Parallelized Jaccard-Based Learning Method and MapReduce Implementation for Mobile Devices Recognition from Massive Network Data

LIU Jun[1], LI Yinzhou[1], Felix Cuadrado[2], Steve Uhlig[2], LEI Zhenming[1]

[1]Beijing Key Laboratory of Network System Architecture and Convergence, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Department of Electronic Engineering and Computer Science, Queen Mary, University of London, London E1 4NS, UK

Abstract: The ability of accurate and scalable mobile device recognition is critically important for mobile network operators and ISPs to understand their customers' behaviours and enhance their user experience. In this paper, we propose a novel method for mobile device model recognition by using statistical information derived from large amounts of mobile network traffic data. Specifically, we create a Jaccard-based coefficient measure method to identify a proper keyword representing each mobile device model from massive unstructured textual HTTP access logs. To handle the large amount of traffic data generated from large mobile networks, this method is designed as a set of parallel algorithms, and is implemented through the MapReduce framework which is a distributed parallel programming model with proven low-cost and high-efficiency features. Evaluations using real data sets show that our method can accurately recognise mobile client models while meeting the scalability and producer-independency requirements of large mobile network operators. Results show that a 91.5% accuracy rate is achieved for recognising mobile client models from 2 billion records, which is dramatically higher than existing solutions.

Key words: mobile device recognition; data mining; Jaccard coefficient measurement; distributed computing; MapReduce

I. INTRODUCTION

With increasing popularity of user-friendly mobile clients (smartphones, pads, and tablets), which are coupled with capable mobile applications (location-aware, multimedia, and social applications) supported by advanced cellular communication technology, mobile clients become a part of people's life. To some extent, mobile client and its usage data could be a natural candidate to support and eventually host a user's digital representative [1]. To be more specific, attributes of a mobile client that operators always concern include model, price, and features convincingly depicting characteristics of a group of users as traditional personal information, such as age, sex, occupation, etc. Moreover, capabilities of a mobile client give a remarkable impact on experience and user's desire of given application. Therefore, it is a new challenge as well as opportunity for mobile network operators and ISPs to understand attributes and capabilities of their customers' mobile clients and associated behaviour patterns for designing more efficient market promotion and achieving better user experience.

As a critical step to address this challenge, it is imperative to extract mobile client models from massive network data. Unfortunately, there is little well done work that can help mo-

Received: 2012-12-12
Revised: 2013-03-11
Editor: YUAN Baozong

We introduce a Jaccard coefficient measurement based mobile device model recognition method with parallel algorithms and MapReduce implementation for automatically processing billions of records, which is demonstrated as an accurate and efficient method by evaluation with massive network data from cellular core network.

bile operators and ISPs to finish this job. To this end, we take the first step to analysis HTTP accessing records from mobile client perspective with massive amounts of web traffic data collected from one of the leading mobile networks in China. Then we propose a novel and scalable Jaccard-based [2] learning method for mobile client model recognition. Objective of this method is to identify a proper keyword that represents a mobile device model from unformatted textual headers of accessing logs. It is designed as a three stages method. The first stage is to extract all keywords that are possibly to be the right description of a device model. The second stage is responsible for decreasing the computing workload through filtering candidate keywords by evaluation of conditional probability value between each keyword and device model. Finally, Jaccard coefficient index is calculated with statistical parameters in above stages and the right keyword with highest Jaccard index is selected to represent a device model in the last stage. To meet the requirements of performance and scalability for handling billions of data record, we designed a set of parallel algorithms and leverage MapReduce framework [3] to implement this method as a system named Mobile Device Recogniser (MODER). Implementation of this method got a remarkable recognition accurate rate with evaluation of real network traffic data. Moreover, the parallel algorithms design of Jaccard-based learning method could be a practical reference for similar research work on massive network data analysis.

The remainder of this paper is organised as follows. Section II describes more details about background of mobile device recognition problem and related work. Section III presents a mathematic definition of the problem we addressed and introduces the overview of our solution. Section IV illustrates the detailed parallel algorithm design and implementation of our method. Section V describes datasets and metrics we used for evaluating the accuracy and performance of this method. We also present empirical evaluation result in this section. We conclude with a summary and direction for future work in Section VI.

II. Background and Related Work

With the high penetration rate of mobile devices and increasing number of fancy mobile applications, mobile devices are becoming indispensable to our daily lives. To best sever their customers, mobile operators, ISPs and application developers need to have a better understanding of capabilities of mobile devices held by customers. For example, WLAN offloading is a good technology for mobile operators to improve the service level of cellular data networks. To plan the WLAN access point in a given area, detailed WLAN capabilities information of mobile devices that are active in this area is important. It is a prior requirement for this work to derive information of mobile devices by recognising the model and associated capabilities. Another requirement comes from the marketing division of a leading mobile operator in China. They have noticed that the traditional region definition catalog (airport, bus station, etc.) became unreliable to represent business value of users that came out in these regions. A proposed alternative definition is to estimate users' business value based on their mobile devices, registered services, and related behaviours. A set of this kind of requirements push device model recognition in network side to be an emerging task. Therefore, recognising model of mobile devices becomes a critical prerequisite for many tasks in network management, service provisioning, and market promotion.

The issue of recognising mobile devices was first addressed by research work for adaptive delivering web content to mobile devices. At first, web developers try to retrieve devices' capabilities from User-Agent Request Headers (said UA) [4] and Accepted Request Headers encapsulated in HTTP requests. It left some best practice experience on how to get important features that web developers care about, such as browser, operation system, character set, and so on. It is an excessive and

inefficient work for web developers to handle various HTTP request formats and changing capabilities set. To ease this kind of work, standard organization formed two specifications. The first one is the standard of Composite Capabilities/Preferences Profiles (CC/PP) which comes from World Wide Web Consortium [5]. CC/PP defines a standard that allows mobile devices to transmit their configuration and capabilities to web services carried in a universal profile. Another standard-based way to identify mobile devices is leveraging User Agent Profile (UAProf) [6] which is a specific CC/PP dictionary defined by Open Mobile Alliance. It specifies solution that can be used to describe mobile devices capabilities and support effective configuration content transition via speed-limited wireless network. A mobile device conformed UAProf will send a URL that linked to its XML format capabilities profile conveyed in HTTP request to server. The server could get device information by accessing this URL and parsing received content. Compared to CC/PP, UAProf is likely a server oriented solution decreasing transmitting content size to adapt slow wireless network.

Limitation of above two standard specification based ways to retrieve mobile devices information comes from market diversity. A lot of mobile devices do not follow these specifications. To solve this problem, an open source project Wireless Universal Resource File (WURFL) [7] came out. WURFL turns its direction back to the UA-based devices recognition approach because of UA's rich information about mobile devices characteristics. WURFL enables web servers to recognise the brand and model of an accessing device by matching the UA content with a predefined configuration file. It enhanced the ability of handling diversity unconfirmed mobile devices. In Ref. [8], the author used WURFL to recognise the multimedia capabilities of mobile device for playing appropriate mobile learning multimedia elements. Another example is Ref. [9] in which the author combined the XML mobile device database of WURFL with the popularity metrics of those devices from Google Trends to estimate the market share of each mobile device. Due to fast evolution of mobile devices and unpredictable application content format, the accuracy of device recognition remains low in today's continuously changing market that we will show detail data in the evaluation section.

