UNIVERSITÉ DE MONTRÉAL

ARCHITECTURE D'APPLICATIONS MOBILES DANS LE DOMAINE DE LA SANTÉ

RONALD JEAN-JULIEN

DÉPARTEMENT DE GÉNIE LOGICIEL ET GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

AVRIL 2013

© Ronald Jean-Julien, 2013.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ARCHITECTURE D'APPLICATIONS MOBILES DANS LE DOMAINE DE LA SANTÉ

présenté par : JEAN-JULIEN Ronald

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme BELLAÏCHE Martine, Ph.D., présidente

M. PIERRE Samuel, Ph.D., membre et directeur de recherche

M. QUINTERO Alejandro, Doct., membre

À mes parents, mes amis et toute ma famille.

REMERCIEMENTS

Je tiens d'abord à remercier mon directeur de recherche, le professeur Samuel Pierre, pour ses conseils, son encadrement et l'aide financière qui m'a été accordée.

Je remercie aussi Mr. Pierre-Alexandre Fournier qui m'a accueilli au sein de Carré Technologies, une jeune entreprise qui développe des applications de santé mobile et qui m'a permis d'appréhender toutes les notions liées à ce secteur.

Mes remerciements s'adressent également aux honorables membres du jury qui ont accepté d'évaluer mon travail.

J'aimerais aussi remercier le CRSNG (Centre de Recherches en Sciences Naturelles et en Génie) pour l'appui financier qu'ils ont consenti à m'apporter durant la recherche.

Un grand merci à la grande famille du Laboratoire de Recherche en Réseautique et Informatique Mobile (LARIM) pour leurs critiques constructives tout au long de la réalisation ce projet.

RÉSUMÉ

Durant ces dernières années, le monde informatique a assisté à une percée spectaculaire des technologies dans le domaine de l'informatique mobile. Cette avancée considérable permet d'offrir de plus en plus de services non seulement aux usagers de l'informatique directement, mais aussi à d'autres secteurs en quête de solutions par rapport à des problèmes jusqu'ici demeurés non résolus. On peut citer notamment le secteur de la santé. Le monde informatique permet aujourd'hui à un médecin, muni d'un terminal mobile, de consulter un patient à distance dans n'importe quelle région du monde sans se déplacer. Les résultats de ses consultations, une fois recueillis, sont exploitables de façon instantanée. Ce qui est encore plus intéressant avec cette technologie est le fait que le médecin dispose aujourd'hui de capteurs biologiques qu'il peut accoler au patient pour pouvoir faire un suivi constant par rapport à une évolution quelconque de sa situation.

Cette façon d'assister les patients à distance a aussi des failles. En effet, on doit prendre en compte la capacité limitée des terminaux mobiles et la complexité de certaines tâches qui demandent une grande puissance de calcul. La solution la plus prisée pour essayer de résoudre ce problème est celle qui permet d'externaliser les tâches de calcul sur des serveurs distants, disposant de la capacité nécessaire à les faire exécuter. Grâce à l'arrivée du *Cloud Computing*, permettant d'exploiter de gros serveurs disposant de ressources virtuellement illimitées, les requêtes qui consomment beaucoup de ressources sont envoyées dans le *Cloud* et le résultat trouvé est exploité par la suite sur le terminal mobile.

Dans le cadre de ce mémoire, nous proposons une architecture sujette à un ensemble de contraintes de performance qui nous permet de tenir compte du meilleur endroit d’exécution des tâches. Pour arriver à cette réalisation, nous proposons un mécanisme de gestion du délai de garde à l'envoi des requêtes dans le *Cloud* pour limiter le temps d'attente sur les terminaux mobiles. Ce mécanisme permet de mieux contrôler les requêtes qui sont envoyées dans le *Cloud* étant donné que l'environnement dans lequel évoluent les terminaux mobiles est exposé à un ensemble de changements qui peuvent arriver n'importe quand et pour des raisons multiples. Pour décider de l'endroit d'exécution d’une tâche évoluant dans le système, le code de notre application doit être partitionné en deux parties. Une première partie qui sera toujours exécutée sur le

téléphone et la seconde partie qui peut être exécutée soit sur le téléphone soit dans le *Cloud*. Le développeur est tenu responsable de l'identification de ces deux parties et doit implémenter la deuxième partition à la fois dans le *Cloud* et sur le téléphone.

Étant donné que l'environnement dans lequel évolue notre application change constamment, nous questionnons les conditions environnementales à chaque fois qu'une tâche est sujette à la déportation dans le *Cloud*. Nous avons pour cela conçu un système expert, composé d'une base de connaissances et d'un moteur d'inférence qui nous permet de décider de l'endroit d'exécution de nos tâches. Le moteur d'inférence décide sur la base d'un ensemble de règles et de faits où exactement une tâche donnée doit être exécutée.

Afin de valider notre architecture, un ensemble de tests a été réalisé. Nous avons obtenu des résultats assez positifs qui nous permettent de conclure que le *Cloud* peut effectivement être utilisé pour améliorer la performance des applications sur les terminaux mobiles. Ces tests sont réalisés en utilisant la solution EC2 d'Amazon qui nous permet d'externaliser nos tâches dans un centre de données situé aux États-Unis.

ABSTRACT

In recent years, the world of information technology has witnessed a spectacular breakthrough in mobile computing area. With those advances, more and more services are being offered not only to people who are in mobile communications industry, but also to those ones who are looking for solutions to some problems that remained so far unresolved. Health care area is a good example. The computing world today allows doctors, staying at home, just by using a mobile device, to offer their services to patients remotely located anywhere in the world. The results, once collected, become fully usable instantly. What is even more interesting with those technologies is that they can use biosensors, that they put on the patients to collect data, to constantly monitor the evolution of any situation.

The way to assist patients remotely, using those technologies, also has vulnerabilities. Indeed, given the limited capacity in terms of resources of mobile devices and the complexity of some algorithms in the medical sector which require a lot of computation, the most popular proposed solution is to offload the most complicated tasks to external powerful servers. Thanks to the arrival of *Cloud Computing* in recent years, that enables computing resources to be accessed and shared as unlimited powerful virtual resources in a scalable way, most complicated tasks are sent to the *Cloud* and the results are used subsequently on mobile devices. Different proposals with computation offloading approaches have been released to try to solve this resource limitation problem.

In this thesis, we propose an architecture subject to a set of constraints to performance allowing us to take into account the best place to execute our tasks. To get this done, we have proposed a mechanism to manage the timeout of the requests that are sent to the *Cloud* given that the system environment is changing constantly and for a lot of reasons out of control of the system. To make use of computation offloading, we have partitioned our codes in two parts. One that will always be running on the mobile device and the other one that can be run on both sides either mobile device or *Cloud*. The developer needs to identify parts of codes that will ask for more resources to get them implemented in the *Cloud* also.

Given that the environment in which our application is being run is changing constantly, we question the environmental parameters on every task that can be offloaded to decide on the spot where exactly this task should be run at this moment. We have design an expert system, using a knowledge base and an inference engine to decide the best place to run our tasks. The inference engine decides, based on a set of rules and a set of facts where exactly it should execute a task.

In order to validate our architecture, a set of tests have been conducted. We got positive results that allow us to conclude that the *Cloud* can actually be used to improve the performance of applications on mobile devices. Those tests are performed by using Amazon EC2 solution that allows us to offload our tasks in a data centre located in the United States.

TABLE DES MATIÈRES

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES FIGURES xiii

LISTE DES SIGLES ET ABRÉVIATIONS xv

CHAPITRE 1 INTRODUCTION 1

1.1 Définitions et concepts de base 2

1.2 Éléments de la problématique 3

1.3 Objectif principal 5

1.4 Plan de mémoire 6

CHAPITRE 2 DÉPORTATION DE CALCULS DANS LE NUAGE 7

2.1 Définitions et concepts de base 8

2.1.1 Le stress 8

2.1.2 Biofeedback 9

2.1.3 Biocapteur 9

2.1.4 Variabilité de la fréquence cardiaque (*HRV ou Heart Rate Variability*) 9

2.1.5 La cohérence cardiaque 10

2.1.6 Mesures de la variabilité du rythme cardiaque 10

2.1.7 Algorithme de *Lomb-Scargle* 11

2.2 Les réseaux mobiles 11

2.2.1 Informatique dans le nuage (*Cloud Computing*) 12

2.2.2 Caractéristiques de l'informatique dans le nuage 12

2.2.3 Modèles de services dans le nuage 14

2.2.4 Les modèles de déploiement du nuage 15

2.2.5 Contraintes liées à la mobilité des terminaux mobiles 16

2.2.6 Informatique mobile dans le nuage 16

2.2.7 Informatique dans le nuage vs Informatique mobile dans le nuage 17

2.3 Les Services Web 18

2.3.1 Le protocole SOAP 18

2.3.2 Architecture REST 20

2.4 Déportation de calculs (*Computation Offloading*) 21

2.4.1 Approche Client-Serveur 21

2.4.2 Migration de machines virtuelles 22

2.4.3 Ericsson Mobile Health (EMH) 22

2.4.4 MAUI (Mobile Assistance Using Infrastructure) 24

2.4.5 Approche de clonage du mobile dans le nuage (*Clone Cloud*) 27

2.4.6 Cloudlet based resource-rich mobile Computing 28

2.4.7 MobiCloud 30

2.4.8 Utilisation de la bande passante 30

2.4.9 *Ad-Hoc Mobile Cloud* 30

2.5 Analyse des travaux 31

CHAPITRE 3 ARCHITECTURE D'UNE APPLICATION MOBILE BASÉE SUR LE CLOUD 33

3.1 Les requis de l'architecture proposée 33

3.1.1 Requis fonctionnels 33

3.1.2 Requis non fonctionnels 34

3.2 Présentation du prototype 35

3.2.1 Hypothèses 35
3.2.2 Période d'apprentissage 35
3.3 Architecture de l'application 38
3.3.1 L'interface utilisateur 38
3.3.2 Le module d'acquisition de données 39
3.3.3 Le module de traitement de données sur le terminal mobile 39
3.3.4 Le module gérant les systèmes hérités 39
3.3.5 Le module décisionnel 40
3.3.6 Le module de traitement de données à distance 40
3.3.7 Le moniteur de ressources 40
3.4 Communication avec le *Cloud* 41
3.5 Utilisation du style architectural REST 42
3.6 Partitionnement du programme 43
3.7 Algorithme de prise de décision 44
3.8 Prédiction du temps d'exécution 45
3.8.1 Moyenne mobile glissante 45
3.9 Authentification et autorisation 45
3.10 Présentation de quelques classes du système 45
3.11 Système Expert 46
3.11.1 Moteur d'inférence 47
3.11.2 Base de faits 48
3.11.3 Base de règles 48
CHAPITRE 4 ÉVALUATION DE L'ARCHITECTURE ET RÉSULTATS 55
4.1 Évaluation des requis fonctionnels 55

