Compte rendu TP Green Computing, Master 2 – TNSI – FA

Maxime Degres
Jean-Baptiste Duriez
Jordane Quincy

Durant ce TP nous avons essayé de voir tous les aspects attendus, aussi bien sur la partie récolte de données (Data collection) que la partie traitement de données (Data processing) ou encore celle de management des ressources (Ressource management / Power management).

Data collection

Notre TP s'oriente pour des utilisateurs de smartphone Android. Nous avons donc voulu pour cette partie de data collection créer une application Android permettant de récupérer différentes informations comme l'état du wifi, le cpu, la luminosité du téléphone, au lieu de simuler nos données. Nous avons donc pu créer cette application en Java et elle tourne correctement sur un Android 4+ (seul version possible avec le téléphone testé). Nous pouvons ainsi récupérer de vraies données en faisant des exports directement depuis l'application. Cependant n'ayant pas eu le temps de faire une collecte de donnée intéressante nous avons quand même simulé nos données afin d'avoir quelque chose d'exploitable. Ne pouvant pas envoyer l'application par email voici le lien de notre github où elle se trouve : https://github.com/jordane-quincy/M2_GreenComputing dans le dossier M2_GreenComputing. Vous pourrez également y trouver une vidéo montrant l'application : viedoApplication.mp4

Data processing et Power management

Pour cette phase de traitement de données nous avons utilisé Weka avec différents algorithmes. Nous avons tout d'abord utilisé un algorithme d'association : GSP – Generalized Sequential Pattern. Cet algorithme nous permet de trouver des « patterns » de données que nous lui donnons en entrée. Pour ce cas-là nous avons utilisé comme données une suite d'applications qui correspond à l'enchaînement des applications pour l'utilisation du téléphone d'une personne sur une journée. Pour le formatage de données avec Weka et GSP il faut 2 attributs, un attribut personID qui distinguera chaque personne et un attribut application qui correspondra au nom de l'application utilisée. Pour avoir une suite pour une personne il faut un ensemble de ligne. Ainsi si on veut la suite Facebook -> Gmail -> Messenger il faudra formater la donnée ainsi :
1,Facebook
1,Gmail
1,Messenger

Et on ajoute ensuite autant de personne que l'on souhaite. On obtient ainsi des couples (ou triplets etc) d'applications qui apparaissent souvent et qui sont donc susceptibles d'arriver. Exemple si on récupère le couple Facebook/PDFReader il sera alors fort possible durant l'utilisation du téléphone que si la personne est sur Facebook alors elle va probablement passer ensuite sur PDFReader, on pourra ainsi potentiellement couper les Data par exemple.

Le fichier que nous avons utilisé est : « donneeAppications - GSP.arff »

Les résultats que nous avons obtenus avec Weka sont les suivants : (nous avons choisi de garder les séquences uniquement s'il y a plus de 80% d'apparition)

```
GeneralizedSequentialPatterns
=============================

Number of cycles performed: 4
Total number of frequent sequences: 17

Frequent Sequences Details (filtered):

- 1-sequences

[1] <{Facebook}> (11)
[2] <{Messenger}> (10)
[3] <{Email}> (9)
[4] <{Internet}> (11)
[5] <{PDFReader}> (10)
[6] <{Musique}> (10)

- 2-sequences

[1] <{Facebook}{Internet}> (11)
[2] <{Facebook}{PDFReader}> (10)
[3] <{Messenger}{Facebook}> (10)
[4] <{Messenger}{Internet}> (10)
[5] <{Messenger}{PDFReader}> (10)
[6] <{Internet}{PDFReader}> (10)

- 3-sequences

[1] <{Facebook}{Internet}{PDFReader}> (10)
[2] <{Messenger}{Facebook}{Internet}> (10)
[3] <{Messenger}{Facebook}{PDFReader}> (10)
[4] <{Messenger}{Internet}{PDFReader}> (10)

- 4-sequences

[1] <{Messenger}{Facebook}{Internet}{PDFReader}> (10)
```

On voit dans nos résultats que l'algorithme trouve des patterns jusque 4 applications !