As the protocol segment of identifying the devices and containing configuration details about the browser, operating system and other hardware and software information, UA brings enough data to recognise mobile devices. But because there is no unified format for web applications to compose it, UA data seems like unstructured data to be processed. That is why method like WURFL could not get high accuracy rate of recognising devices by directly matching captured UA with predefined UA. Figure 1 shows part of textual UA records for Nokia5320 in a set of HTTP accessing records captured from mobile network. Based on the study of these records, we observed the following:

1) The TAC code (such as "35570402" in *r*1, *r*2, and *r*3), which represents the mobile device model, could be retrieved from the first eight number of IMEI in the record [10]. Each TAC is an exclusive code for device model which is allocated by GSM Association (GSMA).

2) Model of mobile device (such as "Nokia5320" in *r*1 and *r*2), the key information of a device, is included in UA as well as other information, such as browser and operating system.

3) UAs generated by different applications are not uniformly formatted. Both sequence of segments and composition styles of a single segment (such as "Nokia5320" in *r*2 vs. "Nokia5320" in *r*3) are various.

4) Unstructured textual UA is non-meaningful without manual knowledge. And there is no sign that could be used to identify possible keyword of device model from disordered mix data.

Above findings lead us to a statistic approach to recognise mobile device models from massive network traffic data. Unlike previous

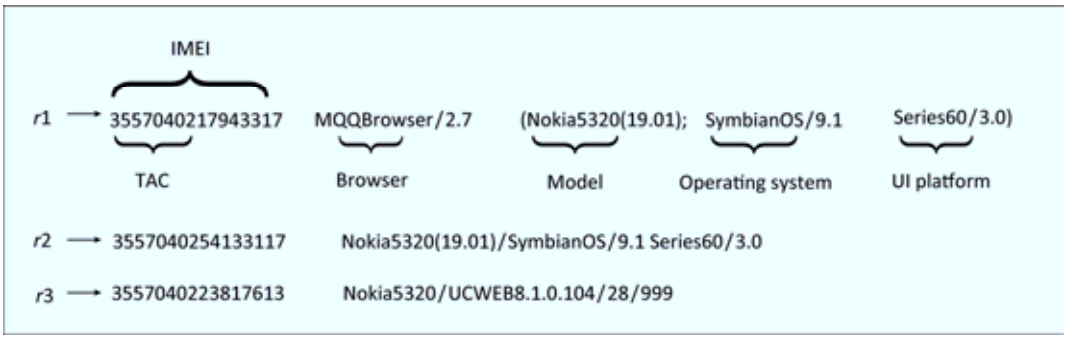

Fig.1 *HTTP accessing records in mobile network*

simple solutions that focus on defining and retrieving predefined information from single device, we have on our hands a hard problem of matching free-text descriptions of mobile devices to their models for which it is desirable to have an algorithmic solution. That is what we presented in this paper. The first step of our solution is to extract and filter candidate keywords that would possibly be the device model from unformatted textual UA header. And then above keywords are sorted by their coefficient values related with the device model, which are calculated by a Jaccard-based formula. The keyword with the biggest Jaccard value is identified as the device model. Jaccard measurement [2] is originally used as a statistic way for comparing the similarity and diversity of sample sets. Due to its simplicity and effectiveness compared with other approaches (Euclidean Distance, Cosine Similarity, Pearson Correlation Coefficient, etc.), it becomes one of the most popular similarity measurements in information retrieval, document clustering, and ontology matching [11-12].

MapReduce is an emerging powerful parallel computing model for big data analytics. In recent years, it is starting to be used as a scalable framework for analysing large volumes of network traffic data [13-14]. Moreover, the efficient and scalable power from combination of MapReduce framework and Jaccard measurement leads some research work on parallel similarity coefficient for huge number of entities. In Ref. [15], Rares et al. introduced how to use MapReduce to calculate Jaccard similarity between records to determine whether join them as output. Another example is a part of the work of the Hadoop-based machine learning framework Mahout [16], which had a similarity computation component for item recommendation. Above works have relied on the presence of some attribute values belonging entities (column of database records, item score from users), and the goal is to find similar entities in record level to link them. This assumption is not valid in our case which is to find a proper keyword from multiple candidates to represent a device model. In this paper, we designed an efficient staged parallel algorithm to disambiguate unclear relations between keywords and device models from billions of data records.

III. Problem Statement and Solution

The original dataset to be processed is a massive number of HTTP accessing records, which are represented as a set $S=\{<r_1(t), r_1(u)>,\ldots, <r_y(t), r_y(u)>\}$. Size of the set S is y, the number of records. Each record $s\in S$ is composed by an attribute pair TAC and UA, which are denoted as $r(t)$ and $r(u)$ respectively. The universe of all TAC is defined as $T=\{t_1, t_2,..., t_M\}$. All UAs compose the set $U=\{u_1, u_2,..., u_L\}$. Each UA is a free-text description that contains characteristic keywords of a mobile device. Keywords extracted from all UAs compose the universe represented as $K=\{k_1, k_2,\ldots, K_N\}$. Our objective is to find out a proper keyword $k_j\in K$ that could represent a $t_i\in T$ which is the model of a mobile device. Based on above definitions, our solution could be described as follows:

Step 1. Keywords Extracting: The first issue we address is how to extract characteristic keywords of a mobile device from unstructured textual UA. After this step, each record $<t_i, u_l>$ is translated to a set of attribute pairs $<t_i, k_j>$ and each k_j is a part of u_l. Thus, we get the universe containing X pieces of records, $R=\{<r_1(t), r_1(k)>,\ldots, <r_X(t), r_X(k)>\}$. The size of R depends on the complexity of UA string in records.

Keyword extraction is well understood in web search, information retrieval and document analysis communities. According to loose guideline of standard specification [4], format of the UA filed in HTTP request is a list of product tokens with optional comments. By convention, multiple product tokens are listed in order of their significance for identifying the application. And the product tokens and optional comments are organised and broken by some reserved formatting letters. Therefore, we take a rule-based heuristic keyword extracting method to achieve comprehensive capability for retrieving unpredictable keywords.

Moreover, with the extracting work of keywords, the joint probability of t_i and k_j, which will be used in following steps, could be computed as follows:

$$P(t_i, k_j) = \frac{\sum_{x=1}^{X} < r_x(t), r_x(k) >, \text{if } r_x(t) = t_i \,\&\, r_x(k) = k_j}{y} \quad (1)$$

In Eq. (1), the denominator y is the number of total records. And the numerator is the number of records whose TAC equals t_i and keywords contain k_j.

