TUGAS AKHIR - KI141502

RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)

PUTRO SATRIO WIBOWO
NRP 5113100130

Dosen Pembimbing I
Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.

Dosen Pembimbing II
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

RANCANG BANGUN MOBILE COMPUTATION OFFLOADING FRAMEWORK UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA SMOARTPHONE (STUDI KASUS LEAF RECOGNITION)

PUTRO SATRIO WIBOWO
NRP 5113100130

Dosen Pembimbing I
Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.

Dosen Pembimbing II
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

DESIGN OF MOBILE COMPUTATION OFFLOADING FRAMEWORK FOR IMPROVE PERFORMANCE AND SAVING OF POWER IN SMARTPHONE (LEAF RECOGNITION CASE STUDY)

PUTRO SATRIO WIBOWO
NRP 5113100130

First Advisor
Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.

Second Advisor
Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
PUTRO SATRIO WIBOWO
NRP: 5113100130

Disetujui oleh Pembimbing Tugas Akhir:

1. Waskitho Wibisono, S.Kom., M.Eng., Ph.D.
   (NIP. 197410222000031001) ..... (Pembimbing 1)

2. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
   (NIP. 197708242006041001) ..... (Pembimbing 2)

SURABAYA
JUNI, 2017

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)

Nama Mahasiswa	**: PUTRO SATRIO WIBOWO**
NRP	**: 5113100130**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**
Dosen Pembimbing 2	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**

Abstrak

Offloading adalah suatu metode pengeksekusian sebuah beban kerja pada sebuah perangkat dengan mengirimkan modul berisi beban kerja tersebut kepada perangkat lain yang memiliki sumber daya dan kemampuan komputasi yang lebih baik. Hasil eksekusi dari beban kerja akan diterima kembali oleh perangkat yang telah mengirim modul beban kerja sebelumnya. Teknik ini dianggap sebagai salah satu cara mengatasi keterbatasan perangkat bergerak yang memiliki sumber daya dan kemampuan komputasi yang terbatas. Oleh karena itu, diperlukan adanya penerapan metode offloading dalam mengeksekusi beban kerja pada perangkat bergerak dengan tujuan dapat melakukan penghematan sumber daya dan peningkatan kinerja pada perangkat bergerak.

Beban kerja dengan kriteria tertentu yang dapat di eksekusi dengan metode offloading dan terdapat faktor yang dapat menghambat proses offloading sehingga dapat tidak menguntungkan pada sumber daya dan kinerja pada perangkat bergerak. Oleh Karena itu, Tugas Akhir ini mengimplementasikan sebuah mobile framework yang dapat menentukan secara dinamis

metode eksekusi yang optimal pada beban kerja yang akan dieksekusi.

Pada tugas akhir ini, penggunaan JADE middleware dalam implementasi metode offloading dianggap efektif dan proses image recognition digunakan sebagai beban kerja. Hasil yang didapat dari kedinamisan mobile framework dalam menentukan keputusan metode eksekusi dianggap dapat meminimalisir penggunaan daya dan meningkatkan kinerja pada perangkat bergerak. Dari uji coba didapatkan penghematan memori paling tinggi 0.259%, penghematan penggunaan CPU paling tinggi 40.379%, penghematan level baterai paling tinggi 1%, dan penghematan waktu eksekusi rata – rata paling tinggi 2.428 detik untuk setiap beban kerja.

Kata kunci: Offloading, mobile framework, image recognition, JADE middleware, beban kerja.

DESIGN OF MOBILE COMPUTATION OFFLOADING FRAMEWORK FOR IMPROVE PERFORMANCE AND SAVING OF POWER IN SMARTPHONE (LEAF RECOGNITION CASE STUDY)

Student's Name	**: PUTRO SATRIO WIBOWO**
Student's ID	**: 5113100130**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**
Second Advisor	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**

Abstract

Offloading is a method of executing a workload on a device by sending a module containing workload to other device that have better resources and computing capabilities. The execution results of the workload will be received by the device that has sent the previous workload module. This technique is considered as one of the ways to overcoming the limitations of mobile devices that have limited resources and computing capabilities. Therefore, it is necessary to apply the offloading method to executing the workload on a mobile device with purpose of saving the resources and improving performance on mobile devices.

Workload with certain criteria that can be executed by offloading method and there are factors that can inhibit the offloading process so it can be unfavorable on the resources and performance of mobile devices. Therefore, this study implements a mobile framework that can dynamically determine the optimal execution method on the workload to be executed.

In this undergraduate thesis, the use of JADE middleware in the implementation of offloading method is considered effective and image recognition process is used as workload. The results obtained from the dynamics of the mobile framework in

determining the decision of the execution method are considered to minimize the use of power and improve performance on mobile devices. From the experiment, the highest memory saving is 0.259%, the maximum CPU usage is 40.379%, the battery saving rate is 1% high, and the average execution time saving is 2,428 sec for each workload.

Keywords : Offloading, mobile framework, image recognition, JADE middleware, workload.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)**"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak Wasisto dan Ibu Nanik Wahyuni selaku orang tua penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku pembimbing I yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini sekaligus dosen wali penulis yang telah memberikan arahan, masukan dan motivasi kepada penulis.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. selaku II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS.
5. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc. selaku koordinator dan sebagai dosen penguji Tugas Akhir penulis.

6. Bapak Tohari Ahmad, S.Kom., MIT., Ph.D. sebagai dosen penguji Tugas Akhir penulis.
7. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
8. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
9. Teman-teman Keluarga Muslim Informatika, yang sudah banyak meluruskan penulis.
10. Teman-teman seperjuangan RMK NCC/KBJ, yang telah menemani dan menyemangati penulis.
11. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
12. Teman-teman angkatan 2013, yang sudah mendukung saya selama perkuliahan.
13. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2017

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi komunikasi saat ini telah berkembang dengan sangat pesat, khususnya dibidang perangkat bergerak. Menurut laporan Indeks Cisco *Visual Networking*, jumlah perangkat bergerak akan melebihi jumlah orang di bumi pada akhir 2015 [1]. Perangkat bergerak yang hampir dimiliki setiap orang saat ini adalah *smartphone*/ponsel cerdas, hal ini dikarenakan *smartphone* sudah menjadi tren disamping harganya yang terjangkau pada kalangan masyarakat. *Smartphone* adalah telepon genggam yang mempunyai kemampuan dengan pengunaan dan fungsi yang menyerupai komputer [2]. *Smartphone* adalah perangkat bergerak yang mayoritas menggunakan OS Android saat ini. Android adalah sistem operasi perangkat bergerak yang bersifat *open source* yang berarti penggunanya dapat mengembangkan aplikasi versi mereka sendiri di dalam sistem operasi. Selain itu, Google Inc. selaku pemilik Android menyediakan *source code* untuk memfasilitasi pengguna Android membangun dan mengembangkan aplikasi. Dengan fakta tersebut, perangkat bergerak berbasis Android adalah perangkat bergerak yang cocok dan mudah untuk digunakan untuk pengembangan aplikasi - aplikasi yang akan dibangun sendiri oleh *developer*.

Berbagai macam aplikasi telah muncul dengan tujuan mempermudah dalam mendapatkan kenyamanan sehingga mengubah gaya hidup masyarakat dunia saat ini. Namun demikian, perangkat bergerak memiliki masalah tersendiri yang tidak bisa dihindari seperti sumber energi baterai yang memiliki daya tahan yang terbatas, konektivitas yang tidak stabil dan terbatas, dan kemampuan komputasi yang terbatas [1]. Faktanya, kebanyakan aplikasi yang dioperasikan pada perangkat bergerak seperti *smartphone* membutuhkan sumber daya yang besar seperti menjalankan navigasi, melakukan pengolahan citra gambar secara

high-definition, *face reognition*, *online mobile games*, dan pengolahan data sensor yang tersedia pada *smartphone* tersebut [1].

Dalam perkembangannya, teknologi komputasi *cloud* berbasis *mobile* telah direncanakan sebagai teknologi yang menjanjikan untuk mengatasi keterbatasan perangkat bergerak [1]. Mekanisme komputasi *cloud* secara *offloading* dianggap sebagai salah satu teknik yang paling baik dan sangat diperlukan yang berpotensi dalam menghemat energi untuk pengguna perangkat bergerak [1]. Namun, upaya penelitian saat ini untuk mekanisme *offloading* masih terbatas dan memiliki banyak kekurangan [1]. Pada aplikasi perangkat bergerak Android seperti *smartphone*, terdapat *task* komputasi yang harus dieksekusi untuk memperoleh data yang diinginkan. Perkembangan terbaru dalam komputasi awan secara *offloading* menunjukkan bahwa tidak semua aplikasi *smartphone* yang cocok untuk diterapkannya *offloading*, hal ini bergantung pada karakteristik beban kerja [1]. Selain itu, *offloading* tidak dapat dilakukan jika *smartphone* tidak memiliki koneksi internet dan juga tidak dapat diterapkan pada beberapa proses komputasi *smartphone* yang harus dilakukan secara lokal. Keputusan dalam menentukan sebuah proses komputasi aplikasi dilakukan secara metode *offloading* atau secara lokal dipengaruhi oleh beberapa faktor – faktor yang menjadi tantangan yang akan diteliti pada tugas akhir ini.

Hasil yang diharapkan dari pengerjaan tugas akhir ini adalah aplikasi perangkat *mobile* Android (*client*) dapat menentukan sebuah komputasi dilakukan secara *offloading* pada *server* atau lokal pada kondisi tertentu secara optimal agar mendapatkan *execution time* paling minimal dan penghematan sumber daya baterai yang maksimal pada perangkat di sisi *client*.

1.2 Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana menerapkan metode *offloading computation* pada *mobile framework* yang dapat meminimalisir penggunaan sumber daya baterai dan meningkatkan performa proses komputasi dalam perangkat *client* ?
2. Bagaimana menentukan penerapan metode *offloading computation* pada *mobile framework* dengan menggunakan faktor – faktor pendukung yang telah disebutkan sebagai acuan ?
3. Bagaimana menentukan format modul pengiriman data yang berisi informasi beban kerja untuk *offloading* ke *server* dan penerimaan hasil proses data pada *client* ?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada tugas akhir ini memiliki batasan sebagai berikut:

1. Sistem operasi pada perangkat *smartphone* adalah Sistem Operasi Android.
2. Data citra daun yang digunakan untuk *image recognition* adalah dataset daun dari UCI Machine Learning [3].
3. Metode yang digunakan untuk meningkatkan kinerja dan penghematan daya pada *smartphone* adalah metode *offloading*.
4. Library yang digunakan untuk pengolahan citra digital adalah OpenCV.
5. Perangkat lunak yang digunakan adalah Android Studio dan Netbeans sebagai IDE serta JADE sebagai *middleware* antara *client* dan *server*.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Membangun sebuah *offloading computation* pada *mobile framework* yang dapat menentukan secara dinamis dan optimal suatu beban kerja diproses secara *offloading computation* ke *server* atau secara lokal berdasarkan kondisi faktor – faktor pendukung yang dialami saat itu oleh perangkat bergerak di sisi *client*.
2. Melakukan implementasi metode *offloading* untuk penghematan daya dan peningkatan kinerja perangkat bergerak.

1.5 Manfaat

Dengan dibuatnya tugas akhir ini diharapkan dapat memberikan manfaat untuk menghasilkan sebuah *Offloading computation mobile framework* yang dapat menentukan secara dinamis pemrosesan suatu beban kerja yang optimal dengan tujuan mengurangi penggunaan sumber daya energi baterai yaitu menggunakan seminimal mungkin sumber daya komputasi pada perangkat bergerak dan meningkatkan kecepatan performa perangkat bergerak dengan mengirimkan beban kerja kepada perangkat yang memiliki sumber daya komputasi yang lebih baik.

Sedangkan bagi penulis, tugas akhir ini bermanfaat sebagai sarana untuk mengimplementasikan ilmu dan algoritma *image recognition* serta pemrosesannya yang dilakukan secara *thread processing* yang telah dipelajari selama kuliah agar berguna bagi masyarakat.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal

Tahap awal tugas akhir ini adalah menyusun proposal tugas akhir. Pada proposal, diajukan gagasan untuk menerapkan

offloading computation mobile framework sebagai solusi dari keterbatasan perangkat bergerak Android khususnya daya tahan baterai sebagai sumber energi serta keterbatasan *processor* yang memproses komputasi pada setiap beban kerja perangkat lunak.

1.6.2 Studi Literatur

Pada tahap ini dilakukan untuk mencari informasi dan studi literatur apa saja yang dapat dijadikan referensi untuk membantu pengerjaan Tugas Akhir ini. Informasi didapatkan dari buku dan literatur yang berhubungan dengan *computation offloading mobile framework* yang akan dibangun dan metode untuk *image recognition*. Informasi yang dicari adalah implementasi *computation offloading* pada perangkat bergerak Android, faktor – faktor yang digunakan untuk menentukan dilakukannya *computation offloading*, dan metode-metode *image recognition* seperti metode SURF (*Speeded Up Robust Features*). Tugas akhir ini juga mengacu pada literatur jurnal karya Hao Qian dan Daniel Andresen dengan judul "*Jade: Reducing Energy Consumption of Android App"* yang diterbitkan pada tahun 2015.

1.6.3 Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun *framework* yang telah disebutkan sebelumnya, maka dilakukan implementasi dengan menggunakan suatu perangkat lunak yaitu Android Studio dan Netbeans sebagai IDE, JADE sebagai *middleware* antara *client* dan *server.*

1.6.4 Pengujian dan Evaluasi

Pada tahap ini *framework* yang telah dibangun diuji coba dengan menggunakan data uji coba yang ada. Data uji coba tersebut diuji coba dengan menerapkan *framework* pada perangkat lunak perangkat bergerak Android sebagai *client* dan perangkat PC

sebagai *server* dengan tujuan mengetahui kemampuan *framework* dalam menentukan penggunaan metode *offloading* atau tidaknya (lokal) pada suatu beban kerja berdasarkan faktor – faktor pendukung yang terjadi saat itu pada perangkat *client* dan mengevaluasi hasil tugas akhir dengan jurnal pendukung yang ada. Hasil evaluasi mencakup penghematan daya sumber energi dan waktu eksekusi dari kemampuan *framework* tersebut dalam menentukan *computation offloading* suatu beban kerja pada perangkat bergerak Android.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang sistem operasi Android, metode *offloading*, faktor – faktor penentu dilakukannya metode *offloading*, JADE sebagai *middleware*, GSON dan Commons-Lang sebagai format modul pengiriman data, serta metode SURF dan OpenCV digunakan dalam penerapan *image recognition* pada citra.
3. Bab III. Perancangan Perangkat Lunak

Bab ini berisi pembahasan mengenai perancangan dari *offloading computation framework* yang diterapkan pada perangkat bergerak Android untuk menentukan dilakukannya metode *offloading* pada suatu beban kerja saat melakukan *image recognition* berdasarkan faktor – faktor tertentu.

4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk *Pseudocode* yang berupa *Pseudocode* dari *offloading computation framework* dari sistem perangkat lunak *image recognition* beserta penerapan JADE *middleware* pada client (perangkat bergerak Android) dan server (PC).
5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dari *offloading computation framework* yang digunakan untuk menentukan dilakukannya metode *offloading* pada suatu beban kerja dari sistem perangkat lunak *image recognition* yang sudah diimplementasikan pada *Pseudocode*. Uji coba dilakukan dengan menggunakan datatest dan dataset citra yang memiliki kualitas rendah. Hasil evaluasi mencakup kedinamisan dari kemampuan *offloading computation framework* dalam menentukan dilakukannya metode *offloading* pada beban kerja berdasarkan faktor – faktor penentu yang di alami perangkat client dan server saat itu.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan *Pseudocode* program secara keseluruhan.

(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Teori-teori tersebut diantaranya adalah sistem operasi Android sebagai OS, metode *offloading*, metode SURF untuk *image recognition*, JADE sebagai *middleware*, modul pengiriman data antara *client* dan *server* dan beberapa teori lain yang mendukung pembuatan tugas akhir.

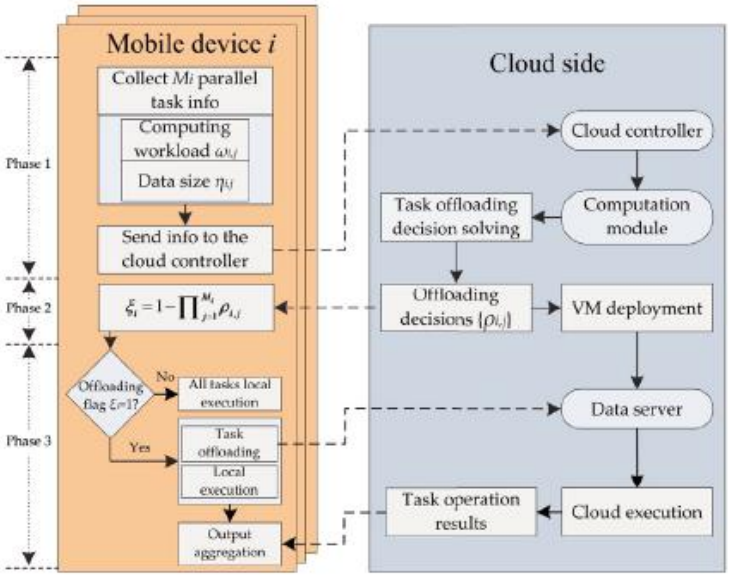
2.1 Sistem Operasi Android

Android adalah sistem operasi berbasis Linux yang dirancang untuk perangkat bergerak layar sentuh seperti telepon pintar dan komputer tablet [4]. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2005 [4]. Android bersifat *open source* yaitu memberikan *developers* untuk mengembangkan versi pribadi mereka dari sistem operasi Android [5]. *Smartphone* berbasis Android umumnya memiliki daya tahan baterai lebih pendek dibandingkan iOS dari iPhone Apple dengan baterai yang lebih besar dan kontrol perangkat lunak pihak ketiga yang ketat. Oleh karena itu, untuk mengurangi tingkat konsumsi daya melalui *background services* merupakan masalah penting untuk komputasi perangkat *mobile* Android. Hal tersebut adalah salah satu alasan Google menerbitkan sebagian besar *source code* untuk Android [5] Dari fakta tersebut, sistem operasi Android sangat cocok untuk digunakan untuk membangun *mobile computation offloading framework*.

2.2 *Offloading*

Metode *Offloading* telah menjadi teknik yang menjanjikan untuk memecahkan masalah yang dihadapi perangkat *smartphone*, yaitu dengan memungkinkan *smartphone* melakukan metode

offload atau mengirimkan suatu beban kerja komputasi yang intensif ke server [5]. Berbagai upaya telah dilakukan untuk menerapkan metode *offload* pada aplikasi Java ke server untuk memperoleh keuntungan dari kapabilitas *crossplatform bytecode* Java [5]. Eksekusi secara *remote* pada server dapat menjadi solusi untuk keterbatasan perangkat *mobile* dan dapat dilihat sebagai cara memperpanjang daya tahan baterai yang merupakan tujuan penerapan *offloading* [6]. Contoh alur eksekusi beban kerja pada penerapan metode *offloading* yang diimplementasikan pada perangkat bergerak dapat dilihat pada Gambar 2.1.



Gambar 2.1 Alur eksekusi beban kerja pada perangkat bergerak ke perangkat komputasi awan [1]

2.3 Faktor Penentu Metode *Offloading*

Keputusan dalam menentukan dilakukannya metode *Offloading* adalah tantangan. Hal ini bergantung pada volume data dan transmisi *bandwidth*, konteks yang di eksekusi dari beban kerja komputasi, perbandingan dalam kemampuan eksekusi antara SoC (*system on chip*) *smartphone* dan *processor* server serta seberapa efek keuntungan jika melakukan metode *Offloading* [5]. Membuat data rinci sebuah beban kerja komputasi dan pemantauan jaringan *quality-of-service* (QoS) sangat penting dalam *framework*. Selain itu, karena komputasi dan karakteristik komunikasi adalah dinamis, sebuah *daemon* yang secara periodik mengukur kapabilitas komputasi, performa komunikasi, konsumsi daya dan secara dinamis membuat keputusan dilakukannya metode *offloading* untuk setiap *task* yang memenuhi syarat untuk dilakukannya *offloading*, hal ini dibutuhkan dalam *framework* [5].

2.4 JADE

JADE adalah singkatan dari *Java Agent DEvelopment Framework*. JADE adalah perangkat lunak *framework* yang sepenuhnya diimplementasikan dalam bahasa Java. JADE mempermudah penggunanya dalam mengimplementasikan sistem *multi-agent* melalui sebuah *middleware* yang sesuai dengan spesifikasi FIPA dan seperangkat alat tampilan grafis yang mendukung fase *debugging* dan *deployment*. Sistem berbasis JADE dapat didistribusikan ke sebagian besar perangkat (meskipun memiliki Sistem Operasi yang berbeda) dan konfigurasi dapat dikendalikan melalui *remote* GUI. Konfigurasi dapat diubah pada saat *runtime* dengan memindahkan agen dari satu perangkat ke perangkat lainnya sesuai kebutuhan. JADE secara penuh diimplementasikan pada Bahasa Java dengan persyaratan minimal sistem adalah versi 5 Java (*runtime environment* atau JDK) [7].

2.5 GSON

GSON *library* awalnya dikembangkan untuk keperluan internal Google. GSON adalah *library* Java yang bisa digunakan untuk mengonversi Objek Java menjadi versi representasi berbentuk JSON. GSON juga dapat digunakan untuk mengubah *string* JSON menjadi Objek Java yang ekuivalen. GSON dapat bekerja Objek Java yang bermacam – macam [8].

2.6 Commons-Lang

Commons-Lang adalah Java *library* yang menyediakan metode untuk memanipulasi kelas inti Java. Apache Commons-Lang menyediakan metode tambahan yaitu sejumlah utilitas pembantu untuk API java.lang, terutama metode manipulasi String, metode numerik dasar, refleksi objek, konkurensi, pembuatan dan serialisasi. Selain itu, perangkat ini berisi perangkat tambahan dasar untuk java.util.Date dan serangkaian utilitas yang didedikasikan untuk menjalankan beberapa metode, seperti *hashCode*, *toString* dan *equals* [9].

2.7 OpenCV

*Open*CV (*Open Source Computer Vision*) adalah *library* yang utamanya digunakan untuk pemrosesan visi komputer. *Open*CV adalah *library* gratis yang dapat digunakan di berbagai *platform*, seperti GNU/Linux maupun Windows. *Open*CV mulanya ditulis dalam bahasa pemrograman C++, namun saat ini *Open*CV dapat digunakan pada berbagai bahasa seperti Python, Java, atau MATLAB [10].

2.8 SURF

Dalam visi computer khususnya *image recognition*, *Speeded Up Robust Features* (SURF) adalah *local feature detector* dan *descriptor* yang dipatenkan. Metode ini dapat digunakan untuk pengenalan objek dan gambar, klasifikasi dan rekonstruksi 3D. Metode ini sebagian terinspirasi oleh *Scale Invariant feature transform* (SIFT) *descriptor*. Versi standar SURF beberapa kali lebih cepat dari SIFT dan diklaim oleh penemunya untuk menjadi lebih tangguh terhadap transformasi citra yang berbeda daripada SIFT. Untuk mendeteksi titik – titik penting fitur pada citra, SURF menggunakan pendekatan bilangan bulat dari determinan *Hessian blob detector*. Deskriptor fitur didasarkan pada jumlah respon *Haar wavelet* di sekitar titik – titik penting fitur. Deskriptor SURF telah digunakan untuk menemukan dan mengenali benda, orang atau wajah, untuk merekonstruksi adegan 3D, untuk melacak objek dan untuk mengekstrak titik penting fitur citra [11].

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN PERANGKAT LUNAK

Bab ini membahas mengenai perancangan dan pembuatan sistem *framework* pada perangkat lunak Android. Sistem *framework* pada perangkat lunak Android yang dibuat pada tugas akhir ini adalah menentukan keputusan dilakukannya metode *offloading* berdasarkan faktor – faktor penentu yang mendukung metode *offloading.*

3.1 Data

Pada sub bab ini akan dijelaskan mengenai data yang digunakan sebagai masukan pada perangkat lunak Android untuk selanjutnya diolah dan dilakukan pengujian secara optimal sehingga menghasilkan data keluaran yang diharapkan dengan waktu pemrosesan dan penggunaan sumber daya seminimal mungkin.

3.1.1 Data Masukan

Data masukan adalah data yang digunakan sebagai masukan awal dari sistem. Data yang digunakan dalam perangkat lunak Android terdiri dari dua jenis data masukan. Data masukan jenis pertama adalah sebuah datatest berupa sebuah citra daun yang dianggap belum diketahui jenisnya. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pengambilan gambar saat menjalankan perangkat lunak Android. Data masukan jenis pertama yang belum diketahui jenisnya tersebut digunakan sebagai datatest untuk uji coba prediksi pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

Sedangkan data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra

daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun. Data masukan jenis kedua telah dilakukan pemrosesan *preprocessing* guna memaksimalkan citra daun untuk proses *image recognition*. Hal yang dilakukan pada tahap *preprocessing* secara umum yaitu mengurangi *noise*, *blurring*, *resizing*, konversi citra ke *grayscale* atau representasi warna lainnya. Data masukan jenis kedua yang berjumlah 40 tersebut digunakan sebagai data masukan untuk uji coba pencocokan dengan datatest pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

Contoh citra sebagai data masukan datatest dan dataset ditunjukkan pada Gambar 3.1 dan Gambar 3.2.

Gambar 3.1 Contoh data masukan citra yang digunakan sebagai datatest

Gambar 3.2 Contoh data masukan citra yang digunakan sebagai dataset

3.1.2 Data Keluaran

Data masukan akan dilakukan *image recognition* dengan menggunakan metode SURF sebagai ekstraksi fitur. Terdapat dua buah data masukan yaitu data citra yang digunakan sebagai datatest dan data citra yang digunakan sebagai dataset. Hasil dari proses *image recognition* antara datatest dengan masing – masing dataset adalah suatu fitur atau ciri – ciri berupa titik – titik pada citra yang dapat digunakan sebagai pembeda pada setiap jenis daun. Dari ciri – ciri tersebut dapat dihitung jarak kedekatan dan kemiripan antara fitur yang dimiliki datatest dan fitur yang dimiliki masing – masing dataset.

3.2 Desain Umum Sistem

Rancangan sistem *framework* pada perangkat lunak *image recognition* menggunakan metode SURF berbasis Android, dimulai dengan mengambil foto gambar citra daun menggunakan kamera yang terpasang pada perangkat bergerak Android. Foto gambar citra daun yang didapat selanjutnya akan digunakan sebagai datatest dan di kompresi agar memiliki kualitas yang sama dengan gambar daun yang digunakan sebagai dataset, kemudian dikonversi menggunakan warna *grayscale*. Tahap selanjutnya adalah membandingkan fitur citra daun datatest dan fitur masing – masing citra daun dataset dengan *image recognition*, yang berarti fitur – fitur yang berada pada satu citra daun datatest akan dibandingkan dengan fitur – fitur yang berada pada setiap citra daun dataset. Metode SURF pada *image recognition* digunakan untuk menghitung jarak kedekatan atau kemiripan antara fitur pada sebuah citra daun datatest dan fitur pada sebuah citra daun dataset. Semakin kecil/mendekati nol nilai jarak kedekatan atau kemiripan antara citra daun datatest dan citra daun dataset maka semakin mirip dan identik kedua citra daun tersebut. Diagram alir langkah kerja secara garis besar *image recognition* pada citra daun datatest dan citra daun dataset menggunakan metode SURF, ditunjukkan pada

Gambar 3.3.