Ce qui est très intéressant pour le power management. En effet avec le pattern Messenger/Facebook/Internet/PDFReader on sait que si l'utilisateur enchaîne Messenger/Facebook/Internet, alors il y a de très fortes chances que l'application suivante soit PDFReader. Ainsi on peut désactiver ce qui n'est pas utile pour l'application PDFReader comme le GPS, le Bluetooth, potentiellement la DATA et le Wifi. Ceci nous donne une règle qui sera applicable plus tard lors de l'utilisation future d'un smartphone.

GSP ne prend pas en compte une suite directe d'application c'est pour ça qu'on retrouve dans les séries de 3 Messenger/Facebook/PDFReader par exemple bien qu'il y ait quasiment toujours dans nos données « Internet » entre Facebook et PDFReader. Mais on peut quand même avoir une règle qui suppose qu'après Messenger et Facebook alors l'utilisateur va lancer PDFReader et on peut donc couper les modules inutiles.

D’autre part, nous avons utilisé un algorithme de clustering afin d’obtenir des « groupes » de personne pour mieux identifier les utilisateurs.

Pour ce faire nous avons utilisé l’algorithme SimpleKMeans, proposé dans Weka. L’algorithme donne des groupes en fonction des données. Une personne appartiendra donc à un groupe qui représente des sortes d’habitudes d’utilisation. Dans Weka nous pouvons choisir le nombre de groupe que nous souhaitons obtenir, nous l’avons fixé à 3. Les données utilisées correspondent à 5 paramètres, la wifi, le GPS, la Data, le mode économie d’énergie, la Bluetooth. Chaque ligne de donnée indique ainsi si l’utilisateur active ou non un des paramètres. 0 si le paramètre n’est pas actif, 1 sinon. Ainsi pour un utilisateur utilisant la wifi et le GPS la ligne correspondante sera : 1,1,0,0,0.

Le fichier que nous avons utilisé est : « donneeTel - Clusturing.arrf ».

Les résultats que nous avons obtenus avec Weka sont les suivants :

```
Final cluster centroids:
                           Cluster#
Attribute       Full Data          0          1          2
                   (54.0)     (21.0)     (15.0)     (18.0)
=========================================================
wifi_ON             0.6667     0.8571          1     0.1667
GPS_ON              0.4444     0.8571        0.2     0.1667
Data_ON             0.6667          1        0.8     0.1667
ecoEnergy_ON        0.3333          0          0          1
Bluetooth_ON        0.4444          1          0     0.1667
```

On obtient donc 3 groupes avec des habitudes d’utilisations bien différentes !

Le groupe 0 a tendance a utilisé tous les modules (> 0,85) sauf celui d’économie d’énergie (= 0), on peut donc dire que c’est le groupe des « non-écolos ».

Le groupe 1 utilise lui quasiment uniquement la Wifi et la Data (> 0,8) on pourrait donc le nommer le groupe des « moyen-écolos ».

Enfin le groupe 2 lui utilise quasiment uniquement le mode économie d’énergie ! C’est donc le groupe des « écolos ».

Ces données sont très utiles pour le power management, en effet en regardant quels modules sont utilisés on peut catégoriser les utilisateurs et faire des actions en fonction du groupe auxquels ils appartiennent. Par exemple si on détecte que la personne est dans le groupe des écolos et qu’elle a activé la Bluetooth depuis longtemps, on peut se dire que la personne a oublié de couper son module de Bluetooth et on pourrait donc le couper pour elle automatiquement.

Et ceci n’est qu’un exemple parmi tant d’autres mais on pourrait faire de même avec les autres groupes. En prenant le groupe des non-écolos, on voit qu’ils utilisent quasiment tous les modules. Si on détecte une personne dans ce groupe on pourrait s’attarder sur l’utilisation de son Bluetooth pour être sûr qu’il l’utilise (même si on sait que le module est actif, cela ne veut pas dire qu’il est utilisé…) Ainsi si on détecte que le module n’est pas vraiment utilisé on pourrait tenter de le désactiver pour voir s’il active parce qu’il en a besoin ou plus par « mauvaise » habitude.