Step 2. Candidate Filtering: For each accessing record, output of Step 1 is a set of attribute pairs $<t_i, k_j>$. Considering a given t_i, there are multiple related k_j that make up the set $K'=\{k_1, k_2,\ldots, k_n\}$. The size n of K' could be large because of various and rich content of UA generated by continuously changing applications with unpredictable behaviour in massive network data environment. That will produce a remarkable heavy computing workload for the following step. That is the reason why candidate filtering step is required to save the computing resources and decrease the processing time. We defined a parameter cn to limit the size of candidate keywords set. Objective of this step is to generate a small set $K_s=\{k_1, k_2,\ldots, k_{cn}\}\subseteq K'$. To accomplish this task, we design a filter with two stages, dictionary filter and correlation weight filter. In the first stage, all keywords pass a domain knowledge based dictionary filter and keywords unrelated with device model are removed. The user-defined dictionary, which can be customised, contains predefined general words that stand for well-known concepts, such as "linux", "android", "symbian", etc. Next, the key of the second stage is the measurement of correlation between keyword and device model. We take a probabilistic approach to find out the correlation. The conditional probability value of each keyword $P(k_j|t_i)$ is defined as the correlation weight between k_j and t_i. It could be calculated by the following formula:

$$P(k_j \mid t_i) = \frac{P(t_i \cap k_j)}{P(t_i)} = \frac{P(t_i, k_j)}{P(t_i)} \quad (2)$$

Notice that we have calculated the joint probability $P(t_i, k_j)$ as Eq. (1) in the last step. The only part $P(t_i)$ we need to get the correlation weight could be computed as follows:

$$P(t_i) = \frac{\sum_{x=1}^{y} < r_x(t), r_x(u) >, \text{if } r_x(t) = t_i}{y} \quad (3)$$

$P(t_i)$ is the probability of t_i. We could compute it by adding numbers of original data records $<r_x(t), r_x(u)>$, whose TAC $r_x(t)$ equal t_i, divided by total records number y.

Keywords passing the first stage are sorted by their correlation weight and top cn keywords are selected as candidate keywords. Defining cn is a tradeoff between recognition accuracy and computing complexity. Computing complexity would be decreased with smaller cn, meanwhile possibility of losing right device model word would be increased. We find a proper value for cn by experiment and get a satisfied result, which will be shown in Section V.

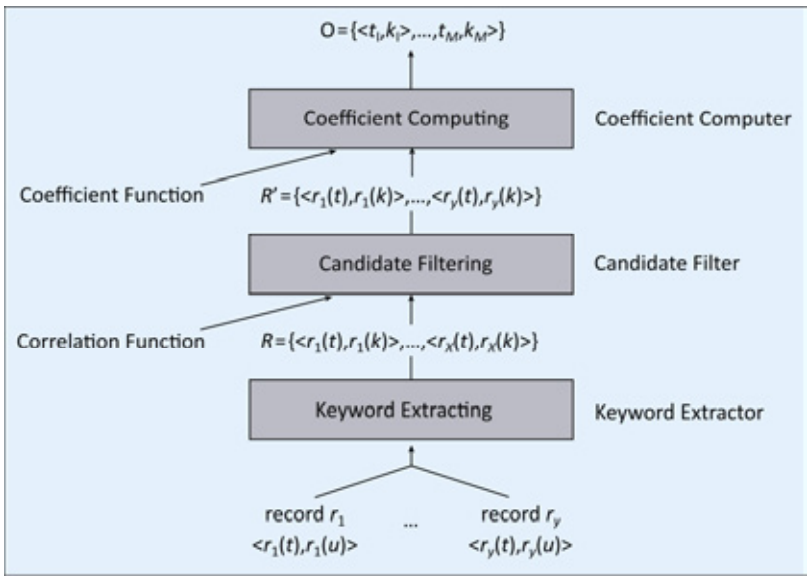

Fig.2 *Overall process of our solution*

Step 3. Coefficient Computing: After prior steps, we get *cn* attribute pairs $<t_i, k_j>$ for each t_i. The complete universe that contains y pieces of record is a set $R'=\{<r_1(t), r_1(k)>,..., <r_Y(t), r_Y(k)>\}$. The size of R' is $Y=y\times cn$. All t_i compose the set $T=\{t_1, t_2,..., t_M\}$ and all k_j compose the set $K=\{k_1, k_2,..., k_N\}$. It should be noticed that the size of K, said N, is not equal to *cn* times M, which is the size of T. The reason is there could be duplicate k_j in different attribute pairs $<t_i, k_j>$. For example, there are duplicate "Nokia5320" in two attribute pairs for $r1$ and $r2$ in Figure 1. The problem to be solved in this step is to identify the right k_j that could represent t_i. It falls in the topic of coefficient measurement of two concepts t_i and k_j. Many practical coefficient measures can be defined based on the joint distribution of two concepts. For this problem, a possible definition for the coefficient measurement between device model and keyword is:

$$\text{Jaccard\_cof}(t_i,k_j)=\frac{P(t_i\cap k_j)}{P(t_i\cup k_j)}=\frac{P(t_i,k_j)}{P(t_i)+P(k_j)-P(t_i,k_j)} \quad (4)$$

This coefficient measure is known as Jaccard coefficient [2]. It took the lowest value 0 when t_i and k_j were irrelevant and the highest value 1 when t_i and k_j were of the same concept. Probabilities of $P(t_i, k_j)$ and $P(t_i)$ are already computed as Eqs. (1) and (3) in Step 2. So we only need to calculate the left required probability value $P(k_j)$ as follows:

$$P(k_j)=\frac{\sum_{x=1}^{X}<r_x(t),r_x(k)>,\text{if } r_x(k)=k_j}{y} \quad (5)$$

$P(k_j)$ is the probability of k_j. In Eq. (5), the denominator y is the number of total records. And the numerator is the number of records whose keywords contain k_j.

Based on this approach, the matching task of given t_i and k_j becomes a two-step work: 1) compute the coefficient value of every k_j. 2) choose the k_j with the highest Jaccard_cof value. Taking these steps for all $t_i\in T$, we will get all TAC and device model description keyword pairs $O=\{<t_1, k_1>,...,<t_M, k_M>\}$ consequently.

With this unsupervised learning method, we implement a mobile devices recognition system named MODER. The overall process and architecture of MODER is shown in Figure 2. In the next section, we will describe the detailed information of how it is implemented in a distributed parallel computing environment.

IV. Parallel Algorithm Design

We now describe the parallel algorithm design of MODER in detail. The basic architecture of this system is shown in Figure 2. It consists of three main modules: Keyword Extractor (KE), Candidate Filter (CF), and Coefficient Computer (CC).

The KE takes original input records as input, together with TAC and UA string. It applies rule-based KE and outputs attribute pair set R. Next, MODER feeds R to the CF, which applies dictionary filtering and correlation function to select first *cn* pairs of attributes. At last the CC takes the coefficient function to identify the keyword k_j that best satisfies the highest coefficient value with t_i. All matching pairs set O is the final output of MODER.

