 MB	1 kB	3.4 MB

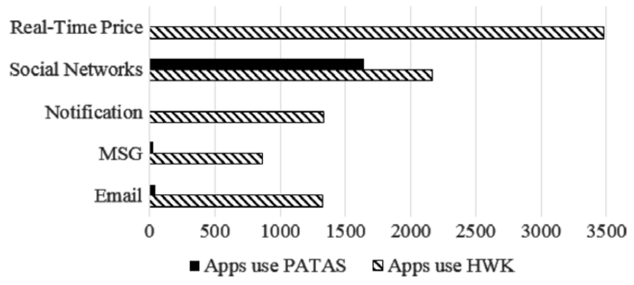

Fig. 3. Comparisons of network traffics between PATAS and HWK.

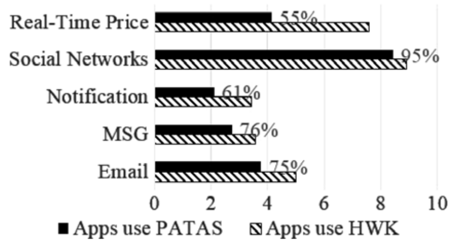

Fig. 4. Comparison of execution latency on HUAWEI U8860.

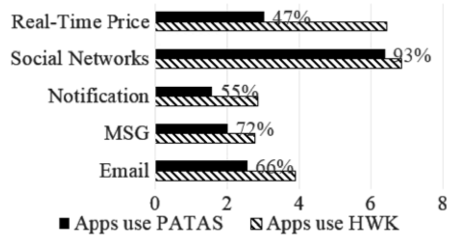

Fig. 5. Comparison of execution latency on Samsung Galaxy S5.

Fig. 3 shows the comparison result of the energy consumptions for two apps using two corresponding algorithms, PATAS and HWK. According to Fig. 3, we find that our algorithm can reduce network traffics effectively, especial the apps that have less PD, such as notification apps and real-time price apps. Therefore, our algorithm can increase the duration of mobile devices.

6.2. Execution latency

Execution latency can be influenced by various factors, such as the CPU frequency, memory usage, communication technology, network traffic, and network bandwidth. In our experiment, we reboot the devices before running every app to reduce the environmental interference. Based on this operation, we can guarantee that each app runs under the same experimental environment.

We run the apps on HUAWEI U8860 Smartphone as the first examined device. Fig. 4 shows the comparison results of the execution latency for two apps with using two corresponding algorithms. The figure represents our approach has low latencies than HWK model. Next, we run the apps on Samsung Galaxy S5 Smartphone. Fig. 5 shows the comparison results of the execution

Table 3
Experiment devices.

Device	HUAWEI U8860	Samsung Galaxy S5	AVD
OS	Android OS v2.3 (Gingerbread)	Android OS v4.4.2 (KitKat)	Android OS v5.0.1 (Lollipop)
Chipset	Qualcomm MSM8255T Snapdragon	Qualcomm MSM8974AC Snapdragon 801	N/A
CPU	1.4 GHz Scorpion (one core)	Quad-core 2.5 GHz Krait 400	armeabi-v7a (Intel Core i5-4210U Host CPU)
GPU	Adreno 205	Adreno 330	Intel HD Graphics Family Host GPU
RAM	512 MB	2 GB	343 MB (8.0 GB Host RAM)
Storage	1 GB	32 GB	512 MB

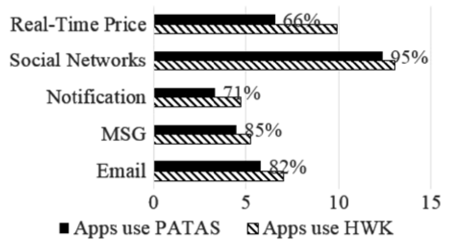

Fig. 6. Comparison of execution latency on AVD.

latency for two apps with using two corresponding algorithms. The result exhibits that our proposed schema even performs much better than HWK model.