4.2 Environnement de développement 56

4.3 Évaluation des requis fonctionnels 58

4.4 Scénario 1 : Exécution sur le terminal mobile 59

4.5 Scénario 2 : Exécution avec REST, réseau Wi-Fi, sans délai de garde 60

4.6 Scénario 3 : Exécution avec REST, sur un réseau 3G, sans délai de garde 62

4.7 Scénario 4 : Comparaison REST et SOAP 63

4.8 Scénario 5 : Exécution avec REST, sur un réseau 3G, avec temps de garde 65

4.9 Scénario 6 : Exécution avec REST, sur un réseau Wi-Fi, avec un temps de garde 67

4.10 Scénario 8 : Influence de la taille des données 68

4.11 Scénario 7 : Exécution avec retour immédiat sur le terminal mobile dans le cas où le délai de garde est dépassé 69

4.12 Scénario 9 : Passage d'un réseau Wi-Fi 1 à un réseau Wi-Fi 2 (Mobilité) 71

4.13 Scénario 10 : Passage du Wi-Fi au 3G (Mobilité : Relève verticale) 71

4.14 Scénario 11 : Test sur un réseau à Montréal 72

4.15 Mise au point autour des scénarios 72

CHAPITRE 5 CONCLUSION 74

5.1 Synthèse des travaux 74

5.2 Limitations des travaux 75

5.3 Travaux futurs 75

BIBLIOGRAPHIE 77

LISTE DES FIGURES

Figure 1.1. L'informatique dans le nuage 2

Figure 2.1 Architecture du *Cloud*.....14

Figure 2.2. L'informatique mobile dans le *Cloud* inspiré de [22].....17

Figure 2.3. Requête SOAP 18

Figure 2.4. Réponse SOAP 19

Figure 2.5. Ericsson Mobile Health inspiré de [25] 23

Figure 2.6. Architecture de MAUI inspiré de [16].....25

Figure 2.7. Architecture de *Clone Cloud* inspiré de [18] 28

Figure 2.8 Concept du *Cloudlet* inspiré de [27] 29

Figure 3.1. Vue globale du système 37

Figure 3.2. Architecture du système.....41

Figure 3.3. Diagramme de séquence d'une requête dans le *Cloud* 43

Figure 3.4. Algorithme de prise de décision 44

Figure 3.5. Concept de base d'un système expert 47

Figure 4.1. Niveau de stress de l'utilisateur (Stressé et non stressé) 59

Figure 4.2. Exécution des tâches sur le terminal mobile.....60

Figure 4.3. Comparaison des temps d'exécution en utilisant le Wi-Fi 61

Figure 4.4. Comparaison des temps d'exécution en utilisant le réseau 3G.....63

Figure 4.5. Comparaison des temps d'exécution (REST vs SOAP) 64

Figure 4.6. Comparaison des temps d'exécution en utilisant un temps de garde 65

Figure 4.7. Décomposition des temps de réponse dans le *Cloud*.....67

Figure 4.8. Comparaison des temps d'exécution avec temps de garde (Wi-Fi vs Local).....68

Figure 4.9. Comparaison des temps d'exécution avec différentes tailles de données 69

LISTE DES SIGLES ET ABRÉVIATIONS

3G	troisième génération
API	Application Program Interface
ARM	Advanced RISC Machine
CPU	Central Processing Unit
EC2	Elastic *Cloud Computing*
ECG	Electrocardiogram
EDGE	Enhanced Data Rates for GSM Evolution
EMH	Ericsson Mobile Health
GAE	Google App Engine
GSM	Global System for Mobile Communication
HRV	Heart Rate Variability
Html	HyperText Markup Language
Http	HyperText Transport Protocol
IaaS	Infrastructure as a Service
Java EE	Java Platform Enterprise Edition
JNI	Java Native Interface
JSON	JavaScript Object Notation
Manet	Mobile Ad Hoc Network
PaaS	Platform as a Service
RISC	Reduced Instruction Set Computing
RMI	Remote Method Invocation
RNF	Requis Non Fonctionnel
RPC	Remote Procedure Call

RTT	Round Trip Time
MAUI	Mobile Assistance Using Infrastructure
REST	Representational State transfer
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
Wi-Fi	Wireless Fidelity
WSDL	Web Service Description Language
XML	eXtensible Markup Language

CHAPITRE 1
INTRODUCTION

Avec le développement important qu'ait connu ces dernières années le marché des téléphones cellulaires, de plus en plus de services sont offerts sur les terminaux mobiles. Les réseaux ont dû évoluer pour permettre le transport de certains types de trafic autres que celui de la voix, ce qui permet à ces terminaux de pouvoir offrir des services qui, jadis, étaient seulement disponibles sur les ordinateurs. Dans le domaine médical, qui fera l'objet d'une analyse dans le cadre de ce mémoire, les spécialistes de la santé les considèrent comme des mini-ordinateurs. Ces appareils offrent une assistance personnelle aux utilisateurs allant de l'interprétation de simples symptômes au diagnostic médical.

Cette technologie naissante et grandissante connait de nombreuses limitations, plus spécifiquement du côté des téléphones mobiles et des tablettes utilisés. En effet, ces appareils, comparativement aux ordinateurs portables ou ordinateurs de bureau sont limités en ressources lorsqu'on considère les applications médicales très gourmandes en ressources. Plusieurs chercheurs proposent des solutions pour pallier cette difficulté [1], [2], [3]. Ainsi, à la fin de l'année 2010, la compagnie Ericsson propose la norme Bluetooth 4.0, qui assure une plus grande portée des dispositifs utilisant cette technologie [4]. Cette version à faible consommation d'énergie constitue un énorme avantage pour le secteur médical par rapport à leur intégration dans les capteurs biologiques. Plus récemment, les recherches ont porté davantage sur l'informatique dans le nuage comme dans les travaux de Giurgi et al. [5], Nkosi, M. and Mercuria [6]. Ce nouveau concept permet d'externaliser certaines tâches sur des serveurs distants disposant d'une plus grande capacité, en comparaison à celle des dispositifs mobiles [7]. De plus, cette nouvelle approche présente comme avantage la disponibilité de plus grands centres de données permettant une puissance de calcul exceptionnelle au service des appareils mobiles. Mais comment arriver à exploiter cette capacité disponible dans le nuage pour permettre aux appareils mobiles de passer outre cette limitation qui leur pose tant de soucis dans les cas d'exécutions de certaines tâches gourmandes en ressources de calcul? C'est dans ce contexte d'externalisation de ces tâches de calculs qu'intervient ce mémoire.

Dans ce chapitre d'introduction, nous allons présenter dans un premier temps l'informatique dans le nuage et le contexte de réalisation de notre projet par rapport au sujet que nous sommes en train de traiter. Ensuite suivront les éléments de la problématique et la formulation de nos objectifs de recherche. Le plan du mémoire clôture enfin le chapitre.

1.1 Définitions et concepts de base

Informatique dans le nuage

L'informatique dans le nuage, illustrée à la figure 1.1, est plus connue par son anglicisme *Cloud Computing*. C'est une technologie qui consiste à externaliser les ressources numériques des entreprises sur des serveurs distants offrant ainsi des capacités de calcul et de stockage virtuellement illimitées, contrairement à l'hébergement traditionnel sur le poste utilisateur [8-11]. Dans la suite du document, le terme *Cloud* sera utilisé pour désigner l'informatique dans le nuage.

Figure 1.1. L'informatique dans le nuage

Capteurs biologiques

Un biocapteur est un dispositif qui transforme un signal biologique en un signal électrique facilement exploitable [12]. A l'aide d'un biocapteur, il devient alors possible d'exploiter les données en provenance du corps humain, les transmettre à un terminal mobile, permettant ainsi de suivre en temps réel le comportement de celui-ci et de pouvoir en tirer des conclusions assez rapides par rapport à certaines situations.

Variabilité du rythme cardiaque

La variabilité du rythme cardiaque ou *Heart Rate Variability* (HRV) est un paramètre utilisé pour calculer le niveau de stress d'un individu. Elle correspond à la variation de la durée de l'intervalle entre chaque battement du cœur [13]. La variabilité du rythme cardiaque est inversement proportionnelle au niveau de stress d'un individu. Plus cette valeur est élevée, plus la personne est détendue.

Application mobile

Dans le cadre de notre projet, nous considérons comme application mobile toute application qui peut être exécutée sur un dispositif mobile tel un téléphone mobile, une tablette, ou tout autre périphérique mobile sujet à des déplacements. Ces applications seront alors disponibles aux utilisateurs n'importe où et n'importe quand. Ceci devient d'autant plus important dans le domaine de la santé où il est possible, à l'aide de capteurs biologiques appliqués aux patients, suivre leur comportement en tout temps et en tout lieu. Ces capteurs biologiques prélèvent des informations sur les patients qui sont envoyées à l'aide d'une puce Bluetooth au terminal mobile qui se charge du traitement des données recueillies.

1.2 Éléments de la problématique

Le Cloud est perçu par plus d'un comme une évolution majeure du monde informatique. La conjugaison de cette technologie, des capteurs biologiques et autres permet d'assister aujourd'hui à un essor considérable des applications mobiles dans le domaine de la santé. En effet, vu les nombreux problèmes auxquels fait face le secteur de la santé tel le vieillissement de la population et le manque de personnels à pouvoir assister les patients, de nombreux travaux ont été effectués afin de pouvoir les assister de façon adéquate [6, 14, 15].

Le domaine d'application est assez vaste, détection des signes vitaux, surveillance à distance des personnes souffrant de l'hypertension, du diabète, consultation à distance. Et, les derniers développements technologiques avec les nouveaux téléphones portables nous fraient un chemin assez prometteur.