Langkah kerja proses *image recognition* pada perangkat lunak ini dimodifikasi dengan menambahkan metode *offloading* dengan tujuan meminimalisir penggunaan sumber daya dengan waktu eksekusi yang optimal. Metode *offloading* dapat meminimalisir penggunaan sumber daya dan mempercepat waktu eksekusi dengan cara mengirimkan beban kerja proses pada perangkat *client* ke *server* yang memiliki kemampuan komputasi jauh lebih baik dan sumber daya tak terbatas, hasil yang didapatkan akan dikirimkan kembali ke perangkat client. Untuk menggabungkan langkah kerja proses *image recognition* dengan metode *offloading*, dibutuhkan sebuah *framework* untuk

mengurutkan setiap langkah kerja secara runtut serta menentukan beban kerja yang akan di *offload* ke *server*. Selain itu, *framework* juga menentukan eksekusi suatu beban kerja dilakukan secara lokal atau *offloading* sebab metode *offloading* memiliki beberapa persyaratan agar mengeksekusi beban kerja secara *offloading* ke *server* lebih optimal dan menghemat sumber daya daripada eksekusi secara lokal pada perangkat *client*.

Sebelum tahap proses *image recognition* menggunakan metode SURF dilakukan, *framework* melakukan pengambilan data inisialisasi yang digunakan sebagai faktor penentu keputusan dilakukannya metode *offloading* pada saat perangkat lunak melakukan pengambilan foto gambar citra daun sebagai datatest. Metode *offloading* diperlukan sebab proses *image recognition* menggunakan metode SURF membutuhkan sumber daya yang cukup besar sedangkan perangkat bergerak Android memiliki sumber daya baterai dan kemampuan komputasi yang terbatas. Data yang diambil oleh *framework* untuk dijadikan faktor penentu keputusan dilakukannya metode *offloading* pada perangkat lunak diantaranya terdiri dari kecepatan *bandwidth* internet, kondisi level baterai pada perangkat bergerak sebagai inisialisasi awal serta waktu eksekusi proses *image recognition* baik eksekusi secara lokal maupun *offloading* untuk proses *image recognition* selanjutnya.

Data – data yang dijadikan faktor penentu keputusan dilakukannya metode *offloading* pada *framework* adalah kecepatan *bandwidth* internet, kondisi level baterai perangkat bergerak dan waktu eksekusi proses *image recognition*.

Framework melakukan pengujian terhadap data - data yang dijadikan sebagai faktor penentu seperti yang telah disebutkan di atas, pengujian dilakukan dengan menggunakan pembobotan nilai terhadap data – data tersebut. Hasil pengujian berupa keputusan pemrosesan *image recognition* pada perangkat bergerak *client* yang optimal baik itu dilakukan secara lokal maupun *offloading*.

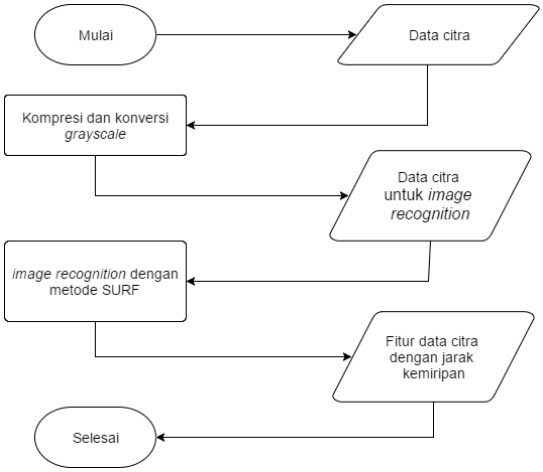
Pada pemrosesan secara lokal, datatest citra daun yang telah diambil dilakukan *image recognition* secara bertahap mulai dari

pengubahan datatest citra daun menjadi bentuk tipe data matriks, pengubahan dataset masing – masing citra daun pada perangkat bergerak *client* menjadi bentuk tipe data matriks, pengambilan *descriptor* menggunakan metode SURF dari masing – masing tipe data matriks datatest maupun dataset dan menghitung jarak kedekatan dan kemiripan dari masing – masing *descriptor* tersebut.

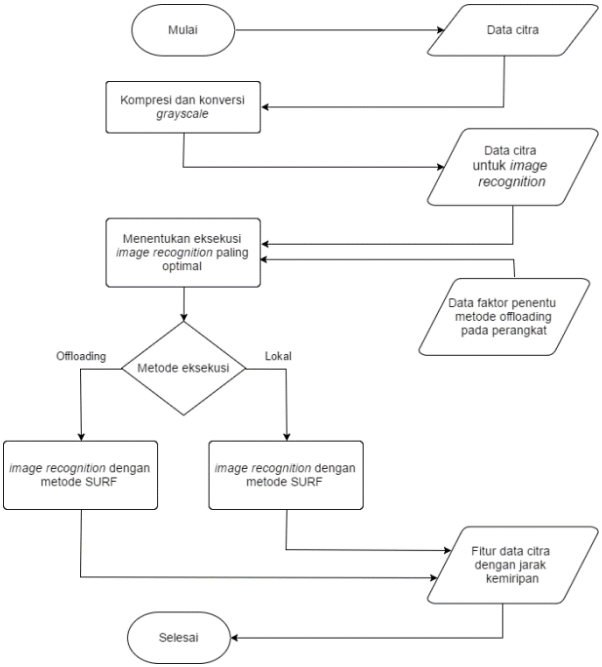
Berbeda halnya dengan pemrosesan secara *offloading*, pada perangkat bergerak *client* dilakukan pengubahan datatest citra daun menjadi bentuk tipe data matriks terlebih dahulu sama seperti pemrosesan secara lokal. Akan tetapi, data matriks tersebut kemudian dikonversi menjadi data *string*, hal ini ditujukan untuk pembuatan modul pengiriman data berbentuk *object serialization* Java yang dikirimkan ke *server*. Konversi dari tipe data matriks menjadi data *string* dilakukan menggunakan *library* GSON. Proses modul pengiriman data pada *server* di awali dengan pembuatan *Agent* JADE pada perangkat bergerak *client* dan *server*. Selanjutnya, perangkat bergerak *client* akan melakukan permintaan dengan mengirimkan pesan *Agent* JADE menuju *server*. *Agent* JADE bertugas untuk mengirimkan dan menerima pesan antara *client* dan *server*. Secara bertahap, *Agent* JADE disisipkan pesan berupa modul pengiriman data yang telah dibuat dan selanjutnya akan dikirimkan pada *server*. *Agent* JADE pada *server* secara responsif akan menerima dan membaca pesan jika ada pesan yang dikirimkan oleh *Agent* JADE *client*. Pesan yang diterima *Agent* JADE *server* yaitu modul pengiriman data yang berbentuk *object serialization* Java. Data *string* dari matriks datatest didalam modul pengiriman data tersebut akan dikonversikan kembali menjadi tipe data matriks menggunakan GSON pada *server*. Data matriks datatest tersebut akan dilakukan pengambilan *descriptor* menggunakan metode SURF untuk dibandingkan dengan *descriptor* dari masing – masing dataset pada *server* yang selanjutnya akan dihitung jarak kedekatan dan kemiripan dari masing – masing *descriptor* tersebut. Data jarak kedekatan antara matriks datatest dan dataset tersebut akan dikirimkan kembali menggunakan modul pengiriman data yang sama yaitu berbentuk

object serialization Java ke perangkat bergerak *client* yang melakukan permintaan dan perangkat bergerak *client* tersebut akan mengambil data yang diperlukan dari modul pengiriman tersebut.

Data hasil dari proses *image recognition* antara datatest dan dataset baik diperoleh dari pemrosesan lokal maupun *offloading* selanjutnya akan diseleksi menurut nilai jarak kedekatannya sesuai dengan kriteria yang ditentukan, hal ini dilakukan untuk mengetahui citra daun dataset yang paling mirip dengan citra daun datatest. Keterangan citra daun dataset yang paling mirip dengan citra daun datatest selanjutnya akan ditampilkan pada tampilan perangkat lunak perangkat bergerak *client*. Pada Gambar 3.4 dijelaskan secara garis besar kolaborasi antara proses *image recognition* dengan *offloading framework*.



Gambar 3.3 Proses *image recognition* menggunakan metode SURF secara garis besar



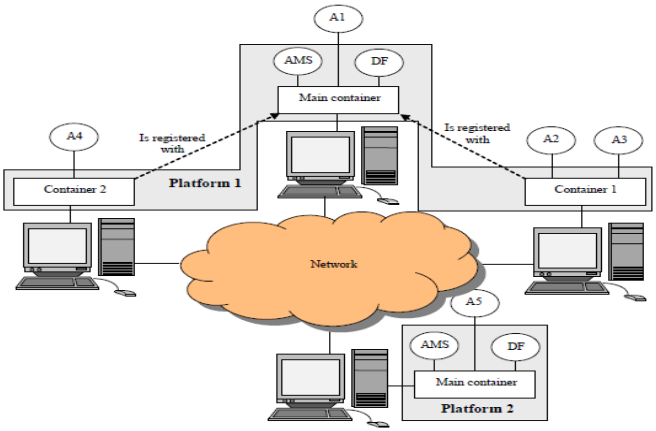
Gambar 3.4 Proses *image recognition* menggunakan metode SURF dengan *offloading framework*

3.3 JADE *Middleware*

JADE merupakan *open source multi agent middleware* dasar yang diimplementasikan dalam Bahasa Java. JADE dapat bertindak sebagai *middleware* yang memfasilitasi pembuatan sistem *multi agent*. Fitur – fitur yang disediakan JADE diantaranya sebuah *runtime environment* dimana JADE *agent* dapat "hidup" dan diaktifkan dengan memberikan alamat *host* sebelum *agent* tersebut dapat dieksekusi, seperangkat kelas *library* yang dibutuhkan dan digunakan *programmer* atau *developer* untuk

pengembangan dan pengaturan *agent* yang telah dibuat, serta seperangkat alat grafis yang memungkinkan pengaturan dan pemantauan aktifitas *agent* yang sedang berjalan.

Setiap *runtime environment* JADE yang berjalan disebut *Container* yang dapat berisi beberapa *agent*. Kumpulan dari *Container* yang aktif disebut *Platform*. Sebuah *Main Container* harus selalu diaktifkan terlebih dahulu dan semua *Container* lain yang terdapat pada sebuah *Platform* harus terdaftar dan dikenali oleh *Main Container* sebelum memulai pengaktifan. Dapat disimpulkan bahwa *Container* pertama yang dibuat oleh *Platform* haruslah sebuah *Main Container* sementara *Container* lain yang bersifat "normal" atau bukan *Main Container* harus "di beri tahu" dimana untuk mencari (*host* dan *port*) *Main Container* dari *Container* tersebut. Jika *Main Container* sebuah *Platform* di aktifkan pada tempat tertentu dalam suatu jaringan, *Platform* lain yang berada dalam satu jaringan dan memiliki *Container* baru bersifat "normal" dapat dimungkinkan untuk dikenali oleh *Main Container* pada *Platform* pertama. Untuk lebih jelasnya dapat dilihat pada Gambar 3.5 di bawah ini.



Gambar 3.5 *Container* dan Platform pada JADE *middleware* [12]

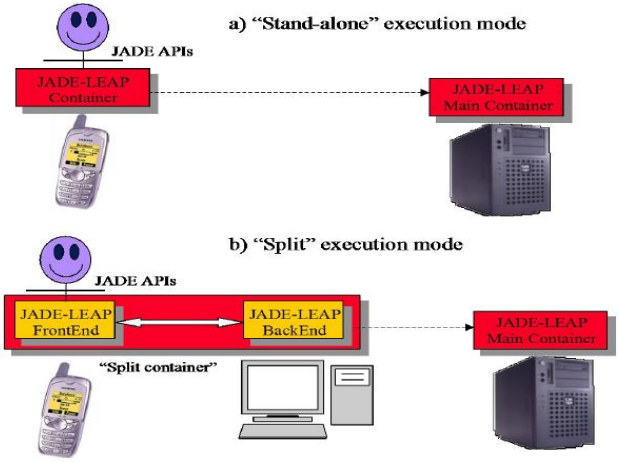
Selain kemampuan untuk mengenal *agent* dari *container* lain, *main container* memiliki perbedaan dengan *container* normal sebab memiliki dua *agent* khusus yang otomatis muncul ketika *main container* dibuat, yaitu AMS (*Agent Management System*) dan DF (*Directory Facilitator*) seperti pada gambar.

AMS menyediakan layanan penamaan *agent* dimana setiap *agent* di dalam *platform* harus memiliki nama yang unik serta mewakili otoritas di dalam *platform* dalam melakukan pembuatan dan penghapusan *agent* pada *remote container*. Pada tugas akhir ini, penulis mengikuti pengaturan dasar dan tidak melakukan perubahan pada AMS.

DF menyediakan layanan *Yellow Pages* yang artinya *agent* dapat mencari *agent* lain yang menyediakan layanan yang ia butuhkan untuk mencapai tujuannya di dalam satu *platform*. Pada tugas akhir ini, penulis tidak menggunakan DF sebab komunikasi yang dibutuhkan antar *agent* hanya berlangsung dua arah pada setiap *client* dan *server*.

JADE *runtime environment* dapat di eksekusi pada dua mode yang berbeda. Dikatakan melakukan secara mode eksekusi normal *Stand – alone* jika *container* yang lengkap (*main container* dan *container* normal lainnya) di eksekusi pada perangkat atau *host* dimana JADE *runtime* diaktifkan. Dikatakan melakukan secara mode eksekusi *Split* jika *container* di pisahkan antara *Front – End* (proses yang berjalan pada perangkat / *host* dimana JADE *runtime* diaktifkan) dan *Back – End* (proses yang diaktifkan pada *remote server*). Pada mode eksekusi *Split*, perangkat *Front – End* dan *Back – End* dihubungkan dengan koneksi internet dan JADE *container* harus diaktifkan terlebih dahulu pada *host* sesuai dengan dimana *Back – End* telah dibuat. *Container* baru yang dibuat antara Front

– End dan Back – End disebut *mediator*. Untuk lebih jelasnya dapat dilihat pada Gambar 3.6 di bawah ini.



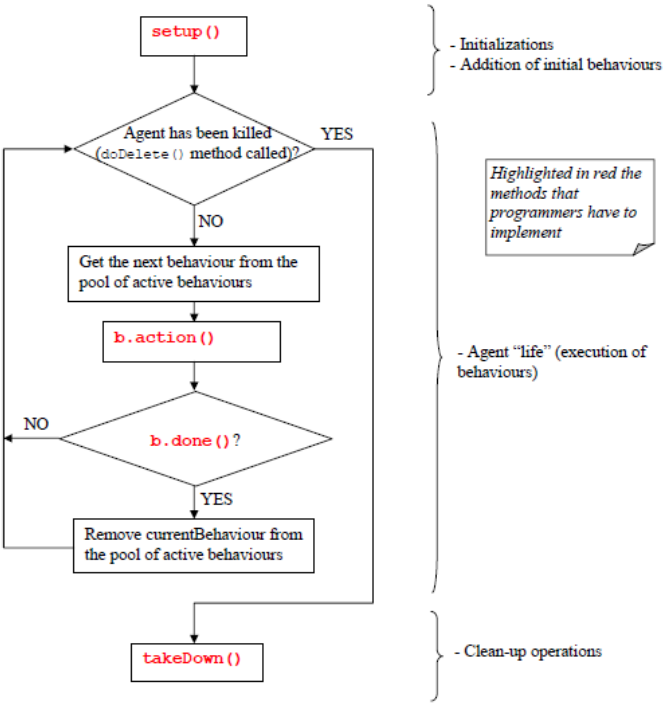
Gambar 3.6 Mode eksekusi pada *runtime* JADE [13]

Dalam pengerjaan pada tugas akhir ini, pembuatan JADE *Main container* dilakukan pada satu *platform*, yaitu pada perangkat PC (*server*). JADE *runtime environment* dilakukan pada mode eksekusi *Split* dimana perangkat bergerak (*client*) bertindak sebagai *Front – End* dan PC (*server*) bertindak sebagai *Back – End*, sehingga perangkat *client* membuat JADE *container* dengan menggunakan *Main container* pada *server*. Proses pada JADE dimulai dengan pengaktifan JADE *runtime* dari PC *server*, dilanjutkan pada Android *Activity* pada perangkat bergerak yang dihubungkan dengan *MicroRuntimeService*. Hasilnya, objek JADE *MicroRuntimeServiceBinder* diaktifkan sedemikian rupa sehingga memungkinkan untuk melakukan semua operasi manajemen JADE, salah satunya yaitu membuat *container* dengan penerapan mode eksekusi *Split*. Hal yang harus diketahui dalam pembuatan *Split container* adalah alamat *host* dan *port* letak *main container* yang sedang berjalan pada *server*. Setelah JADE *runtime*

terhubung dan berjalan, hal ini memungkinkan *client* dan *server* untuk berkomunikasi serta saling mengirim dan menerima pesan.

Langkah pertama yang dilakukan setelah JADE *runtime* yang terhubung adalah melakukan pengaturan pada *agent* yang telah dibuat yaitu menginisialisasi identitas *agent* lain yang menjadi tujuan pengiriman pesan seperti nama *agent*, nama *platform* dan alamat *host* menggunakan *AID*. Begitu pun sebaliknya pada JADE agent *server*, pengaturan *AID* dilakukan dengan menggunakan informasi identitas dari *agent* yang melakukan pengiriman sebelumnya.

Langkah kedua, setiap operasi yang akan dikerjakan oleh JADE *agent* . *Behaviour* dapat ditambahkan secara fleksibel baik ketika pembuatan *agent* (metode *setup*) maupun didalam *behaviour* yang lain. Setiap *behaviour* terdapat metode *action* yang mendefinisikan operasi yang dijalankan ketika *behaviour* dieksekusi dan metode *done* yang menentukan akhir pengeksekusian suatu *behaviour* serta melakukan penghapusan *agent* (metode *dodelete*) dari kumpulan *behaviour* lain dari suatu *agent*. *Agent* dapat mengeksekusi *behaviour* secara bersamaan serta eksekusi akan dimulai saat metode *action* dipanggil dan berjalan sampai mendapatkan hasil dari operasi pada *behaviour*. Untuk lebih jelasnya dapat dilihat pada Gambar 3.7 di bawah ini.

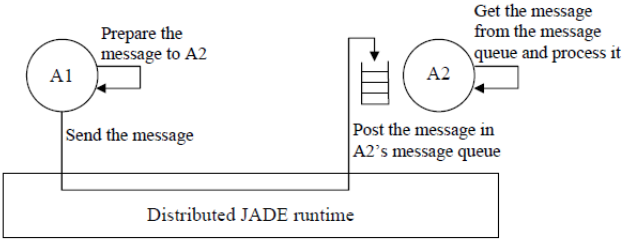


Gambar 3.7 Langkah eksekusi *agent* menggunakan *thread* [12]

Terdapat beberapa macam tipe *behaviour* yang disediakan oleh JADE *agent*. Berdasarkan eksekusi operasi, *behaviour* dibagi menjadi 3 jenis, diantaranya *one – shot behaviour*, *cyclic behaviour*, dan *generic behaviour*. *One – shot behaviour* mengeksekusi metode *action* hanya sekali dengan otomatis memberikan nilai *true* pada metode done. *Cyclic behaviour* mengeksekusi metode *action* yang berisi operasi yang sama secara berulang tanpa akhir dengan otomatis memberikan nilai *false* pada metode done. *Generic behaviour* menyisipkan status dalam mengeksekusi metode *action* yang berisi operasi yang berbeda, *behaviour* ini akan menyelesaikan operasi tertentu pada metode

action ketika syarat kondisi status yang diberikan sesuai. Berdasarkan penjadwalan operasi, *behaviour* dibagi menjadi 2 jenis diantaranya *Waker behaviour* dan *Ticker behaviour*. *Waker behaviour* mengeksekusi operasi dengan waktu tunggu yang ditentukan. *Ticker behaviour* mengeksekusi operasi secara periodik dengan waktu yang ditentukan. Pada tugas akhir ini, tipe *behaviour* yang digunakan adalah *generic behaviour* pada perangkat *client* dan *ticker behaviour* pada perangkat *server* disebabkan terdapat status tertentu yang harus terpenuhi untuk melakukan operasi dalam metode *action*, misalnya pada saat *agent client* melakukan operasi penerimaan pesan dari *agent server* harus dilakukan setelah *agent client* melakukan operasi pengiriman pesan ke *agent server*. Selain itu, *server* harus melakukan pengecekan secara periodik terhadap pesan yang telah dikirimkan oleh perangkat *client* untuk dieksekusi.

Pada JADE *agent client*, pengiriman dan penerimaan pesan dilakukan dengan menggunakan kelas *ACLMessage*. Paradigma komunikasi yang diadopsi adalah pengiriman pesan secara *asynchronous*. Setiap *agent* memiliki *queue* untuk pesan yang digunakan jika JADE *runtime* menampilkan pesan yang dikirimkan oleh *agent* lain. Setiap kali ada pesan yang dikirim didalam *queue*, *agent* penerima akan diberitahu dan mengambil pesan tersebut. Seperti dapat dilihat pada Gambar 3.8 di bawah ini



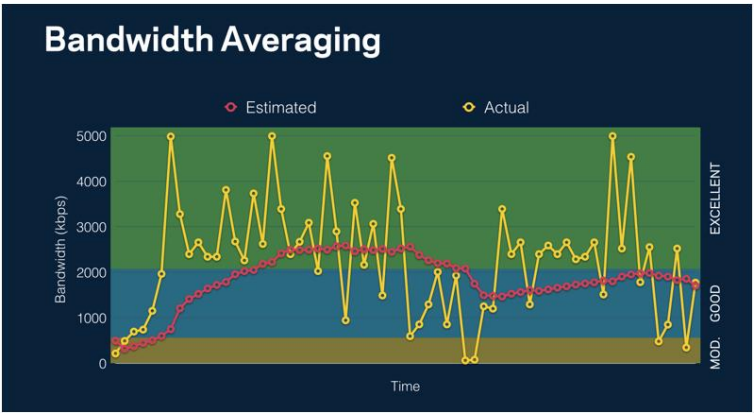
Gambar 3.8 Paradigma pengiriman pesan JADE secara *asynchronous* [12]

Pengiriman pesan ke *agent* lain dilakukan dengan mengisi *field* pada objek *ACLMessage* dan memanggil metode *send* dari kelas *agent*. Pengisian *field* pada *ACLMessage* untuk *container* lain yang berisi informasi *agent* untuk tujuan pengiriman pesan antara lain pengisian nama *agent*, nama *platform* dan alamat *host* menggunakan *AID*. Begitu pun sebaliknya pada JADE agent *server*, pesan yang diterima akan diproses menggunakan metode *receive* kelas *agent* dan akan dibalas dengan pesan yang berisi hasil pemrosesan menggunakan *ACLMessage* dengan pengisian *field* sama seperti sebelumnya. Pengaturan *AID* dilakukan dengan menggunakan informasi dari *container* yang melakukan pengiriman sebelumnya.

3.4 Faktor Penentu Metode *Offloading*

3.4.1 Kualitas Koneksi Internet

Data yang pertama adalah kecepatan *bandwidth* internet yang terbagi menjadi 4 kelas urut dari kualitas koneksi yang baik ke buruk yaitu *Excellent*, *Good*, *Moderate*, dan *Poor* serta jika koneksi internet tidak tersedia pada perangkat *client* maka dimasukkan dalam kelas *Unknown*. Data ini diperoleh dari fungsi eksekusi mengunduh data gambar kualitas rendah yang terpasang pada perangkat lunak sebagai penguji kualitas koneksi internet yang dimiliki oleh perangkat bergerak saat itu. Pengujian dilakukan dengan mengecek *buffer* data aktual yang terjadi saat mengunduh data gambar dengan estimasi *buffer* data berdasarkan besarnya ukuran data yang di unduh. Untuk lebih jelasnya dapat dilihat pada Gambar 3.9 Informasi estimasi *buffer* data diperoleh dari kode sumber *connection class facebook network*.



Gambar 3.9 Contoh pengelompokan kualitas koneksi berdasarkan estimasi *buffer* data dan *buffer* data actual [14]

3.4.2 Kondisi Level Baterai Perangkat Bergerak

Data yang kedua adalah kondisi level baterai perangkat bergerak yang berupa nilai *range* mulai dari 0 hingga 100. Nilai ini diperoleh dengan mengambil nilai level baterai perangkat bergerak secara *real time* menggunakan kode sumber IDE Android Studio.

3.4.3 Waktu Eksekusi Proses *Image Recognition*

Data yang ketiga adalah waktu eksekusi proses *image recognition* yang diperoleh dari menghitung waktu dimulainya proses *image recognition* menggunakan metode SURF hingga diperolehnya data jarak kedekatan citra daun datatest dengan masing – masing citra daun dataset. Perhitungan waktu eksekusi dilakukan baik pada eksekusi secara lokal maupun eksekusi secara *offloading*.

3.5 *Offloading Framework*

Offloading framework merupakan salah satu metode untuk penghematan daya pada perangkat bergerak dengan cara melakukan *offload* beban kerja dari perangkat bergerak ke *server*. Metode ini sangat mendukung untuk diimplementasikan sebab konektivitas internet yang bertindak sebagai *virtual bus* dapat diperoleh atau didapatkan dimana saja dan kapan saja saat ini. Studi kasus pada tugas akhir ini, *server* tidak dibatasi oleh sumber daya baterai dan memiliki *processor* dengan performa jauh lebih cepat dibandingkan dengan perangkat bergerak *client* sehingga apabila melakukan *offloading* beban kerja ke server, diharapkan eksekusi operasi terhadap beban kerja lebih cepat dan meminimalisir waktu eksekusi.

Tidak semua beban kerja pada perangkat bergerak *client* dapat di *offloading*, misalnya pada layanan *background* data koordinat lokasi menggunakan GPS yang sensitif berubah secara berkelanjutan sehingga pengiriman data secara *offload* dilakukan secara terus menerus. Hal ini menyebabkan penggunaan sumber daya baterai lebih boros dan tidak sesuai dengan konsep dasar metode *offloading*. Pada tugas akhir ini, beban kerja yang akan di *offload* ke *server* sudah ditentukan yaitu *image recognition* antara datatest dan dataset citra daun menggunakan metode SURF.

Dalam memulai eksekusi *offload* beban kerja pada *server*, ada beberapa hal yang harus dilakukan sebelumnya agar hasil dari eksekusi *offload* beban kerja ke *server* maksimal dan sesuai yang diharapkan. Identifikasi beban kerja, kondisi perangkat bergerak *client* dan pemantauan kualitas jaringan internet sangat penting dalam *framework* ini. Identifikasi beban kerja seperti perhitungan waktu eksekusi baik dilakukan secara lokal maupun *offloading*, kondisi perangkat bergerak *client* seperti kondisi level baterai, serta pemantauan kualitas jaringan internet yang dilakukan secara dinamis dan berkala setiap adanya beban kerja yang akan di eksekusi diperlukan dalam *framework*. *Framework* yang melakukan eksekusi beban kerja secara dinamis dan berkala

diharapkan dapat memberikan keputusan metode eksekusi yang tepat dan optimal baik secara lokal maupun *offloading* terhadap hal – hal yang dapat mempengaruhi pengambilan keputusan metode eksekusi yang contohnya telah disebutkan sebelumnya. Selanjutnya, hal – hal yang dapat mempengaruhi pengambilan keputusan metode eksekusi kita sebut sebagai faktor – faktor penentu metode *offloading* serta fungsi dalam *framework* yang digunakan untuk menentukan keputusan metode eksekusi kita sebut sebagai *decision maker*.