In the remainder of this section, we provide an introduction to the MapReduce paradigm and present the parallel implementation of above three main components.

4.1 MapReduce framework

The dataset needs to be processed has more than 2 billion records. This kind of situation drives us to find a proper parallel computing solution to handle such a huge dataset. After the survey of some possible technologies including parallel DBMS and MPI, we choose the MapReduce programming model due to its impressive high efficiency and low cost characteristics.

MapReduce is a powerful programming model designed for data-intensive parallel computing in shared-nothing cluster environment with up to thousands of low-cost commodity computing nodes. A MapReduce application consists of a sequence of stages that transforms a set of input data into a set of output data. A stage is composed by a set of operation functions named map or reduce. Data is represented as (key, value) pairs and the user defined "map" or "reduce" computation is expressed as:

map: ($k1$, $v1$) → list($k2$, $v2$)

reduce: ($k2$, list($v2$)) → list($k3$, $v3$)

Figure 3 shows an example data flow in a MapReduce program. The input data is partitioned to data splits that are feed into different maps. A map receives a set of inputs as key-value pair ($k1$, $v1$) and produces a set of outputs in the form list ($k2$, $v2$). In this phase, the map functions are applied in parallel on different data splits. And the computation in each map is stateless that every output depends only on the current input. After that, the output <$k2$, $v2$> pairs by each map function are hash-partitioned on the key. Then, they are sent to the reduce node after being merged in a sorted order by the key. That means all the pair values with the same key will be processed in a single reduce node. Reduce function applies a user defined processing logic on each <$k2$, list($v2$)> and produces output data <$k3$, $v3$>. The final result which is typically a list of values comes from the aggregation of all output pairs of reduce nodes.

The actual execution of a MapReduce program is supported by a MapReduce runtime framework implementation such as Apache Hadoop [17]. The MapReduce program is scheduled and executed by specified number of map tasks and reduce tasks with allocated computing resources. Furthermore, the MapReduce framework provides more powerful capabilities, such as combine function, customizable hashing and partitioning functions, to support flexible data processing and algorithm implementation.

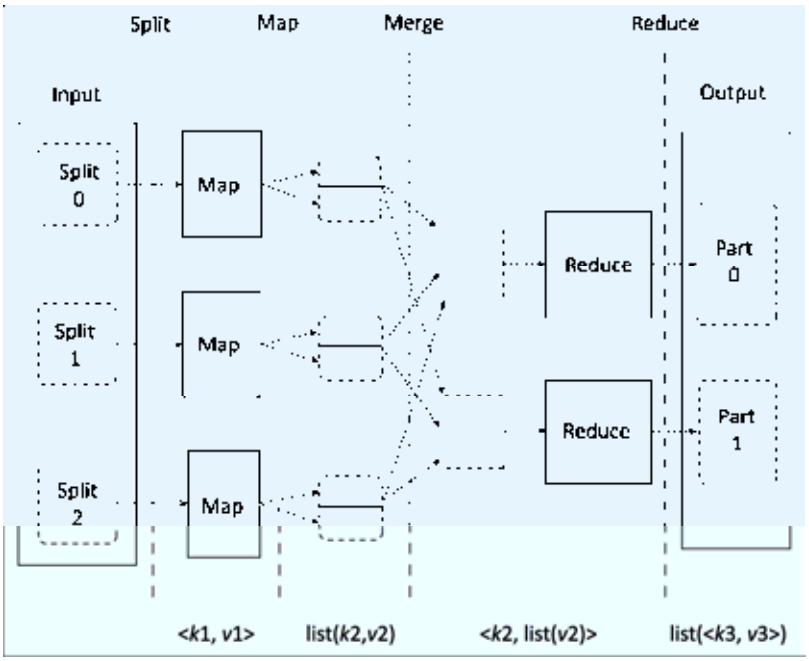

Fig.3 *Data flow in MapReduce program*

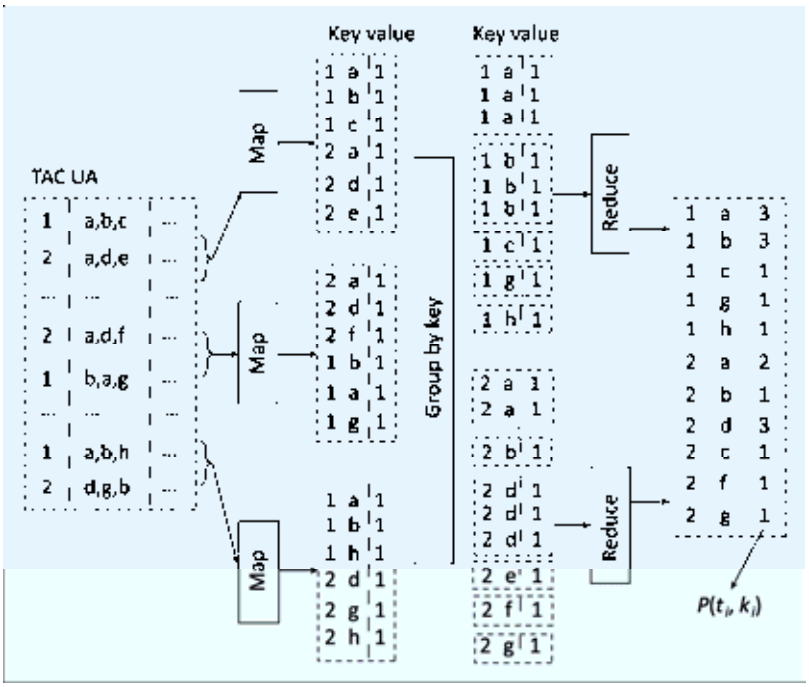

Fig.4 *Data flow in KE*

4.2 Keyword Extractor (KE)

KE takes the responsibility of first stage com-

putation, extracting all possible keywords from UA string in records and counting the number of each $<t_i, k_j>$ pair appearing. Data flow of this stage is shown as Figure 4.

To make our description clear and simple, we use number 1 and 2 to represent the eight digits TAC, and letters to represent the keywords, for example a, b, etc. The map function gets input as the original HTTP accessing log records including TAC, UA and other segments. For each record, the function extracts the value of TAC and cuts the UA string to a set of keywords according to the guideline of specification [4]. Other useless segments in a record for device model recognition are discarded by this function. At the end of map function, a set of attribute pairs ($<t_i, k_j>$, 1) is emitted. The key of output data is $<t_i, k_j>$, and the value is 1 for the following counting work. The map function of KE is introduced in Algorithm 1.

Algorithm 1 Map function of KE

```
Input:   Key: orgFileName  Value: fileContent
Output:  Key: <ti, kj>     Value: 1
for each line in fileContent
  tac=getTAC(line)
  ua=getUA(line)
  while ua not empty
    keyword=getKeyword(ua, rule)
    ua=ua.remove(keyword)
    key=<tac,keyword>
    Emit(key, 1)
  end while
end for
```

Output pairs of above map function are grouped by their key and sent to corresponding reduce node. Subsequently, the reduce function computes the total count for each key and value pairs and generates all possible $<t_i, k_j>$ pairs and their appearing count count_$<t_i, k_j>$ in all records. The reduce function of this step is shown in Algorithm 2.

