Cuckoo Manual

August 28, 2013

1 Introduction

Cuckoo is a framework for computation offloading applications on Android. Some applications might contain compute intensive tasks that can better be executed on a *remote* resource, such as a laptop, desktop or cloud environment, than on the device itself. For such tasks computation offloading can be used. Computation offloading transfers execution of compute tasks to remote resources to get the result of the task quicker or to save energy usage on the mobile device.

The Cuckoo framework aids application developers in transforming a regular Android application into a computation offloading application. Furthermore, at runtime it decides whether or not to offload the particular task. This decision has to be made, because the circumstances (network type and status, invocation paramaters of the method call) of method invocations on mobile devices change continuously, making offloading sometimes beneficial and sometimes not.

Furthermore, the Cuckoo framework contains a server application that can be installed on remote resources and a resource manager app for mobile phones, that can manage the remote resources that are known to the mobile device and can be used for apps on that mobile device that are built with Cuckoo.

This document describes how to use Cuckoo. It describes how to write a Cuckoo app and how to run this app. For more in depth discussion of the framework, please read Chapter 2 of the PhD thesis: Programming Frameworks for Distributed Smartphone Computing by Roelof Kemp.

2 Write a Cuckoo App

In this section we provide a step by step guide on how to write a Cuckoo application.

2.1 Prerequisites

- Eclipse integrates with the Android plugin for the Eclipse IDE. Download and install Eclipse and the Android plugin (ADT) as described at: `http://www.eclipse.org` and `http://developer.android.com/tools/sdk/eclipse-adt.html`. Please check the versions of the Android Plugin and

SDK with the SDK Manager in Eclipse. The Android SDK Build Tools are tested with version 18.0.1, the Android SDK Tools with version 22.0.5.

- Download the Cuckoo library from github:
  `https://github.com/interdroid/cuckoo-library`.
  Compile the library to generate the appropriate jar files using ant in the project root:
  `$ ant`

- Install the Cuckoo plugin for Eclipse. Open Eclipse. Click the "Help" menu, select "Install New Software...". Enter the following in the "Work with" field: `https://raw.github.com/interdroid/cuckoo-plugin-feature/master/exported`, press Enter. Make sure that you uncheck: "Group items by category". Select "Cuckoo Plugin Feature". Press the "Next" button. Continue the wizard until the plugin is installed. Restart Eclipse.

- Set the Cuckoo Library Location in the Eclipse Preferences (Mac: Eclipse > Preferences, Ubuntu: Window > Preferences). Open the Eclipse Preferences from the menu. Select "Cuckoo Preferences", enter the path to the root directory of the Cuckoo library, make sure that the library contains the jar files in the "lib" directory.

2.2 Make a project offloadable

Write the Android application. Make sure that any compute intensive code is performed in an Android Service (see: `developer.android.com/reference/android/app/Service.html`) and that this service runs in a separate process. To make a service running in a separate process, you can add the following to the service tag in the AndroidManifest.xml: android:process=":remote". You should communicate with the service by binding to it and invoking methods on the proxy. For this reason you should write an AIDL file that exposes the interface of the service and implement this interface accordingly in the service (see: `http://developer.android.com/guide/components/aidl.html`).

So far it's all basic Android stuff. Now the only thing you have to do to make the app offloadable is to add the Cuckoo "nature". You can do this by right-clicking on the project and selecting "Make Offloadable with Cuckoo" from the "Android Tools" menu item in the context menu.

Then the Cuckoo libraries are added to your project (i.e. in the libs directory), a dummy remote implementation is generated in the directory "remote" within your project root directory, and the jar resulting from this remote implementation is put in the "assets" folder of your project. Furthermore the code that is generated for the stub/proxy pair as a result of the AIDL specification is altered, so that Cuckoo calls are inserted.

Now you should replace the dummy remote implementation with a real remote implementation. You might want to copy-paste the local implementation from your service, or create an entirely different implementation. Keep

in mind that the remote code cannot access Android specific libraries that are not purely implemented in Java. The Eclipse plugin automatically compiles your Java code into the jar that is placed in the assets folder. You can use external libries for your remote implementation. Place them in the "remote/<packagename>/external" directory.

Make sure you add the following permissions to the AndroidManifest as they are required by the Cuckoo runtime:

- android.permission.INTERNET
- android.permission.ACCESS_WIFI_STATE
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_FINE_LOCATION

Now the application is ready to be deployed on the device.

2.2.1 Control over offloading

- In the AIDL file one can describe the offloading strategy. The strategy can be "local", "remote", "energy", "speed" and "parallel". Cuckoo also supports a multifold offloading strategy of "energy" combined with "speed". In this case Cuckoo will offload the execution of the method if any of the strategies results in a positive answer on the question: Should we offload? The order of consulting the model for energy usage and execution time depends on the order within the purpose description – "energy/speed" will first consult the energy model. If the strategy is remote, Cuckoo will always try to offload the method, because the remote implementation will for instance deliver a better quality result. By default the purpose of offloading is "speed/energy". You can add a strategy by providing a special line with comments above the method in the AIDL file as shown in Figure 1:

- You can help Cuckoo in estimating the runtime of a method by overriding the method weight function in the Stub implementation in the service. This weight method has the same signature as the method defined in the AIDL file, but its name is prefixed with `weight_` and the return type is of type double. The default implementation returns 1, but if the execution time of the method depends on the variables, you can assign a weight based on the variables. For instance an image operation that scales per pixel can return the number of pixels as the method weight. Any measured runtime results will then be normalized to the time per pixel and subsequently a better estimate can be made for a new invocation with a known amount of pixels.