Decision maker adalah fungsi yang dipanggil setiap beban kerja yang telah ditentukan pada perangkat bergerak akan dieksekusi. Data faktor – faktor penentu metode *offloading* akan digunakan sebagai catatan histori performa oleh *decision maker* untuk memperkirakan dan membandingkan biaya dari kedua metode eksekusi tersebut sebelum memutuskan metode eksekusi yang akan dipilih dan memberikan keuntungan yang lebih besar dalam penghematan sumber daya dan waktu eksekusi. Pada perancangan tugas akhir ini, *decision maker* melakukan pengecekan data terhadap kualitas koneksi internet perangkat *client*, kondisi level baterai perangkat *client* dan rata – rata waktu eksekusi masing – masing metode pengeksekusian beban kerja *image recognition*. Untuk eksekusi pertama, akan dilakukan eksekusi secara lokal untuk mengetahui waktu eksekusi secara lokal yang digunakan sebagai pembanding dengan metode offloading selanjutnya. Jika dari data – data tersebut memenuhi untuk dilakukannya metode *offloading*, maka dilakukanlah eksekusi secara *offloading* serta hasil waktu eksekusinya akan dibandingkan dengan metode secara lokal. Dalam menentukan metode eksekusi selanjutnya setelah pengeksekusian beban kerja menggunakan waktu eksekusi digunakan Persamaan 3.1 dan Persamaan 3.2 [15].

$$\sum_{i \in I} (1 - I_i) \times T_i^l + I_i \times T_i^r + I_i \times T_i^t \leq l \qquad (3.1)$$

$$T_i^r + T_i^t - T_i^l \leq l \quad (3.2)$$

Penjelasan pada persamaan di atas, I mewakili rangkaian beban kerja yang dilakukan dalam perangkat lunak. Untuk setiap beban kerja $i \in I$ yang dapat di eksekusi secara *offloading*, T_i^l adalah waktu eksekusi dari *i* pada *client*. T_i^r adalah waktu eksekusi dari *i* pada *server*. T_i^t adalah waktu pengiriman dari *i*. I_i adalah variabel indikator: I_i = 0 jika *i* dieksekusi secara lokal, I_i = 1 jika *i* dieksekusi secara *remote* pada *server* atau *offloading*, *l* didasarkan pada kebutuhan waktu yang digunakan oleh pengembang.

Untuk studi kasus menentukan metode eksekusi, akan dilakukan eksekusi secara *offloading* jika selisih waktu eksekusi pada perangkat *server* terhadap waktu eksekusi pada perangkat *client* kurang dari sama dengan waktu yang telah ditentukan.

Penerapan menentukan keputusan sebagian besar didasarkan pada Persamaan 3.2 dengan beberapa modifikasi seperti penggunaan waktu rata – rata eksekusi metode *offloading* ($T_i^r + T_i^t$) dan lokal (T_i^l) serta *l* diinisialisasi dengan nilai 8 detik. Jika memenuhi lagi untuk metode *offloading* maka dilakukan metode *offloading* begitu seterusnya sampai semua beban kerja pada proses *image recognition* telah selesai dieksekusi.

3.6 Metode SURF

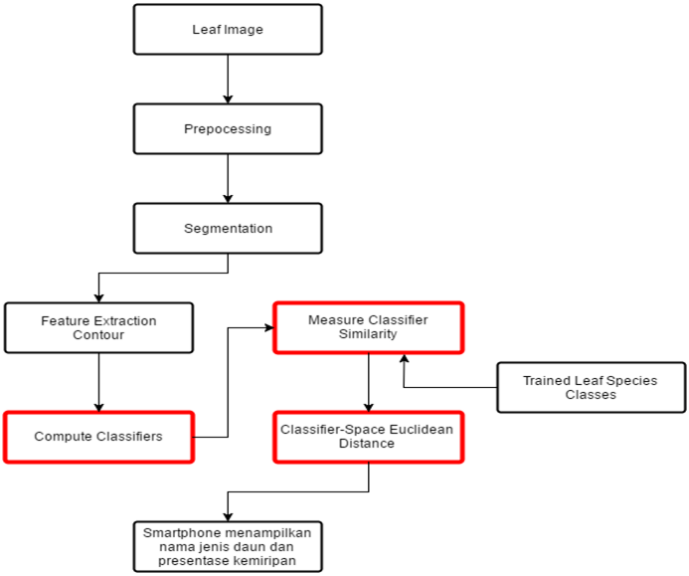
SURF singkatan dari *Speed Up Robust Features* merupakan salah satu metode yang digunakan untuk *feature extraction* pada proses *image recognition*. Pada *image recognition* sendiri terdapat empat proses dalam pengolahan citra diantaranya *preprocessing*, *segmentation*, *feature extraction* dan *classification*.

Tahap pertama, *preprocessing* adalah metode untuk memaksimalkan citra yang didapat untuk proses *image recognition*. Pada tugas akhir ini, tahap *preprocessing* dilakukan dengan mengkompresi citra (datatest) yang didapat oleh kamera perangkat bergerak (*client*) agar sesuai dengan ukuran dataset dan konversi citra ke pewarnaan *grayscale*.

Tahap kedua, *Segmentation* adalah metode untuk identifikasi *image recognition* dari gambar dengan cara mengenali ciri – ciri unik gambar sehingga dapat dibedakan antara yang satu dengan yang lainnya. Pada tugas akhir ini, tahap *segmentation* dilakukan dengan menemukan titik – titik penting fitur citra dengan menggunakan *feature detector* metode SURF yang disediakan *library* OpenCV.

Tahap ketiga, *feature extraction* adalah metode untuk menghitung ciri – ciri citra yang diperoleh dari tahap *segmentation* ke dalam bentuk tertentu. Ciri – ciri citra yang di dapat umumnya disebut *descriptor*. *Descriptor* memiliki beberapa bentuk yang digunakan sebagai ciri – ciri pada citra seperti garis luar/*outline* objek pada citra, garis kerangka pada citra, serta umumnya titik – titik ujung maupun sudut – sudut pada citra. Pada tugas akhir ini, tahap *feature extraction* dilakukan dengan menghitung *descriptor* berupa titik – titik yang digunakan sebagai fitur baik pada citra datatest maupun citra dataset. *Descriptor extractor* metode SURF yang disediakan *library* OpenCV digunakan sebagai penghitung nilai *descriptor* fitur kedua citra tersebut .

Tahap akhir, *classification* adalah metode untuk mengklasifikasi gambar dari hasil *feature extraction* berdasarkan kelas yang telah didefinisikan. Kelas ini didapatkan dari pengklasifikasian data *training* / dataset yang sudah dilakukan terlebih dahulu. Pada tugas akhir ini, tahap *classification* dilakukan dengan menghitung jarak kedekatan dan kemiripan nilai *descriptor extractor* antara datatest dan dataset dengan *descriptor matcher*. *Descriptor matcher* dilakukan dengan metode FLANNBASED yang disediakan metode OpenCV dengan ketentuan semakin dekat dan mirip nilai kedua *descriptor extractor* maka nilai *descriptor matcher* semakin mendekati nol. Pada tugas akhir ini, nilai jarak kedekatan *descriptor* yang memenuhi kriteria bernilai kurang dari 0.015. Urutan langkah pada proses *image recognition* dapat terlihat pada Gambar 3.10.



Gambar 3.10 Proses *image recognition* pada citra daun

3.7 Java *Class Object Serialization* dari *Client* ke *Server*

Java *Class Object Serialization Client* ke *Server* adalah sebuah kelas java yang terdiri data – data yang akan dikirimkan dari *client* ke *server* dan di serialiasasi menjadi *object* java. Serialisasi dari data java menjadi *object* Java dapat dilakukan menggunakan *library* Commons – Lang yang berfungsi menambah fungsionalitas pada inti Java. Pada tugas akhir ini, serialisasi data menjadi *object* java digunakan sebagai modul pengiriman data dari *client* ke *server*. Data – data tersebut terdiri dari data *string descriptor* citra daun datatest yang dikonversi dari tipe data matriks OpenCV menggunakan GSON, data *string* nama file citra daun dataset yang di akan dilakukan *image recogniton*, data *string* waktu

agent JADE memulai modul pengiriman data dari *client* ke *server* yang terdiri dari keterangan jam, menit, detik dan milidetik. Data – data *string* yang dikirimkan dalam modul data akan diterima oleh *server* dan dikonversikan kembali menjadi tipe data dalam java seperti semula. Misalnya, data *string descriptor* citra daun datatest akan dikonversikan kembali menjadi tipe data matriks OpenCV pada *server* untuk pemrosesan *image recognition* citra daun.

3.8 Java *Class Object Serialization* dari *Server* ke *Client*

Java *Class Object Serialization Server* ke *Client* adalah sebuah kelas java yang terdiri dari data – data yang akan dikirimkan dari *server* ke *client* dan di serialisasi menjadi *object* java seperti yang telah dijelaskan sebelumnya. Perbedaan dengan kelas serialisasi objek sebelumnya yaitu data – data pada kelas serialisasi objek dari *server* ke *client* berisi data – data hasil *image recognition* citra daun menggunakan metode SURF yang akan ditampilkan pada perangkat bergerak (*client*). Data – data tersebut terdiri dari data *string* nama file citra daun dataset yang telah dilakukan *image recognition*, data *integer* jumlah titik – titik yang jarak kemiripannya diterima / valid sesuai dengan batas jarak minimal yang telah ditentukan, data *double* jarak kedekatan minimal yang dapat diperoleh antara citra daun datatest dan dataset, data *double* jarak kedekatan maksimal yang dapat diperoleh antara citra daun datatest dan dataset, data *double* waktu yang dibutuhkan *agent* dalam pengiriman modul data, data *string* waktu agent JADE memulai modul pengiriman data dari *server* ke *client* yang terdiri dari keterangan jam, menit, detik dan milidetik dan data *double* waktu yang dibutuhkan *agent* dalam penerimaan modul data dari server.

BAB IV
IMPLEMENTASI

Bab ini berisi penjelasan mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa *Pseudocode* untuk membangun program.

4.1 Lingkungan Implementasi

Implementasi penerapan *offloading computation mobile framework* pada sistem perangkat lunak *image recognition* menggunakan metode SURF menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

Privilege	Perangkat	Jenis	Spesifikasi
Client	Perangkat Keras	Prosesor	Intel(R) Atom(TM) CPU Z2580 2,00 GHz
		Memori	2 GB DDR3
	Perangkat Lunak	Sistem Operasi	Android 4.4.2 KitKat
Server	Perangkat Keras	Prosesor	Intel(R) Core(TM) i7-2.5 GHz
		Memori	8 GB 800 MHz DDR3
	Perangkat Lunak	Sistem Operasi	Windows 10 dan Ubuntu 16.04

		Perangkat Pengembang	Android Studio IDE dan Netbeans IDE 8.2

4.2 Implementasi

Pada sub bab implementasi ini menjelaskan mengenai pembangunan perangkat lunak secara detail dan menampilkan *Pseudocode* yang digunakan mulai tahap *preprocessing* hingga *feature extraction* antara datatest dan dataset yang digunakan dalam penerapan *image recognition* dengan metode SURF, selain itu, terdapat mekanisme eksekusi menggunakan *offloading framework* untuk penghematan daya dan peningkatan kinerja perangakat *client*. Pada tugas akhir ini data yang digunakan, seperti yang telah dijelaskan pada bab sebelumnya, yaitu terdiri dari dua jenis data masukan. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pemfotoan saat menjalankan perangkat lunak Android. Data masukan jenis pertama yang belum diketahui jenisnya tersebut digunakan sebagai datatest untuk uji coba prediksi pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur. Sedangkan data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun. Data masukan jenis kedua yang berjumlah 40 tersebut digunakan sebagai data masukan untuk uji coba pencocokan dengan datatest pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

4.2.1 JADE *Middleware*

4.2.1.1 Client

Hal pertama yang dilakukan pada JADE *middleware* adalah mengaktifkan JADE *runtime* dari Android *Activity* untuk menghubungkannya dengan *MicroRuntimeService* melalui *Pseudocode* 4.1. Hal ini bertujuan untuk memungkinkan perangkat bergerak *client* menjalankan operasi manajemen JADE. Terdapat variabel – variabel yang harus diinisialisasi untuk terhubung dengan *MicroRuntimeService*. Variabel – variabel tersebut digunakan sebagai parameter dalam menghubungkan perangkat lunak dengan *MicroRuntimeService*, variabel tersebut diantaranya *host* dan *port* yang diletakkan dalam tipe data `properties` yang digunakan untuk mereferensi alamat *platform* serta `agentController` yang digunakan untuk pembuatan *container* baru. Sebelum menghubungkan *MicroRuntimeService*, dilakukan pengecekan dahulu terhadap *MicroRuntimeService* apakah sudah terhubung atau belum. Jika belum terhubung, maka *MicroRuntimeService* dihubungkan dan dilanjutkan pada tahap selanjutnya.

```
1   function bindService(String host, String port,
       RuntimeCallback<AgentController>
       agentStartupCallback)
2          Properties pp := new Properties()
3          pp.setProperty(Profile.MAIN_HOST, host)
4          pp.setProperty(Profile.MAIN_PORT, port)
5          pp.setProperty(Profile.JVM,
              Profile.ANDROID)
6          if (microRuntimeServiceBinder == NIL)
7              mServiceConnection := new
                  ServiceConnection()

8              @Override
9              function
10               onServiceConnected(ComponentName
                   componentName, IBinder service)
```

```
11                      microRuntimeServiceBinder :=
                          (MicroRuntimeServiceBinder)
                            service
12                      Log.i(TAG,"###Gateway
                            successfully bound to
                            RuntimeService")
13                      startMainContainer(pp,
                            agentStartupCallback)
14                  @Override
15              function
                 onServiceDisconnected(ComponentName
                    componentName)
16                      Log.i(TAG,"###Gateway unbound
                            from RuntimeService")
17              Log.i(TAG,"###Binding Gateway to
                   RuntimeService...")
18              bindService(new
                  Intent(getApplicationContext(),
                  MicroRuntimeService.class),
                  mServiceConnection,
                  Context.BIND_AUTO_CREATE)
19          else
20              startMainContainer(pp,
                  agentStartupCallback)
```

Pseudocode* 4.1 Kode program menghubungkan JADE *runtime* dengan *MicroRuntimeService

Tahap selanjutnya adalah membuat *container* dari *MicroRuntimeService* melalui *Pseudocode* 4.2. Nilai parameter yang dibutuhkan dalam membuat *container* adalah nilai – nilai yang disisipkan dalam `properties` dan `agentController` seperti yang dijelaskan pada fungsi `bindService` (dapat dilihat pada *Pseudocode* 4.1). Nilai *host* pada `properties` berisi alamat *http* dari *server*. Sehingga *client* tidak membuat *main container* sendiri pada perangkat bergerak yang dimilikinya tetapi pada perangkat *server*. Penerapan metode container seperti ini disebut *Split container*.

```
1   function startMainContainer(Properties profile,
       RuntimeCallback<AgentController>
       agentStartupCallback)
2            microRuntimeServiceBinder.
                startAgentContainer(profile,new
                RuntimeCallback<Void>()
3                     @Override
4                     function onSuccess(Void
                        thisIsNull)
5                           Log.i(TAG, "###Agent-
                               Container
6                              created...")
7                           bindservice := true
8                     @Override
9                     function onFailure(Throwable
                        throwable)
10                          Log.i(TAG, "###Failed to
                               create
                               Main Container")
```

Pseudocode* 4.2 Kode Program membuat *Container

Agent dapat dibuat setelah *container* pada *platform server* telah berjalan. Pembuatan *agent* dapat dilakukan melalui *Pseudocode* 4.3. Terdapat nilai – nilai parameter yang dibutuhkan dalam pembuatan *agent* seperti nama *agent*, nama kelas java yang mengimplementasikan fungsional *agent*, dan *agentController*. Selain itu, terdapat data – data yang diinisialisasi ke dalam beberapa `Object args` yang akan disisipkan dalam pembuatan *agent* yaitu *context* dari kelas java tempat pembuatan *agent*, *descriptor* dari datatest citra daun yang telah di konversi menjadi *string*, nama file dataset citra daun yang akan dilakukan *image recognition*, dan waktu dimulainya eksekusi pembuatan *agent client* yang disamakan dengan waktu pengiriman pesan ke *agent server*. Selain itu, disediakan *error handling* seperti pengecekan keberadaan *MicroRuntimeService* pada baris 2, pengecekan keberhasilan eksekusi pembuatan *agent* baru pada baris 16.

```
1   function createAgent(String name, String
      className,RuntimeCallback<AgentController>
      agentStartupCallback)
```

```
2           if (microRuntimeServiceBinder != NIL)
3             TimeZone local :=
              TimeZone.getTimeZone("Asia/Jakarta")
4             Calendar now :=
              Calendar.getInstance(local)
5             SimpleDateFormat formatter := new
              SimpleDateFormat("HH:mm:ss.SSS")
6             formatter.setTimeZone(local)
7             String dateString :=
              formatter.format(now.getTime())
8             Log.i(TAG, "Time Send Start : " +
              dateString)
9             Object[] args := Object[0..3]
10            args[0] := SURFExampleActivity.this
11            args[1] := JSONdescCamera
12            args[2] := ImageName
13            args[3] := dateString
14            microRuntimeServiceBinder.startAgent
               (name, className, args, new
              RuntimeCallback<Void>()

15                  @Override
16                  function onSuccess(Void
                      thisIsNull)
17                    Log.i(TAG,"###Success to
                        create agent")
18                    try
19                       agentStartupCallback.
                          onSuccess(MicroRuntime.
                          getAgent(name))
20                    catch(ControllerException
                        e)
21                       agentStartupCallback.
                          onFailure(e)
22                  @Override
23                  function onFailure(Throwable
                      throwable)
24                    Log.i(TAG, "###Failed to
                         created an Agent")

25                         agentStartupCallback.
                             onFailure(throwable)
26            )
27          else
```

```
28          Log.e(TAG, "###Can't get Main-
              Container to create agent")
```

Pseudocode* 4.3 Kode Program membuat *Agent

Pada kelas *agent*, terdapat variabel – variabel yang di inisialisasi yang dapat dilihat melalui *Pseudocode* 4.4. Variabel `StatusTask` dan `agentCreated` digunakan untuk proses pada *offloading framework* (Bab 4.2.2). Variabel `desc` dan `ImageName` digunakan untuk proses *image recognition* serta variabel `SendAgentStartTime`, `ReceiveAgentStartTime` dan `receiveAgentTime` digunakan untuk menghitung waktu eksekusi.

```
1   class SendObjectAgent inherits Agent implements
    SimpleAgentInterface
2       String TAG := "SendObjectAgent"
3       StatusTask := false
4       agentCreated := false
5       Context context
6       String ipAddress
7       String agentName
8       String desc
9       String ImageName
10      String SendAgentStartTime
11      String ReceiveAgentStartTime
12      receiveAgentTime
```

Pseudocode* 4.4 Kode program inisialisasi variabel *agent* pada *client

Pada fungsi `setup()` pada kelas *agent* digunakan untuk memulai operasi – operasi yang dilakukan oleh *agent* seperti yang telah dijelaskan pada Bab 3.3. Pengaturan `setup()` dapat dilihat pada *Pseudocode* 4.5. Pertama, kita ambil dahulu nilai parameter dari pembuatan *agent* pada *Pseudocode* 4.3 sebelumnya. Nilai parameter kita tampung pada variabel `context`, `desc`, `ImageName` dan `SendAgentStartTime`. Selanjutnya, kita tambahkan *behaviour* untuk menjalankan operasi. Pada fungsi `setup()`, untuk mengetahui *host* dan *port* server digunakan `SharedPreferences` yang akan dijelaskan pada Bab 4.2.2.

```
1   function setup()
2           Object[] args := getArguments()
3           if (args != NIL AND args.length > 0)
4               if (args[0] instanceof Context)
5                   context := (Context) args[0]
6               if (args[1] instanceof String)
7                   desc := (String) args[1]
8               if (args[2] instanceof String)
9                   ImageName := (String) args[2]
10              if (args[3] instanceof String)
11                  SendAgentStartTime := (String)
                      args[3]
12          addBehaviour(new ReceiveMessage())

13          registerO2Ainterface
               (SimpleAgentInterface.class, this)
14          Intent broadcast := new Intent()
15          broadcast.setAction
               ("jade.demo.agent.SEND_MESSAGE")
16          Log.i(TAG, "###Sending broadcast " +
              broadcast.getAction())
17          context.sendBroadcast(broadcast)
18          SharedPreferences sharedPreferences :=
               context.getSharedPreferences
             (Constants.PREFS_FILE_NAME,
               Context.MODE_PRIVATE)
19          ipAddress := sharedPreferences.getString
              (Constants.PREFS_HOST_ADDRESS,
              ipAddress)
20          agentName := sharedPreferences.getString
              (Constants.PREFS_AGENT_NAME,
               agentName)

21  function onHostChanged(String host)
22          ipAddress := host

23  function onAgentNameChanged(String name)
24          agentName := name
```

Pseudocode* 4.5 Kode program metode *Setup* pada *client

Behaviour yang digunakan *agent* pada client adalah *generic behaviour* melalui *Pseudocode* 4.6, untuk penjelasannya

bisa dilihat pada Bab 3.3. Pada *behaviour*, terdapat operasi yang dapat di pisah menjadi dua sub operasi. Sub operasi yang pertama yaitu mengirim pesan menuju *server*. Pada sub operasi ini, langkah awal adalah membuat *AID* beserta pengaturan nama dan alamat *agent* pada *server*. Langkah selanjutnya menyisipkan variabel `desc`, `ImageName` dan `SendAgentStartTime` pada kelas serialisasi *ObjectDataMat* yang akan dijelaskan pada Bab 4.2.4. Langkah akhir yaitu mengirim pesan kepada server. Jika sub operasi satu selesai, sub operasi yang kedua langsung dijalankan. Sub operasi yang kedua adalah menerima pesan yang berisi hasil operasi beban kerja dari *server*. Langkah awal adalah menunggu adanya pesan diterima dari *server*. Jika pesan sudah diterima, maka langkah selanjutnya pesan akan ditampung menggunakan kelas serialisasi *data* yang lebih jelasnya akan dijelaskan pada Bab 4.2.5 beserta pencatat waktu penerimaan pesan. Langkah akhir adalah mengirim kembali hasil operasi dari pesan yang telah diterima ke kelas java pembuat *agent* melalui fungsi `exportLog`. Jika tugas *behaviour* selesai, maka dilanjutkan memanggil fungsi `doDelete()` untuk menghapus *behaviour* dari *agent*. Jika salah satu *behaviour* dari *agent* dihapus maka fungsi `done()` otomatis dipanggil.

Pseudocode* 4.6 Kode program generic *behaviour* pada *client

```
1   class ReceiveMessage inherits Behaviour
2      step := 0

3      @Override
4      function action()
5          switch (step) :
6             case 0:
7                 try
8                    agentCreated := true
                     ACLMessage message := new
                       ACLMessage(ACL
                       Message.INFORM)
9                    AID dummyAid := new AID()
10                   dummyAid.setName
                       (agentName + "@" +
                       ipAddress +
                       ":1099/JADE")
```

```
11                      dummyAid.addAddresses
                          ("http://" + ipAddress
                          + ":7778/acc")

12                      message.addReceiver(dummyAid)
13                      String convId := "C-" +
                          myAgent.getLocalName()
14                      ObjectDataMat p := new
                          ObjectDataMat(desc, ImageName,
                          SendAgentStartTime)

15                      message.setContentObject(p)
16                      message.setLanguage
                          ("JavaSerialization")
17                      step := 1
18                      myAgent.send(message)
19                      Log.i(TAG, "###Send
                          message:" + message.
                          getContent())
20                  catch (IOException e)
21                      e.printStackTrace()

22                  break
23              case 1:
24                  agentCreated := true
25                  ACLMessage messageserver :=
                      myAgent.receive()
26                  if (messageserver != NIL)
27                      if ("JavaSerialization".equals
                         (messageserver.getLanguage()))
28                          try
29                              data d := (data)
                                  messageserver.
                                  getContentObject()
30                              TimeZone local :=
                                  TimeZone.getTime
                                  Zone("Asia/Jakarta")
31                              Calendar now :=
                                  Calendar.get
                                  Instance(local)
32                              SimpleDateFormat
                                  formatter := new
                                  SimpleDateFormat
                                  ("HH:mm:ss.SSS")
```

```
33                          formatter.set
                              TimeZone(local)

34                          String ReceiveAgent
                              FinishTime := formatter.
                              format(now.getTime())
35                          String[] time_receive2 :=
                              ReceiveAgentFinishTime.
                              split(":")
36                          hour_receive2 :=
                              Integer.parseInt
                              (time_receive2[0])
37                          minute_receive2 :=
                              Integer.parseInt
                              (time_receive2[1])
38                          seconds_receive2 :=
                              Double.parseDouble
                              (time_receive2[2])
39                          ReceiveAgentStart
                              Time := d.getReceive
40                            AgentStartTime()
                            String[] time_receive1 :=
                              ReceiveAgentStartTime.
                              split(":")
41                          hour_receive1 :=
                              Integer.parseInt
                              (time_receive1[0])
42                           minute_receive1 :=
                               Integer.parseInt
                               (time_receive1[1])
43                           seconds_receive1 :=
                               Double.parseDouble
                               (time_receive1[2])

44                           Log.i("SURFExample
                               Activity", "Time
                               Receive Start : "
                               + ReceiveAgent
                               StartTime)

45                           Log.i("SURFExample
                               Activity", "Time
                               Receive Finish :
```

```
                                " + ReceiveAgent
                                FinishTime)

46                            hour_receive :=
                                hour_receive2 -
                                hour_receive1
47                            minute_receive :=
                                minute_receive2 -
                                minute_receive1
48                            seconds_receive :=
                                seconds_receive2 -
                                seconds_receive1
49                            receiveAgentTime:= 0
50                               if (hour_receive
                                   != 0)
51                                  receiveAgentTime
                                      := receiveAgent
                                      Time + (hour_
                                      receive * 3600)
52                               if (minute_receive
                                  != 0)

53                                  receiveAgentTime
                                     := receiveAgent
                                     Time + (minute_
                                     receive * 60)
54                               if (seconds_receive
                                   != 0)

55                                   receiveAgentTime
                                     := receiveAgent
                                     Time + seconds_
                                     receive

56                                  d.setReceiveAgent
                                     Time(receive
                                     AgentTime)
57                                  exportLog(d)
58                          catch (UnreadableException e)
59                              e.printStackTrace()

60                        else
61                             Log.d("result", "Failed")
62                        StatusTask := true
```

```
63                      doDelete()
64                  else
65                      block()


66      @Override
67      function done()
68          return false

69  function exportLog(data log)
70      SURFExampleActivity surfExampleActivity :=
          (SURFExampleActivity) context
71      surfExampleActivity.exportLogConsole(log)
```