Algorithm 2 Reduce function of KE

```
Input:   <<ti, kj>,list(1)>
Output:  <<ti, kj>,count_<ti, kj>>
count=0
for each item in list(1)
    count++
end for
Emit(<ti, kj>, count)
```

From the description of coefficient computing in previous section, we know that all possibility value is calculated through a conditional count number divided by the total records number y. Therefore, we could eliminate y in our computation and take the count number to represent the possibility value in the following description. After KE finished its reduce work, we get all $<t_i, k_j>$ pairs and $P(t_i, k_j)$=count_$<t_i, k_j>$, which are stored in a file fileKE.

4.3 Candidate Filter (CF)

Recall Step 2, CF in Section III, we take a staged approach to remove unrelated and low possibility keywords with device models. For the first stage, we build a domain knowledge based dictionary to remove unrelated keywords. We will not go into detail for this straightforward method and just leave it on below pseudo code. Key point of the second stage is sorting keywords based on their conditional probability $P(k_j|t_i)$. From formula of conditional probability, we get the following:

$$P(k_j \mid t_i) = \frac{P(t_i, k_j)}{P(t_i)} = \frac{\text{count}\_ < t_i, k_j >}{\text{count}\_ < t_i >} \quad (6)$$

In KE, we already get the count_$<t_i, k_j>$. For various k_j with a given t_i, the count_$<t_i>$ is obvious the same. So we could use count_$<t_i, k_j>$ as the weight for keywords sorting. Data flow of the whole CF process is shown in Figure 5. An MapReduce process named MRP*t* is needed is to compute count_$<t_i>$ representing $P(t_i)$ for the following computation. And then another MapReduce process following MRP*t* named MRCF takes the work of selecting keywords by sorted conditional probabilities as well as dictionary-based filtering. As an example, we define the parameter of candidate number *cn* as 2.

The map function of MRP*t* MapReduce process tags the record projections with their TAC segment. Thus the reduce function receives a list of record projections grouped by TAC and counts all number within records with the same t_i, which is count_$<t_i>$ that we need to compute $P(k_j|t_i)$.

Unlike just counting work in previous MapReduce processes, the logic of MRCF is with some kind of complication. Problems to be solved in MRCF is how to handle multiple different format files and link related items together for composing a complete record ($P(t_i, k_j)$ and $P(t_i)$ sharing the same t_i) for further computation on a parallel behaviour. We use two advanced MapReduce technologies to meet these requirements, MultipleInputs and Reduce-side Join, which are explained in Algorithm 3.

Algorithm 3 Map function for fileMRP*t*

Input: Key: fileMPR*t* Value: fileContent
Output: Key: <t_i, 0> Value: count_<t_i>

```
1  for each line in fileContent
2    tac=getTAC(line)
3    count=getCount(line)
4    key=<tac, 0>
5    Emit(key, count)
6  end for
```

Besides map function illustrated in Algorithm 3, we have another independent map function for fileKE shown in Algorithm 4.

Algorithm 4 Map function for fileKE

Input: Key: fileKE Value: fileContent
Output: Key: <t_i,1> Value: <k_j, count_<t_i, k_j>>

```
1  for each line in fileContent
2    tac=getTAC(line)
3    keyword=getKeyword(line)
4    count=getCount(line)
5    key=<tac, 1>
6    value=<keyword, count>
7    Emit(key, value)
8  end for
```

MapReduce MultipleInputs allows developers to specify different map functions for different files. Thus, we develop two map functions to handle fileMRP*t* and fileKE. Key of the output of these two map functions is differentiated by a tag (0 or 1) appending with t_i. The tag is used by the following reduce function to identify whether a record contains count_<t_i> or count_<t_i, k_j>. Recall the introduction of MapReduce programming model in this section, it seems we break the rule that keys of records computing in the same reduce function should be the same. After we introduce the tag into key, output pairs of map

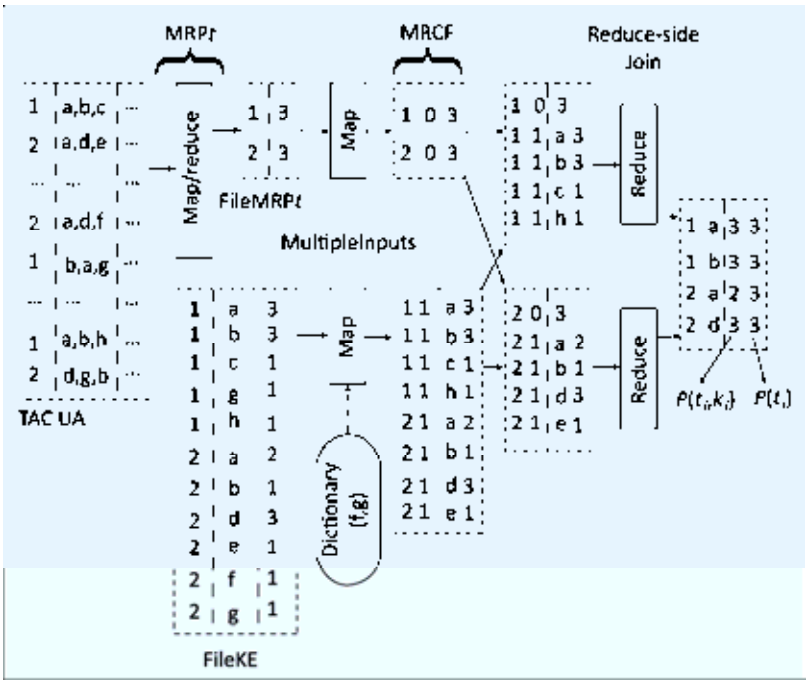

Fig.5 *Data flow in CF*

function what we want to link together in the same reduce function have different keys even their t_i is same. This problem is solved by our customised Partitioner class and TextPair. First-Comparator class. Therefore, the Reduce-side Join is successfully accomplished by the following reduce function:

Algorithm 5 Reduce function of MRCF

Input: <<t_i, tag>, list(value)>
Output: <<t_i, k_j>,< k_j, count_<t_i, k_j>>

```
1  tac=getTAC(<ti, tag>)
2  count_t=popFirstItem(list(value))
3  while list(value) not empty
4    item=popFirstItem(list(value))
5    keyword=getKeyword(item)
6    count_t_k=getCount(item)
7    key=<tac, keyword>
8    value=<count_t_k, count_t>
9    Emit(key, value)
10 end while
```

After working of CF, we get a set of four attributes pairs <t_i, k_j, $P(t_i, k_j)$, $P(t_i)$> stored in a file fileCF.