Pour continuer avec le power management, nous avons parlé de quelques règles précédemment et il faudrait donc une application pour détecter si on se trouve dans l'une des règles ou non et donc s'il faut faire une action ou non. Nous n'avons pas fait cette application mais nous avons testé différents bouts de codes qui ont pour but de désactiver ou non un module ou de régler la luminosité etc. Nous avons donc déjà tout le code permettant de faire les actions.

Luminosité :

```
private static void setBrightness(Context context, int brightnessValue) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.FROYO) {
        Settings.System.putInt(context.getContentResolver(),
                Settings.System.SCREEN_BRIGHTNESS_MODE,
Settings.System.SCREEN_BRIGHTNESS_MODE_MANUAL);
    }

    Settings.System.putInt(context.getContentResolver(),
Settings.System.SCREEN_BRIGHTNESS, brightnessValue);
}
```

Volume sonore :

```
private static void adjustVolume(Context context, boolean isVolumeMustBeLower) {
    Log.d(TAG, "adjustVolume :" + isVolumeMustBeLower);
    AudioManager audioManager = (AudioManager)
context.getSystemService(Context.AUDIO_SERVICE);

    audioManager.adjustVolume(isVolumeMustBeLower ? AudioManager.ADJUST_LOWER :
AudioManager.ADJUST_RAISE, AudioManager.FLAG_PLAY_SOUND);
}
```

Changement module Bluetooth :

```
private static void bluetoothToggleState() {
    BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    if (bluetoothAdapter == null) {
        Log.e(TAG, "Bluetooth is not supported on this hardware platform.");
        return;
    }

    if (bluetoothAdapter.isEnabled()) {
        bluetoothAdapter.disable();
    } else {
        bluetoothAdapter.enable();
    }
}
```

Changement module WIFI :

```
private static void wifiToggleState(Context context) {
    WifiManager wifiManager = (WifiManager)
context.getSystemService(Context.WIFI_SERVICE);

    if (wifiManager == null) {
```

```
        Log.e(TAG, "Wifi is not supported on this hardware platform.");
        return;
    }

    wifiManager.setWifiEnabled(!wifiManager.isWifiEnabled());
}
```

Changement module DATA :

```
private static void dataToggleState(Context context) {
    try {
        final ConnectivityManager conman = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
        final Class conmanClass = Class.forName(conman.getClass().getName());
        final Field connectivityManagerField =
conmanClass.getDeclaredField("mService");
        connectivityManagerField.setAccessible(true);
        final Object connectivityManager = connectivityManagerField.get(conman);
        final Class connectivityManagerClass =
Class.forName(connectivityManager.getClass().getName());


        final Method getMobileDataEnabledMethod =
connectivityManagerClass.getDeclaredMethod("getMobileDataEnabled");
        getMobileDataEnabledMethod.setAccessible(true);
        boolean isDataEnabled = (boolean)
getMobileDataEnabledMethod.invoke(connectivityManager);
        Log.d(TAG, "dataToggleState isDataEnabled ? " + isDataEnabled);

        final Method setMobileDataEnabledMethod =
connectivityManagerClass.getDeclaredMethod("setMobileDataEnabled", Boolean.TYPE);
        setMobileDataEnabledMethod.setAccessible(true);
        setMobileDataEnabledMethod.invoke(connectivityManager, !isDataEnabled);

    } catch (ClassNotFoundException | NoSuchFieldException |
IllegalAccessException | NoSuchMethodException | InvocationTargetException e) {
        //FIXME : Ces méthodes ne sont utilisables jusque android kitkat mais pas
de tél android marshmallow pour tenter mieux
        Log.e(TAG, "Error during dataToggleState :" + e);
    }
}
```