Ces applications mobiles font face à l'heure actuelle à des besoins grandissants en termes de puissance de calcul informatique. La capacité limitée des terminaux mobiles pose un problème qui oblige la communauté scientifique à explorer différentes solutions. Avec l'arrivée du *Cloud* sur le marché ces dernières années, beaucoup de regards se sont fixés sur ces centres de données disposant d'une puissance de calcul virtuellement illimitée, pour essayer de résoudre ce problème. La déportation des tâches de calcul dans le *Cloud* a été proposée pour pallier les problèmes de performance sur le terminal mobile. Cette approche d'externalisation, bien que très intéressante, n'est pas toujours la solution idéale et ne constitue pas la panacée des problèmes de performance des terminaux mobiles. Plusieurs propositions sont faites dans la littérature, les unes plus performantes que les autres en vue de trouver une réponse aux problèmes de performance. De façon générale, les questions posées par rapport à cette approche d'externalisation sont multiples. Quand est-ce que la déportation des tâches dans le *Cloud* doit être faite? Comment identifier les différentes parties du code à déporter? Comment choisir le serveur externe sur lequel déporter le code?

Pour décider des tâches à déporter dans le *Cloud*, [Cuervo, E., A. Balasubramanian, et al.] [16] présentent une approche qui permet aux développeurs d'annoter les méthodes qui doivent être assujetties à la déportation. Ceci conduit à un partitionnement du code en deux groupes, un premier groupe qui doit être toujours exécuté sur le terminal mobile et un deuxième groupe qui peut être exécuté soit sur le terminal mobile soit dans le *Cloud*. La décision de déportation se fait en général au moment de l'exécution de l'application en tenant compte du caractère évolutif de l'environnement d'exécution. Cette approche tient seulement compte des applications utilisant la plate-forme .Net de Microsoft. Elle nécessite aussi l'identification des tâches à être exécutées dans le *Cloud* par le développeur.

Chun et al. [1, 17, 18], proposent une approche de clonage complet du terminal mobile dans le *Cloud*. Cette approche permet de bénéficier du même environnement d'exécution que sur le téléphone, ce qui constitue un énorme avantage. Le partitionnement du code est réalisé à chaud

en faisant migrer le thread d'exécution du terminal mobile au clone situé dans le *Cloud*. Une fois l'exécution terminée dans le *Cloud*, le thread est réintégré sur le terminal mobile, rendant ainsi exploitable le résultat d'exécution trouvé à distance. L'approche de clonage est encore à ces débuts et il reste beaucoup à faire pour mieux assoir cette idée. Les plateformes mobiles étant nombreuses aujourd'hui, il n'existe pas encore une approche satisfaisante permettant de faire le clonage du terminal mobile dans le *Cloud*. Le temps de migration à chaud constitue un problème majeur pour cette approche.

Verbelen et al. [3] ont approché le problème en proposant d'externaliser les tâches de calcul dans un *Cloudlet*. Ce dernier peut être défini comme un centre de données dans une boite situé dans le voisinage immédiat du terminal mobile, donnant ainsi un accès plus rapide une fois toutes les mises en place effectuées. Leur point d'appui étant que les réseaux Wi-Fi situés dans le voisinage du téléphone offrent un accès beaucoup plus rapide aux serveurs exécutant les tâches requises. Le problème avec cette approche est l'instanciation des machines virtuelles qui prend trop de temps à être réalisée. L'infrastructure d'exécution des tâches n'étant pas implémentée au départ dans le *Cloudlet*, le système prend un certain temps pour la mise en place des serveurs avant de commencer l'exécution des différentes tâches.

1.3 Objectif principal

L'objectif principal de ce mémoire est de concevoir l'architecture d'une application mobile tout en ayant soin d'intégrer un module de prise de décision par rapport au meilleur endroit à faire exécuter les tâches qui consomment beaucoup de ressources, soit dans le *Cloud* ou directement sur le téléphone. Cette application se situe dans le domaine de la santé et permettra à l'aide de l'analyse de la variabilité du rythme cardiaque ou *Heart Rate Variability* (HRV), de contrôler le niveau de stress d'un utilisateur.

De façon spécifique, ce mémoire vise à :

- Analyser les architectures et algorithmes existants dans le domaine;
- Proposer une architecture prenant en compte la capacité limitée des appareils mobiles;
- Proposer un mécanisme de prise de décision par rapport au meilleur endroit à faire; exécuter les tâches de calcul comme celle qui nous permet d'obtenir le niveau de stress d'un individu;

- Évaluer la performance de l’architecture proposée.

1.4 Plan de mémoire

La suite du mémoire est organisée de la manière suivante. Le chapitre 2 présente les différentes approches architecturales qui ont été présentées dans la littérature, leurs faiblesses et leurs points forts. Ces approches sont classées en plusieurs catégories suivant que le *Cloud* soit utilisé ou que toutes les tâches soient exécutées sur le téléphone. Nous avons identifié deux catégories d’approches. La première catégorie désigne les applications qui font appel au clonage dans le *Cloud*, le téléphone se trouve ainsi reproduit dans le *Cloud* dans toute sa globalité. Une autre approche fait appel au partitionnement du code, une première partie est exécutée sur le téléphone et une autre partie dans le *Cloud*. Le chapitre 3 présente notre solution architecturale. Le chapitre 4 présente l’évaluation de performance de l’architecture proposée. Nous terminons par une conclusion où nous présentons les limites de notre travail et proposons certains travaux à réaliser dans le futur.

CHAPITRE 2
DÉPORTATION DE CALCULS DANS LE NUAGE

Le secteur de la santé se trouve aujourd'hui confronté à de nombreux défis. Avec une population mondiale de plus en plus vieillissante, la prise en charge de nouvelles maladies chroniques et le besoin de faire appel de plus en plus à des mesures préventives, le personnel de la santé essaie de profiter le plus possible des facilités que lui offrent les nouvelles technologies de l'information. En effet, avec les percées technologiques de ces dernières années, le secteur informatique joue un rôle majeur en contribuant à de nouveaux champs d'épanouissement du secteur médical. Nous allons faire ressortir quelques-uns de ces exemples qui nous intéressent par rapport à notre sujet. 1) Les capteurs biologiques qui permettent de capter les signaux biologiques en provenance d'un être humain. 2) Le Bluetooth 4.0 qui permet une communication sur une plus grande distance avec les terminaux mobiles. 3) Les nouveaux téléphones intelligents utilisant différents plates-formes tels qu'Android ou iOS qui font du téléphone un outil indispensable dans la société d'aujourd'hui. 4) Les réseaux mobiles d'aujourd'hui avec un débit relativement élevé et aussi grâce à l'arrivée des téléphones intelligents permettent d'offrir des services aussi divers que variés. 5) L'informatique dans le nuage ou "*Cloud Computing*" qui permet de faire de la déportation de calculs sur des serveurs puissants. Les dernières évolutions dans les réseaux mobiles permettent d'être joignable presque partout à travers différentes options tels les réseaux cellulaires, le Wi-Fi, les réseaux satellitaires à une vitesse relativement élevée. Ainsi avec la technologie au bout du doigt, il est possible d'offrir au médecin la possibilité d'être en tout temps et en tout lieu à la traque de ce qui se passe chez son patient sans délai et de façon automatique. Pour ce faire, le médecin peut s'appuyer sur une large gamme de produits disponibles grâce à ces dernières évolutions technologiques que nous venons d'évoquer. Étant donné la capacité limitée des terminaux mobiles et la complexité de certains traitements sur les données médicales, nous sommes obligé de déporter bien des calculs dans le nuage informatique. En ce sens, dans ce chapitre, nous nous donnons pour tâche dans notre revue de littérature de se pencher sur ces nouvelles technologies, regarder en profondeur ce qui se fait dans le secteur avant de présenter notre travail. Le cas d'application que nous utilisons étant la gestion du stress par la méthode du *biofeedback*, nous allons présenter d'abord certaines définitions et concepts de base liées au

domaine, ensuite nous présenterons l'informatique dans le nuage avec ces défis et ces avantages. Puis nous mettons le focus sur les services Web qui peuvent être utilisés pour communiquer avec certaines applications dans le *Cloud*. Toute la revue sera faite en s'appuyant sur les différents travaux relatifs aux différents points clés ci-dessus mentionnés.

2.1 Définitions et concepts de base

2.1.1 Le stress

Dans le monde d'aujourd'hui, personne ne peut se vanter d'être à l'abri du stress. Ce dernier envahit tous les couloirs de notre société. Qu'on soit aux études, au travail, au chômage, etc., tout le monde fait face d'une façon ou d'une autre, un jour ou l'autre à une situation estimée stressante. Bien des personnes se voient perdre subitement un travail sans préparation pour le chômage qui s'en suit. Un étudiant qui a eu à subir un examen très difficile aujourd'hui alors qu'il a une présentation à faire le lendemain. Une personne qui s'en va à un entretien d'embauche. Un candidat à la présidence qui vient de perdre les élections. Un patient qui reçoit des résultats compromettants après certains examens médicaux. Un employé qui doit remettre un rapport à son patron alors que le temps presse. Bref, ils sont vraiment nombreux les exemples de situations stressantes qui peuvent être retracés dans notre société, le stress est partout et se fait présent à tout instant dans notre vie.

Les exemples de stress cités plus haut sont tous des exemples négatifs, mais il existe aussi des situations stressantes perçues positivement. Quelqu'un qui va se marier par exemple, qui gagne à la loterie, ou qui s'en va en vacances, a toutes les raisons du monde pour se voir envahir de stress mais cette fois-ci dans le bon sens.

La gestion du stress est quelque chose de très complexe et il y a eu beaucoup de recherches à être faites dans ce sens. A chaque situation stressante correspond une approche de résolution qui lui est propre. Il est conseillé aux personnes qui en souffrent de façon chronique d'aller voir un spécialiste. Parmi les méthodes utilisées pour résoudre les problèmes de stress, nous pouvons citer le sport, le yoga, le contrôle de la respiration. S'appuyant sur les dernières avancées dans le monde informatique, le monde médical essaie aujourd'hui de tirer profit des terminaux mobiles leur permettant de suivre les patients en tout temps et en tout lieu à travers les

applications mobiles se connectant aux capteurs qui prélèvent des données directement de ces derniers.