4.4 Coefficient Computer (CC)

With previous computed attribute pairs <t_i, k_j, $P(t_i, k_j)$, $P(t_i)$>, the only missing part is $P(k_j)$ to fulfill the computation of Jaccard coefficient value. For this, we take the fileKE as input of

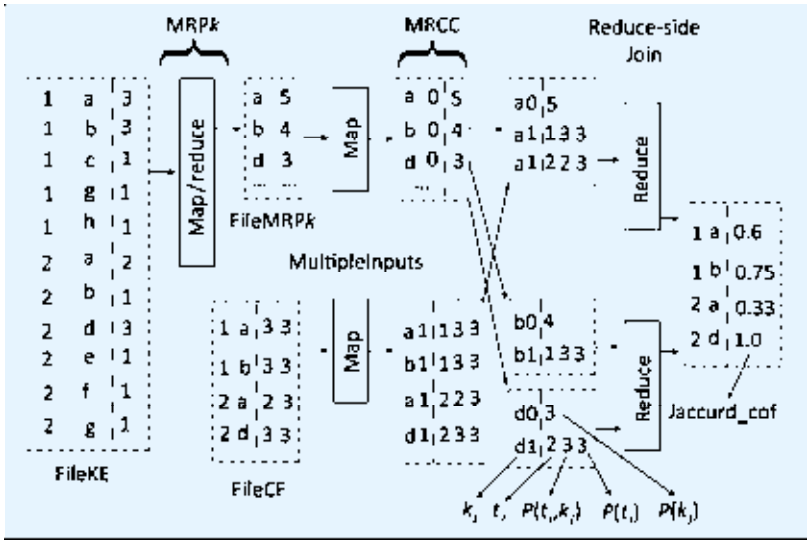

Fig.6 *Data flow in CC*

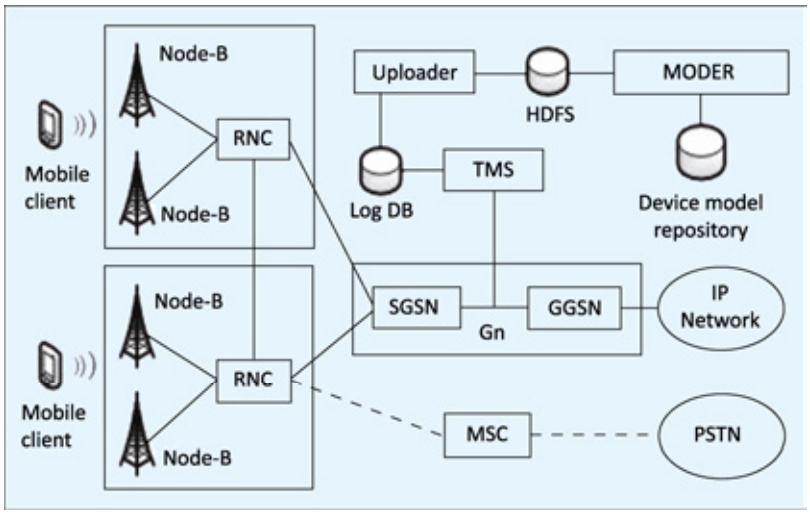

Fig.7 *UMTS network architecture*

TAC	Model	Jaccard	...
...	...	...	...
01284400	iphone	0.00606313	...
01284500	iphone	0.00607453	...
01284600	iphone	0.00593558	...
...	...	...	...
35199404	nokia2700c	0.03362297	...
35199604	nokiac52-1	0.16986312	...
35200004	nokia5233	0.03912787	...
...	...	...	...
35218904	zte-tu236_td	0.15381341	...
35219004	zte-tu260_td	0.15391763	...
...	...	...	...

Fig.8 *Segment of device model repository*

MapReduce process MRP*k* to count number of all k_j. The map function of MRP*k* retrieves keyword from each record and emits key-value pairs $<k_j, P(k_j)>$. Then the MultipleInputs and Reduce-side Join are used again to compose full possibility values $<t_i, k_j, P(t_i, k_j), P(t_i), P(k_j)>$ for computing Jaccard_cof. Data flow of CC, which is shown as Figure 6.

The two map functions for fileMRP*k* and fileCF are similar to Algorithms 3 and 4 with a little difference on preparing data for appending $P(k_j)$ to $<t_i, k_j, P(t_i, k_j), P(t_i)>$. Here we focus on the reduce function to see how is the final Jaccard_cof value produced, which is described as Algorithm 6.

Algorithm 6 Reduce function of MRCC

Input: $<< k_j, \text{tag}>, \text{list(value)}>$
Output: $<<t_i, k_j>, \text{Jaccard\_cof}>$

```
keyword=getKeyword(<kj, tag>)
count_k=popFirstItem(list(value))
while list(value) not empty
    item=popFirstItem(list(value))
    tac=getTAC(item)
    count_t_k=getCount_t_k(item)
    count_t=getCount_t(item)
    jac=count_t_k/(count_t+count_k-count_t_k)
    Emit(<tac, keyword>, jac)
end while
```

Finally, we have a file containing records dictionary ordered by $<t_i, k_j>$. The list composing by the highest Jaccard_cof value is the output list of device models. For our example case, words that represent the two sample devices are b for TAC 1 and d for TAC 2.

V. Evaluation of Real-World Data

The central goal of our work is to develop a functional system that can be de deployed in a real mobile network operation environment to recognise device models. The critical requireements for us are being able to recognise at least 90% captured records and achieve 85% above accuracy rate. Generally speaking, this is a tall order for any existing solution to satisfy. Sources of difficulty to meet above requirements mainly come from the huge size of records and disordered textual data. More than 2 billion accessing records with mixing application and unstructured format prevent recogniser from performing analysis effectively and affect the learning of the model identification

function. In this section, we will show the experiments we conducted to evaluate how we meet above critical requirements.

5.1 Dataset and recognition result

The experimental dataset to be processed is captured from a living Universal Mobile Telecommunication System (UMTS) network [18] of a leading mobile network operator in China. Key components of the monitoring system and MODER system what we developed for device model recognition are illustrated in Figure 7.

A mobile device directly talks with a cell tower (node-B) which forwards its voice or data traffic to a Radio Network Controller (RNC). In case of mobile data service, the RNC delivers the data service request to a Serving PRS Support Node (SGSN) that establishes a tunnel on Gn interface with a Gateway GPRS Support Node (GGSN) through which the data enters the IP network.

Data is collected from a large UMTS network for five days with the size of 2.2 billion records. This dataset includes: 1) IMEI, which indicates the unified identity of a mobile device; and 2) User-Agent field of HTTP request, which is produced by mixed applications accessing data services. It is captured by the operator through Traffic Monitoring System (TMS) and stored in a log database for system experimental evaluation. Given the sensitivity of the data, privacy related information is removed by the Uploader component when it transmits logs to the distributed file system of experimental platform. And all our results stored in the Device Model Repository which is produced by the MODER system are presented as aggregated report to protect the privacy of individuals.