Finally, we run the apps on AVD. Fig. 6 shows the comparison results of execution latency for two apps with using two corresponding algorithms. The figure shows a similar result to Figs. 4 and 5. According to Figs. 4, 5, and 6, we find that apps using PATAS algorithm have lower execution latencies than others on each device. It implies that the level of device specifications has a positive relationship with the declining proportion of the execution latency. For example, Samsung Galaxy S5 Smartphone has the highest computation capability among the examined mobile devices. Our approach has the greatest advantage by using this Samsung device.

In summary, our algorithm has provided an approach for generating apps user interface with using HTML. It can efficiently integrate traditional PC with mobile access into one system, which results in an efficient Android system. The experimental results show that our proposed mechanism can enhance the performances in network traffic control, power management, and executive latency. The reasons causing this result are that PATAS can avoid unnecessary network traffics and the capability of the device has a positive relationship with the computation speed under the same networking environment. The portion of latency in runtime will become greater while the computation time becomes shorter.

7. Conclusions

This paper presents a novel schema for high performance Android systems with using pre-cache technologies, which is PATAS. The proposed mechanism mainly consists of three algorithms, including 2SEPA, OPLPA, and PSA. The key part of our approach is that we introduce a middleware component in web pages for reducing network traffics. Our method can be broadly applied for increasing the capability of mobile systems in heterogeneous distributed computing systems. The experimental results show that our proposed approach has better performances in saving energy, lowering latency, and reducing networking traffics.

Acknowledgments

This work has been partially supported by the projects "National Science Foundation (No. 1457506)", "National Science Foundation (No. 1359557)", and the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China ICT1535 (Prof. M. Qiu). Prof. M. Chen was supported in part by China National Natural Science Foundation under Grant 61300224, the International Science and Technology Collaboration Program (2014DFT10070) funded by China Ministry of Science and Technology (MOST), Hubei Provincial Key Project under grant 2013CFA051, and Program for New Century Excellent Talents in University (NCET).