4.2.1.2 Server

Pada *server*, kelas *agent* memiliki variabel – variabel yang diinisialisasi melalui *Pseudocode* 4.7. Variabel `Mat descriptor`, `MatOfDMatch matches`, `Bitmap inputImage`, `FeatureDetector detector`, `DescriptorExtractor descriptorExtractor` dan `DescriptorMatcher descriptorMatcher` digunakan untuk *image recognition*. Kelas *data* digunakan menampung hasil operasi dari *image recognition* serta `sendAgentTime` digunakan untuk perhitungan waktu eksekusi.

```
1   class TestObject inherits Agent
2      Mat descriptor
3      MatOfDMatch matches
4      Bitmap inputImage
5      FeatureDetector detector
6      DescriptorExtractor descriptorExtractor
7      DescriptorMatcher descriptorMatcher
8      String path := "/home/putrosw/training/"
9      data d
10     sendAgentTime
```

Pseudocode* 4.7 Kode program inisialisasi variabel *agent* pada *server

Fungsi `setup()` pada kelas *agent server* digunakan untuk menjalankan *behaviour* yag berhubungan dengan operasi *image recognition*. Pengaturan `setup()` pada *server* dapat dilihat pada *Pseudocode* 4.8. Pertama, kita panggil dahulu *native library* agar metode yang digunakan dalam *image recognition* menggunakan OpenCV dapat diterapkan. Selanjutnya, kita tambahkan *behaviour* untuk menjalankan operasi *image recognition*.

```
1   function setup()
2       System.loadLibrary
          (Core.NATIVE_LIBRARY_NAME)
3       writeln "Hallo! Receivermessage-agent " +
          getAID().getName() + " is ready."
4       addBehaviour(new TickerBehaviour(this, 1000)
```

Pseudocode* 4.8 Kode program metode *Setup* pada *server

Behaviour yang digunakan *agent* pada *server* adalah *ticker behaviour* bias dilihat melalui *Pseudocode* 4.9, untuk penjelasannya bias dilihat pada Bab 3.3. Pada *behaviour*, terdapat operasi yang digunakan untuk melakukan *image recognition*. Operasi pada *behaviour* diawali dengan pemilihan metode fitur *descriptor*, *descriptor extractor*, dan *descriptor matcher*. Pada tugas akhir ini, metode fitur *descriptor* sekaligus *descriptor extractor* adalah SURF, dan *descriptor matcher* yang digunakan adalah metode FLANNBASED. Selanjutnya, *behaviour* akan menunggu sampai adanya pesan masuk dari *agent client*. Jika pesan masuk, data dalam pesan tersebut akan ditampung dalam kelas *ObjectDataMat* disertai dengan pencatatan waktu sebagai penanda jika pengiriman *agent* telah selesai dilakukan. *Descriptor* datatest yang bertipe data *string* diubah menjadi tipe data *Mat* menggunakan fungsi `matFromJson` agar dapat diproses dalam *image recognition*, untuk lebih jelasnya bisa dilihat pada Bab 4. Setelah *descriptor* datatest berubah menjadi tipe data *Mat*, maka fungsi `surf()` dipanggil untuk menjalankan *image recognition*, untuk lebih jelasnya bisa dilihat pada Bab 4.2.3. Setelah *image recognition* selesai dilakukan, data yang dihasilkan akan

dikirimkan kembali ke *agent client*. *AID* baru dibuat dengan pengaturan nama dan alamat *agent* disesuaikan dengan *agent client* pengirim pesan. Langkah terakhir adalah mengirimkan kembali pesan yang berisi data hasil *image recognition* ke *agent client* pengirim pesan serta menyisipkan waktu yang dicatat saat *server* mengirimkan pesan ke *agent client*.

```
1   function onTick()
2      try
3         Mat rgba := new Mat()
4         detector := FeatureDetector.create
           (FeatureDetector.SURF)
5         descriptorExtractor := DescriptorExtractor.
           create(DescriptorExtractor.SURF)
6         descriptorMatcher := DescriptorMatcher.
           create(DescriptorMatcher.FLANNBASED)
7         writeln "Trying receive object"
8         ACLMessage msg := blockingReceive()

9         if("JavaSerialization".equals
           (msg.getLanguage()))
10           ObjectDataMat p := (ObjectDataMat)
              msg.getContentObject()
11           writeln getLocalName() + " read Java
              Object " + p.getClass().getName()
12           TimeZone local :=
              TimeZone.getTimeZone("Asia/Jakarta")
13           Calendar now :=
              Calendar.getInstance(local)
14           SimpleDateFormat formatter := new
              SimpleDateFormat("HH:mm:ss.SSS")
15           formatter.setTimeZone(local)
16           String SendAgentFinishTime :=
              formatter.format(now.getTime())
17           writeln "Send Agent Finish Time : " +
              SendAgentFinishTime
18           String[] time_send2 :=
              SendAgentFinishTime.split(":")
19           hour_send2 :=
              Integer.parseInt(time_send2[0])
20           minute_send2 :=
              Integer.parseInt(time_send2[1])
21           seconds_send2 :=
```

```
          Double.parseDouble(time_send2[2])
22        String SendAgentStartTime :=
           p.getSendAgentStartTime()
23        String[] time_send1 :=
           SendAgentStartTime.split(":")
24        hour_send1 :=
           Integer.parseInt(time_send1[0])
25        minute_send1 :=
           Integer.parseInt(time_send1[1])
26        seconds_send1 :=
           Double.parseDouble(time_send1[2])
27        hour_send := hour_send2 - hour_send1
           minute_send := minute_send2 -
           minute_send1
28        seconds_send := seconds_send2 -
           seconds_send1
29        sendAgentTime := 0
30        if (hour_send != 0)
31          sendAgentTime := sendAgentTime +
             (hour_send * 3600)
32        if (minute_send != 0)
33          sendAgentTime := sendAgentTime +
             (minute_send * 60)
34        if (seconds_send != 0)
35          sendAgentTime := sendAgentTime +
             seconds_send
36        writeln "sendAgentTime : " +
           sendAgentTime + " seconds"
37        descriptor :=
           matFromJson(p.getDescriptor())
38        writeln descriptor
39        writeln p.getImageName()
40        surf(p.getImageName())
41        String SenderAgent :=
           msg.getSender().getName()
42        String[] parts := SenderAgent.split("@")
43        String part1 := parts[0]
44        String part2 := parts[1]
45        String[] parts2 := part2.split(":")
46        String IpAddress := parts2[0]
47        writeln "IpAddress : " + IpAddress
48        ACLMessage reply := new
           ACLMessage(ACLMessage.CONFIRM)
```

```
49          reply.setLanguage("JavaSerialization")
50          writeln msg.getSender()
51          writeln "Name : " +
             msg.getSender().getName() + " Local
             Name : " +
             msg.getSender().getLocalName() + "
             Address " +
             msg.getSender().getAddressesArray()[0]

52           reply.addReceiver(msg.getSender())
53           try
54              reply.setContentObject(d)
55           catch (IOException ex)
56              Logger.getLogger(TestObject.
                 class.getName()).log(Level.SEVERE,
                 NIL, ex)
57              myAgent.send(reply)
58       else
59          writeln getLocalName() + " read Java
             String " + msg.getContent()
60          String[] parts :=
             msg.getContent().split("/")
61          String part1 := parts[0]
62          String part2 := parts[1]
63          writeln part1 + " " + part2
64    catch (UnreadableException e3)
65      System.err.println(getLocalName() + "
         catched exception " + e3.getMessage())
66  )
```

Pseudocode* 4.9 Kode program *Ticker behaviour* pada *server

4.2.2 *Offloading Framework*

Hal pertama yang dilakukan pada *framework* yang melakukan penerepan metode *offloading* adalah menginisialisasi tampilan layar perangkat lunak menjadi tampilan mode kamera untuk mengambil datatest melalui *Pseudocode* 4.10. Pemanggilan fungsi untuk pengecekan kondisi koneksi internet juga dilakukan pada fungsi `onCreate`. Gambar citra datatest akan diperoleh dengan menekan tombol yang telah diatur pada fungsi `onKeyDown`. Gambar citra yang diperoleh dilakukan kompresi agar sesuai

dengan citra dataset saat dilakukan *image recognition*. Selanjutnya akan dimulai eksekusi *image recognition* secara *asynchronous*.

```
1   @Override
2   function onCreate(Bundle savedInstanceState)
3       super.onCreate(savedInstanceState)
4       getWindow().setFlags(WindowManager.
         LayoutParams.FLAG_FULLSCREEN,
         WindowManager.LayoutParams.
         FLAG_FULLSCREEN)

5       requestWindowFeature(Window.
         FEATURE_NO_TITLE)
6       mResultView := new ResultView(this)
7       mPreview := new Preview(this)

8       setContentView(mPreview)
9       addContentView(mResultView,new LayoutParams
         (LayoutParams.WRAP_CONTENT,
         LayoutParams.WRAP_CONTENT))
10      checkConnection()
11      new DownloadImage().execute(mURL)

12  @Override
13  function onKeyDown(keycode, KeyEvent event)
14      if (keycode == KeyEvent.KEYCODE_VOLUME_UP)
15          if (mResultView.IsShowingResult)
16              mResultView.IsShowingResult :=
                 false
17          else if (mCameraReadyFlag == true)
18              mCameraReadyFlag := false
19              mPreview.camera.takePicture
                 (shutterCallback, rawCallback,
                 jpegCallback)
20          return true
21      return super.onKeyDown(keycode, event)

22  ShutterCallback shutterCallback := new
     ShutterCallback()
23      function onShutter()

24  PictureCallback rawCallback := new
     PictureCallback()
25      @Override
```

```
26      function onPictureTaken([] arg0,
         android.hardware.Camera arg1)

27  PictureCallback jpegCallback := new
     PictureCallback()
28      @Override
29      function onPictureTaken([] imageData,
         android.hardware.Camera camera)
30          if (imageData != NIL)
31              Intent mIntent := new Intent()
32              compressByteImage(mContext,
                 imageData, 75)
33              setResult(0, mIntent)
34              ServerTask task := new ServerTask()
                 task.execute(Environment.
                 getExternalStorage
                 Directory().toString() +
                 INPUT_IMG_FILENAME)
35              camera.startPreview()
```

Pseudocode* 4.10 Kode program inisialisasi perangkat *client* dalam memulai *framework

Tahap selanjutnya adalah mengeksekusi *image recognition* yang dilakukan secara *asynchronous* melalui *Pseudocode* 4.11 yang berarti proses komunikasi data terikat dengan waktu tetap serta kecepatan eksekusi cukup relatif bergantung kondisi perangkat pengirim dan penerima. Variabel `dialog` adalah sebuah *pop – up* pada tampilan untuk *client* yang digunakan untuk memantau proses yang sedang berjalan pada perangkat lunak. Pada fungsi `doInBackground`, dilakukan inisialisasi untuk memulai *image recognition* seperti inisialisasi *path* letak dataset dan datatest, jumlah dataset serta pengambilan *descriptor* dari datatest yang digunakan untuk eksekusi proses *image recognition* melaui `surfDatatest()` yang akan dijelaskan pada Bab 4.2.3. Selain itu, pemanggilan fungsi `startRepeatingTask()` dilakukan untuk memulai eksekusi secara *asynchronous* baik secara lokal maupun *offload* ke *server*. Pada fungsi `onProgressUpdate`, dilakukan pembaruan status dari *pop – up* tampilan. Pada fungsi `onPostExecute`, pemanggilan

fungsi `startRepeatingBar()`, digunakan sebagai pemantau proses *image recognition*. Fungsi ini digunakan untuk melihat progress berjalannya eksekusi *image recognition* yang ditampilkan pada *pop – up* `dialog`.

```
1   class ServerTask inherits AsyncTask<String,
     Integer, Void>
2      ServerTask()
3         dialog := new ProgressDialog(mContext)

4      function onPreExecute()
5         dialog.setMessage("Photo captured")
6         dialog.show()

7      @Override
8      function doInBackground(String... params)
          publishProgress(UPLOADING_
9          PHOTO_STATE)
10        surfDatatest()
11        publishProgress(SERVER_PROC_STATE)

12        startRepeatingTask()

13        mCameraReadyFlag := true
14        return NIL

15     @Override
16     function onProgressUpdate(Integer...
        progress)
17     if (progress[0] == UPLOADING_PHOTO_STATE)
18        dialog.setMessage("Uploading")
19        dialog.setCanceledOnTouch
           Outside(false)
20        dialog.show()
21     else if (progress[0] ==
        SERVER_PROC_STATE)
22        if (dialog.isShowing())
23           dialog.dismiss()
24        dialog.setMessage("Processing")
25        dialog.show()

26     @Override
27     function onPostExecute(Void param)
```

```
28          startRepeatingBar()
```

Pseudocode* 4.11 Kode program inisialisasi penerapan *Asynchronous Task* pada *framework

Selanjutnya dilakukan pengecekan status sebelum dilakukannya eksekusi beban kerja *image recognition* melalui *Pseudocode* 4.12. Pada fungsi `run()` secara *asynchronous* dari pemanggilan fungsi `startRepeatingTask()` diawali dengan pengecekan status dari eksekusi beban kerja yang sedang berjalan. Syarat dieksekusinya beban kerja baru adalah jika beban kerja sebelumnya selesai dikerjakan.

Jika eksekusi baru dimulai, maka beban kerja sebelumnya dianggap sudah selesai. Jika eksekusi sebelumnya dilakukan secara *offloading*, dilakukan pengecekan status beban kerja pada kelas java yang mengimplementasikan *agent* dan jika eksekusi beban kerja selesai dilakukan maka di cetak waktu lama eksekusi beban kerja pada *console* Android Studio dan disimpan untuk data histori waktu eksekusi *offloading* yang dibutuhkan oleh *decision maker* pada *framework*. Selain itu, juga dilakukan pengecekan jika eksekusi sebelumnya dilakukan secara *offloading* akan tetapi dibatalkan menjadi eksekusi lokal karena koneksi internet yang buruk. Pengecekan yang terakhir adalah jika eksekusi sebelumnya dilakukan secara lokal dan selesai dilakukan maka di cetak waktu lama eksekusi beban kerja pada *console* Android Studio dan disimpan untuk data histori waktu eksekusi lokal yang dibutuhkan oleh *decision maker* pada *framework*.

```
1   function startRepeatingTask()
2      AgentTask.run()

3   function stopRepeatingTask()
4      handler.removeCallbacks(AgentTask)

5   Runnable AgentTask := new Runnable()
6      i := 0
7      @Override
8      function run()
```

```
9         agentCreated := SendObjectAgent.
           AgentCreated()
10        StatusTask := false
11        if (i == 0)
12           StatusTask := true
13        else if (Offloading == true)
14           if (StatusTaskLocal)
15              StatusTask := StatusTaskLocal
16              StatusTaskLocal := false
17           else
18              StatusTask := SendObjectAgent.
                 FinishTask()
19           if (StatusTask)
20              endTimeTaskOffload()

21        else if (StatusTaskLocal & Offloading ==
           false)
22           StatusTask := StatusTaskLocal
23           StatusTaskLocal := false
24           endTimetaskLocal()
```

Pseudocode* 4.12 Kode program pengecekan status eksekusi beban kerja *image recognition

Dimulainya eksekusi beban kerja *image recognition* pada *framework* dapat dilihat melalui *Pseudocode* 4.13. Ada dua studi kasus dalam memulai eksekusi beban kerja. Studi kasus pertama adalah beban kerja tersebut adalah beban kerja baru disebabkan beban kerja sebelumnya sudah selesai di eksekusi baik secara lokal maupun *offloading* dan hasil eksekusi telah didapatkan. Studi kasus kedua adalah beban kerja tersebut adalah beban kerja yang belum selesai dieksekusi secara sempurna oleh metode *offloading* disebabkan perubahan koneksi internet menjadi buruk sehingga dialihkan menjadi eksekusi secara lokal. Untuk menangani studi kasus tersebut, jika beban kerja tersebut adalah beban kerja baru maka secara langsung dilakukan pengecekan koneksi internet untuk keputusan metode eksekusi selanjutnya. Jika beban kerja tersebut adalah beban kerja yang belum selesai dieksekusi secara sempurna maka status beban kerja dianggap selesai dikerjakan

akan tetapi urutan file nama gambar dataset diundur satu kali agar dialihkan pada metode eksekusi secara lokal.

Pada beban kerja yang diputuskan untuk dieksekusi menggunakan metode *offloading*, akan dilakukan pengambilan beberapa data yang dibutuhkan untuk dilakukan *image recognition* dengan datatest pada *server* yang telah dijelaskan pada Bab 4.2.1.1 seperti nama file dataset, data *string* dari *descriptor* datatest yang didapat menggunakan fungsi `matToJson` yang akan dijelaskan pada Bab 4.2.4. Dalam pembuatan *agent*, dibutuhkan beberapa data seperti nama *agent* yang terdiri dari kombinasi model perangkat client, tanggal, bulan, tahun, jam, menit dan detik yang bertujuan untuk membedakan *agent* yang satu dengan yang lainnya. Selain itu, `agentStartupCallback` dibutuhkan untuk menemukan *container* tempat dimana *agent* dibuat serta `SendObjectAgent.class.getName()` digunakan untuk mereferensi kelas java yang mengimplementasi *agent* yang akan dibuat. Selain itu, dilakukan pengambilan data waktu dimulainya pembuatan *agent* untuk kebutuhan faktor penentu metode *offloading* yang akan dijelaskan pada Bab 4.2.6. Pada beban kerja yang diputuskan untuk dieksekusi secara lokal, akan dilakukan pengambilan data hanya nama file dataset untuk dilakukan proses *image recognition* secara lokal dengan fungsi `surf()` serta dilakukan pengambilan data waktu dimulainya eksekusi secara lokal sama seperti pada metode *offloading*. Jika semua dataset telah dilakukan *image recognition* dengan datatest, maka dipanggil fungsi `stopRepeatingTask()` untuk menghentikan eksekusi beban kerja. Berjalannya eksekusi beban kerja pada *framework* dapat dilihat pada **`finally`** dimana eksekusi beban kerja dilakukan secara teratur selama 1 detik dengan syarat beban kerja sebelumnya telah selesai dilakukan.

```
1  Try
2      if (corruptTask & i != 0)
3          StatusTask := true
4      if (StatusTask)
5          checkConnection()
```

```
6       if (i < files.length & StatusTask == true &
         Offloading == true)
7           if (corruptTask & i != 0)
8               i := i - 1
9               corruptTask := false
10          Log.d(TAG, "CurrentFileName:" +
             currentFileTrain)
11          Log.d(TAG, "FileName:" +
             files[i].getName())
12          file_train := files[i].getName()
13          if (currentFileTrain.equals(file_train))
14              i++
15              file_train := files[i].getName()
16              Log.d(TAG, "NewFileName:" +
                 files[i].getName())
17          JSONdescCamera := matToJson(descCamera)
18          ImageName := file_train
19          String deviceModel := Build.MANUFACTURER
             + "/" + Build.MODEL +  "/" +
             Build.VERSION.RELEASE + "/" +
             Build.VERSION_CODES.class.getFields()
             [android.os.Build.VERSION.SDK_INT].
             getName()
20          Calendar c := Calendar.getInstance()
21          date := c.get(Calendar.DATE)
22          month := c.get(Calendar.MONTH)
23          year := c.get(Calendar.YEAR)
24          hour := c.get(Calendar.HOUR)
25          min := c.get(Calendar.MINUTE)
26          seconds := c.get(Calendar.SECOND)
27          startTimeTask()
28          createAgent("agentOf" + deviceModel + "-
             in-" + date + "/" + month + "/" + year +
             "at" + hour + ":" + min + ":" + seconds,
29          SendObjectAgent.class.getName(),
             agentStartupCallback)
30          i++
31      if (i < files.length & StatusTask == true &
         Offloading == false)
32          if (corruptTask & i != 0)
33              i := i - 1
34              corruptTask := false
35          Log.d(TAG, "CurrentFileName:" +
             currentFileTrain)
```

```
36          Log.d(TAG, "FileName:" +
             files[i].getName())
37          file_train := files[i].getName()
38          if (currentFileTrain.equals(file_train))
39              i++
40              file_train := files[i].getName()
41              Log.d(TAG, "NewFileName:" +
                 files[i].getName())
42          startTimeTask()
43          surf(file_train)
44          i++
45      if (i >= files.length)
46          stopRepeatingTask()
47  finally
48      StatusTask := false
49      handler.postDelayed(AgentTask, 1000)
```

Pseudocode* 4.13 Kode program metode eksekusi beban kerja *image recognition* pada *framework

Selain eksekusi yang dilakukan secara bertahap dan teratur pada beban kerja, dibutuhkan juga suatu eksekusi secara bertahap dan teratur terhadap pemantauan berjalannya eksekusi beban kerja yang diperlukan oleh client sebagai *user interface* dalam perangkat lunak melalui *Pseudocode* 4.14. Pada fungsi `run()` secara *asynchronous* dari pemanggilan fungsi `startRepeatingBar()` diawali dengan pengecekan status dari eksekusi beban kerja yang telah selesai dilakukan dan pembaruan pada tampilan *pop – up* `dialog`. Selain itu, terdapat pengecekan jika semua beban kerja telah selesai dilakukan pada *image recognition* maka dihilangkannya *pop – up* `dialog` untuk menghentikan pemantauan eksekusi beban kerja. Berjalannya pemantauan eksekusi beban kerja pada *framework* dapat dilihat pada **`finally`** dimana pemantauan dilakukan secara teratur selama 0.5 detik.

```
1   function startRepeatingBar()
2       AgentBar.run()

3   function stopRepeatingBar()
4       handler.removeCallbacks(AgentBar)
```

```
5       Collections.sort(listdata, new
         Comparator<data>()
6           @Override
7           function compare(data t1, data t2)
8               return t2.getAcceptpoint() -
                 t1.getAcceptpoint()
9       )
10      Intent result := new Intent(getBaseContext(),
         PageResult.class)
11      result.putExtra("resultListData",
         (Serializable) listdata)
12      startActivity(result)

13  Runnable AgentBar := new Runnable()
14      @Override
15      function run()
16          try
17              if (GetFinishAgentTask !=
                 FinishAgentTask)
18                  dialog.setMessage("Processing " +
                     FinishAgentTask)
19                  GetFinishAgentTask :=
                     FinishAgentTask
20              if (dialog.isShowing() &
                 FinishAgentTask >= files.length)
21                  dialog.dismiss()
22                  mResultView.invalidate()
23                  stopRepeatingBar()
24           finally
25              handler.postDelayed(AgentBar, 500)
```

Pseudocode* 4.14 Kode program kode pemantauan eksekusi beban kerja *image recognition* pada *framework

Untuk menentukan keputusan metode eksekusi yang optimal antara *offloading* dan lokal. Dibutuhkan *decision maker* yang menentukan keputusan berdasarkan faktor – faktor penentu *offloading* yang dapat dilihat secara bertahap melalui *Pseudocode* 4.15, *Pseudocode* 4.16, *Pseudocode* 4.17, *Pseudocode* 4.18 dan *Pseudocode* 4.19. Pada *Pseudocode* 4.15, dilakukan pengecekan adanya koneksi internet pada perangkat. Pada fungsi `showSnack()` dilakukan inisialisasi variabel `batteryLevel` yang digunakan

pada salah satu faktor penentu metode *offloading*. Nilai `batteryLevel` didapatkan dari fungsi `getBatteryPercent()` yang akan dijelaskan pada Bab 4.2.6. Variabel `weightOffload` digunakan untuk pembobotan nilai yang digunakan dalam dasar pengambilan keputusan oleh *decision maker*.

Jika dalam pengecekan koneksi internet dinyatakan adanya koneksi pada perangkat, maka selanjutnya akan dilakukan pengecekan terhadap tersedianya `microRuntimeServiceBinder` pada perangkat *client*. Jika belum tersedia, maka dibuat `microRuntimeServiceBinder` baru menggunakan fungsi `bindService()` dengan parameter *host* dan *port* yang ditampung oleh `SharedPreferences`. Pada studi kasus lain, `microRuntimeServiceBinder` sudah terbuat dan menjalankan eksekusi beban kerja secara *offloading* akan tetapi koneksi internet terputus sehingga eksekusi dialihkan secara lokal. Apabila koneksi tersambung kembali dan diputuskan untuk melakukan metode *offloading*, maka harus dilakukan pembuatan `microRuntimeServiceBinder` yang baru dan menghapus yang lama.

```
1   function checkConnection()
2      isConnected :=
        ConnectivityReceiver.isConnected()
3      showSnack(isConnected)

4   function showSnack(isConnected)
5      batteryLevel := getBatteryPercent()
6      weightOffload := 0
7      if (isConnected)
8          if (NOT bindservice)
9              if (rateExecutionTimeOffload != 0)
10                 unbindService(mServiceConnection)
11                 microRuntimeServiceBinder := NIL
12             SharedPreferences sharedPreferences :=
                SURFExampleActivity.this.
                getSharedPreferences(Constants.
                PREFS_FILE_NAME, Context.
                MODE_PRIVATE)
```

```
13              String host :=
                 sharedPreferences.getString
                 (Constants.PREFS_HOST_ADDRESS,
                 ipAddress)
14              String port := settings.getString
                 ("defaultPort", "")
15              Log.e(TAG, "Connecting to --> host :
                 " + host + " - " + "port : " + port)
16              bindService(host, port,
                 agentStartupCallback)
```

Pseudocode* 4.15 Kode program *Decision Maker* melakukan pengecekan terhadap ketersediaan koneksi internet dan *microRuntimeServiceBinder

Pada *Pseudocode* 4.16, dilakukan pengecekan kualitas koneksi internet dengan 4 pembagian, yaitu *Excellent*, *Good*, *Moderate* dan *Poor*. Jika dinyatakan kualitas internet *Excellent*, maka pembobotan ditambahkan nilai tiga. Jika dinyatakan kualitas internet *Good*, maka pembobotan ditambahkan nilai dua. Jika dinyatakan kualitas internet *Moderate*, maka pembobotan ditambahkan nilai satu. Jika dinyatakan kualitas internet *Poor*, maka tidak ada penambahan bobot nilai.

```
1   Connectivity := new ConnectivityReceiver()
2   NetworkInfo info := Connectivity.
     getNetworkInfo(getBaseContext())
3   if (mConnectionClassManager.getCurrentBandwidth
     Quality() == ConnectionQuality.EXCELLENT |
     mConnectionClassManager.getCurrentBandwidth
     Quality() == ConnectionQuality.GOOD |
     mConnectionClassManager.getCurrentBandwidth
     Quality() == ConnectionQuality.MODERATE)
4        if (mConnectionClassManager.getCurrent
          BandwidthQuality() ==
          ConnectionQuality.EXCELLENT)
5           weightOffload := weightOffload + 3
6           Log.e(TAG, "+++weightOffload
             Connection: " + mConnectionClass
             Manager.getCurrent
             BandwidthQuality())
```

```
7       if (mConnectionClassManager.getCurrent
         BandwidthQuality() ==
         ConnectionQuality.GOOD)
8           weightOffload := weightOffload + 2
9           Log.e(TAG, "++weightOffload
             Connection: " + mConnectionClass
             Manager.getCurrent
             BandwidthQuality())
10      if (mConnectionClassManager.getCurrent
         BandwidthQuality() ==
         ConnectionQuality.MODERATE)
11          weightOffload++
12          Log.e(TAG, "+weightOffload
             Connection: " + mConnectionClass
             Manager.getCurrent
             BandwidthQuality())
13  if (mConnectionClassManager.getCurrentBandwidth
     Quality() == ConnectionQuality.POOR)
14      Log.d(TAG, "Internet Poor")
15      StatusTaskLocal := true
```

***Pseudocode* 4.16 Kode program *Decision Maker* melakukan pengecekan terhadap kualitas koneksi internet**

Pada *Pseudocode* 4.17, dilakukan pengecekan terhadap level baterai sebagai salah satu faktor penentu metode *offloading*, Jika level baterai berada pada nilai dibawah dua puluh yang berarti harus dilakukan penghematan lebih agar perangkat *client* tidak mati akibat kehabisan baterai, maka ditambahkan satu nilai pembobotan untuk lebih cenderung dilakukan metode *offloading*.

```
1   Log.e(TAG, "Battery level : " + batteryLevel)
2   if (batteryLevel < 20)
3       weightOffload++
4       Log.e(TAG, "+weightOffload Battery: " +
         batteryLevel)
```

***Pseudocode* 4.17 Kode program *Decision Maker* melakukan pengecekan terhadap kondisi level baterai**

Pada *Pseudocode* 4.18, dilakukan pengecekan tehadap berjalannya eksekusi beban kerja serta pengambilan keputusan

oleh *Decision Maker*. Hal yang dilakukan pertama kali adalah eksekusi beban kerja yang pertama harus dilakukan secara lokal sesuai dengan penjelasan pada Bab 3.5 untuk memperoleh data waktu eksekusi secara lokal yang selanjutnya akan dibandingkan waktu eksekusi secara *offloading*. Eksekusi beban kerja selanjutnya akan dilakukan secara *offloading* jika tersedia koneksi internet dan kualitas koneksi tidak buruk. Setelah dilakukan kedua metode eksekusi beban kerja pada *framework*, *Decision Maker* akan membandingkan waktu eksekusi tiap metode eksekusi. Jika selisih rata – rata waktu eksekusi secara *offloading* dengan rata – rata waktu eksekusi secara lokal kurang dari sama dengan delapan detik, maka ditambahkan satu nilai pembobotan.