Part of example result we recognised from the experimental dataset is shown in Figure 8. MODER system takes first 90% records with TAC sorted by appearing number as input and achieve 91.5% accuracy rate of device model recognition. Results are stored in a RDBMS maintained device model repository in which additional recognition set will be incrementally inserted with accumulated uploading logs.

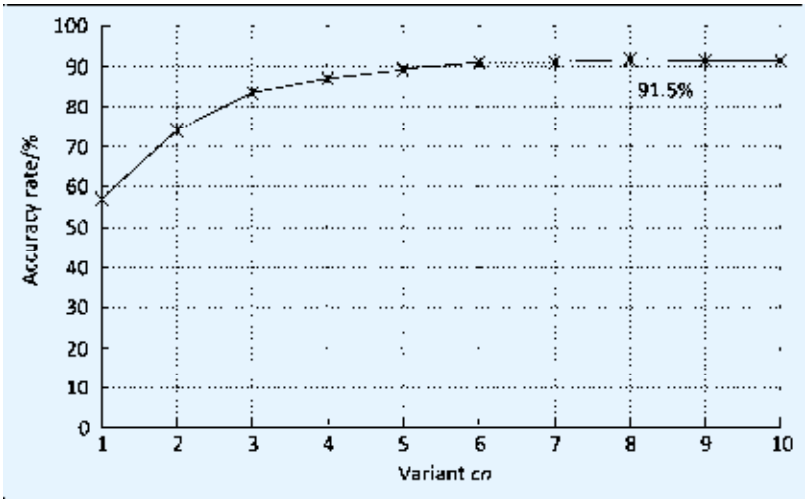

Fig.9 *Accuracy rate of various cn*

From the example result shown in Figure 8, we can see that our algorithm can accurately recognise unusual situations. For example, there are multiple TACs belonging to one device model (iPhone), which are different with normal one-on-one matching scenario (nokia and zte).

5.2 Study of algorithm variant *cn*

Definition of accuracy rate is the radio of the sum of all correctly recognised mobile device models to the sum of all models residing in testing data records. This metric is the most important parameter to evaluate the practical value of a recognition method.

Before we evaluate the accuracy rate of our method, we study the variable parameter *cn* that limits the size of candidate keywords set in Step 2 of our method. We randomly extract 2 billion records from the total set in 3 times to get 3 record sets. Then, the system MODER is executed on these 3 record sets with various parameter *cn* from 1 to 10. The number of all device models for each record set is calculated by summing the count of TAC numbers, which is the same value 1944 for above record sets. Recognised result is evaluated manually by identifying if each recognised model word is an existing mobile device model. Accuracy rate of each *cn* is the average value of 3 record sets. Result is shown in Figure 9.

From Figure 9, we can see that the accuracy rate is increasing with the rising of *cn* value.

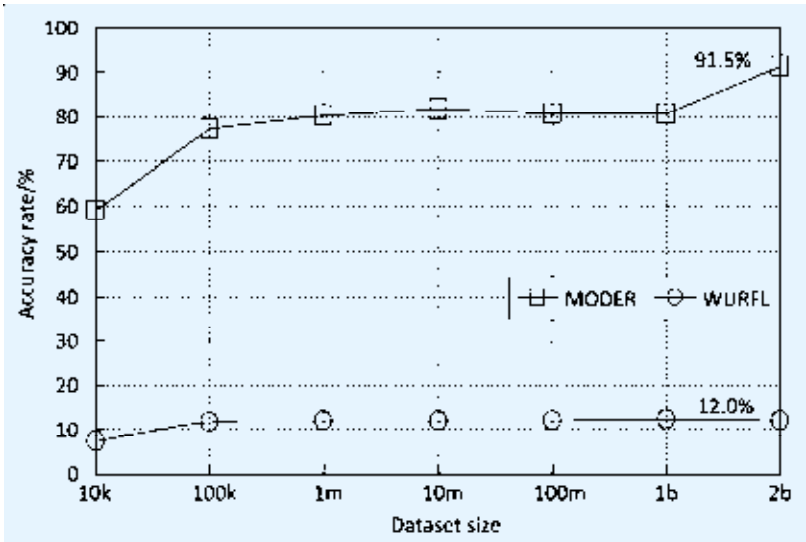

Fig.10 *Accuracy rate evaluation*

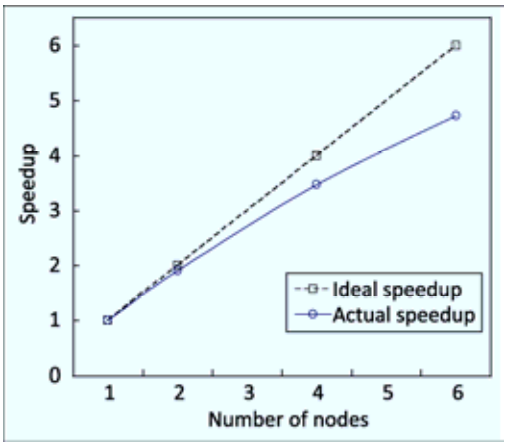

Fig.11 *Speedup evaluation*

The accuracy rate becomes relatively stable from 7 and reaches the highest value 91.5% on 8. After 8, the accuracy rate is slightly dropped (around 0.1%) by few incorrect matching words entering Step 3 with loose number limit. According to the result of this experiment, we choose 8 as the value for *cn*.

5.3 Accuracy rate evaluation

As the most widely used solution by server log analyser and application developer for mobile client recognition, WURFL [7] is taken as a comparator for accuracy rate evaluation in our experiment. To study the accuracy rate of our method in different data sizes, we randomly select 7 different sizes of data records, 10 thousand to 2 billion respectively. For each size, we take three times random selection. Then, our method and WURFL based program are executed on all 21 datasets. Final accuracy rate value for each size is calculated as the average accuracy rate of 3 datasets. The result is shown as Figure 10. It should be noticed that we only compare the accuracy rate of MODER with WURFL instead of performance evaluation because the WURFL is not a parallelized method.

As a string matching approach, the accuracy rate of WURFL remains respectively stable around 12%. The reason for such a low accuracy rate is UA fields in records are generated by both web browsers and applications running on http protocol with unpredictable behaviour. Even for records from web browsers, WURFL cannot recognise them very well because of fast evolution of browsers version, which brings fast changing of http header content. It is the natural defect of string matching based method.

From Figure 10, we can see that the accuracy rate of our method is increasing with the growing size of datasets. We get 91.5% accuracy rate finally which exceeds the critical requirement 85%. According to some study on matters of dataset size for accuracy rate of statistical learning method [19], we can expect our method to achieve higher accuracy recognition if MODER accumulates more http accessing records.