References

[1] J. Araújo, M. Mazo, A. Anta, P. Tabuada, K. Johansson, System architectures, protocols and algorithms for aperiodic wireless control systems, IEEE Trans. Ind. Inf. 10 (1) (2014) 175–184.
[2] L. Zhang, D. Bild, R. Dick, Z. Mao, P. Dinda, Panappticon: event-based tracing to measure mobile application and platform performance. in: IEEE International Conference on Hardware/Software Codesign and System Synthesis, Montreal, QC, Canada, 2013, pp. 1–10.
[3] P. Rost, C. Bernardos, A. Domenico, M. Girolamo, M. Lalam, A. Maeder, D. Sabella, et al., Cloud technologies for flexible 5G radio access networks, IEEE Commun. Mag. 52 (5) (2014) 68–76.
[4] X. Zhang, W. Cheng, H. Zhang, Heterogeneous statistical QoS provisioning over 5G mobile wireless networks, IEEE Netw. 28 (6) (2014) 46–53.
[5] M. Qiu, Z. Chen, L. Yang, X. Qin, B. Wang, Towards power-efficient smartphones by energy-aware dynamic task scheduling, in: IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems, Liverpool, UK, 2012, pp. 1466–1472.
[6] J. Li, M. Qiu, J. Niu, Y. Zhu, M. Liu, T. Chen, Three-phase algorithms for task scheduling in distributed mobile DSP system with lifetime constraints, J. Signal Process. Syst. 67 (3) (2012) 239–253.
[7] C. Wang, Y. Zhu, S. Zhou, X. Gu, J. Jiang, M. Qiu, A scalable embedded system for massive medical signal processing, in: IEEE 12th International New Circuits and Systems Conference, Trois-Rivieres, QC, Canada, 2014, pp. 432–435.
[8] X. Zhang, Z. Yang, L. Shangguan, Y. Liu, L. Chen, Boosting mobile apps under imbalanced sensing data, IEEE Trans. Mob. Comput. PP (99) (2014) 1–12.
[9] D. Wubben, P. Rost, J. Bartelt, M. Lalam, V. Savin, M. Gorgoglione, A. Dekorsy, G. Fettweis, Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN, IEEE Signal Process. Mag. 31 (6) (2014) 35–44.
[10] M. Chen, A. Ksentini, Cache in the air: exploiting content caching and delivery techniques for 5G systems, IEEE Commun. Mag. 52 (2) (2014) 131–139.
[11] J. Li, M. Qiu, J. Niu, Y. Chen, Z. Ming, Adaptive resource allocation for preemptable jobs in cloud systems, in: 10th International Conference on Intelligent Systems Design and Applications, Cairo, Egypt, 2010, pp. 31–36.
[12] W. Shi, M. Wu, S. Wang, M. Guo, B. Peng, B. Ouyang, T. Chen, Local resource accessing mechanism on multiple mobile platform, in: IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems, IEEE, Liverpool, UK, 2012, pp. 1716–1721.
[13] M. Qiu, E. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Trans. Des. Autom. Electron. Syst. 14 (2) (2009) 25.
[14] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu, Online optimization for scheduling preemptable tasks on IaaS cloud systems, J. Parallel Distrib. Comput. 72 (5) (2012) 666–677.
[15] K. Gai, S. Li, Towards cloud computing: a literature review on cloud computing and its development trends, in: IEEE 4th International Conference on Multimedia Information Networking and Security, 2012, pp. 142–146.
[16] M. Qiu, J. Niu, L. Yang, X. Qin, S. Zhang, B. Wang, Energy-aware loop parallelism maximization for multi-core DSP architectures, in: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, 2010, pp. 205–212.
[17] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in mobile cloud computing: taxonomy and open challenges, IEEE Commun. Surv. Tutor. 16 (1) (2014) 369–392.

[18] J. Li, C. Liu, J. Yu, Context-based diversification for keyword queries over XML data, IEEE Trans. Knowl. Data Eng. 27 (99) (2014) 1–14.
[19] J. Ohrt, V. Turau, Cross-platform development tools for smartphone applications, Computer 45 (9) (2012) 72–79.
[20] P. Samadi, H. Mohsenian-Rad, V. Wong, R. Schober, Real-time pricing for demand response based on stochastic approximation, IEEE Trans. Smart Grid 5 (2) (2014) 789–798.
[21] A. Forkan, I. Khalil, Z. Tari, CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living, Future Gener. Comput. Syst. 35 (2014) 114–127.
[22] W. He, D. Xu, Integration of distributed enterprise applications: a survey, IEEE Trans. Ind. Inf. 10 (1) (2014) 35–42.

Hui Zhao received the B.E. and M.S. degrees from Xi'an Technology University, Shanxi and Henan University, Henan, China, in 2000 and 2008, respectively. He is a Ph.D. student at the Seidenberg School of Computer Science and Information Systems of Pace University. He is currently an associate professor in the Software school of Henan University.