Untuk melakukan eksekusi beban kerja secara *offloading*, nilai pembobotan harus lebih dari dua. Akan tetapi, jika dalam eksekusi beban kerja secara *offloading* koneksi internet terputus secara otomatis eksekusi beban kerja dialihkan secara lokal oleh *framework*. Ketika koneksi internet kembali tersedia pada perangkat, maka dilakukan pengecekan terhadap kualitas koneksi internet. Jika kualitas koneksi internet buruk maupun normal, maka eksekusi beban kerja tetap dilakukan secara lokal sebab koneksi internet saat awal tersedia pada perangkat *client* tidak stabil sehingga menyebabkan gagalnya pembuatan *agent* pada perangkat *client*.

```
1   if (rateExecutionTimeOffload == 0 &
     mConnectionClassManager.getCurrent
     BandwidthQuality() != ConnectionQuality.POOR &
     mConnectionClassManager.getCurrent
     BandwidthQuality() != ConnectionQuality.UNKNOWN)
2       Offloading := true
3   if (rateExecutionTimeLocal == 0)
4       Offloading := false
5   else if (rateExecutionTimeLocal != 0)
6       if (rateExecutionTimeOffload != 0)
7           Log.e(TAG, "Delta Execution Time: " +
             (rateExecutionTimeOffload –
             rateExecutionTimeLocal))
8           if (rateExecutionTimeOffload –
```

```
          rateExecutionTimeLocal <= 8)
9            Log.e(TAG, "+weightOffload Delta
              Execution Time: " +
              (rateExecutionTimeOffload -
              rateExecutionTimeLocal))
10           weightOffload++
11        if (weightOffload >= 2)
12           Offloading := true
13        else
14           Offloading := false
15        if (corruptTaskOffload)
16           if
    (mConnectionClassManager.getCurrent
              BandwidthQuality() ==
              ConnectionQuality.MODERATE &
              mConnectionClassManager.getCurrent
              BandwidthQuality() ==
17            ConnectionQuality.POOR)
18               Offloading := false
              corruptTaskOffload := false
```

***Pseudocode* 4.18 Kode program *Decision Maker* melakukan pembobotan nilai untuk keputusan metode eksekusi**

Pada *Pseudocode* 4.19, dilakukan eksekusi secara lokal jika tidak tersedia koneksi pada perangkat *client*. Inisialisasi pada variabel *Decision Maker* dilakukan agar *framework* melakukan eksekusi beban kerja secara lokal. Selain itu, terdapat beberapa variabel yang perlu di inputkan nilainya jika dalam eksekusi beban kerja secara *offloading* koneksi internet terputus dan dialihkan pada eksekusi beban kerja secara lokal

```
1   else
2       Log.e(TAG, "Not Connected")
3       mConnectionClassManager.reset()
4       Log.e(TAG, "Connection class : " +
         mConnectionClassManager.getCurrent
         BandwidthQuality().toString())
5       if (Offloading)
6           corruptTask := true
7           corruptTaskOffload := true
8           corruptTaskHandler := true
```

```
9           file_train_corrupt := file_train
10      Offloading := false
11      bindservice := false
```

Pseudocode* 4.19 Kode program *Decision Maker* melakukan metode eksekusi lokal jika tidak tersedianya koneksi internet pada perangkat *client

Tahap akhir dari *framework* adalah menyimpan data hasil eksekusi beban kerja pada suatu *array* bertipe kelas *data* yaitu `listdata` melalui *Pseudocode* 4.20 yang selanjutnya akan diurutkan secara *descending* (besar ke kecil) menurut jumlah titik fitur yang sesuai kriteria pada fungsi `stopRepeatingBar()` pada *Pseudocode* 4.14 serta ditampilkan pada *console* Android Studio IDE untuk pengecekan. Selain itu, dilakukan pengecekan koneksi internet kembali melalui `DownloadImage().execute()` yang akan dijelaskan pada Bab 4.2.6.

```
1   function exportLogConsole(data log)
2       Message logMessage := new Message()
3       logMessage.obj := log
4       handlerMessage.sendMessage(logMessage)

6   function onCreate(Bundle savedInstanceState)
7       handlerMessage := new Handler()
8           @Override
9           function handleMessage(Message
             dataServer)
10              if (NOT
                 file_train_corrupt.equals(((data)
                 dataServer.obj).getNama()) |
                 corruptTaskHandler)
11                  corruptTaskHandler := false
12                  listdata.add((data)
                     dataServer.obj)
13                  Log.i(TAG,
                     dataServer.obj.toString())
14                  Log.i(TAG, "name : " + ((data)
                     dataServer.obj).getNama() + "
                     min distance : " + ((data)
                     dataServer.obj).getMin_dist() +"
```

```
                    max distance : " + ((data)
                    dataServer.obj).getMax_dist() +"
                    accepted : " + ((data)
                    dataServer.obj).
                    getAcceptpoint()+"\n" + "
                    Send Agent Time : " + ((data)
                    dataServer.obj).
                    getSendAgentTime() + "\n" + "
                    Receive Agent Time : " + ((data)
                    dataServer.obj).
                    getReceiveAgentTime() + "\n" + "
                    AgentTransportTime : " + ((data)
                    dataServer.obj).
                    getTransportAgentTime())
15                  currentFileTrain := ((data)
                     dataServer.obj).getNama()
                    FinishAgentTask++
16                  Log.i(TAG, " Task : " +
                     FinishAgentTask)
17                  new DownloadImage().execute(mURL)
```

Pseudocode* 4.20 Kode program menyimpan hasil eksekusi beban kerja pada *framework

4.2.3 Metode SURF

Hal yang dibutuhkan untuk melakukan metode SURF adalah memperoleh *descriptor* dari matriks citra daun datatest dan dataset. Untuk agar memperoleh *descriptor* yang optimal dari matriks citra datatest maka dilakukan kompresi melalui *Pseudocode* 4.21.

```
1   function compressByteImage(Context mContext, []
2    imageData, quality)
3       File sdCard := Environment.getExternal
         StorageDirectory()
4       FileOutputStream fileOutputStream := NIL
5       try
6           BitmapFactory.Options options := new
             BitmapFactory.Options()
7           options.inSampleSize := 1
8           Bitmap myImage :=
             BitmapFactory.decodeByteArray
```

```
          (imageData, 0, imageData.length,options)
9         fileOutputStream := new FileOutputStream
          (sdCard.toString() + INPUT_IMG_FILENAME)
10        BufferedOutputStream bos := new Buffered
          OutputStream(fileOutputStream)
11        myImage.compress(CompressFormat.JPEG,
          quality, bos)
12        bos.flush()
13        bos.close()
14        fileOutputStream.close()
15    catch (FileNotFoundException e)
16        Log.e(TAG, "FileNotFoundException")
17        e.printStackTrace()
18    catch (IOException e)
19        Log.e(TAG, "IOException")
20        e.printStackTrace()
21    return true
```

***Pseudocode* 4.21 Program kompresi citra datatest**

Setelah dikompresi, dilakukan pengambilan *descriptor* pada citra datatest melalui *Pseudocode* 4.22 dengan menggunakan `FeatureDetector` SURF serta `DescriptorExtractor` SURF. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*.

```
1   function surfDatatest()
2       System.loadLibrary("opencv_java")
3       System.loadLibrary("nonfree")

4       Bitmap inputImage, cameraImage
5       FeatureDetector detector :=
6       FeatureDetector.create(FeatureDetector.SURF)
7       DescriptorExtractor descriptorExtractor :=
         DescriptorExtractor.create
         (DescriptorExtractor.SURF)
8       DescriptorMatcher descriptorMatcher :=
         DescriptorMatcher.create
         (DescriptorMatcher.FLANNBASED)
9       Mat rgbaCamera, descCamera, result
10      MatOfDMatch matches
11      MatOfKeyPoint keyPointsCamera

12      rgbaCamera := new Mat()
```

```
13      descCamera := new Mat()
14      matches := new MatOfDMatch()
15      result := new Mat()
16      BitmapFactory.Options bmOptions := new
         BitmapFactory.Options()
17      File testimage := new
         File(Environment.getExternalStorage
         Directory().toString(), "
         INPUT_IMG_FILENAME")
18      cameraImage := BitmapFactory.decodeFile
         (testimage.getAbsolutePath(), bmOptions)
19      cameraImage := Bitmap.createScaledBitmap
         (cameraImage, cameraImage.getWidth(),
20       cameraImage.getHeight(), true)
21      Utils.bitmapToMat(cameraImage, rgbaCamera)
22      keyPointsCamera := new MatOfKeyPoint()
23      Imgproc.cvtColor(rgbaCamera, rgbaCamera,
         Imgproc.COLOR_RGBA2GRAY)
24      detector.detect(rgbaCamera, keyPointsCamera)
25      descriptorExtractor.compute(rgbaCamera,
         keyPointsCamera, descCamera)
26      Features2d.drawKeypoints(rgbaCamera,
         keyPointsCamera, rgbaCamera)
27      Utils.matToBitmap(rgbaCamera, cameraImage)
```

***Pseudocode* 4.22 Kode Program mendapatkan *descriptor* pada datatest**

Pada eksekusi beban kerja secara lokal, dilakukan juga pengambilan *descriptor* pada citra dataset hampir sama dengan datatest melalui *Pseudocode* 4.23 dengan menggunakan `FeatureDetector` SURF serta `DescriptorExtractor` SURF. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*. Selain itu, untuk menghitung jarak kemiripan digunakan `DescriptorMatcher` FLANNBASED. Selanjutnya jarak fitur antara datatest dan dataset yang didapatkan berupa titik – titik tersebut dilakukan pengecekan, jika nilai jarak kemiripan diterima yaitu 0.15 maka akan `acceptPoint` pada kelas *Java Object Serialization data* yang berarti semakin besar `acceptPoint` maka datatest dan dataset semakin mirip.

```
1   function surf(String imageName)
2       FeatureDetector detector2 :=
3       FeatureDetector.create(FeatureDetector.SURF)
4       DescriptorExtractor descriptorExtractor2 :=
         DescriptorExtractor.create
         (DescriptorExtractor.SURF)
5       DescriptorMatcher descriptorMatcher2 :=
         DescriptorMatcher.create
         (DescriptorMatcher.FLANNBASED)
6       Mat rgba := new Mat()
7       Mat desc := new Mat()
8       BitmapFactory.Options bmOptions := new
         BitmapFactory.Options()
9       File image := new File(path, imageName)
10      inputImage := BitmapFactory.decodeFile
         (image.getAbsolutePath(), bmOptions)
11      inputImage := Bitmap.createScaledBitmap
         (inputImage, inputImage.getWidth(),
         inputImage.getHeight(), true)
12      Utils.bitmapToMat(inputImage, rgba)
13      Imgproc.cvtColor(rgba, rgba,
14      Imgproc.COLOR_RGBA2GRAY)
15      detector2.detect(rgba, keyPoints)
16      descriptorExtractor2.compute
         (rgba, keyPoints, desc)
17      descriptorMatcher2.match
         (descCamera, desc, matches)
18      count := 0
19      max_dist := 0
20      min_dist := 0.15
21      List<DMatch> matchesList := matches.toList()
22      Features2d.drawKeypoints
         (rgba, keyPoints, rgba)
23      for i := 0 loop till i < descCamera.rows() by
         i++ each step
24          dist := matchesList.get(i).distance
25          if (dist < 0.15)
26              if (dist < min_dist)
27                  min_dist := dist
28              count++
29          if (dist > max_dist)
30              max_dist := dist
31      data newdata := new data(imageName, count,
         min_dist, max_dist, 0, "0", 0)
```

```
32      exportLogConsole(newdata)
33      StatusTaskLocal := true
34      Log.e(TAG, "Execution Local")
```

***Pseudocode* 4.23 Kode program eksekusi *image recognition* secara lokal**

Pada eksekusi beban kerja secara *offloading*, dilakukan juga pengambilan *descriptor* pada citra dataset yang disediakan oleh *server* melalui *Pseudocode* 4.24 dengan menggunakan `FeatureDetector` SURF serta `DescriptorExtractor` SURF sama seperti pada eksekusi secara lokal. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*. Hampir seluruh urutan tahapnya sama dengan metode eksekusi beban kerja secara lokal yang telah dijelaskan sebelumnya, hanya perbedaan dalam menulis kode sumber.

```
1   String path :=
     "C://xampp//htdocs//ServerCode//training//
     test//training//"
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME)

2   function surf(String imageName)
3       detector := FeatureDetector.
         create(FeatureDetector.SURF)
4       descriptorExtractor :=
         DescriptorExtractor.
         create(DescriptorExtractor.SURF)
5       descriptorMatcher :=
         DescriptorMatcher.create
6        (DescriptorMatcher.FLANNBASED)
7       Mat rgba := new Mat()
8       Mat desc := new Mat()
        BitmapFactory.Options bmOptions := new
9        BitmapFactory.Options()
10      File image := new File(path, imageName)
        rgba := Highgui.imread(path + imageName,
11       Highgui.CV_LOAD_IMAGE_COLOR)
        MatOfKeyPoint keyPoints := new
12       MatOfKeyPoint()
        Imgproc.cvtColor(rgba, rgba,
```

```
13          Imgproc.COLOR_RGBA2GRAY)
14        detector.detect(rgba, keyPoints)
          descriptorExtractor.compute(rgba, keyPoints,
15         desc)
16        matches := new MatOfDMatch()
          descriptorMatcher.match(descriptor, desc,
17         matches)
18        count := 0
19        max_dist := 0
20        min_dist := 0.15
21        List<DMatch> matchesList := matches.toList()
          Features2d.drawKeypoints(rgba, keyPoints,
22         rgba)
          for i := 0 loop till i < descriptor.rows() by
23         i++ each step
24            dist := matchesList.get(i).distance
25            if (dist < 0.15)
26                if (dist < min_dist)
27                    min_dist := dist
28                count++
29            if (dist > max_dist)
30                max_dist := dist
          System.err.println("result " + " min distance
           : " + min_dist + " max distance : " + max_dist
           + " accepted : " + count)
31
          TimeZone local :=
32         TimeZone.getTimeZone("Asia/Jakarta")
33        Calendar now := Calendar.getInstance(local)
          SimpleDateFormat formatter := new
34         SimpleDateFormat("HH:mm:ss.SSS")
35        formatter.setTimeZone(local)
          String ReceiveAgentStartTime :=
36         formatter.format(now.getTime())
          d := new data(imageName, count, min_dist,
           max_dist, sendAgentTime,
           ReceiveAgentStartTime, 0)
```

Pseudocode* 4.24 Kode Program eksekusi *image recognition* secara *offloading

4.2.4 Java *Class Object Serialization* dari *Client* ke *Server*

Proses awal yang dilakukan dalam membuat kelas *Java Object Serialization* dari *client* ke *server* adalah mengubah data matriks citra daun datatest menjadi data string agar dapat disisipkan pada variabel kelas *Java Object Serialization*. Pengubahan data matriks citra daun menjadi string dapat dilihat pada *Pseudocode* 4.25. Pada fungsi `matToJson()` dilakukan inisialisasi baris dan kolom serta tipe data dari nilai di dalam matriks. Pada studi kasus matriks citra daun pada tugas akhir ini, tipe data pada nilai matriks adalah *float* sehingga pada fungsi akan dikenal dengan tipe `CvType.CV_32F`. Fungsi `matToJson()` akan mengembalikan nilai string dari konversi matriks citra daun datatest.

```
1   function matToJson(Mat mat)
2       JsonObject obj := new JsonObject()
3       cols := mat.cols()
4       rows := mat.rows()
5       elemSize := mat.elemSize()
6       type := mat.type()
7       obj.addProperty("rows", mat.rows())
8       obj.addProperty("cols", mat.cols())
9       obj.addProperty("type", mat.type())
10      String dataString
11      if (type == CvType.CV_32S OR type ==
         CvType.CV_32SC2 OR type == CvType.CV_32SC3
         OR type == CvType.CV_16S)
12          [] data := [0..cols * rows * elemSize-1]
13          mat.get(0, 0, data)
14          dataString := new
             String(Base64.encode(Serialization
             Utils.serialize(data), Base64.DEFAULT))
15      else if (type == CvType.CV_32F OR type ==
         CvType.CV_32FC2)
16          [] data := [0..cols * rows * elemSize-1]
17          mat.get(0, 0, data)
18          dataString := new
             String(Base64.encode(Serialization
             Utils.serialize(data), Base64.DEFAULT))
19      else if (type == CvType.CV_64F OR type ==
         CvType.CV_64FC2)
```

```
20          [] data := [0..cols * rows * elemSize-1]
21          mat.get(0, 0, data)
22          dataString := new
             String(Base64.encode(Serialization
             Utils.serialize(data), Base64.DEFAULT))
23      else if (type == CvType.CV_8U)
24          [] data := [0..cols * rows * elemSize-1]
25          mat.get(0, 0, data)
26          dataString := new String(Base64.encode
             (data, Base64.DEFAULT))
27      else
28          throw new UnsupportedOperation
             Exception("unknown type")
29      obj.addProperty("data", dataString)
30      Gson gson := new Gson()
31      String json := gson.toJson(obj)
32      return json
```

Pseudocode* 4.25 Kode program pengubahan data matriks menjadi string pada perangkat *client

Untuk dapat menyisipkan berbagai data pada modul pengiriman data yang dikirim oleh *agent*, dibuatlah kelas *Java Object Serialization* `ObjectDataMat` seperti pada Pseudocode 4.26. Pada kelas tersebut akan diinisialisasi `serialVersionUID` yang harus bernilai sama dengan `serialVersionUID` pada kelas *Java Object Serialization* `ObjectDataMat` yang berada pada *server* dengan tujuan agar modul pengiriman data dapat terkirim ke *server*. Inisialisasi data lain yang dilakukan adalah data yang dibutuhkan untuk *image recognition* seperti *descriptor* datatest, nama file dataset, dan waktu dimulainya eksekusi.

```
1   class ObjectDataMat implements Serializable
2       serialVersionUID := 8551377534047755760L

3           System.loadLibrary("opencv_java")
4           System.loadLibrary("nonfree")

5       String descCamera, ImageName,
         SendAgentStartTime
```

```
6       ObjectDataMat(String desc, String ImgName,
         String SendAgentStartTime)
7           this.descCamera := desc
8           this.ImageName := ImgName
9           this.SendAgentStartTime :=
             SendAgentStartTime
```

Pseudocode* 4.26 Kode Program kelas *Java Object Serialization* ObjectDataMat pada perangkat *client

Pada kelas *Java Object Serialization* `ObjectDataMat` pada *server* yang diperlihatkan pada *Pseudocode* 4.27, kontennya hampir sama dengan kelas *Java Object Serialization* `ObjectDataMat` pada perangkat *client*. Perbedaannya hanya terletak pada penambahan fungsi untuk mengambil nilai dari masing – masing variabel.

```
1   class ObjectDataMat implements Serializable
2       serialVersionUID := 8551377534047755760L
3       String descCamera, ImageName,
         SendAgentStartTime
4       ObjectDataMat(String desc, String ImgName,
         String SendAgentStartTime)
5           this.descCamera := desc
6           this.ImageName := ImgName
7           this.SendAgentStartTime :=
             SendAgentStartTime

8       function getDescriptor()
9           return this.descCamera

10      function getImageName()
11          return this.ImageName

12      function getSendAgentStartTime()
13          return this.SendAgentStartTime

14      function toString()
15          return ("desc : " + descCamera.toString()
             + "\n" + "image name : " +
             ImageName.toString() + "\n" + "Send Agent
             Start Time : " +
```

```
    SendAgentStartTime.toString())
```

Pseudocode* 4.27 Kode Program kelas *Java Object Serialization* ObjectDataMat pada perangkat *server

Agar data *descriptor* yang berupa *string* harus dirubah terlebih dahulu menjadi tipe data *Mat* agar dapat diproses dalam *image recognition*. Pengubahan tipe data *string* menjadi tipe data *Mat* dilakukan oleh fungsi `matFromJson()` dapat dilihat pada *Pseudocode* 4.28. Proses pengubahannya hampir sama dengan fungsi `matToJson()` pada perangkat *client*.

```
1   function matFromJson(String json)
2       JsonParser parser := new JsonParser()
3       JsonObject JsonObject :=
         parser.parse(json).getAsJsonObject()
4       rows := JsonObject.get("rows").getAsInt()
5       cols := JsonObject.get("cols").getAsInt()
6       type := JsonObject.get("type").getAsInt()
7       Mat mat := new Mat(rows, cols, type)
8       String dataString :=
         JsonObject.get("data").getAsString()
9       if (type == CvType.CV_32S OR type ==
         CvType.CV_32SC2 OR type == CvType.CV_32SC3
         OR type == CvType.CV_16S)
10          [] data := SerializationUtils.deserialize
             (Base64.decodeBase64
             (dataString.getBytes()))
11          mat.put(0, 0, data)
12      else if (type == CvType.CV_32F OR type ==
         CvType.CV_32FC2)
13          [] data := SerializationUtils.deserialize
             (Base64.decodeBase64
             (dataString.getBytes()))
14          mat.put(0, 0, data)
15      else if (type == CvType.CV_64F OR type ==
         CvType.CV_64FC2)
16          [] data := SerializationUtils.deserialize
             (Base64.decodeBase64
             (dataString.getBytes()))
17          mat.put(0, 0, data)
18      else if (type == CvType.CV_8U)
```

```
19          [] data := Base64.decodeBase64
             (dataString.getBytes())
20          mat.put(0, 0, data)
21      else
22          throw new UnsupportedOperation
             Exception("unknown type")
23      return mat
```

Pseudocode* 4.28 Kode program pengubahan string menjadi Data Matriks pada perangkat *server

4.2.5 Java *Class Object Serialization* dari *Server* ke *Client*

Proses awal yang dilakukan dalam membuat kelas *Java Object Serialization* dari *server* ke *client* adalah memastikan data hasil proses *image recognition* telah didapatkan dan siap untuk di inisialisasi pada kelas *Java Object Serialization* `data` yang dapat dilihat melalui *Pseudocode* 4.29. Implementasi kelas *Java Object Serialization* `data` untuk modul pengiriman data dari *server* ke *client* hampir sama dengan penerapan kelas *Java Object Serialization* `ObjectDataMat` yang telah dijelaskan sebelumnya. Terdapat data – data hasil *image recognition* yang diinisialisasi pada kelas *Java Object Serialization* `data` seperti `acceptpoint`, `min_dist`, `max_dist`, `SendAgentTime`, `ReceiveAgentStartTime`, dan `ReceiveAgentTime`. Terdapat juga fungsi – fungsi yang digunakan untuk mengambil dan mengubah data tiap – tiap variabel.

```
1   class data implements Serializable
2       serialVersionUID := 8551377534047755760L

3       String name
4       acceptpoint
5       min_dist
6       max_dist
7       String ReceiveAgentStartTime
8       SendAgentTime
9       ReceiveAgentTime
```

```
10      data(String name, acceptpoint, min_dist,
         max_dist, SendAgentTime, String
         ReceiveAgentStartTime, ReceiveAgentTime)
11          this.name := name
12          this.acceptpoint := acceptpoint
13          this.min_dist := min_dist
14          this.max_dist := max_dist
15          this.SendAgentTime := SendAgentTime
16          this.ReceiveAgentStartTime :=
             ReceiveAgentStartTime
17          this.ReceiveAgentTime := ReceiveAgentTime

18      function getNama()
19          return name

20      function setNama(String name)
21          this.name := name

22      function getAcceptpoint()
23          return acceptpoint

24      function setAcceptpoint(acceptpoint)
25          this.acceptpoint := acceptpoint

26      function getMin_dist()
27          return min_dist

28      function setMin_dist(min_dist)
29          this.min_dist := min_dist

30      function getMax_dist()
31          return max_dist

32      function setMax_dist(max_dist)
33          this.max_dist := max_dist

34      function getSendAgentTime()
35          return SendAgentTime

36      function setSendAgentTime(SendAgentTime)
37          this.SendAgentTime := SendAgentTime

38      function getReceiveAgentStartTime()
39          return ReceiveAgentStartTime
```

```
40      function setReceiveAgentStartTime(String
         ReceiveAgentStartTime)
41          this.ReceiveAgentStartTime :=
             ReceiveAgentStartTime

42      function getReceiveAgentTime()
43          return ReceiveAgentTime

44      function setReceiveAgentTime
         (ReceiveAgentTime)
45          this.ReceiveAgentTime := ReceiveAgentTime
```

Pseudocode* 4.29 Kode Program kelas *Java Object Serialization* data pada perangkat *server