Sur le marché des applications mobiles, il existe différentes approches pour essayer d'aider les gens à se débarrasser de leur stress. Malheureusement ces applications ne permettent pas aux utilisateurs d'avoir un contrôle instantané sur leur situation de stress. Ces applications sont là surtout pour donner des conseils aux individus, mais elles ne les aident pas vraiment à contrôler leur stress de façon instantanée, d'où l'apport du *biofeedback*.

2.1.2 Biofeedback

Le mot le plus approprié que nous utilisons pour traduire le *biofeedback* est peut-être le rétrocontrôle. Le *biofeedback* peut être vu comme un processus qui permet de mettre à la disposition de l'utilisateur les informations biologiques qui seront prélevées à partir d'un biocapteur. Ces données donneront des informations à l'utilisateur par rapport à son niveau de relaxation, ce qui lui permettra de pouvoir prendre des décisions pour améliorer son état de santé. Imaginons que le niveau de relaxation d'un individu varie de 0 à 100 par exemple. Utilisant les données provenant d'un biocapteur, une application lui affiche son niveau de relaxation qui évolue de façon dynamique lui permettant de réagir à chaud pour modifier son état qu'il regarde évoluer devant lui sur le petit écran de son terminal.

2.1.3 Biocapteur

Pour faire l'analyse des données biologiques, les biocapteurs sont de plus en plus utilisés. Ces derniers peuvent être définis comme des dispositifs qui transforment un signal biologique observé en un signal électrique facilement manipulable. Ils sont utilisés comme éléments de base pour faire l'acquisition des données provenant des êtres humains [12]. Ils sont constitués d'un détecteur biologique et d'un transducteur servant à faire la conversion du signal.

2.1.4 Variabilité de la fréquence cardiaque (*HRV ou Heart Rate Variability*)

La variabilité du rythme cardiaque, également présentée sous les termes de "variabilité sinusale" ou encore "d'arythmie respiratoire", est le paramètre étudié pour donner le niveau de stress d'un individu. Elle peut être vue comme une méthode permettant d'apprécier l'état de la régulation biologique, faisant référence aux oscillations spontanées du rythme cardiaque. Une

personne disposant d'un biocapteur pouvant générer des signaux *ECG* (Électrocardiogrammes) peut à l'aide de ces informations calculer la variabilité du rythme cardiaque en analysant les intervalles cardiaques successifs [13]. A l'aide du *HRV*, un ensemble d'informations riches et variées est immédiatement disponible par rapport au niveau de stress de l'individu.

Différents indices sont utilisés pour quantifier la variabilité de la fréquence cardiaque. Certaines analyses sont effectuées dans le domaine temporel et d'autres dans le domaine fréquentiel. Dans le domaine temporel, nous utilisons surtout l'écart-type des séries d'intervalles par rapport aux battements de cœur successifs. Dans le domaine fréquentiel, l'analyse spectrale de l'évolution des intervalles *R-R* en fonction du temps permet de générer un périodogramme illustrant la répartition de densité spectrale en fonction de la fréquence des oscillations présentes dans le tracé initial.

2.1.5 La cohérence cardiaque

La cohérence cardiaque est une mesure de la variabilité du rythme cardiaque. Même si le cœur bat régulièrement, par exemple à 60 pulsations par minute (un battement par seconde), les intervalles exacts entre deux battements varient constamment : parfois un peu plus d'une seconde, parfois un peu moins. Quand ces variations augmentent et diminuent en suivant un rythme régulier, on dit qu'il y a cohérence cardiaque. Cela se produit quand on est calme et détendu. Par contre, en cas de stress, de colère ou de frustration, les variations se produisent de façon chaotique.

A partir de ces constatations, un ensemble de techniques ont été développées afin de susciter la cohérence cardiaque. Cela permet de réduire le stress et l'anxiété et de renforcer le système immunitaire. Une des techniques consiste simplement à ajuster son rythme respiratoire à 6 cycles par minute (inspirations et expirations de 5 secondes chacune).

2.1.6 Mesures de la variabilité du rythme cardiaque

La variabilité du rythme cardiaque fait référence aux variations dans la durée de l'intervalle entre les battements du cœur. En général, les plus grandes variations témoignent d'une capacité de régulation végétative de l'organisme plus élevée et par conséquent, d'une énergie vitale plus développée. C'est pourquoi la mesure de la variabilité de la fréquence cardiaque peut déterminer si une personne est en bonne santé et permet aussi de savoir si une

thérapie sera ou non efficace dans son cas. Différentes méthodes ont été proposées dans la littérature pour calculer cette variabilité. Les méthodes qui relèvent du domaine temporel sont divisées en méthodes statistiques et méthodes géométriques. Les méthodes les plus utilisées sont celles relevant du domaine fréquentiel où l'on fait un calcul du ratio des composantes se trouvant dans les hautes fréquences par rapport à celles qui se trouvent dans les basses fréquences. Les données situées dans la fourchette 0.04 – 0.15 Hz sont du domaine des basses fréquences tandis que celles qui sont situées dans la fourchette 0.15 – 0.40 relèvent du domaine des hautes fréquences. Les données recueillies, étant distribuées de façon aléatoire dans le temps, nous sommes obligés d'utiliser l'algorithme de *Lomb-Scargle* pour trouver ce ratio. La transformée de *Fourier* ne répond pas à cette situation étant donné que les données reçues ne sont pas espacées de façon régulière.

2.1.7 Algorithme de *Lomb-Scargle*

Le traitement des données dans le domaine médical n'est pas une trivialité. C'est un domaine où les données pullulent à un rythme effréné. Outre la grande quantité de données à traiter, on se trouve la plupart du temps confronté à des contraintes de temps réel pour le traitement des données, ce qui complique énormément la tâche dans pareilles situations.

Dans la littérature, différentes méthodes ont été proposées pour arriver à un calcul tout à fait appréciable de la variabilité du rythme cardiaque [19]. Notre point focal, en ce sens, sera les oscillations dans les intervalles qui séparent les battements du cœur. La variabilité du rythme cardiaque est le terme le plus accepté de nos jours pour décrire les variations du rythme cardiaque. Ce qu'il faut retenir est qu'au lieu de mettre le focus sur les battements du cœur, les analyses sont faites de préférence sur les fluctuations entre les différents battements.

Les sections précédentes ont fait l'objet de certains concepts et définitions dans le secteur de la santé. Dans la deuxième section, nous allons présenter d'autres concepts ayant surtout rapport à l'informatique mobile dans le *Cloud*.

2.2 Les réseaux mobiles

Un réseau mobile est un réseau où ses terminaux sont appelés à se déplacer dans le temps. Il existe présentement deux types d'infrastructures réseautiques permettant l'utilisation de ces réseaux, celles des réseaux Wi-Fi et celles des réseaux cellulaires. Pour utiliser un réseau Wi-Fi,

il suffit d'avoir à sa disposition un point d'accès Wi-Fi. Ils sont présents presque partout aujourd'hui mais l'accès à l'utilisation peut être refusé pour des raisons de sécurité. Les réseaux cellulaires sont presque partout aujourd'hui dans le monde. Dans les pays développés, on a accès à un débit relativement élevé avec la troisième et quatrième génération. Les réseaux de quatrième génération peuvent atteindre théoriquement une vitesse de téléchargement allant jusqu'à 1 Gbps.

Les applications mobiles utilisent ces deux types d'infrastructure (Wi-Fi et cellulaire) pour accéder aux centres de données situées dans le Cloud.

2.2.1 Informatique dans le nuage (*Cloud Computing*)

L'informatique dans le nuage ou *Cloud Computing* connait un succès assez considérable aujourd'hui. De grandes compagnies dépensent des centaines de millions de dollars pour mettre en place des parcs informatiques pouvant sauvegarder les données des utilisateurs et permettre en même temps de bénéficier de la toute-puissance de gros serveurs qui sont mis à leur disposition. Certaines entreprises aujourd'hui, au lieu de construire leur propre parc informatique, utilisent les centres de données du *Cloud* pour héberger l'infrastructure informatique de base de l'entreprise. Ainsi leurs employés ne disposent que de simples terminaux pour communiquer avec des serveurs disposés dans le *Cloud*. Il existe d'autres types d'utilisation du *Cloud* et celui sur lequel nous voulons nous attarder le plus revient à la déportation des tâches dans le nuage par rapport à la capacité limitée des terminaux mobiles. Au lieu d'exécuter en local sur un téléphone mobile certaines tâches qui consomment beaucoup de ressources par exemple, il est préférable de les faire exécuter dans le nuage, ce qui apporte un gain considérable en termes de temps d'exécution et de conservation d'énergie.

L'informatique dans le nuage est vue aussi comme une forme particulière de gérance de l'informatique, les clients ne sachant pas exactement où sont placées leurs données. Il n'y a pas un endroit exact pour la localisation de ces données dans le nuage, les centres de données étant éparpillés partout dans le monde.

2.2.2 Caractéristiques de l'informatique dans le nuage

Le *Cloud Computing* ou informatique dans le nuage, pour être accepté par la communauté, doit répondre aux caractéristiques suivantes [9] :

1. **Élasticité**

L'élasticité peut être décrite comme la capacité de pouvoir mettre des ressources supplémentaires à la disposition des applications dépendamment de leurs besoins. Une application se trouvant proche de la saturation se voit attribuer automatiquement des ressources supplémentaires. Dans les cas classiques, le serveur doit être arrêté pour se faire ajouter les ressources nécessaires puis redémarrer par la suite. La virtualisation des ordinateurs dans le *Cloud* permet de mettre fin à cette façon de faire très couteuse en termes de temps pour les utilisateurs. C'est l'une des caractéristiques qui a été mise en relief par Amazon grâce à son offre *EC2* (*Elastic Compute Cloud*) qui fournit une capacité de calcul redimensionnable.

2. **Facturation à la consommation**

L'un des avantages majeurs du *Cloud* est le fait que le client qui l'utilise ne paie que pour ce qu'il consomme, ni plus, ni moins. Il dispose seulement des ressources dont il a besoin. Le fournisseur le fait payer pour sa consommation réelle.