- Likewise, you can help Cuckoo in estimating the return size of the method by overriding the `returnSize_` variant of the method in the Stub.

```
package interdroid.photoshoot;

import interdroid.photoshoot.Face;

// cuckoo.strategy=speed
interface FaceDetectorInterface {
  List<Face> findFaces(in byte[] jpegData, int width, int height, int ↩
      maxFaces);
}
```

```
float weight_findFaces(in byte[] jpegData, int width, int height, int ↩
    maxFaces) {
    return width * height;
}
```

```
long returnSize_findFaces(in byte[] jpegData, int width, int height, ↩
    int maxFaces) {
    return 100;
}
```

Figure 1: The interface of the face detection service in AIDL. The services will search for a maximum number of faces in the provided image and returns a list of Face objects that it has found.

- If you want to disable a particular method for offloading, put this line in front of it in the AIDL file:
  `// cuckoo:enabled=false`
- Some phones put the WiFi radio in a lower power state after some time, which can delay receiving the response with up to hundreds of milliseconds. To prevent this Cuckoo can keep the radio active continuously, or try to activate the radio just in time. This can be configured through the following system property.
  ```
  System.setProperty(Cuckoo.WIFI_WAKE_STRATEGY,
      Cuckoo.WIFI_WAKE_STRATEGY_SLEEP);
  ```
  Cuckoo supports the following strategies WIFI_WAKE_STRATEGY_SLEEP (does not try to keep the radio active, default if the offloading strategy is "energy"), WIFI_WAKE_STRATEGY_AWAKE, (default if the offloading strategy is "speed") and WIFI_WAKE_STRATEGY_JIT (which tries to wake the WiFi radio just in time).

3 Run a Cuckoo App

In this section we describe how to run a Cuckoo App. To run a Cuckoo app a couple of things have to be done. There must be a remote Cuckoo server running. This server should be bound to the mobile device. And the mobile device should contain an app with offloadable code.

3.1 Start Cuckoo Server

To start a Cuckoo server, download the code from: `https://github.com/interdroid/cuckoo-library`. Compile the code using ant:

```
$ ant
```

Then start the Cuckoo server using the command:

```
$ java -cp .:lib/server/* -Dlog4j.configuration=file:log4j.properties
  interdroid.cuckoo.server.CuckooServer
```

If you run this command from graphic user environment, a popup with a QR-code is shown, otherwise the server generates a file 'qr.png' in the root directory. The QR-code is needed to bind the server to the mobile device.

3.1.1 Advanced Items for the Cuckoo Server

- If you use SSH to access your remote machine you can start the server with 'nohup' to keep it running even if your SSH connection terminates.
- By default the server runs on port 9000 and assumes its IP-address can be reached by the mobile device (e.g. public or in the same local area network). Firewalls may prevent connections.
- The server produces output to standard err/out and with log4j. Both can with the appropriate redirecting be stored in different files.
- The server caches code it receives from mobile devices. If you update your code on the mobile device you should delete the code from the 'services' directory at your server and also restart the server.
- The Cuckoo server can be configured using a configuration file called "cuckoo.properties" in the root directory of the project. Such a file looks like this:

```
# cuckoo properties

# ipaddress is determined automatically, if this use the property
cuckoo.server.ipaddress = 123.123.123.123
# upload speed to internet in bytes/ms (equiv. to kilobytes/↩
    second)
cuckoo.server.upload = 10000
# upload variance
cuckoo.server.upload.variance = 1000
# download speed from internet in bytes/ms (equiv. to kilobytes/↩
    second)
cuckoo.server.download = 7500
# download variance
cuckoo.server.download.variance = 1000
# geolocation of server (latitude,longitude)
cuckoo.server.location = 52.333045,4.867841
# bssids of accesspoints within the LAN of the resource (comma ↩
    separated list)
cuckoo.server.bssids = 0:12:7f:50:a4:10,0:12:7f:50:a4:13
```

3.2 Connect Server with Mobile Device

Cuckoo comes with a separate app, which is called the Resource Manager, which you can download from `https://github.com/interdroid/cuckoo-resource-manager`. Use this app to scan the QR code generated by starting the service. By scanning the QR-code the mobile device knows the remote resource and can use it for offloading. You can scan the code by using the "Add resource" option in the options menu. After scanning, the app will add the scanned resource to the list of known resources, and will show you the list, after which you can run the Cuckoo app.

3.3 Run the Cuckoo App

Now everything is ready for the offloading app to run. The first time it will run in parallel, both local and remote to get statistics. Also the jars from the assets directory will be transfered to the remote resource.