Pada kelas *Java Object Serialization* `data` pada perangkat *client* yang dapat dilihat melalui *Pseudocode* 4.30, inisialisasi variabelnya sama dengan kelas *Java Object Serialization* `data` pada *server*. Begitu juga dengan penerapan fungsi yang digunakan untuk mengambil dan mengubah data tiap – tiap variabel.

```
1   class data implements Serializable
2       serialVersionUID := 8551377534047755760L

3       String name
4       acceptpoint
5       min_dist
6       max_dist
7       String ReceiveAgentStartTime
8       SendAgentTime
9       ReceiveAgentTime

10      data(String name, acceptpoint, min_dist,
         max_dist, SendAgentTime, String
         ReceiveAgentStartTime, ReceiveAgentTime)
11          this.name := name
12          this.acceptpoint := acceptpoint
13          this.min_dist := min_dist
14          this.max_dist := max_dist
15          this.SendAgentTime := SendAgentTime
16          this.ReceiveAgentStartTime :=
```

```
          ReceiveAgentStartTime
17        this.ReceiveAgentTime := ReceiveAgentTime

18    function getNama()
19        return name

20    function setNama(String name)
21        this.name := name

22    function getAcceptpoint()
23        return acceptpoint

24    function setAcceptpoint(acceptpoint)
25        this.acceptpoint := acceptpoint

26    function getMin_dist()
27        return min_dist

28    function setMin_dist(min_dist)
29        this.min_dist := min_dist

30    function getMax_dist()
31        return max_dist

32    function setMax_dist(max_dist)
33        this.max_dist := max_dist

34    function getSendAgentTime()
35        return SendAgentTime

36    function setSendAgentTime(SendAgentTime)
37        this.SendAgentTime := SendAgentTime

38    function getReceiveAgentStartTime()
39        return ReceiveAgentStartTime

40    function setReceiveAgentStartTime(String
41     ReceiveAgentStartTime)
          this.ReceiveAgentStartTime :=
           ReceiveAgentStartTime

42    function getReceiveAgentTime()
43        return ReceiveAgentTime
```

```
44      function setReceiveAgentTime
         (ReceiveAgentTime)
45          this.ReceiveAgentTime := ReceiveAgentTime

46      function getTransportAgentTime()
47          return (SendAgentTime + ReceiveAgentTime)
```

Pseudocode* 4.30 Kode Program kelas *Java Object Serialization* data pada perangkat *client

4.2.6 Faktor Penentu Metode *Offloading*

4.2.6.1 Kualitas Koneksi Internet

Pengecekan kualitas koneksi internet melalui *Pseudocode* 4.31 diawali dengan inisialisasi *library* konektivitas yang diimplementasikan oleh kelas utama `SURFExampleActivity`. Pada fungsi `onCreate()` dilakukan pemanggilan fungsi `checkConnection()` dan `new DownloadImage().execute()` dengan tujuan pengecekan koneksi internet dilakukan secara berkala dan dimunculkannya pemberitahuan jika terdapat perubahan terhadap kualitas koneksi internet selama proses eksekusi beban kerja image recognition berlangsung.

```
1   class SURFExampleActivity inherits Activity
     implements ConnectivityReceiver.
     ConnectivityReceiverListener

2       ConnectivityReceiver Connectivity
3       ConnectionClassManager
         mConnectionClassManager
4       DeviceBandwidthSampler
         mDeviceBandwidthSampler
5       ConnectionChangedListener mListener
6       String mURL := "https://c1.staticflickr.com
         /6/5646/30422515475_5482a5e51b_b.jpg"
7       mTries := 0


8       ConnectionQuality mConnectionClass :=
```

```
       ConnectionQuality.UNKNOWN

9      @Override
10     function onCreate(Bundle savedInstanceState)
11         super.onCreate(savedInstanceState)

12         mConnectionClassManager :=
            ConnectionClassManager.getInstance()
13         mDeviceBandwidthSampler :=
            DeviceBandwidthSampler.getInstance()
14         mListener := new
            ConnectionChangedListener()
15         checkConnection()
16         new DownloadImage().execute(mURL)
```

Pseudocode* 4.31 Kode Program inisialisasi pengecekan koneksi internet pada perangkat *client

Pseudocode 4.32 memperlihatkan fungsi dalam kode sumber yang digunakan untuk pengecekan secara berkala ketersediaan koneksi internet pada perangkat *client* melalui `onNetworkConnectionChanged()` maupun kualitas koneksi internet melalui `ConnectionChangedListener`.

```
1      @Override
2      function onNetworkConnectionChanged(isConnected)
           showSnack(isConnected)

3      class ConnectionChangedListener implements
        ConnectionClassManager.Connection
        ClassStateChangeListener
4          @Override
5          function onBandwidthState
            Change(ConnectionQuality bandwidthState)
6              mConnectionClass := bandwidthState
7              runOnUiThread(new Runnable()
8                  @Override
9                  function run()
10                     Log.e(TAG, "Connection Class: " +
                        mConnectionClass.toString())
11             )
```

Pseudocode 4.32 Kode Program kondisi saat koneksi internet mengalami perubahan kualitas dan pengecekan ketersediaan

Untuk mengetahui kualitas koneksi internet pada perangkat *client*, dijalankan kelas `DownloadImage` yang berjalan secara *asynchronous* dalam pengecekan kualitas koneksi internet menggunakan *buffer* data saat melakukan *transmit* data melalui *Pseudocode* 4.33. Transmit dilakukan dengan mengunduh file random yang memiliki ukuran yang disesuaikan. Pada tugas akhir ini, data yang digunakan adalah file gambar dengan ukuran data kurang lebih 500 KB. *Buffer* data saat mengunduh file akan dibandingkan dengan referensi *buffer* data sesuai ukuran file yang telah dijelaskan pada Bab 3.4.1. Transmit data akan dihentikan jika kelas tersebut menemukan kualitas koneksi internet perangkat *client*.

```
1   class DownloadImage inherits AsyncTask<String,
    Void, Void>
2       @Override
3       function onPreExecute()
4           mDeviceBandwidthSampler.startSampling()
5       @Override
6       function doInBackground(String... url)
7           String imageURL := url[0]
8           try
9               URLConnection connection := new
                 URL(imageURL).openConnection()
10              connection.setUseCaches(false)
11              connection.connect()
12              InputStream input :=
                 connection.getInputStream()
13              try
14                  [] buffer := [0..1023]
15                  while (input.read(buffer) != -1)
16              finally
17                  input.close()
18          catch (IOException e)
19              Log.e(TAG, "Error while downloading
```

```
                image.")
20          return NIL
21      @Override
22      function onPostExecute(Void v)
23          mDeviceBandwidthSampler.stopSampling()
24          if (mConnectionClass ==
             ConnectionQuality.UNKNOWN AND mTries <
             10)
25              mTries++
26              new DownloadImage().execute(mURL)
27          if (NOT
             mDeviceBandwidthSampler.isSampling())
28              Log.e(TAG, "Finish Check Connection")
29              mTries := 0
30              Log.e(TAG, "+" + mConnectionClass
                 Manager.getCurrentBandwidth
                 Quality().toString())
31              if (mConnectionClassManager.
                 getCurrentBandwidthQuality() !=
                 ConnectionQuality.UNKNOWN)
32                  Log.e(TAG, "OK")
```

Pseudocode 4.33 Kode program inisialisasi penerapan *Asynchronous Task* pada pengecekan buffer data

4.2.6.2 Kondisi Level Baterai Perangkat Bergerak

```
1   function getBatteryPercent()
2       IntentFilter ifilter := new IntentFilter
         (Intent.ACTION_BATTERY_CHANGED)
3       Intent batteryStatus := getBaseContext().
         registerReceiver(NIL, ifilter)
4       level := batteryStatus.getIntExtra
         (BatteryManager.EXTRA_LEVEL, -1)
5       return level
```

Pseudocode 4.34 memperlihatkan fungsi untuk mendapatkan level baterai dari perangkat *client*. Nilai level akan berubah sesuai kondisi level baterai perangkat *client* jika fungsi ini dipanggil saat adanya perubahan pada level baterai perangkat *client*.

```
1   function getBatteryPercent()
```

```
2       IntentFilter ifilter := new IntentFilter
         (Intent.ACTION_BATTERY_CHANGED)
3       Intent batteryStatus := getBaseContext().
         registerReceiver(NIL, ifilter)
4       level := batteryStatus.getIntExtra
         (BatteryManager.EXTRA_LEVEL, -1)
5       return level
```

Pseudocode* 4.34 Kode program mendapatkan nilai level baterai pada perangkat *client

4.2.6.3 Waktu Eksekusi Proses *Image Recognition*

Pseudocode 4.35 memperlihatkan fungsi untuk memulai perhitungan waktu saat dimulainya eksekusi beban kerja *image recognition* baik secara lokal maupun *offloading* yang dapat dilihat pada pemanggilan fungsi `startTimeTask()` pada *Pseudocode* 4.13.

```
1   function startTimeTask()
2           startTime := SystemClock.
             elapsedRealtime()
```

***Pseudocode* 4.35 Kode program perhitungan waktu dimulainya beban kerja dieksekusi**

Pseudocode 4.36 memperlihatkan fungsi untuk mengakhiri perhitungan saat selesainya eksekusi beban kerja *image recognition* secara *offloading* yang dapat dilihat pada pemanggilan fungsi `endTimeTaskOffload()` pada *Pseudocode* 4.12. Hasil waktu eksekusi secara *offloading* selanjutnya akan di rata – rata setiap dilakukannya eksekusi beban kerja secara *offloading* dengan tujuan untuk dipergunakan oleh *Decision Maker*.

```
1   function endTimeTaskOffload()
2         endTime :=
           SystemClock.elapsedRealtime()
3         elapsedMilliSeconds := endTime -
```

```
        startTime
4       elapsedSeconds := elapsedMilliSeconds
        / 1000.0
5       Log.i(TAG, "Time Elapsed Offload " +
        currentFileTrain + " : " +
        elapsedSeconds + " seconds")
6       TaskCountOffload++
7       rateExecutionTimeOffload :=
        ((rateExecutionTimeOffload *
        (TaskCountOffload - 1)) +
        elapsedSeconds) / TaskCountOffload
8       Log.i(TAG, "Time Elapsed Rate
        Offload: " +  rateExecutionTimeOffload + "
        seconds")
```

Pseudocode* 4.36 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara *offloading

Pseudocode 4.37 memperlihatkan fungsi untuk mengakhiri perhitungan saat selesainya eksekusi beban kerja *image recognition* secara lokal yang dapat dilihat pada pemanggilan fungsi `endTimeTaskOffload()` pada *Pseudocode* 4.37. Hasil waktu eksekusi secara lokal selanjutnya akan di rata – rata setiap dilakukannya eksekusi beban kerja secara lokal dengan tujuan untuk dipergunakan oleh *Decision Maker*.

```
1   function endTimeTaskLocal()
2         endTime :=
          SystemClock.elapsedRealtime()
3         elapsedMilliSeconds := endTime –
          startTime
4         elapsedSeconds := elapsedMilliSeconds /
          1000.0
          Log.i(TAG, "Time Elapsed Local " +
5         currentFileTrain + " : " +
6         elapsedSeconds + " seconds")
          TaskCountLocal++
          rateExecutionTimeLocal :=
          ((rateExecutionTimeLocal *
7         (TaskCountLocal - 1)) + elapsedSeconds)
           / TaskCountLocal
8         Log.i(TAG, "Time Elapsed Rate Local: " +
```

```
        rateExecutionTimeLocal + " seconds")
```

***Pseudocode* 4.37 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara lokal**

(Halaman ini sengaja dikosongkan)

BAB V
HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi penentuan keputusan eksekusi beban kerja pada *offloading computation framework* pada perangkat lunak Android. Hasil uji coba didapatkan dari implementasi pada Bab 4 dengan skenario yang berbeda. Bab ini berisikan pembahasan mengenai lingkungan pengujian, data pengujian, dan uji kinerja.

5.1 Lingkungan Pengujian

Lingkungan pengujian pada uji coba permasalahan penentuan keputusan eksekusi beban kerja oleh *framework* melalui faktor – faktor penentu metode *offloading* menggunakan spesifikasi keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Lingkungan Pengujian

Privilege	Perangkat	Jenis	Spesifikasi
Client	Perangkat Keras	Prosesor	Intel(R) Atom(TM) CPU Z2580 2,00 GHz
		Memori	2 GB DDR3
	Perangkat Lunak	Sistem Operasi	Android 4.4.2 KitKat
Server	Perangkat Keras	Prosesor	Intel(R) Core(TM) i7-2.5 GHz
		Memori	8 GB 800 MHz DDR3

	Perangkat Lunak	Sistem Operasi	Windows 10 dan Ubuntu 16.04
		Perangkat Pengembang	Android Studio IDE dan Netbeans IDE 8.2

5.2 Data Pengujian

Subbab ini menjelaskan mengenai data yang digunakan pada uji coba. Seperti yang telah dijelaskan sebelumnya, terdiri dari dua jenis data masukan. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pengambilan gambar saat menjalankan perangkat lunak Android. Data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun.

Kedua data tersebut akan diolah menggunakan kompresi pada untuk citra datatest dan pewarnaan *grayscale* untuk kedua citra datatest dan dataset sehingga menghasilkan citra dengan matriks yang lebih baik jika akan diambil masing – masing *descriptor* matriksnya. Data hasil *preprossesing* tersebut akan dihitung kemiripannya menggunakan metode SURF untuk proses *image recognition*. Selanjutnya dilakukan pengecekan terhadap perhitungan kemiripan dengan melakukan seleksi jarak kedekatan titik fitur yang didapatkan sesuai kriteria yang ditentukan dengan dasar nilai terkecil (mendekati nol) memiliki kemiripan yang lebih besar.

5.3 *Preprocessing* citra

Preprocessing citra yang digunakan dalam skenario uji coba adalah pada tahap kompresi citra dan pewarnaan *grayscale* yaitu

tahap untuk menghasilkan citra dengan matriks yang lebih baik jika akan diambil masing – masing *descriptor* matriksnya untuk tahap *image recognition*.

5.3.1 Skenario Uji Coba

Sebelum melakukan uji coba, perlu ditentukan skenario yang akan digunakan dalam uji coba. Melalui skenario ini, perangkat akan diuji apakah sudah berjalan dengan benar dan bagaimana performa pada masing-masing skenario. Dan membandingkan skenario manakah yang memiliki hasil lebih baik.

Di dalam skenario uji coba, terdapat kondisi koneksi internet dan level baterai pada perangkat *client* yang digunakan dalam menentukan metode pengeksekusian beban kerja. Kondisi koneksi internet dan level baterai akan dikategorikan ke dalam syarat tertentu untuk keperluan uji coba.

Kondisi koneksi internet akan dikategorikan menjadi 3 kategori yang berbeda yaitu Koneksi A, Koneksi B, dan Koneksi C. Pembagian kategori ini didasarkan pada nilai kualitas koneksi internet perangkat *client* seperti yang telah dijelaskan pada Bab 3.4.1. Pada Koneksi C, untuk mendapatkan nilai kualitas *unkown* pada uji coba dilakukan dua kali penonaktifan koneksi internet pada saat perangkat *client* melakukan proses *image recognition*. Untuk lebih jelasnya mengenai pembagian kategori kondisi koneksi internet akan dijelaskan pada Tabel 5.2

Tabel 5.2 Pembagian kategori koneksi internet

Kategori	Nilai Kualitas
Koneksi A	*Excellent*, *Good* dan *Moderate*
Koneksi B	*Poor*
Koneksi C	*Unknown*

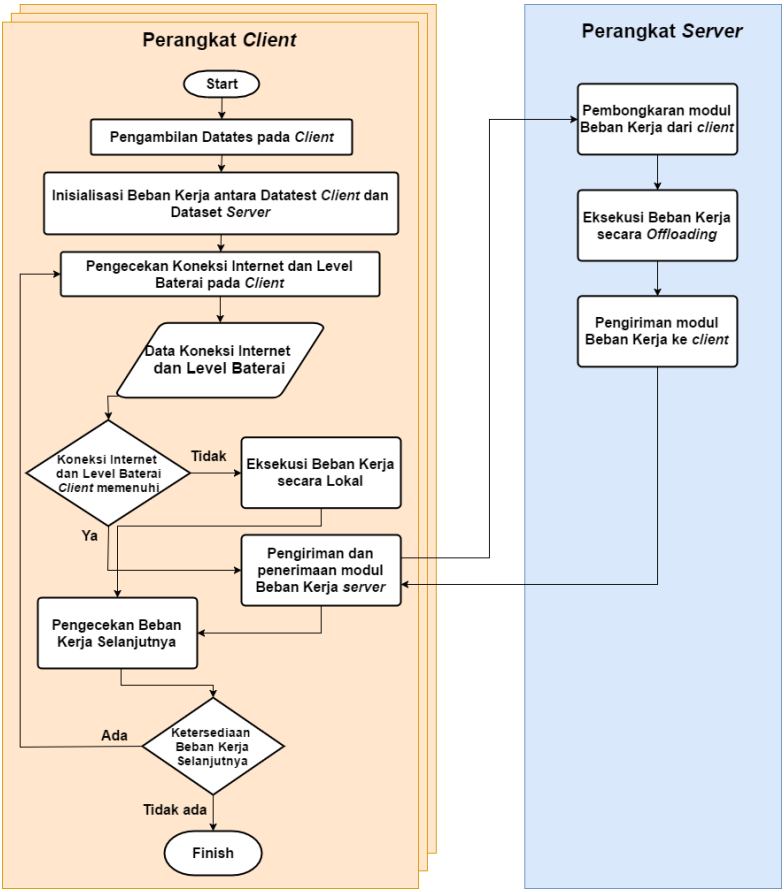
Kondisi level baterai akan dikategorikan menjadi 2 kategori yang berbeda yaitu Baterai A dan Baterai B. Pembagian kategori ini didasarkan pada nilai level baterai perangkat *client*

seperti yang telah dijelaskan pada Bab 3.4.2. Untuk lebih jelasnya mengenai pembagian kategori kondisi level baterai akan dijelaskan pada Tabel 5.3.

Tabel 5.3 Pembagian kategori level baterai

Kategori	Nilai level
Baterai A	21% - 100%
Baterai B	0% - 20%

Dalam menjalankan eksekusi beban kerja proses *image recognition*, *framework* akan melakukan urutan langkah – langkah kerja yang telah dijelaskan sebelumnya pada Bab 3.5. Pada skenario uji coba kali ini akan digabungkan antara langkah – langkah kerja dari *framework* dengan kondisi koneksi internet dan level baterai seperti yang telah dijelaskan sebelumnya. Pada Gambar 5.1 akan dijelaskan bagaimana langkah – langkah kerja *framework* dan kondisi koneksi internet dan level baterai dilakukan.



Gambar 5.1 Bagan alur kerja skenario uji coba

Terdapat 6 macam skenario uji coba, yaitu:

1. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan level Baterai A.

2. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A.
3. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B.
4. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B.
5. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A.
6. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B.

5.3.2 Skenario Uji Coba 1

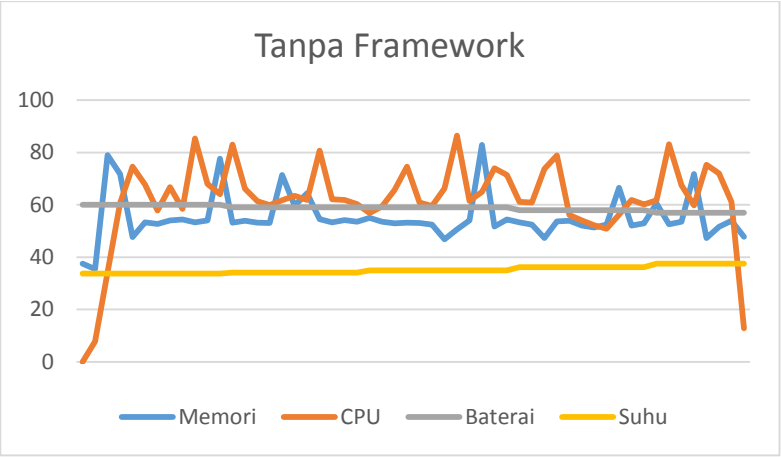
Skenario uji coba 1 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image*

recognition. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 1 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.6. Hasil waktu eksekusi pada uji coba 1 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.7.

Tabel 5.4 Performa tanpa penerapan *framework* pada perangkat Android

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	37.47	1.89	0	0	60	33.7
2	35.5	1.79	7.84	0	60	33.7
3	78.97	3.98	34.25	0	60	33.7
4	71.71	3.61	60.61	0	60	33.7
5	47.65	2.4	74.58	0	60	33.7
6	53.33	2.69	67.71	0	60	33.7
7	52.65	2.65	57.85	0	60	33.7
8	54.04	2.72	66.8	0	60	33.7
9	54.41	2.74	58.37	0	60	33.7
10	53.33	2.69	85.36	0	60	33.7
11	54.05	2.72	67.94	0	60	33.7
12	77.6	3.91	64.06	0	60	33.7
13	53.2	2.68	82.94	0	59	34.1
14	53.86	2.71	66.14	0	59	34.1
15	53.22	2.68	61.4	0	59	34.1
16	53.08	2.67	59.84	0	59	34.1
17	71.44	3.6	61.77	0	59	34.1
18	59.66	3.01	63.41	0	59	34.1
19	64.45	3.25	61.83	0	59	34.1
20	54.52	2.75	80.63	0	59	34.1
21	53.31	2.69	62.09	0	59	34.1
22	54.2	2.73	61.88	0	59	34.1
23	53.59	2.7	60.3	0	59	34.1

24	55.05	2.77	56.87	0	59	34.9
25	53.6	2.7	59.44	0	59	34.9
26	52.98	2.67	65.62	0	59	34.9
27	53.18	2.68	74.61	0	59	34.9
28	53.09	2.68	60.91	0	59	34.9
29	52.45	2.64	59.58	0	59	34.9
30	46.83	2.36	66.31	0	59	34.9
31	50.66	2.55	86.47	0	59	34.9
32	53.97	2.72	61.49	0	59	34.9
33	82.91	4.18	64.88	0	59	34.9
34	51.71	2.61	73.9	0	59	34.9
35	54.38	2.74	71.35	0	59	34.9
36	53.35	2.69	61.14	0	58	36.2
37	52.44	2.64	60.86	0	58	36.2
38	47.31	2.38	73.74	0	58	36.2
39	53.62	2.7	78.8	0	58	36.2
40	53.93	2.72	56.12	0	58	36.2
41	52.13	2.63	54	0	58	36.2
42	51.38	2.59	52.15	0	58	36.2
43	52.38	2.64	50.9	0	58	36.2
44	66.46	3.35	56.17	0	58	36.2
45	52.08	2.62	61.87	0	58	36.2
46	52.98	2.67	60.19	0	58	36.2
47	60.49	3.05	61.79	0	57	37.5
48	52.58	2.65	83.17	0	57	37.5
49	53.58	2.7	67.52	0	57	37.5
50	71.75	3.62	59.81	0	57	37.5
51	47.26	2.38	75.3	0	57	37.5
52	51.56	2.6	71.96	0	57	37.5
53	53.93	2.72	61.11	0	57	37.5
54	47.75	2.41	12.86	0	57	37.5
Rata2	55.24	2.784	61.639	0	-	-



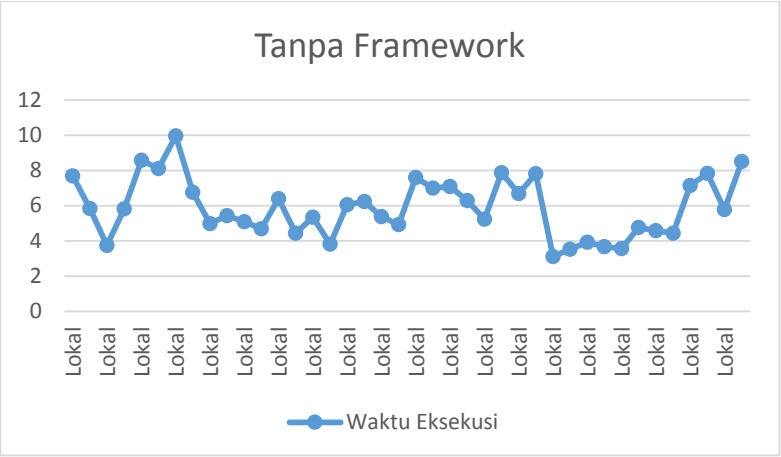
Gambar 5.2 Grafik performa tanpa penerapan *framework* pada perangkat Android

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* mempunyai penggunaan memori rata – rata 55.24 MB atau 2.784% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 61.639% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 0 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 3.8 C.

Tabel 5.5 Waktu eksekusi tanpa penerapan *framework* pada perangkat lunak *image recognition*

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	7.695	-
2	Lokal	5.834	-
3	Lokal	3.739	-
4	Lokal	5.822	-

5	Lokal	8.575	-
6	Lokal	8.098	-
7	Lokal	9.957	-
8	Lokal	6.762	-
9	Lokal	4.974	-
10	Lokal	5.435	-
11	Lokal	5.094	-
12	Lokal	4.684	-
13	Lokal	6.398	-
14	Lokal	4.424	-
15	Lokal	5.338	-
16	Lokal	3.809	-
17	Lokal	6.05	-
18	Lokal	6.242	-
19	Lokal	5.383	-
20	Lokal	4.923	-
21	Lokal	7.603	-
22	Lokal	6.998	-
23	Lokal	7.092	-
24	Lokal	6.279	-
25	Lokal	5.238	-
26	Lokal	7.872	-
27	Lokal	6.686	-
28	Lokal	7.818	-
29	Lokal	3.106	-
30	Lokal	3.517	-
31	Lokal	3.93	-
32	Lokal	3.666	-
33	Lokal	3.563	-
34	Lokal	4.756	-
35	Lokal	4.575	-
36	Lokal	4.429	-
37	Lokal	7.137	-
38	Lokal	7.838	-
39	Lokal	5.776	-
40	Lokal	8.5	-
Rata - rata	=	5.89	



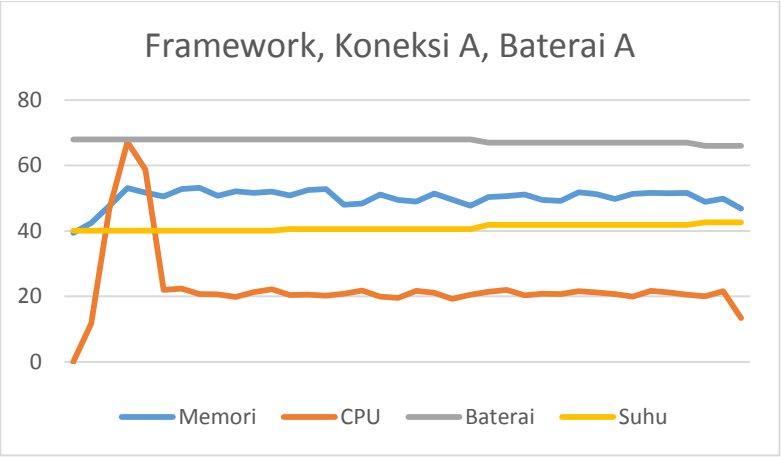
Gambar 5.3 Grafik waktu eksekusi tanpa penerapan *framework* pada perangkat lunak *image recognition*

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dilakukan secara lokal mencapai 100% dan mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 5.89 detik.