3. **Accès aux services à la demande**

Les fournisseurs du *Cloud* mettent à la disposition des clients des interfaces de communication qui leur facilitent l'accès aux différents services. Le client a l'avantage de pouvoir configurer et utiliser les services dont il a besoin uniquement. Quand il ne veut plus payer pour ce service, il n'a qu'à y mettre fin. En ce sens, le client utilise les services du fournisseur seulement sur demande. Il ne paie que pour un service utilisé et quand il en a effectivement besoin.

4. **Accès haut-débit**

Les grands centres de données des fournisseurs du *Cloud* sont directement connectés au *backbone* internet, ce qui permet au client de bénéficier d'une bande passante très intéressante. Le temps d'accès recherché par les fournisseurs aujourd'hui ne doit pas aller au-delà de 50 millisecondes. Ils travaillent de plein fouet pour pouvoir apporter les services aux clients dans ce délai record.

5. **Abstraction sur la localisation des ressources (non localisées)**

L'utilisateur du *Cloud* n'a aucune connaissance de l'endroit exactement où sont exécutées ses applications, ni de l'endroit de stockage de ses données. Les fournisseurs de services du *Cloud* peuvent déporter leurs ressources dépendamment des requêtes qui leur sont adressées, ce qui leur

permet un équilibrage de la charge par rapport aux ressources disponibles. Peu importe l'endroit d'exécution de l'application dans le monde, l'utilisateur ne doit percevoir aucune différence.

2.2.3 Modèles de services dans le nuage

Trois modèles de services sont offerts dans le nuage : IaaS, SaaS et PaaS.

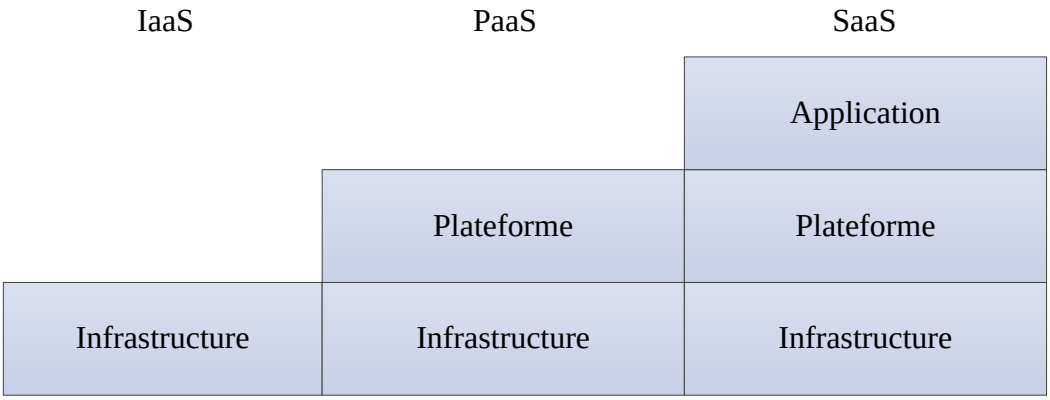

Figure 2.1 Architecture du *Cloud*

SaaS ou Software as a Service

Le SaaS ou Software as a Service, avec sa traduction logiciel en tant que service, est un modèle où le fournisseur est responsable de l'infrastructure, de la plateforme et des applications déployées dans *Cloud*. Les applications peuvent être configurées par le client sans que celui-ci ne puisse contrôler la plateforme ou les infrastructures sous-jacentes. La messagerie Hotmail de Microsoft représente un exemple typique de ces applications configurable par le client.

PaaS ou Platform as a Service

Dans le modèle PaaS ou encore Plateforme en tant que service, le fournisseur est responsable de l'infrastructure et de la plateforme. Le client dispose d'un environnement d'exécution qui lui est rapidement disponible. Il peut créer et faire la maintenance de ses applications dans le *Cloud*. Il peut les déployer et les configurer. Google Apps Engine (GAE) représente un exemple de Plateforme en tant que Service.

IaaS (Infrastructure as a Service)

"L'infrastructure as a Service" est parfois appelé "Hardware as a Service". C'est un modèle où le fournisseur est responsable de l'infrastructure. Les usagers bénéficient des moyens de stockage, des serveurs, des capacités réseaux et d'autres ressources mis à leur disposition. La maintenance et l'hébergement des équipements demeurent à la charge du fournisseur. Le client peut utiliser l'infrastructure disponible pour installer n'importe quel système d'exploitation et/ou autres applications sur les équipements.

2.2.4 Les modèles de déploiement du nuage

Le *National Institute for Standards and Technology* (NIST) définit quatre modèles de déploiement du *Cloud* [20]. Ils sont élaborés dans les sections suivantes:

1. Nuage privé

Dans le cas où l'infrastructure du *Cloud* est utilisée par une seule et même entreprise, le nuage est dit privé [21]. Toute la gestion de l'infrastructure est faite par les employés de l'entreprise contrairement au nuage public. Une grande compagnie peut tout de même décider de sous-traiter une bonne partie ou toute la gestion de son *Cloud* à une entité tierce disposant des compétences requises en la matière. Suivant la politique de la compagnie, le centre de données peut être construit dans leurs propres locaux ou ailleurs.

2. Nuage public

Le nuage public est le cas le plus répandu du *Cloud Computing*. Il peut être défini par rapport au fait que certaines grandes entreprises permettent d'exploiter de grands centres de données à travers internet, les rendant ainsi accessibles à tout le monde [21]. Les entreprises les plus réputées à offrir ce service sont Google, Amazon, et Microsoft.

3. Nuage communautaire

Quand au moins deux entreprises se mettent ensemble pour partager une infrastructure de *Cloud*, il s'agit d'un nuage communautaire. Un ensemble de politiques bien défini par la communauté garantit une exploitation normative des infrastructures. Le partage des ressources disponibles dans le *Cloud* permet à ces entreprises de faire une économie assez intéressante et de garantir une plus grande sécurité des données. Une entité tierce peut avoir un mandat pour gérer l'infrastructure du nuage communautaire moyennant un accord de toutes les parties prenantes.

4. Nuage hybride

Un nuage hybride est un cas de combinaison d'au moins deux des types de *Cloud* mentionnés ci-dessus.

2.2.5 Contraintes liées à la mobilité des terminaux mobiles

Les terminaux mobiles, qui remplissent aujourd'hui les mêmes fonctions que les ordinateurs, sont très limités par rapport aux ressources dont ils disposent. Nous allons présenter dans les sections suivantes quelques-unes de ces contraintes.

1. La bande passante

Étant donné la grande mobilité des usagers des terminaux mobiles, ils se trouvent très souvent connectés à Internet via les réseaux cellulaires tels que les réseaux EDGE, 3G ou 4G. Occasionnellement, il leur arrive d'être connecté via un réseau Wi-Fi. La position de l'usager par rapport à ces réseaux s'est révélée d'une importance capitale. La mobilité de l'usager peut donner lieu à des accès interrompus aux ressources disponibles. Dans le cadre du développement des applications mobiles, les développeurs doivent tenir compte de l'environnement très flexible dans lequel évoluent les usagers.

2. Les ressources matérielles

Les processeurs des terminaux mobiles, la capacité mémoire ou de stockage interne sont nettement en deçà de ce dont disposent les ordinateurs de bureaux et les ordinateurs portables. Étant donné la grande mobilité des usagers, il leur arrive assez souvent de n'être connecté à aucune source d'énergie. Un développeur d'application doit tenir compte de cette capacité limitée en termes de ressources matérielles sur ces dispositifs mobiles.

2.2.6 Informatique mobile dans le nuage

L'informatique mobile dans le nuage est une extension de l'informatique dans le nuage. C'est le fait de mettre à la disposition des terminaux mobiles la puissance de calcul du nuage, l'approche traditionnelle étant l'utilisation par des terminaux se trouvant à des points fixes où l'on n'a pas à gérer la mobilité des nœuds bénéficiaires. Vu la capacité limitée des terminaux mobiles en termes de puissance de calcul et considérant le fait qu'ils ne puissent bénéficier d'une source d'énergie en continue dû à la mobilité, on essaie aujourd'hui de déporter certaines tâches

dans le nuage diminuant ainsi leur dépendance énergétique et améliorant assez souvent leur puissance de calculs.