Min Chen is a professor in School of Computer Science and Technology at Huazhong University of Science and Technology (HUST). He is the Chair of IEEE Computer Society (CS) Special Technical Communities (STC) on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU) from September 2009 to February 2012. He was R&D director at Confederal Network Inc. from 2008 to 2009. He worked as a Post-Doctoral Fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years. Before joining UBC, he was a Post-Doctoral Fellow at SNU for one and half years. He received Best Paper Award from IEEE ICC 2012, and Best Paper Runner-up Award from QShine 2008. He has more than 180 paper publications, including 85 SCI papers. He serves as editor or associate editor for Information Sciences, Wireless Communications and Mobile Computing, IET Communications, IET Networks, Wiley I. J. of Security and Communication Networks, Journal of Internet Technology, KSII Trans. Internet and Information Systems, International Journal of Sensor Networks. He is managing editor for IJAACS and IJART. He is a Guest Editor for IEEE Network, IEEE Wireless Communications Magazine, etc. He is Co-Chair of IEEE ICC 2012-Communications Theory Symposium, and Co-Chair of IEEE ICC 2013-Wireless Networks Symposium. He is General Co-Chair for IEEE CIT-2012 and Mobimedia 2015. He is General Vice Chair for Tridentcom 2014. He is Keynote Speaker for CyberC 2012 and Mobiquitous 2012. He is a TPC member for IEEE INFOCOM 2013 and INFOCOM 2014. His research focuses on Internet of Things, Big Data, Machine to Machine Communications, Body Area Networks, E-healthcare, Mobile Cloud Computing, Ad Hoc Cloudlet, Cloud-Assisted Mobile Computing, Ubiquitous Network and Services, and Multimedia Transmission over Wireless Network, etc.

Meikang Qiu (SM'07) received the BE and ME degrees from Shanghai Jiao Tong University, China. He received the M.S. and Ph.D. degree in Computer Science from University of Texas at Dallas in 2003 and 2007, respectively. Currently, he is an associate professor of Computer Engineering at Pace University. He has worked at Chinese Helicopter R&D Institute, IBM, etc. Currently, he is an IEEE Senior member and ACM Senior member. His research interests include cyber security, embedded systems, cloud computing, smart grid, microprocessor, data analytics, etc. A lot of novel results have been produced and most of them have already been reported to research community through high-quality journal (such as IEEE Trans. on Computer, ACM Trans. on Design Automation, IEEE Trans. on VLSI, and JPDC) and conference papers (ACM/IEEE DATE, ISSS+CODES and DAC). He has published 4 books, 200+ peer-reviewed journal and conference papers (including 100+ journal articles, 100+ conference papers), and 3 patents. He has won ACM Transactions on Design Automation of Electrical Systems (TODAES) 2011 Best Paper Award. His paper about cloud computing has been published in JPDC (Journal of Parallel and Distributed Computing, Elsevier) and ranked #1 in 2012 Most Downloaded Paper of JPDC. He has won another 4 Conference Best Paper Award (IEEE/ACM ICESS'12, IEEE GreenCom'10, IEEE EUC'10, IEEE CSE'09) in recent four years. Currently he is an Associate Editor of IEEE Transactions on Computers and IEEE Transactions on Cloud Computing. He is the General Chair of the IEEE HPCC/ICESS?CSS 2015, the General Chair of IEEE CSCloud'15 and NSS'15, Steering Committee Chair of IEEE BigDataSecurity 2015. He was also a recipient of the Navy Summer Faculty Award in 2012 and Air Force Summer Faculty Award in 2009. His research is supported by NSF and Industrial such as Nokia, TCL, and Cavium.

Keke Gai is a Ph.D. student in Computer Science department at Pace University, New York, USA. He holds degrees from Nanjing University of Science and Technology (B.Eng.), the University of British Columbia (M.E.T.) and Lawrence Technological University (M.B.A. and M.S.). His research interests include mobile cloud computing, cyber security, combinatorial optimization, business process modeling, enterprise architecture, and Internet computing.

Meiqin Liu received the B.E. and Ph.D. degrees in Control Theory and Control Engineering from Central South University, Changsha, China, in 1994 and 1999, respectively. She was a Post-Doctoral Research Fellow with the Huazhong University of Science and Technology, Wuhan, China, from 1999 to 2001. She was a Visiting Scholar with the University of New Orleans, New Orleans, LA, USA, from 2008 to 2009. She is currently a Professor with the College of Electrical Engineering, Zhejiang University, Hangzhou, China. She is a senior member of IEEE and has participated in organizing several IEEE international conferences. She has authored more than 90 peer reviewed papers, including 48 journal papers. Her current research interests include intelligent systems, information fusion, and nonlinear control.