Tabel 5.6 Performa dengan penerapan *framework* pada Koneksi A dan Baterai A

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	39.45	1.99	0	0	68	40.1
2	42.46	2.14	11.73	412	68	40.1
3	47.62	2.4	46.46	947	68	40.1
4	53.1	2.68	67.22	947	68	40.1
5	51.78	2.61	58.76	1309	68	40.1
6	50.56	2.55	22	2332	68	40.1

7	52.82	2.66	22.39	3340	68	40.1
8	53.24	2.68	20.77	4158	68	40.1
9	50.77	2.56	20.64	4923	68	40.1
10	52.11	2.63	19.87	5506	68	40.1
11	51.65	2.6	21.31	6223	68	40.1
12	52.07	2.62	22.2	6864	68	40.1
13	50.87	2.56	20.4	7700	68	40.6
14	52.48	2.64	20.5	8413	68	40.6
15	52.79	2.66	20.25	8995	68	40.6
16	48	2.42	20.85	9577	68	40.6
17	48.36	2.44	21.85	10552	68	40.6
18	51.1	2.58	19.96	11318	68	40.6
19	49.45	2.49	19.53	11901	68	40.6
20	49.04	2.47	21.74	12616	68	40.6
21	51.43	2.59	21.14	13287	68	40.6
22	49.55	2.5	19.31	14146	68	40.6
23	47.73	2.41	20.58	14806	68	40.6
24	50.39	2.54	21.41	15388	67	41.9
25	50.67	2.55	22.04	16197	67	41.9
26	51.12	2.58	20.31	17024	67	41.9
27	49.51	2.5	20.87	17710	67	41.9
28	49.23	2.48	20.73	18426	67	41.9
29	51.79	2.61	21.58	19007	67	41.9
30	51.2	2.58	21.18	19875	67	41.9
31	49.73	2.51	20.71	20614	67	41.9
32	51.31	2.59	19.98	21335	67	41.9
33	51.63	2.6	21.73	21917	67	41.9
34	51.56	2.6	21.19	22785	67	41.9
35	51.59	2.6	20.56	23525	67	41.9
36	48.91	2.46	20.06	24107	66	42.6
37	49.87	2.51	21.58	24822	66	42.6
38	46.84	2.36	13.41	24822	66	42.6
Rata2	50.09	2.525	43.39	12311.2	-	-



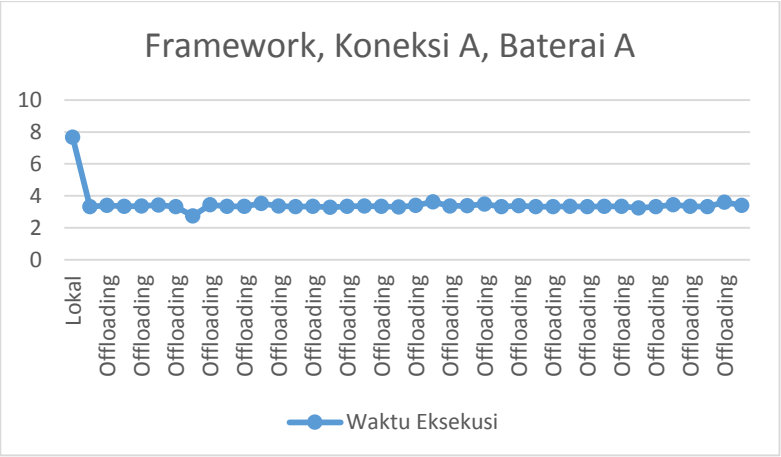
Gambar 5.4 Grafik performa dengan penerapan *framework* pada Koneksi A dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A mempunyai penggunaan memori rata – rata 50.09 MB atau 2.525% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 43.39% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 12311.2 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 2.5 C.

Tabel 5.7 Waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai A

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	7.666	*GOOD*
2	Offloading	3.319	*GOOD*
3	Offloading	3.395	*GOOD*

4	Offloading	3.334	*EXCELLENT*
5	Offloading	3.355	*EXCELLENT*
6	Offloading	3.418	*EXCELLENT*
7	Offloading	3.314	*GOOD*
8	Offloading	2.742	*GOOD*
9	Offloading	3.442	*GOOD*
10	Offloading	3.347	*GOOD*
11	Offloading	3.348	*EXCELLENT*
12	Offloading	3.522	*EXCELLENT*
13	Offloading	3.36	*EXCELLENT*
14	Offloading	3.329	*EXCELLENT*
15	Offloading	3.337	*EXCELLENT*
16	Offloading	3.287	*EXCELLENT*
17	Offloading	3.346	*EXCELLENT*
18	Offloading	3.357	*EXCELLENT*
19	Offloading	3.349	*EXCELLENT*
20	Offloading	3.308	*EXCELLENT*
21	Offloading	3.396	*EXCELLENT*
22	Offloading	3.625	*EXCELLENT*
23	Offloading	3.351	*GOOD*
24	Offloading	3.39	*GOOD*
25	Offloading	3.476	*GOOD*
26	Offloading	3.321	*EXCELLENT*
27	Offloading	3.373	*GOOD*
28	Offloading	3.32	*GOOD*
29	Offloading	3.328	*GOOD*
30	Offloading	3.349	*GOOD*
31	Offloading	3.312	*GOOD*
32	Offloading	3.34	*GOOD*
33	Offloading	3.348	*GOOD*
34	Offloading	3.246	*GOOD*
35	Offloading	3.323	*GOOD*
36	Offloading	3.434	*GOOD*
37	Offloading	3.331	*GOOD*
38	Offloading	3.315	*GOOD*
39	Offloading	3.601	*GOOD*
40	Offloading	3.41	*GOOD*
Rata - rata	=	3.4616	



Gambar 5.5 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A dilakukan secara lokal mencapai 2.5% dan secara *offloading* mencapai 97.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 3.4616 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 50.09 MB atau 2.525%.

Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 43.39%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar 5.16 MB atau 0.259%, penggunaan CPU rata – rata sebesar 18.249% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 3.4616 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 2.4284 detik setiap eksekusi beban kerjanya.

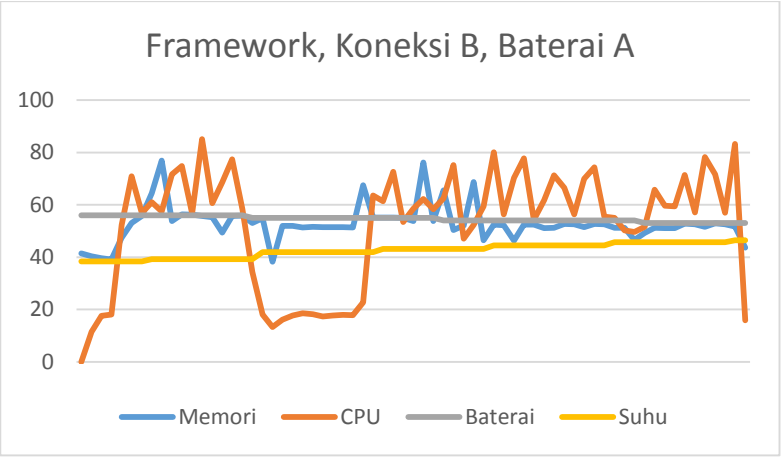
5.3.3 Skenario Uji Coba 2

Skenario uji coba 2 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 2 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.8. Hasil waktu eksekusi pada uji coba 2 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.9.

Tabel 5.8 Performa dengan penerapan *framework* pada Koneksi B dan Baterai A

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	41.47	2.09	0	0	56	38.4
2	40.38	2.04	11.51	43	56	38.4
3	39.55	1.99	17.62	147	56	38.4
4	39.12	1.97	18.15	249	56	38.4
5	47.4	2.39	52.27	295	56	38.4
6	53.02	2.67	70.86	295	56	38.4
7	55.72	2.81	56.8	341	56	38.4
8	64.38	3.24	61.01	477	56	39.2
9	76.91	3.88	57.47	592	56	39.2
10	53.8	2.71	71.68	712	56	39.2
11	56.37	2.84	74.81	841	56	39.2
12	56.22	2.83	56.73	911	56	39.2
13	55.77	2.81	85.12	1016	56	39.2
14	55.28	2.79	60.58	1128	56	39.2
15	49.39	2.49	68.54	1242	56	39.2
16	55.73	2.81	77.42	1344	56	39.2
17	55.87	2.82	58.36	1470	56	39.2
18	53.07	2.67	34.34	1572	55	39.2
19	54.83	2.76	18.05	1690	55	41.9
20	38.27	1.93	13.27	1817	55	41.9
21	51.97	2.62	16.14	1931	55	41.9
22	51.97	2.62	17.69	2016	55	41.9
23	51.35	2.59	18.56	2135	55	41.9
24	51.54	2.6	18.2	2253	55	41.9
25	51.41	2.59	17.35	2305	55	41.9
26	51.44	2.59	17.7	2386	55	41.9
27	51.41	2.59	18.02	2506	55	41.9
28	51.34	2.59	17.8	2637	55	41.9
29	67.53	3.4	22.74	2727	55	41.9
30	55.13	2.78	63.58	2736	55	41.9
31	55.14	2.78	61.31	2857	55	43.1
32	55.14	2.78	72.67	2967	55	43.1
33	55.01	2.77	53.4	3065	55	43.1
34	53.73	2.71	58.58	3115	55	43.1

35	76.1	3.83	62.21	3156	55	43.1
36	53.77	2.71	58.26	3229	55	43.1
37	65.46	3.3	62.49	3317	54	43.1
38	50.41	2.54	75.23	3414	54	43.1
39	52.05	2.62	47.01	3484	54	43.1
40	68.72	3.46	52.29	3579	54	43.1
41	46.44	2.34	59.44	3701	54	43.1
42	52.48	2.64	80.06	3828	54	44.5
43	52.24	2.63	56.38	3950	54	44.5
44	46.3	2.33	70.34	4023	54	44.5
45	52.43	2.64	77.8	4143	54	44.5
46	52.45	2.64	54.26	4255	54	44.5
47	51.13	2.58	61.79	4354	54	44.5
48	51.18	2.58	71.22	4474	54	44.5
49	52.74	2.66	66.54	4591	54	44.5
50	52.51	2.65	56.34	4713	54	44.5
51	51.45	2.59	69.91	4812	54	44.5
52	52.72	2.66	74.28	4913	54	44.5
53	52.55	2.65	55.54	4974	54	44.5
54	51.23	2.58	55	4983	54	45.7
55	51.25	2.58	50.2	5072	54	45.7
56	46.62	2.35	49.68	5148	54	45.7
57	49.27	2.48	51.55	5242	53	45.7
58	51.25	2.58	65.75	5314	53	45.7
59	51.1	2.58	59.6	5394	53	45.7
60	51.1	2.58	59.36	5476	53	45.7
61	52.77	2.66	71.34	5584	53	45.7
62	52.6	2.65	57.03	5680	53	45.7
63	51.56	2.6	78.23	5811	53	45.7
64	52.93	2.67	71.67	5941	53	45.7
65	52.62	2.65	56.92	6069	53	45.7
66	51.65	2.6	83.3	6172	53	46.5
67	43.62	2.2	15.89	6231	53	46.5
Rata2	52.826	2.662	51.869	3087.24	-	-



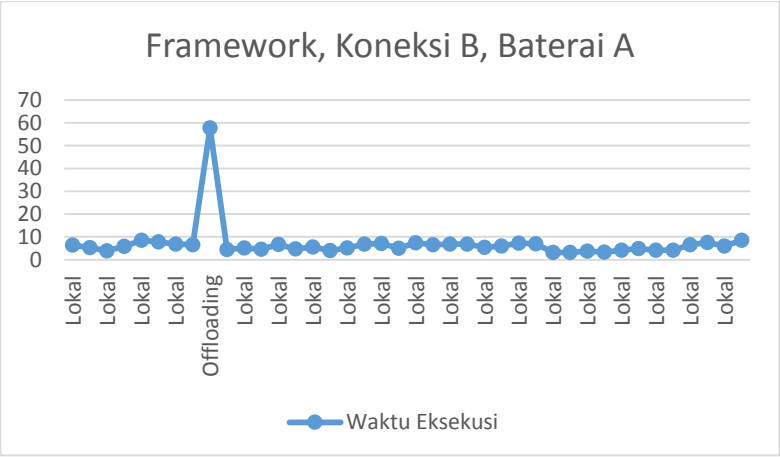
Gambar 5.6 Grafik performa dengan penerapan *framework* pada Koneksi B dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A. mempunyai penggunaan memori rata – rata 52.826 MB atau 2.662% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 51.869% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 3087.24 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 8.1 C.

Tabel 5.9 Waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai A

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	6.438	*POOR*
2	Lokal	5.211	*POOR*
3	Lokal	3.806	*POOR*
4	Lokal	5.894	*POOR*
5	Lokal	8.506	*POOR*

6	Lokal	7.749	*POOR*
7	Lokal	6.777	*POOR*
8	Lokal	6.606	*POOR*
9	Offloading	57.738	*MODERATE*
10	Lokal	4.428	*POOR*
11	Lokal	5.143	*POOR*
12	Lokal	4.626	*POOR*
13	Lokal	6.677	*POOR*
14	Lokal	4.729	*POOR*
15	Lokal	5.504	*POOR*
16	Lokal	3.953	*POOR*
17	Lokal	5.173	*POOR*
18	Lokal	6.777	*POOR*
19	Lokal	7.057	*POOR*
20	Lokal	5.052	*POOR*
21	Lokal	7.404	*POOR*
22	Lokal	6.57	*POOR*
23	Lokal	6.821	*POOR*
24	Lokal	6.889	*POOR*
25	Lokal	5.407	*POOR*
26	Lokal	6.009	*MODERATE*
27	Lokal	7.263	*MODERATE*
28	Lokal	6.958	*MODERATE*
29	Lokal	3.214	*POOR*
30	Lokal	3.188	*POOR*
31	Lokal	3.696	*POOR*
32	Lokal	3.37	*POOR*
33	Lokal	4.158	*POOR*
34	Lokal	4.89	*POOR*
35	Lokal	4.167	*POOR*
36	Lokal	4.216	*POOR*
37	Lokal	6.547	*POOR*
38	Lokal	7.593	*POOR*
39	Lokal	6.048	*POOR*
40	Lokal	8.483	*POOR*
Rata - rata	=	7.018	



Gambar 5.7 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A dilakukan secara lokal mencapai 97.5% dan secara *offloading* mencapai 2.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 7.018 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 52.826 MB atau 2.662%. Penggunaan CPU rata – rata pada perangkat *client* pada metode

eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 51.869%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar 2.414 MB atau 0.122%, penggunaan CPU rata – rata sebesar 9.77% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 7.018 detik sehingga dihasilkan penghematan waktu eksekusi mencapai -1.128 detik setiap eksekusi beban kerjanya.

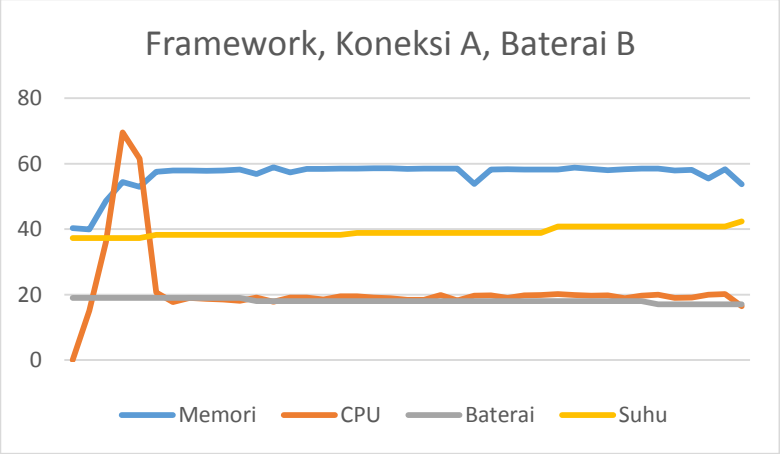
5.3.4 Skenario Uji Coba 3

Skenario uji coba 3 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 3 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.10. Hasil waktu eksekusi pada uji coba 3 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.11.

Tabel 5.10 Performa dengan penerapan *framework* pada Koneksi A dan Baterai B

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	40.28	2.03	0	140	19	37.3
2	39.97	2.01	15.12	2280	19	37.3
3	48.65	2.45	36.13	2957	19	37.3
4	54.41	2.74	69.55	2957	19	37.3
5	52.94	2.67	61.53	3229	19	37.3
6	57.52	2.9	20.68	4517	19	38.3
7	57.88	2.92	17.78	5255	19	38.3
8	57.94	2.92	18.97	5977	19	38.3
9	57.85	2.92	18.72	6680	19	38.3
10	57.96	2.92	18.53	7342	19	38.3
11	58.23	2.93	18.13	7970	19	38.3
12	56.86	2.87	19.13	8739	18	38.3
13	58.92	2.97	17.86	9361	18	38.3
14	57.29	2.89	19.13	10036	18	38.3
15	58.4	2.94	19.15	11010	18	38.3
16	58.42	2.94	18.47	11536	18	38.3
17	58.53	2.95	19.45	12325	18	38.3
18	58.52	2.95	19.51	13105	18	38.9
19	58.56	2.95	19.14	13757	18	38.9
20	58.56	2.95	18.94	14465	18	38.9
21	58.41	2.94	18.45	15261	18	38.9
22	58.48	2.95	18.42	16143	18	38.9
23	58.51	2.95	19.89	16721	18	38.9
24	58.5	2.95	18.18	17414	18	38.9
25	53.83	2.71	19.68	18209	18	38.9
26	58.25	2.94	19.77	18706	18	38.9
27	58.27	2.94	19.05	19346	18	38.9
28	58.26	2.94	19.75	19991	18	38.9
29	58.26	2.94	19.86	20740	18	38.9
30	58.25	2.94	20.18	21427	18	40.8
31	58.82	2.96	19.93	22123	18	40.8
32	58.42	2.94	19.67	22793	18	40.8
33	58.02	2.92	19.79	23404	18	40.8
34	58.35	2.94	18.92	24153	18	40.8

35	58.48	2.95	19.69	24874	18	40.8
36	58.5	2.95	19.95	25519	17	40.8
37	57.95	2.92	18.97	26262	17	40.8
38	58.12	2.93	19.07	26864	17	40.8
39	55.44	2.79	19.95	27539	17	40.8
40	58.35	2.94	20.13	28161	17	40.8
41	53.74	2.71	16.45	28297	17	42.4
Rata2	61.25	2.85	21.26	15063.049	-	-



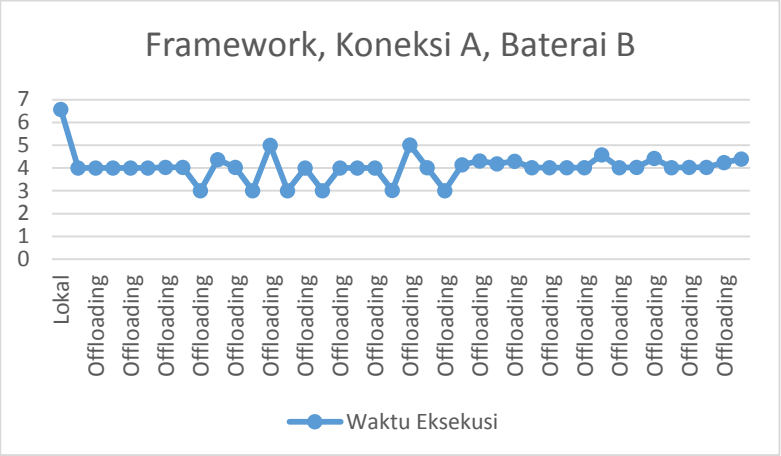
Gambar 5.8 Grafik performa dengan penerapan *framework* pada Koneksi A dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 61.25 MB atau 2.85% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 21.869% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 15063.049 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 5.1 C.

Tabel 5.11 Waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai B

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	6.568	*EXCELLENT*
2	Offloading	4.002	*EXCELLENT*
3	Offloading	4.005	*EXCELLENT*
4	Offloading	4.003	*EXCELLENT*
5	Offloading	4.008	*EXCELLENT*
6	Offloading	4.006	*EXCELLENT*
7	Offloading	4.025	*EXCELLENT*
8	Offloading	4.03	*EXCELLENT*
9	Offloading	3.005	*EXCELLENT*
10	Offloading	4.37	*EXCELLENT*
11	Offloading	4.033	*EXCELLENT*
12	Offloading	3.001	*EXCELLENT*
13	Offloading	5.003	*EXCELLENT*
14	Offloading	3.004	*EXCELLENT*
15	Offloading	4.006	*EXCELLENT*
16	Offloading	3.003	*EXCELLENT*
17	Offloading	4.002	*EXCELLENT*
18	Offloading	4.008	*EXCELLENT*
19	Offloading	4.003	*EXCELLENT*
20	Offloading	3.016	*EXCELLENT*
21	Offloading	5.01	*EXCELLENT*
22	Offloading	4.014	*EXCELLENT*
23	Offloading	3.009	*GOOD*
24	Offloading	4.137	*GOOD*
25	Offloading	4.314	*GOOD*
26	Offloading	4.188	*GOOD*
27	Offloading	4.296	*GOOD*
28	Offloading	4.024	*GOOD*
29	Offloading	4.017	*GOOD*
30	Offloading	4.016	*GOOD*
31	Offloading	4.013	*GOOD*
32	Offloading	4.576	*GOOD*
33	Offloading	4.013	*GOOD*
34	Offloading	4.037	*EXCELLENT*

35	Offloading	4.429	*EXCELLENT*
36	Offloading	4.024	*EXCELLENT*
37	Offloading	4.026	*EXCELLENT*
38	Offloading	4.027	*GOOD*
39	Offloading	4.238	*GOOD*
40	Offloading	4.402	*GOOD*
Rata - rata	=	4.048	



Gambar 5.9 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B dilakukan secara lokal mencapai 2.5% dan secara *offloading* mencapai 97.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.048 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja

proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 61.25 MB atau 2.85%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 21.26%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -6.01 MB atau -0.066%, penggunaan CPU rata – rata sebesar 40.379% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.048 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.842 detik setiap eksekusi beban kerjanya.

5.3.5 Skenario Uji Coba 4

Skenario uji coba 4 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang

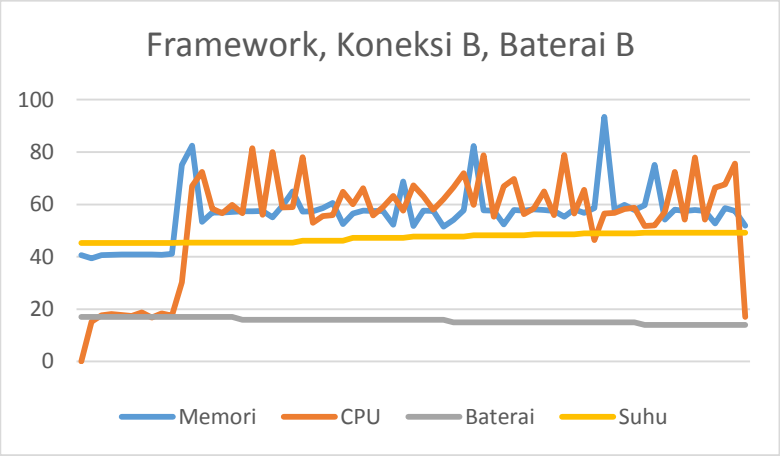
terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 4 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.12. Hasil waktu eksekusi pada uji coba 4 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.13.

Tabel 5.12 Performa dengan penerapan *framework* pada Koneksi B dan Baterai B

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	40.62	2.05	0	3	17	45.3
2	39.37	1.98	15.26	7	17	45.3
3	40.7	2.05	17.66	54	17	45.3
4	40.73	2.05	18.17	80	17	45.3
5	40.88	2.06	17.83	85	17	45.3
6	40.88	2.06	17.36	119	17	45.3
7	40.9	2.06	18.71	155	17	45.3
8	40.93	2.06	16.8	191	17	45.3
9	40.78	2.06	18.36	234	17	45.3
10	41.11	2.07	17.53	271	17	45.3
11	75.11	3.79	30.22	300	17	45.4
12	82.47	4.16	66.99	300	17	45.4
13	53.36	2.69	72.36	301	17	45.4
14	56.87	2.87	58.41	353	17	45.4
15	56.95	2.87	56.65	392	17	45.4
16	57.18	2.88	59.78	423	17	45.4
17	57.38	2.89	56.7	443	16	45.4
18	57.33	2.89	81.5	462	16	45.4
19	57.46	2.9	56.1	492	16	45.4
20	55.11	2.78	79.98	510	16	45.4
21	59.41	2.99	58.81	522	16	45.4
22	64.99	3.27	58.92	544	16	45.4
23	57.29	2.89	78.05	563	16	46.1
24	57.42	2.89	52.94	584	16	46.1
25	58.62	2.95	55.59	596	16	46.1

26	60.59	3.05	55.97	608	16	46.1
27	52.5	2.65	64.79	618	16	46.1
28	56.5	2.85	60.13	624	16	47.2
29	57.59	2.9	66.14	632	16	47.2
30	57.49	2.9	55.76	641	16	47.2
31	57.49	2.9	59.01	656	16	47.2
32	52.3	2.64	63.3	686	16	47.2
33	68.72	3.46	57.61	701	16	47.2
34	51.72	2.61	67.33	715	16	47.7
35	57.57	2.9	63.09	740	16	47.7
36	57.56	2.9	58.05	763	16	47.7
37	51.49	2.59	62.05	785	16	47.7
38	54.08	2.73	66.54	797	15	47.7
39	57.8	2.91	71.97	806	15	47.7
40	82.26	4.15	59.83	818	15	48.2
41	57.75	2.91	78.74	827	15	48.2
42	57.69	2.91	55.26	836	15	48.2
43	52.42	2.64	66.97	853	15	48.2
44	57.82	2.91	69.71	874	15	48.2
45	57.66	2.91	56.32	891	15	48.2
46	58.07	2.93	58.49	907	15	48.6
47	57.91	2.92	64.95	923	15	48.6
48	57.49	2.9	55.88	956	15	48.6
49	55.28	2.79	78.93	983	15	48.6
50	58.16	2.93	56.5	1017	15	48.6
51	56.72	2.86	65.57	1048	15	48.9
52	58.57	2.95	46.36	1078	15	48.9
53	93.48	4.71	56.5	1090	15	48.9
54	57.93	2.92	56.8	1115	15	48.9
55	59.79	3.01	58.18	1149	15	48.9
56	57.92	2.92	58.73	1203	15	48.9
57	59.68	3.01	51.76	1266	14	49.2
58	75.11	3.78	51.96	1316	14	49.2
59	54.2	2.73	57.12	1360	14	49.2
60	58.01	2.92	72.4	1435	14	49.2
61	57.54	2.9	54.22	1494	14	49.2
62	57.83	2.91	77.9	1530	14	49.2
63	57.53	2.9	54.19	1561	14	49.2
64	52.78	2.66	66.44	1592	14	49.2

65	58.6	2.95	67.65	1625	14	49.2
66	57.48	2.9	75.58	1663	14	49.2
67	51.94	2.62	17.04	1693	14	49.2
Rata2	56.49	2.847	54.364	759.089	-	-



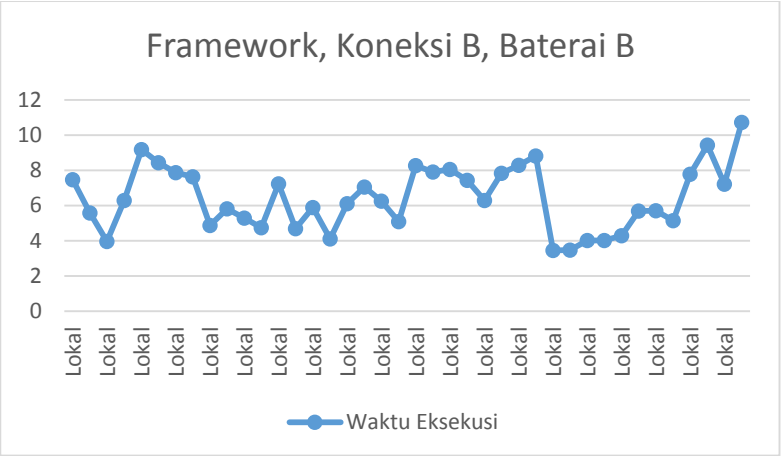
Gambar 5.10 Grafik performa dengan penerapan *framework* pada Koneksi B dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 56.49 MB atau 2.847% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 54.364% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 759.089 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 3.9 C.