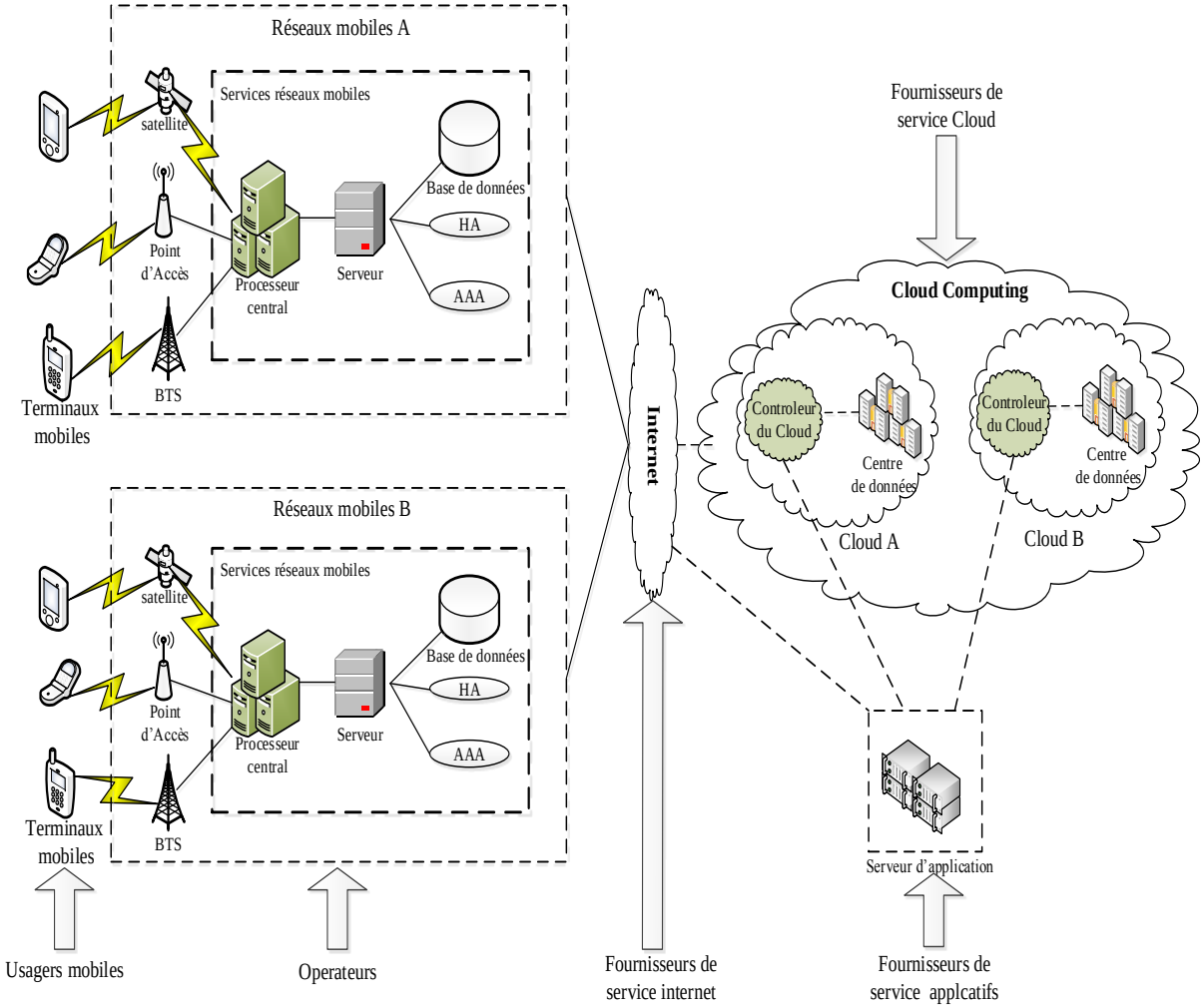

Figure 2.2. L'informatique mobile dans le *Cloud* inspiré de [22]

2.2.7 Informatique dans le nuage vs Informatique mobile dans le nuage

On trouve différentes approches pour définir l'informatique mobile dans le nuage.

1. Assez souvent, on fait référence au terme informatique mobile dans le nuage lorsqu'on utilise un terminal mobile pour communiquer avec des serveurs se trouvant dans de gros centres de données, l'accès à ces ressources se faisant via le réseau 3G ou Wi-Fi [22]. C'est le cas par exemple du service Gmail de Google, Facebook, Twitter qu'on utilise sur les téléphones mobiles ou les tablettes.

2. Dans certains cas, l'informatique mobile dans le nuage fait référence à un réseau pair à pair construit avec les téléphones mobiles servant de fournisseurs de ressources. Les terminaux mobiles qui se trouvent dans le voisinage peuvent mettre leurs ressources à disposition d'autres téléphones en cas de besoin.

2.3 Les Services Web

Un service web peut être considéré comme un programme informatique accessible à travers un réseau (internet ou intranet d'entreprise) et qui ne dépend d'aucun système d'exploitation ou de langage de programmation. Sa procédure de fonctionnement est la suivante :

1. Le service Web définit un format pour les requêtes et les réponses.
2. Le programme client envoie une requête au serveur.
3. Le serveur effectue une action par rapport à la requête reçue
4. Le serveur renvoie la réponse au programme client
5. Le client utilise le résultat reçu.

2.3.1 Le protocole SOAP

SOAP est un protocole d'échange d'information introduit par Microsoft et IBM et standardisé par le W3C. Il est utilisé dans les environnements distribués pour invoquer des méthodes définies à distance. Un client appelant un service Web défini sur un serveur lui envoie une requête SOAP et le serveur lui envoie une réponse utilisant le protocole SOAP. Les messages SOAP sont encodés dans des documents XML ayant la forme présentée à la figure ci-dessous.

```
<soap:Envelope xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle=http://www.w3.org/2001/12/soap-encoding>
  <soap:Body xmlns:m=http://carre.org/stress">
    <m:OtenirNiveauDeStress>
      <m:battementCardiaque>1299</m:battementCardiaque>
    </m:ObtenirNiveauDeStress>
```

Figure 2.3. Requête SOAP

```
<soap:Envelope xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle=http://www.w3.org/2001/12/soap-encoding>
  <soap:Body xmlns:m=http://carre.org/stress”>
    <m:OtenirNiveauDeStressResponse>
      <m:niveauStress>89</m:niveauStress>
    </m:ObtenirNiveauDeStressResponse>
```

Figure 2.4. Réponse SOAP

Pour décrire les services Web, le protocole SOAP utilise le WSDL (Web Service Description Language). C’est un document écrit en XML qui donne des informations concernant l’emplacement du service web, les méthodes qui y sont exposées et les paramètres utilisées dans ces méthodes. L’élément le plus important à regarder dans un document WSDL est le type de port qui donne les points de connexion au service Web. Il est comparable à une bibliothèque de fonctions par rapport à l’approche qui est faite d’habitude dans les langages de programmation traditionnels.

Pour faire la publication des services Web SOAP, nous utilisons aujourd’hui UDDI (Universal Description, Discovery and Integration). C’est une sorte d’annuaire qui permet de référencer gratuitement un service Web.

Un message SOAP est constitué de trois parties : 1) une enveloppe (soap-Envelope) contenant un en-tête optionnel et un corps obligatoire, c’est l’élément racine d’un message soap, 2) l’encodage pour les différents types de données utilisés, 3) l’invocation des méthodes. Les principaux avantages à utiliser le protocole sont : son indépendance par rapport aux langages de programmation, aux plateformes de déploiement et à la couche transport. L’utilisation du standard XML peut être vue comme un avantage et un inconvénient. Comme avantage, XML permet d’avoir un format d’échange standardisé entre les entités de programmation. Comme principal inconvénient, il est lourd comparativement à d’autres langages assez légers comme JSON par exemple. Ce qui peut engendrer des problèmes de performance. Un autre inconvénient du SOAP est le coût de la mise en œuvre qui est souvent très élevé et la maintenabilité quand on

doit faire certaines modifications. Ces inconvénients ont remis en cause les avantages du protocole et on a vu les gens se pencher ces dernières années sur le style architectural REST (Representational State Transfer) qui devient du coup un concurrent direct de SOAP. On va présenter dans les lignes qui suivent le style architectural REST qui a été introduit au cours de l'année 2000 par Roy Fielding [23].

2.3.2 Architecture REST

Le style architectural REST devient de plus en plus un standard et se fraie un chemin assez intéressant aujourd'hui. REST n'est ni un protocole ni un format, c'est un style d'architecture logicielle, c'est-à-dire un ensemble de contraintes bien adapté au Web. Les principaux objectifs qu'on cherche à atteindre à travers le style REST sont l'évolutivité, la maintenabilité, le couplage faible des services, l'encapsulation des services hérités. Les applications qui répondent aux contraintes imposées par ce style architectural sont dites "RESTFul". Ce style architectural a été introduit par Roy Fielding dans son travail de doctorat en 2002. Ce dernier est un co-fondateur du projet Apache et a travaillé sur des sujets variés touchant le Web tels que HTTP, HTML, URI, etc. Pour Roy Fielding, une application web n'est autre qu'un réseau de pages Web. C'est une machine à états finis où les pages Web représentent les états et les hyperliens représentent les transitions entre ces états. Il s'en va vers un modèle permettant de faire une description du mode de fonctionnement qu'on devrait trouver sur le Web aujourd'hui.

L'une des contraintes du style architectural REST est que les applications doivent être constituées de clients de serveurs séparés par une interface uniforme.

L'information de base dans le style architectural REST est appelée ressource. Le style architectural REST fournit une interface simple et uniforme cristallisée par l'utilisation des opérations GET, PUT, POST, DELETE et met un accent sur la représentation des ressources. On entend par ressource sur la toile tout ce qui peut être référencé par un lien. Ces ressources sont identifiées par des URI (Uniform Resource Identifier) qui facilitent leurs manipulations par les composantes de l'architecture qui transfèrent aux clients leurs représentations par la suite.

Le style architectural REST s'impose de plus en plus sur le marché aujourd'hui par rapport à SOAP. Il présente en effet de nombreux avantages parmi lesquels l'utilisation d'une interface uniforme pour accéder aux services Web. Ils n'imposent pas l'utilisation d'un entête XML aux fournisseurs qui peuvent utiliser JSON, ce qui entraine une meilleure utilisation de la bande passante. Du point de vue de l'interopérabilité, REST présente beaucoup plus d'avantages par rapport à SOAP. Tout ce dont vous avez besoin sur votre terminal pour utiliser SOAP est la pile http contrairement à SOAP qui fait appel à une trousse à outils de votre plateforme. L'un des inconvénients majeurs de ces deux styles architecturaux est leur limitation en matière de sécurité.

2.4 Déportation de calculs (*Computation Offloading*)

Le terme *Computation Offloading* ou déportation de calculs se trouve étroitement lié à l'informatique dans le nuage. C'est un mécanisme qui consiste à déporter l'exécution d'algorithmes lourds sur des serveurs distants en général dans le nuage [7]. Ces tâches, la plupart du temps consomment beaucoup de ressources, font appels à des processeurs d'une vitesse non disponible sur les téléphones. Aujourd'hui, il y a de nombreuses recherches qui analysent la meilleure approche pour déporter ces algorithmes. Les deux qui sont retenues ici pour notre revue de littérature et qui sont les plus considérées sont le partitionnement du code et la virtualisation. Ces deux approches seront exposées à travers les lignes qui suivent.

Pour résoudre le problème des ressources limitées sur le téléphone, on sépare les applications en différentes parties qui sont exécutées soit sur le téléphone, soit dans le nuage. Le partitionnement du code peut être fait de façon statique ou de façon dynamique. En ce qui concerne la virtualisation, il existe une approche de migration de la machine virtuelle dans le nuage et une approche de clonage du téléphone mobile dans le nuage.

2.4.1 Approche Client-Serveur

L'approche client-serveur est une approche statique faisant appel à un partitionnement du code de l'application par le développeur. Pour ce faire, on utilise des protocoles tels que RPC (*Remote Procedure Call* ou appel de procédure à distance), RMI (*Remote Method Invocation* ou Invocation de méthode à distance), et les sockets. Les services qui seront utilisés dans ces cas doivent être préalablement installés sur les serveurs distants. Dans le cas de l'informatique

mobile dans le nuage, ceci constitue un désavantage à cause de la nature mobile des périphériques qui mettent leurs ressources à disposition du client.