5.4 Speedup evaluation

Speedup is the key parameter that is used to evaluate the efficiency of a parallel algorithm. To evaluate speedup of our method, we choose the dataset with fixed size of 1 billion records. The number of Hadoop cluster nodes is changed as 1, 2, 4, and 6 to study the execution time for recognising work. The *y*-axis of Figure 11 is the speedup value which is calculated by the execution time of changing number of nodes scenarios divided by the execution time of 1 node scenario. The dash line indicates the ideal speedup in this relative scale. The ideal speedup should mathematically be a linear speedup that means doubling the number of

computing nodes doubles the processing speed. From the result, we can see that our method approaches closely to linear speedup. The reason for not matching the ideal speedup is that extra workload of data I/Os, task scheduling, and communications between "map" and "reduce" nodes do not speed up linearly, which are internal mechanism residing in Hadoop MapReduce execution environment. Although our parallel algorithm on larger data sets tends to speed up slower than the linear speedup, the total execution times, which are 33 m 53 s and 1 h 9 m 31 s for computing 1 billion and 2 billion data records respectively, satisfy the performance requirement as a batch processing program running over only 6 working nodes.

VI. Conclusion and Future Work

In this article, we designed a Jaccard-based learning method as a solution to recognise mobile device model from massive network traffic data. Utilizing a staged architecture, our method decomposed the recognition task into multiple subtasks including keyword extracting, candidate filtering, and coefficient computing. To achieve critically high recognition accuracy rate, high volumes of accessing records are required as input to the recognition system. Therefore, a set of parallel algorithms are designed and implemented on MapReduce framework for each stage. Through extensive experiments using real world data, we demonstrated that our system achieved an accuracy rate of 91.5%, which is dramatically higher than existing solution. Furthermore, we evaluated that our parallelization method is scalable for processing more traffic data to get better recognition result.

As future work, aside from continually striving to improve the accuracy of the method, our work will extend to build an automatically accumulating solution for a complete repository covering all captured mobile device models. Another line of future research work involves studying user behaviours from device model's perspective by leveraging our recognition results.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61072061; the National Science and Technology Major Projects under Grant No. 2012ZX03002008; and the Fundamental Research Funds for the Central Universities under Grant No. 2012RC0121.

References

[1] CHITTARANJAN G, BLOM J, GATICA-PEREZ D. Who's Who with Big-Five: Analysing and Classifying Personality Traits with Smartphones [C]// Proceedings of the 15th Annual International Symposium on Wearable Computers: June 12-15, 2011, San Francisco, CA, USA, 2011: 29-36.

[2] JACCARD P. Etude Comparative de la Distribution Orale Dans Une Portion des Alpes et des Jura[J]. In Bulletin del la Socit Vaudoise des Sciences Naturelles, 1901, 37: 547-579.

[3] DEAN J, GHEMAWAT S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communications of the ACM — 50th Anniversary Issue: 1958-2008, 2008, 51(1): 107-113.

[4] FIELDING R, GETTYS J, MOGUL J, *et al*. Hypertext Transfer Protocol — HTTP/1.1, RFC2616[S]. June 1999.

[5] W3C Recommendation Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0[S]. Jan, 2004.

[6] Specification of Open Mobile Alliance, OMA User Agent Profile V2.0[S]. June 25, 2007.

[7] Wireless Universal Resource File Open Source Project (WURFL)[EB/OL]. [2013-4-23]. http://wurfl.sourceforge.net/.

[8] GEORGIEVA E, GEORGIEV T. Methodology for Mobile Devices Characteristics Recognition [C]// Proceedings of International Conference on Computer Systems and Technologies (CompSysTech'07): June 14-15, 2007. University of Rousse, Bulgaria, 2007.

[9] ALMEIDAA A, ORDUN P, CASTILLEJO E, *et al*. A Method for Automatic Generation of Fuzzy Membership Functions for Mobile Device's Characteristics Based on Google Trends[J]. Computers in Human Behaviour, 2012, 29(2): 510-517.

[10] GSM Association Official Document. IMEI Allocation and Approval Guidelines[S]. July 27, 2011.

[11] SUBHASHINI R, KUMAR V J S. Evaluating the Performance of Similarity Measures Used in

Document Clustering and Information Retrieval[C]// Proceedings of 1st International Conference on Integrated Intelligent Computing (ICIIC): August 5-7, 2010. Changsha, China, 2010: 27-31.

[12] DOAN A, MADHAVAN J, DHAMANKAR R, *et al*. Learning to Match Ontologies on the Semantic Web[J]. The International Journal on Very Large Data Bases, 2003, 12(4): 303-319.

[13] LEE Y, KANG W, SON H. An Internet Traffic Analysis Method with MapReduce[C]// Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS Wksps): April 19-23, 2010. Osaka, Japan, 2010: 357-361.

[14] SAMAK T, GUNTER D, HENDRIX V. Scalable Analysis of Network Measurements with Hadoop and Pig[C]// Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS): April 16-20, 2012. Hawaii, USA, 2012: 1254-1259.

[15] VERNICA R, CAREY M J, LI Chen. Efficient Parallel Set-similarity Joins Using MapReduce[C] // Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD'10): June 6-11, 2010. Indiana, USA, 2010: 495-506.

[16] Apache Mahout Machine Learning Library [EB/OL]. [2013-4-23]. http://mahout.apache.org/.

[17] Apache Hadoop[EB/OL]. [2013-4-23]. http://hadoop.apache.org/.

[18] 3GPP Technical Specification 23.101. General Universal Mobile Telecommunications System (UMTS) Architecture[S]. 2004.

[19] BANKO M, BRILL E. Scaling to Very Very Large Corpora for Natural Language Disambiguation [C]// Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL'01): July 9-11, 2001. Toulouse, France, 2001: 26-33.

Biographies

LIU Jun, Head of Network Monitoring R&D Base in School of Information and Communication Engineering, Beijing University of Posts and Telecommunications (BUPT), China. He received his B.E. and Ph.D. degrees from Department of Information Engineering, BUPT, China in 1998 and 2003, respectively. His research interests include network traffic monitoring and telecom big data analysis. Email: liujun@bupt.edu.cn

LI Yinzhou, graduate student in the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications (BUPT), China. She received her B.E. degree in communication engineering from BUPT in 2011. She is engaged in the research of broadband IP network traffic measurement and data analysis.

Felix Cuadrado, Assistant Professor with the Department of Electronic Engineering and Computer Science at Queen Mary, University of London, UK. He received his Ph.D. degree from Universidad Politécnica de Madrid, Spain in 2009. His current researches focus on autonomic computing, cloud computing, and software engineering applied to distributed services.

Steve Uhlig, Professor of Networks and Head of the Networks Research group at Queen Mary, University of London, UK. He obtained his Ph.D. degree from the University of Louvain, Belgium in 2004. Prior to joining Queen Mary, he was a Senior Research Scientist with Technische University Berlin/Deutsche Telekom Laboratories, Berlin, Germany. His current research interests include Internet measurements, software-defined networking, content delivery, and privacy-preserving analytics.

LEI Zhenming, Professor in the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, China. He received his Ph.D. degree from Beijing Institute of Posts and Telecommunications, China in 1986. His research interests include network traffic monitoring, controlling and analysis.