Tabel 5.13 Waktu eksekusi dengan penerapan *framework* pada Koneksi B Baterai B

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	7.471	*POOR*
2	Lokal	5.582	*POOR*
3	Lokal	3.968	*POOR*
4	Lokal	6.289	*POOR*
5	Lokal	9.186	*POOR*
6	Lokal	8.447	*POOR*
7	Lokal	7.875	*POOR*
8	Lokal	7.637	*POOR*
9	Lokal	4.868	*POOR*
10	Lokal	5.823	*POOR*
11	Lokal	5.303	*POOR*
12	Lokal	4.753	*POOR*
13	Lokal	7.248	*POOR*
14	Lokal	4.702	*POOR*
15	Lokal	5.887	*POOR*
16	Lokal	4.116	*POOR*
17	Lokal	6.111	*POOR*
18	Lokal	7.066	*POOR*
19	Lokal	6.266	*POOR*
20	Lokal	5.093	*POOR*
21	Lokal	8.284	*POOR*
22	Lokal	7.905	*POOR*
23	Lokal	8.05	*POOR*
24	Lokal	7.44	*POOR*
25	Lokal	6.293	*POOR*
26	Lokal	7.84	*POOR*
27	Lokal	8.29	*POOR*
28	Lokal	8.818	*POOR*
29	Lokal	3.459	*POOR*
30	Lokal	3.48	*POOR*
31	Lokal	4.02	*POOR*
32	Lokal	4.024	*POOR*
33	Lokal	4.293	*POOR*
34	Lokal	5.7	*POOR*
35	Lokal	5.717	*POOR*

36	Lokal	5.145	*POOR*
37	Lokal	7.785	*POOR*
38	Lokal	9.441	*POOR*
39	Lokal	7.223	*POOR*
40	Lokal	10.741	*POOR*
Rata - rata	=	6.441	



Gambar 5.11 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B dilakukan secara lokal mencapai 100% dan secara *offloading* mencapai 0% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 6.441 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading*

computation framework pada kondisi Koneksi A dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 56.49 MB atau 2.847%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 54.364%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -1.25 MB atau -0.063%, penggunaan CPU rata – rata sebesar 7.275% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 6.411 detik sehingga dihasilkan penghematan waktu eksekusi mencapai -0.521 detik setiap eksekusi beban kerjanya.

5.3.6 Skenario Uji Coba 5

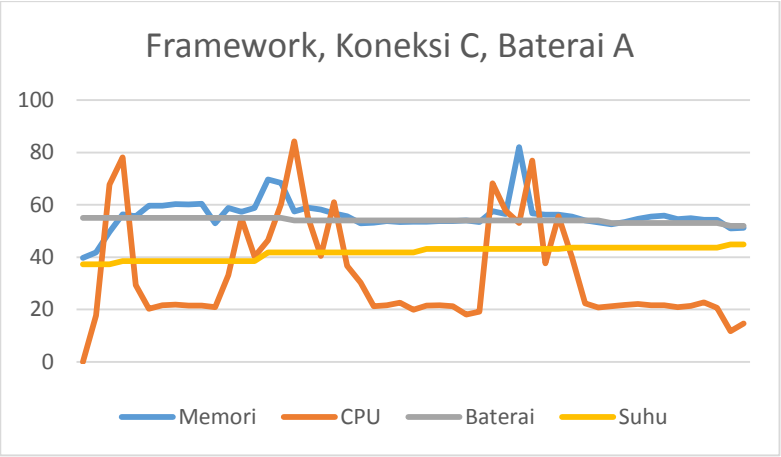
Skenario uji coba 5 adalah perhitungan penghematan daya dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai.

Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 5 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.14. Hasil waktu eksekusi pada uji coba 5 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.15.

Tabel 5.14 Performa dengan penerapan *framework* pada Koneksi C dan Baterai A

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	39.7	2	0	140	55	37.3
2	41.78	2.11	17.78	2823	55	37.3
3	49.69	2.5	67.7	2957	55	37.3
4	56.33	2.84	78.12	2957	55	38.5
5	55.6	2.8	29.34	3803	55	38.5
6	59.63	3	20.31	4518	55	38.5
7	59.66	3.01	21.66	5100	55	38.5
8	60.25	3.04	21.86	5696	55	38.5
9	60.19	3.03	21.57	6563	55	38.5
10	60.4	3.04	21.53	7206	55	38.5
11	52.87	2.66	20.93	7803	55	38.5
12	58.83	2.96	33.11	7958	55	38.5
13	57.38	2.89	54.89	7958	55	38.5
14	58.83	2.96	40.16	7958	55	38.5
15	69.65	3.51	46.43	7958	55	41.8
16	68.31	3.44	60.33	7958	55	41.8
17	57.49	2.9	84.19	7958	54	41.8
18	58.94	2.97	55.53	7958	54	41.8
19	58.14	2.93	40.44	7958	54	41.8
20	56.76	2.86	61.05	7958	54	41.8
21	55.65	2.8	36.7	8095	54	41.8
22	52.97	2.67	30.34	8232	54	41.8
23	53.18	2.68	21.28	8680	54	41.8
24	53.77	2.71	21.67	9526	54	41.8
25	53.46	2.69	22.65	10422	54	41.8
26	53.56	2.7	19.87	11006	54	41.8

27	53.52	2.7	21.5	11721	54	43.1
28	53.76	2.71	21.65	12302	54	43.1
29	53.78	2.71	21.23	12886	54	43.1
30	53.97	2.72	18.14	13308	54	43.1
31	53.39	2.69	19.24	13308	54	43.1
32	57.61	2.9	68.18	13308	54	43.1
33	56.49	2.85	57.74	13308	54	43.1
34	81.98	4.13	53.11	13308	54	43.1
35	56.71	2.86	76.89	13308	54	43.1
36	56.29	2.84	37.6	13891	54	43.1
37	56.17	2.83	55.61	13891	54	43.1
38	55.46	2.79	39.92	14027	54	43.6
39	54.09	2.73	22.4	14182	54	43.6
40	53.29	2.69	20.77	14881	54	43.6
41	52.61	2.65	21.32	15638	53	43.6
42	53.43	2.69	21.77	16242	53	43.6
43	54.58	2.75	22.07	17147	53	43.6
44	55.46	2.79	21.64	17962	53	43.6
45	55.85	2.81	21.6	18545	53	43.6
46	54.5	2.75	20.93	19261	53	43.6
47	54.94	2.77	21.36	19842	53	43.6
48	54.27	2.73	22.69	20711	53	43.6
49	54.25	2.73	20.61	21004	53	43.6
50	51.02	2.57	11.77	21004	52	44.9
51	51.18	2.58	14.63	21004	52	44.9
Rata2	55.823	2.812	33.052	1142.8	-	-



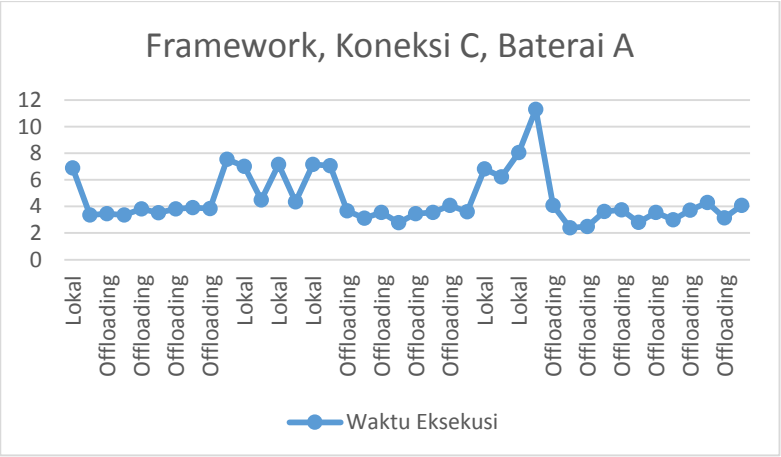
Gambar 5.12 Grafik performa dengan penerapan *framework* pada Koneksi C dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A. mempunyai penggunaan memori rata – rata 55.823 MB atau 2.812% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 33.052% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 1142.8 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 6.4 C.

Tabel 5.15 Waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai A

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	6.891	*EXCELLENT*
2	Offloading	3.353	*EXCELLENT*
3	Offloading	3.455	*EXCELLENT*
4	Offloading	3.369	*EXCELLENT*

5	Offloading	3.817	*EXCELLENT*
6	Offloading	3.52	*EXCELLENT*
7	Offloading	3.823	*EXCELLENT*
8	Offloading	3.91	*EXCELLENT*
9	Offloading	3.836	*EXCELLENT*
10	Lokal	7.552	*UNKNOWN*
11	Lokal	7.007	*UNKNOWN*
12	Lokal	4.482	*UNKNOWN*
13	Lokal	7.156	*UNKNOWN*
14	Lokal	4.344	*UNKNOWN*
15	Lokal	7.154	*UNKNOWN*
16	Lokal	7.063	*UNKNOWN*
17	Offloading	3.683	*EXCELLENT*
18	Offloading	3.128	*EXCELLENT*
19	Offloading	3.549	*GOOD*
20	Offloading	2.78	*GOOD*
21	Offloading	3.449	*GOOD*
22	Offloading	3.541	*GOOD*
23	Offloading	4.091	*EXCELLENT*
24	Offloading	3.599	*GOOD*
25	Lokal	6.816	*UNKNOWN*
26	Lokal	6.224	*UNKNOWN*
27	Lokal	8.045	*UNKNOWN*
28	Offloading	11.292	*MODERATE*
29	Offloading	4.087	*GOOD*
30	Offloading	2.408	*GOOD*
31	Offloading	2.49	*MODERATE*
32	Offloading	3.613	*MODERATE*
33	Offloading	3.742	*MODERATE*
34	Offloading	2.816	*GOOD*
35	Offloading	3.543	*GOOD*
36	Offloading	3.001	*GOOD*
37	Offloading	3.725	*GOOD*
38	Offloading	4.309	*GOOD*
39	Offloading	3.148	*GOOD*
40	Offloading	4.087	*EXCELLENT*
Rata - rata	=	4.547	



Gambar 5.13 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A dilakukan secara lokal mencapai 27.5% dan secara *offloading* mencapai 72.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.547 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 55.823 MB atau 2.812%.

Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 33.052%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -0.583 MB atau -0.028%, penggunaan CPU rata – rata sebesar 28.587% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.547 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.343 detik setiap eksekusi beban kerjanya.

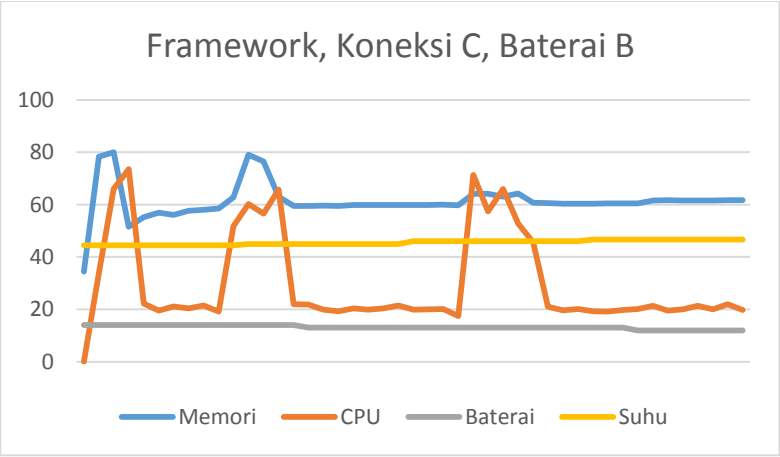
5.3.7 Skenario Uji Coba 6

Skenario uji coba 6 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 6 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.16. Hasil waktu eksekusi pada uji coba 6 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.17.

Tabel 5.16 Performa dengan penerapan *framework* pada Koneksi C dan Baterai B

No. Cek	Penggunaan Memori		Penggunaan CPU	Bandwidth Internet	Level Baterai	Suhu perangkat
(5s)	(MB)	(%)	(%)	(KB)	(%)	(C)
1	34.45	1.74	0	140	14	44.5
2	78.36	3.95	33.94	1752	14	44.5
3	79.99	4.03	65.93	1752	14	44.5
4	51.51	2.6	73.61	1753	14	44.5
5	55.17	2.78	22.2	3676	14	44.5
6	56.9	2.87	19.56	4219	14	44.5
7	56.11	2.83	21.1	4850	14	44.5
8	57.65	2.91	20.43	5632	14	44.5
9	58.06	2.93	21.43	6286	14	44.5
10	58.46	2.95	19.13	6929	14	44.5
11	62.78	3.16	51.78	6929	14	44.5
12	79.06	3.98	60.21	6929	14	45
13	76.51	3.86	56.62	6929	14	45
14	63.05	3.18	65.71	6930	14	45
15	59.46	3	21.99	7070	14	45
16	59.5	3	21.81	7772	13	45
17	59.67	3.01	19.84	8719	13	45
18	59.54	3	19.32	9442	13	45
19	59.87	3.02	20.35	10089	13	45
20	59.91	3.02	19.88	10714	13	45
21	59.88	3.02	20.42	11407	13	45
22	59.91	3.02	21.47	12188	13	45
23	59.91	3.02	19.86	12775	13	46.1
24	59.88	3.02	20.07	13442	13	46.1
25	59.95	3.02	20.19	14172	13	46.1
26	59.79	3.01	17.47	14444	13	46.1
27	64.19	3.23	71.3	14444	13	46.1
28	64.19	3.23	57.36	14444	13	46.1
29	63.01	3.18	65.96	14444	13	46.1
30	64.28	3.24	52.83	14444	13	46.1
31	60.69	3.06	45.54	14582	13	46.1
32	60.59	3.05	21	15264	13	46.1
33	60.31	3.04	19.62	16056	13	46.1
34	60.31	3.04	20.08	16725	13	46.1

35	60.34	3.04	19.25	17435	13	46.6
36	60.5	3.05	19.17	18088	13	46.6
37	60.48	3.05	19.79	18950	13	46.6
38	60.47	3.05	20.18	19705	12	46.6
39	61.62	3.11	21.36	20459	12	46.6
40	61.64	3.11	19.49	21379	12	46.6
41	61.61	3.1	19.96	22094	12	46.6
42	61.61	3.1	21.37	22840	12	46.6
43	61.62	3.11	19.97	23449	12	46.6
44	61.65	3.11	21.91	24152	12	46.6
45	61.64	3.11	19.71	24715	12	46.6
Rata2	61.246	3.088	30.01	12235.8	-	-



Gambar 5.14 Grafik performa dengan penerapan *framework* pada Koneksi C dan Baterai B

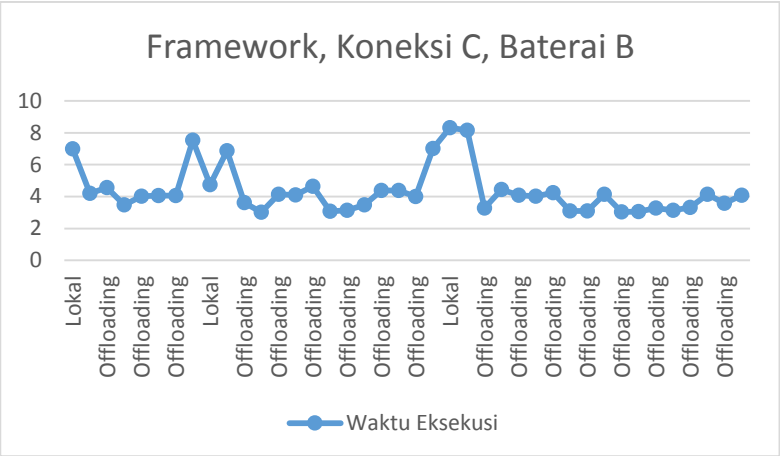
Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 61.246 MB atau 3.088% dari keseluruhan memori yang dimiliki perangkat *client*,

penggunaan CPU rata – rata 30.01% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 12235.8 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 2.1 C.

Tabel 5.17 Waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai B

No. Eksekusi	Metode	Waktu (s)	Kualitas Koneksi
1	Lokal	6.996	*EXCELLENT*
2	Offloading	4.201	*EXCELLENT*
3	Offloading	4.571	*EXCELLENT*
4	Offloading	3.476	*GOOD*
5	Offloading	4.011	*GOOD*
6	Offloading	4.065	*GOOD*
7	Offloading	4.065	*GOOD*
8	Lokal	7.537	*UNKNOWN*
9	Lokal	4.742	*UNKNOWN*
10	Lokal	6.86	*UNKNOWN*
11	Offloading	3.619	*EXCELLENT*
12	Offloading	3.007	*GOOD*
13	Offloading	4.132	*GOOD*
14	Offloading	4.093	*GOOD*
15	Offloading	4.635	*GOOD*
16	Offloading	3.073	*GOOD*
17	Offloading	3.145	*GOOD*
18	Offloading	3.472	*GOOD*
19	Offloading	4.374	*GOOD*
20	Offloading	4.38	*GOOD*
21	Offloading	4.008	*EXCELLENT*
22	Lokal	7.011	*UNKNOWN*
23	Lokal	8.32	*UNKNOWN*
24	Lokal	8.149	*UNKNOWN*
25	Offloading	3.284	*MODERATE*
26	Offloading	4.436	*GOOD*
27	Offloading	4.089	*GOOD*
28	Offloading	4.014	*GOOD*
29	Offloading	4.246	*EXCELLENT*
30	Offloading	3.105	*GOOD*

31	Offloading	3.095	*GOOD*
32	Offloading	4.137	*GOOD*
33	Offloading	3.038	*GOOD*
34	Offloading	3.052	*GOOD*
35	Offloading	3.276	*GOOD*
36	Offloading	3.143	*GOOD*
37	Offloading	3.318	*GOOD*
38	Offloading	4.139	*GOOD*
39	Offloading	3.569	*GOOD*
40	Offloading	4.071	*GOOD*
Rata - rata	=	4.349	



Gambar 5.15 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B dilakukan secara lokal mencapai 17.5% dan secara *offloading* mencapai 82.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.349 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 61.246 MB atau 3.088%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 30.01%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -6.006 MB atau -0.304%, penggunaan CPU rata – rata sebesar 31.629% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.349 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.541 detik setiap eksekusi beban kerjanya.

5.4 Evaluasi Umum Skenario Uji Coba

Hasil performa dan waktu eksekusi dari ke enam skenario dapat dilihat pada Tabel 5.18.

Tabel 5.18 Hasil performa dan waktu eksekusi dari 6 skenario uji coba

Uji Coba			1	2	3	4	5	6
Performa	Memori	(MB)	50.09	52.85	61.25	56.49	55.82	61.25
		(%)	2.525	2.662	2.85	2.847	2.812	3.088
	CPU	(%)	43.39	51.87	21.26	54.36	33.05	30.01
	Baterai	(%)	2	3	2	3	3	2
	Waktu	(s)	3.462	7.018	4.048	6.411	4.547	4.349
Penghematan	Memori	(MB)	5.16	2.41	-6.01	-1.25	-0.58	-6.01
		(%)	0.259	0.122	-0.07	-0.06	-0.03	-0.35
	CPU	(%)	18.249	9.77	40.379	7.275	28.59	31.629
	Baterai	(%)	1	0	1	0	0	1
	Waktu	(s)	2.428	-1.128	1.842	-0.521	1.343	1.541

Berdasarkan skenario uji coba ke satu yang telah dilakukan, dapat diketahui bahwa adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation*

framework yaitu pada memori terdapat peningkatan sebesar **5.16 MB** atau **0.259%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **9.249%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi rata - rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* sebesar **2.428 detik.**

Sedangkan berdasarkan skenario uji coba ke dua yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya peningkatan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu pada memori terdapat peningkatan sebesar **2.41 MB** atau **0.122%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **9.77%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui bahwa adanya tingkat penurunan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **-1.128 detik.**

Sedangkan berdasarkan skenario uji coba ke tiga yang telah dilakukan, dapat diketahui bahwa adanya adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat

peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu pada memori terdapat penurunan sebesar **-6.01 MB** atau **-0.07%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **40.379%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui bahwa adanya tingkat peningkatan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **1.842 detik**.

Sedangkan berdasarkan skenario uji coba ke empat yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu pada memori terdapat penurunan sebesar **-1.25 MB** atau **-0.06%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **7.275%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat penurunan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* sebesar **-0.521 detik.**

Sedangkan berdasarkan skenario uji coba ke lima yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu pada memori terdapat penurunan sebesar **-0.58 MB** atau **-0.03%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **28.59%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* sebesar **1.343 detik.**

Sedangkan berdasarkan skenario uji coba ke enam yang telah dilakukan, dapat diketahui bahwa adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation framework* yaitu pada memori terdapat penurunan sebesar **-6.01 MB** atau **-0.35%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **31.629%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi beban

rata – rata kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation framework* sebesar **1.541 detik.**

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan perangkat lunak selanjutnya.

6.1 Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba penghematan daya dan peningkatan kinerja pada perangkat bergerak Android menggunakan *offloading computation framework* yang diterapkan pada perangkat lunak *image recognition* adalah sebagai berikut:

1. Implementasi metode *offloading* dapat digunakan sebagai salah satu metode penghematan daya dan peningkatan kinerja pada perangkat bergerak Android. Namun juga dapat menurunkan waktu eksekusi beban kerja dan penghematan daya pada perangkat bergerak Android jika koneksi tidak stabil.
2. Implementasi JADE *middleware* dapat digunakan dalam menerapkan metode *offloading* dengan cara melakukan pengiriman dan penerimaan modul pengiriman data yang bertipe *string* antara *client* dan *server*.
3. Implementasi *mobile framework* dapat dijadikan sebagai metode yang cocok untuk meminimalisir kerugian menggunakan metode *offloading computation* dengan cara menentukan secara dinamis metode pengeksekusian beban kerja pada perangkat Android.
4. Pada kasus Tugas Akhir ini, *mobile computation offloading framework* melakukan penghematan daya dan peningkatan kinerja yang optimal pada saat perangkat bergerak Android memiliki kualitas koneksi internet yang stabil dalam melakukan *transmit* dan *receive* data (kategori Koneksi A).

5. *Decision Maker* pada *mobile computation offloading framework* dapat menentukan eksekusi beban kerja secara dinamis untuk meningkatkan penghematan daya serta meningkatkan kecepatan performa pada perangkat bergerak Android yang hasil penerapannya dapat dilihat pada Tabel 5.18.
6. Perhitungan penghematan daya, performa, dan waktu eksekusi terbaik yang didapatkan masing – masing penghematan daya 1% (uji coba 1, 2, 3), penggunaan memori 2.525% sehingga melakukan penghematan penggunaan memori sebesar 0.259% dibandingkan dengan eksekusi beban kerja tanpa *framework* (uji coba 1), penggunaan CPU 21.26% sehingga melakukan penghematan penggunaan CPU sebesar 40.379% (uji coba 3) dan waktu eksekusi rata - rata beban kerja 3.462 detik sehingga melakukan penghematan waktu eksekusi rata – rata sebesar 2.428 detik (uji coba 1).

6.2 Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah:

1. Menambah jumlah data karena dari total 40 data citra yang digunakan sebagai dataset dianggap masih terlalu sedikit. Ditambahkannya kondisi jika perubahan kualitas koneksi internet menjadi buruk terjadi sesaat setelah pengiriman beban kerja oleh *offloading framework* menuju *server* dilakukan. Sehingga perangkat bergerak *client* akan menunda pemrosesan beban kerja *image recognition* selanjutnya akibat menunggu hasil eksekusi dari *server*. Pada studi kasus ini, *server* tidak dapat mengirim kembali hasil eksekusi akibat koneksi buruk yang dimiliki perangkat bergerak *client*.
2. Pengecekan sumber daya baterai lebih detail pada perangkat bergerak *client* yang bisa dilakukan dengan alat – alat tertentu.

DAFTAR PUSTAKA

[1] K. Liu, J. Peng, H. Li, X. Zhang, dan W. Liu, "Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing." [Daring]. Tersedia pada: http://www.sciencedirect.com/science/article/pii/S0167739X16300905. [Diakses: 01-Jun-2017].

[2] "Smartphone definition (Phone Scoop)." [Daring]. Tersedia pada: http://www.phonescoop.com/glossary/term.php?gid=131. [Diakses: 31-Mei-2017].

[3] "UCI Machine Learning Repository: Leaf Data Set." [Daring]. Tersedia pada: https://archive.ics.uci.edu/ml/datasets/leaf. [Diakses: 01-Jun-2017].

[4] "Google's Android OS: Past, Present, and Future," *Phone Arena*. [Daring]. Tersedia pada: http://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future_id21273. [Diakses: 31-Mei-2017].

[5] M. Wang, "Novel Mobile Computation Offloading Framework for Android Devices," *Eng. Appl. Sci. Theses Diss.*, Des 2014.

[6] "A survey on clustering algorithms for wireless sensor networks." [Daring]. Tersedia pada: http://www.sciencedirect.com/science/article/pii/S0140366407002162. [Diakses: 31-Mei-2017].

[7] "Jade Site | Java Agent DEvelopment Framework." .

[8] *gson: A Java serialization/deserialization library to convert Java Objects into JSON and back*. Google, 2017.

[9] "Lang – Home." [Daring]. Tersedia pada: https://commons.apache.org/proper/commons-lang/. [Diakses: 31-Mei-2017].

[10] "OpenCV library." [Daring]. Tersedia pada: http://opencv.org/. [Diakses: 31-Mei-2017].

[11] P. Sykora, P. Kamencay, dan R. Hudec, “Comparison of SIFT and SURF Methods for Use on Hand Gesture Recognition based on Depth Map,” *AASRI Procedia*, vol. 9, hal. 19–24, Jan 2014.
[12] G. Caire, “JADEProgramming-Tutorial-for-beginners.pdf.” 30-Jun-2009.
[13] G. Caire dan F. Pieri, “LEAPUserGuide.pdf.” 15-Nov-2011.
[14] *network-connection-class: Listen to current network traffic in the app and categorize the quality of the network*. Facebook, 2017.
[15] H. Qian dan D. Andersen, “Jade: Reducing Energy Consumption of Android App.” .

LAMPIRAN

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Putro Satrio Wibowo merupakan anak dari pasangan Bapak Wasisto dan Ibu Nanik Wahyuni. Lahir di Lumajang pada tanggal 4 November 1994. Penulis menempuh pendidikan formal dimulai dari TK PGRI Pasirian (1999-2001), SDN 4 Pasirian (2001-2007), SMPN 1 Pasirian (2007-20010), SMAN 2 Lumajang (2010-2013) dan S1 Teknik Informatika ITS (2013-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2014-2015) dan KMI (2014-2016). Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2014 divisi National Programming Competition dan SCHEMATICS 2015 divisi Perlengkapan dan Transportasi. Penulis juga menyukai kegiatan sosial dan pecinta alam. Penulis memiliki hobi futsal dan bermain game. Penulis dapat dihubungi melalui email: putrosatrio27@gmail.com.