2.4.2 Migration de machines virtuelles

La migration de machine virtuelle fait référence à un transfert de l'image qu'on a en mémoire d'une source vers une destination sans interruption. On parle de migration à chaud car cette migration se fait pendant que le système est en production. Les pages en mémoire de la machine virtuelle sont copiées au préalable sans aucune interruption du système d'exploitation ou de ses applications, donnant ainsi l'illusion d'une migration sans coupure. L'avantage qu'on a avec cette façon de faire la migration est qu'il n'y a aucune nécessité de modifier le code lors de sa déportation dans le nuage. Le désavantage qui sort de l'utilisation de cette méthode est la lenteur de la migration des machines virtuelles aussi bien que la charge qui peut être un handicap majeur dépendamment de sa taille.

Une autre solution a été proposée par Satyanarayan et al. [24] qui propose de faire la déportation à travers un petit nuage similaire à un centre de données situées dans une zone spécifique et qui lui-même se trouve connecté à un grand nuage disposant de beaucoup plus de capacités à travers l'internet.

2.4.3 Ericsson Mobile Health (EMH)

EMH est une solution architecturale proposée par Ericsson pour le développement des applications mobiles dans le domaine de la santé. Cette architecture a été proposée dans le cadre du projet *MobiHealth* mis en place en Europe [25]. La figure 2.5 illustre cette architecture.

Nous avons identifié dans ce système les capteurs biologiques qui sont portés par les patients. Les capteurs biologiques sont munis d'un module Bluetooth leur permettant d'acheminer les données à destination soit vers un téléphone mobile ou un autre périphérique, lui-même aussi muni d'un module Bluetooth comme celui proposé par Ericsson. Le signal biologique qui est capté est ensuite transféré au périphérique qui est responsable de son acheminement ou de sa sauvegarde au cas où le réseau ne serait pas disponible pour l'acheminement. Les valeurs qui sont prélevées sont transmises à travers un réseau mobile à un nœud central.

L'objectif principal de l'architecture présenté par Ericsson est de permettre à un client de pouvoir bénéficier des services de santé en tout temps et en tout lieu. Le client, muni d'un capteur biologique, achemine à travers un réseau mobile les données recueillies à un nœud central. Un périphérique développé par Ericsson a été proposé pour faciliter les utilisateurs qui ne sont pas assez doués pour ces technologies de pouvoir faire une meilleure utilisation. Ces utilisateurs n'auront aucune configuration à faire par rapport au module Bluetooth.

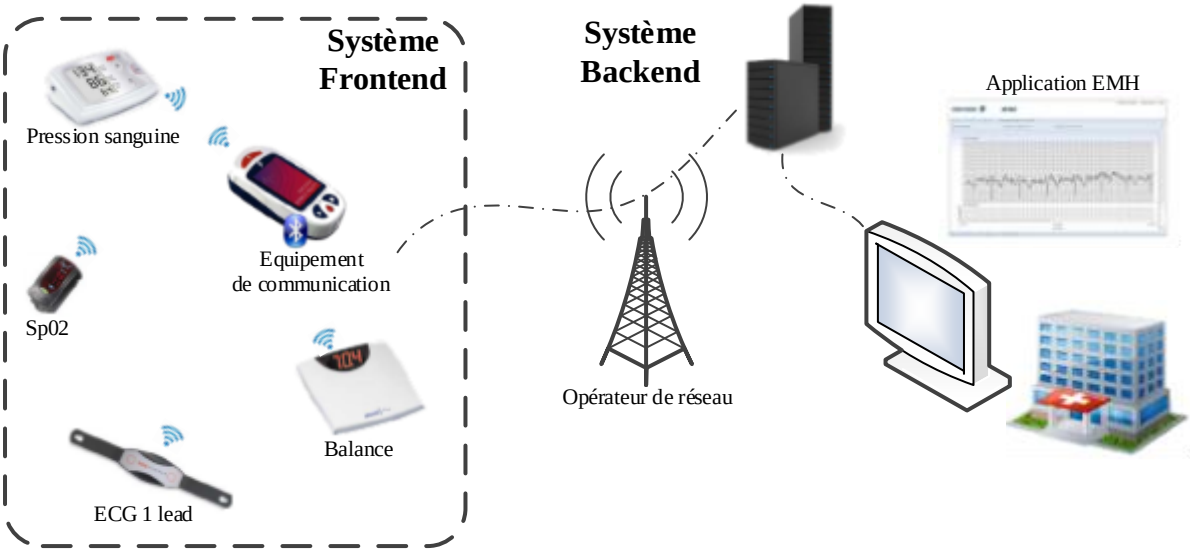

Figure 2.5. Ericsson Mobile Health inspiré de [25]

Dans l'architecture proposée par Ericsson, aucune analyse n'est faite sur les données qui sont reçues des capteurs biologiques. Les données une fois reçues sont automatiquement acheminées au nœud central et les résultats transmis au personnel médical qui décideront en fonction des résultats après traitement sur le nœud central. Ce qu'il faut signaler par rapport à cette architecture est que le nœud central est muni de capacités extraordinaires pour l'analyse des données provenant du patient, d'où une déportation sans aucune analyse préalable des tâches de calculs. Le nuage utilisé peut être vu comme faisant partie d'un nuage privé appartenant à une quelconque entreprise. C'est un nuage qui, dans la plupart des cas, appartient à une institution médicale ou construite à cette fin, la gestion des données médicales demandant un service de sécurité assez robuste. Cette architecture peut s'étendre aussi à un nuage public, vu l'ampleur et la maturité de l'informatique dans le nuage aujourd'hui et les services de sécurisation qui y sont offerts. D'autres analyses sont à faire étant donné qu'il s'agit de données médicales.

Le rôle joué par le périphérique proposé par Ericsson peut être joué par un téléphone mobile vu toute l'intelligence qui leur est implantée. Ces téléphones sont par ailleurs qualifiés de téléphones intelligents. Un développeur conscient des enjeux devra prendre des mesures afin de permettre au client d'utiliser le module Bluetooth du téléphone de façon efficace. La crainte nourrie par Ericsson à cet effet peut être dissipée par une bonne programmation de l'application sur le téléphone, le développeur pouvant activer ou désactiver le module Bluetooth à n'importe quel moment donné.

2.4.4 MAUI (Mobile Assistance Using Infrastructure)

Eduardo et al. [16] ont proposé MAUI qui est un système qui permet de déporter des tâches de calculs dans le nuage. L'objectif visé à travers ce système est surtout la conservation de l'énergie au niveau du terminal mobile. **MAUI** combine à la fois la migration et le partitionnement du code. Ce dernier se fait à chaud, au moment de l'exécution de l'application.

Afin de pouvoir atteindre son objectif de partitionnement sur des plateformes différentes, MAUI apporte des réponses à certains défis tels que énumérés ci-dessous :

1) La portabilité du code fait référence au fait que le même code d'un programme puisse être exécuté dans des environnements différents sans avoir besoin d'être recompilé. Le principal avantage avec la portabilité est la maintenabilité du code par le fait que la même version est implémentée sur toutes les plateformes. Afin de pouvoir exécuter ses tâches sur différents processeurs, étant donné que les terminaux mobiles(ARM) utilisent typiquement des architectures différentes de celles des serveurs (x86), MAUI utilise le code managé de Microsoft lui permettant d'exécuter le même code sur les différentes plateformes de Microsoft (mobile, serveurs, tablettes, etc.).
2) MAUI utilise l'approche d'annotation pour identifier les méthodes qui peuvent être déportées dans le *Cloud*. Tout développeur utilisant MAUI doit modifier son code source en ajoutant des métadonnées tels que des attributs "remoteable" à toutes méthodes ou classes qui peuvent être déportées dans le *Cloud*.
3) MAUI génère une enveloppe à la compilation. Elle sert à sauvegarder l'état de l'application juste avant la déportation dans le *Cloud* et sert aussi à récupérer son état après l'exécution d'une tâche dans le *Cloud*. Pour sauvegarder cet état, ils apportent deux modifications aux niveaux de la signature d'une méthode, leur ajoutant un

argument servant à recevoir l'état de l'application à l'aller et une valeur de retour servant à recueillir l'état au retour.

4) MAUI génère deux proxies à la compilation, une devant être exécutée sur le serveur et l'autre sur le terminal mobile. Ces proxies implémentent les décisions qui doivent être prises par le solveur, afin de décider à partir des données recueillies du profiler de l'endroit d'exécution des tâches qui peuvent être déportées.

MAUI peut être décrit comme un système dynamique qui décide de l'endroit d'exécution du code. Ils utilisent la plateforme applicative .NET de Microsoft pour construire leur système profitant ainsi de l'approche de code géré de ce dernier. L'utilisation du code géré leur permet de faire de la portabilité qui consiste en l'implémentation du même code sur différentes plateformes. En effet, ils peuvent utiliser le même code développé pour le téléphone dans le nuage sur les différents types de terminaux mobiles utilisant les systèmes d'exploitation de Microsoft, ce qui constitue un énorme avantage. Les fonctions qui peuvent être déportées doivent être marquées par les développeurs pour que l'analyse de déportation puisse être faite au moment de l'exécution. Cette analyse de déportation est basée sur les conditions du réseau et de la capacité du téléphone au moment de l'exécution du code. Deux versions du code sont ainsi créées, une pouvant être exécutée dans le *Cloud* et une autre sur le terminal mobile.

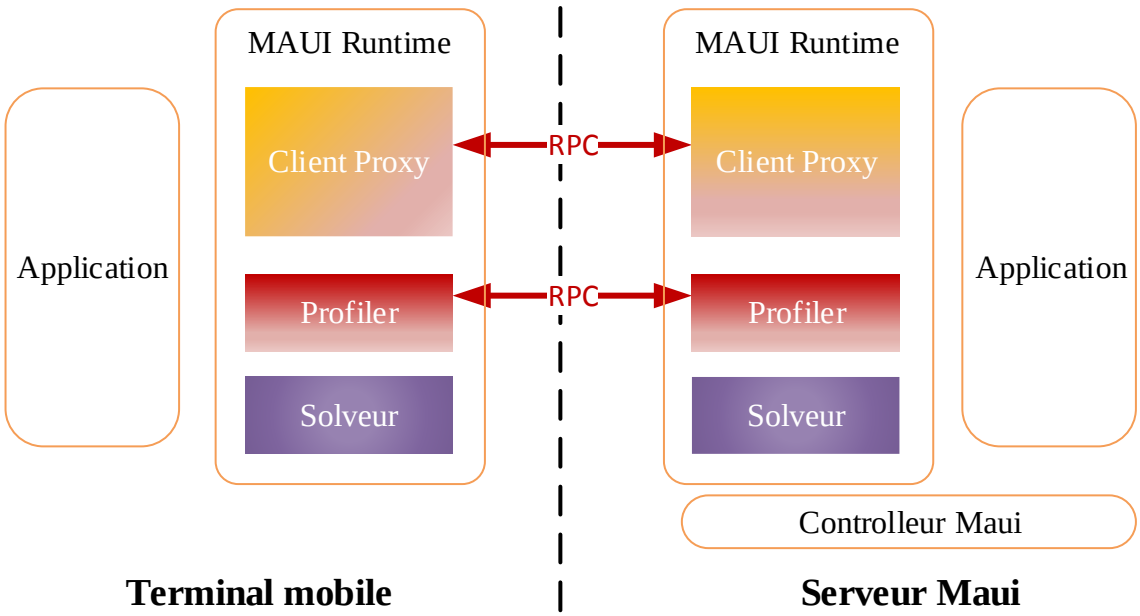

Figure 2.6. Architecture de MAUI inspiré de [16]

L'architecture du système est présentée ci-dessus.

Les principales composantes du système peuvent être ainsi présentées :

1. Le profiler qui sert à collecter les données concernant les ressources utilisées dans le système. Trois facteurs sont considérés afin de prendre leur décision : les caractéristiques de consommation d'énergie de l'appareil mobile, les caractéristiques des applications utilisées, et les caractéristiques du réseau tels que la bande passante, la latence et le taux de perte des paquets.
2. Le solveur utilisé pour analyser les données du profiler et qui sert à décider de l'endroit d'exécution des différentes tâches. C'est aussi au niveau du solveur qu'on fait l'analyse pour voir si une méthode peut être déportée dans le *Cloud*. Son objectif est de trouver la meilleure stratégie de partitionnement du code afin de minimiser la consommation d'énergie sur le terminal mobile sujet à des contraintes de latence. Afin de conserver l'énergie disponible sur le terminal mobile, le solveur est seulement implémenté au niveau du serveur.
3. Les deux proxies qui gèrent le contrôle et le transfert de données se basant sur les décisions qui sont prises par le solveur. Le proxy se trouvant sur l'appareil mobile fait la sérialisation avant d'appeler une méthode et la dé-sérialisation de l'état de l'application après exécution de cette méthode. Le proxy se trouvant sur le serveur fait aussi de la sérialisation avant de passer le contrôle au proxy en local après exécution à distance.
4. Le contrôleur est seulement implémenté au niveau du serveur et fait de l'authentification des usagers et l'allocation de ressources.

MAUI fournit un environnement de programmation où les développeurs peuvent annoter les méthodes d'une application sujettes à la déportation. Ils utilisent le mécanisme de réflexion leur permettant d'analyser le code en exécution et de prendre certaines décisions par rapport à certaines données internes au code. A chaque fois qu'une de ces méthodes sera appelée, MAUI fait une analyse pour décider de son endroit d'exécution. Une fois cette méthode exécutée, MAUI fait appel à son "profiler" pour enregistrer les informations nécessaires à la prédiction du temps d'exécution de la méthode dans le futur.

2.4.5 Approche de clonage du mobile dans le nuage (*Clone Cloud*)

Chun et al. introduisent l'idée d'améliorer la performance des terminaux mobiles en utilisant leur architecture de clonage du téléphone dans le nuage. En effet, ***Clone Cloud*** utilise aussi la migration de machine virtuelle pour déporter une partie de leurs applications à travers un réseau 3G ou Wi-Fi [1, 17, 18]. Ils créent des clones virtuels de l'environnement d'exécution du terminal mobile dans le nuage et leur transfèrent les taches d'exécution par la suite. A proprement parler, ils déportent les tâches d'exécution du terminal mobile sur une infrastructure très rapide hébergeant à l'avance un ensemble de clones. L'avantage avec le Clone *Cloud*, contrairement à MAUI par exemple, est qu'ils n'ont pas besoin de marquer des fonctions à être déportées car ils font un clonage complet du téléphone avant d'exécuter les applications dans le nuage. Tout est fait dans le nuage et seulement le résultat est renvoyé au terminal à des fins d'affichage. Leur modèle de coût permet de faire de la prédiction par rapport au temps d'exécution du programme dans le nuage et son temps d'exécution sur le téléphone. Leur système a été testé en utilisant des téléphones Android avec des clones s'exécutant sur un PC Dell utilisant le système d'exploitation Ubuntu. Pour tester leurs applications, ils ont fait usage de trois applications assez gourmandes en ressources, un numériseur de virus, un logiciel de recherche d'images et un autre logiciel visant la publicité. Par défaut, ils assument que leur environnement de machine virtuelle est fiable. Des travaux supplémentaires doivent être encore faits pour valider leur approche. Ils doivent tenir compte des environnements de travail qui ne sont pas forcément fiables.

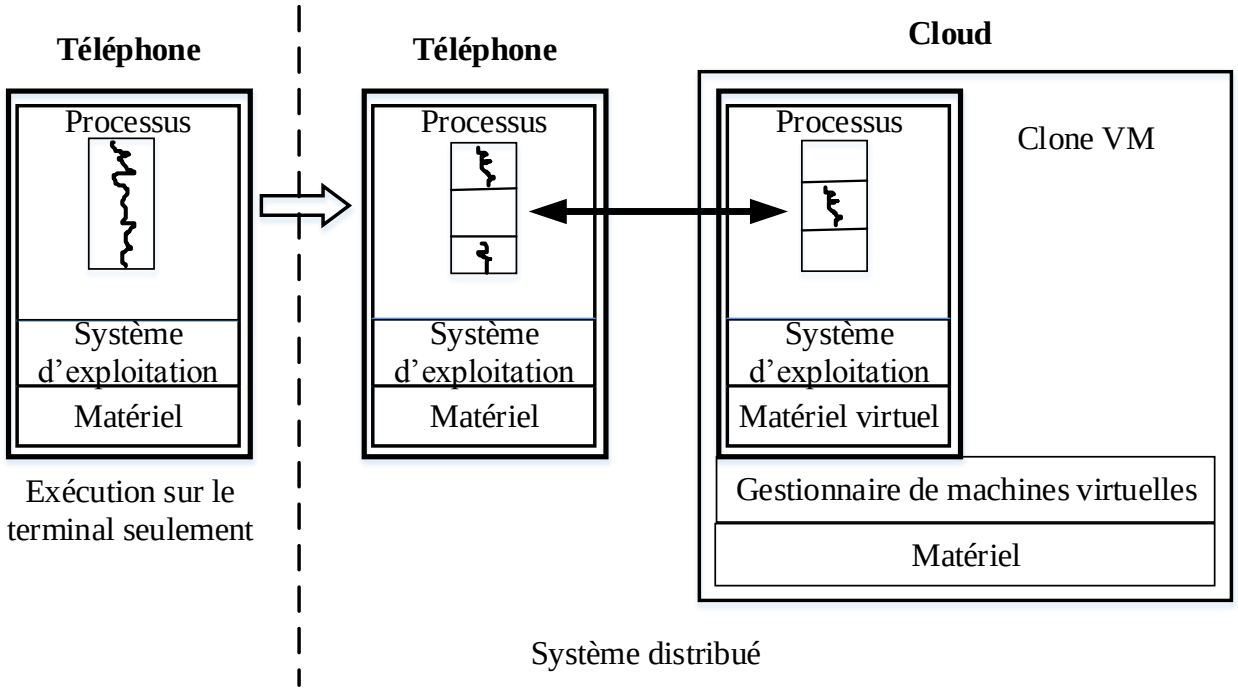

Figure 2.7. Architecture de *Clone Cloud* inspiré de [18]

2.4.6 Cloudlet based resource-rich mobile Computing

L'informatique dans le nuage consiste à déporter dans des centres de données distants de capacité infinie des traitements informatiques en général effectués sur des ordinateurs personnels. Un *Cloudlet* utilise à peu près la même approche mais fait la déportation dans des réseaux privés situés dans les parages. Le *Cloudlet* peut être vu comme le nuage informatique ramené au niveau d'une boite [3, 26, 27]. L'approche est de pouvoir profiter au maximum de sa proximité avec les équipements, ce qui permet de diminuer la latence comparativement au *Cloud* qui se trouve le plus souvent à une grande distance de l'usager. Le délai s'avère le plus souvent insignifiant vu la distance qui sépare terminaux mobiles de ces ordinateurs situés dans ces réseaux locaux. Pour mettre en place un *Cloudlet*, il suffit d'avoir du courant électrique, de l'internet, d'un ordinateur ou d'un ensemble d'ordinateurs assez puissant pour fournir les services nécessaires. Le terminal de l'usager fonctionne dans ce cas comme un client léger qui utilise le *Cloudlet* pour l'ensemble de ses traitements.

La proximité physique est le facteur clé qui pourrait expliquer la diminution du temps d'exécution par rapport au *Cloud*. Mais assez souvent les utilisateurs n'ont pas de *Cloudlet* à leur

disposition et dans ce cas l'application fait choix du *Cloud* si les analyses le révèlent plus rapide par rapport au temps d'exécution sur le téléphone.

Comme on peut le constater, l'approche du *Cloudlet* s'avère intéressante mais il reste malheureusement beaucoup à faire à ce niveau. Les services qui sont exécutés sur le téléphone ne sont pas disponibles dans le *Cloud*. Pour les rendre disponible, il y a malheureusement encore du chemin à parcourir. L'approche qui prévaut aujourd'hui est la création de machines virtuelles sur des réseaux préparés à cette fin. Une fois ces machines créées, le système doit d'abord les lancer et par la suite lancer les services qui doivent être utilisés. Cela peut prendre un temps non négligeable. Une approche beaucoup plus intéressante doit être trouvée afin de diminuer le temps d'exécution qui dans ces cas s'accompagnent d'autres tâches qui peuvent augmenter ce temps de façon considérable.

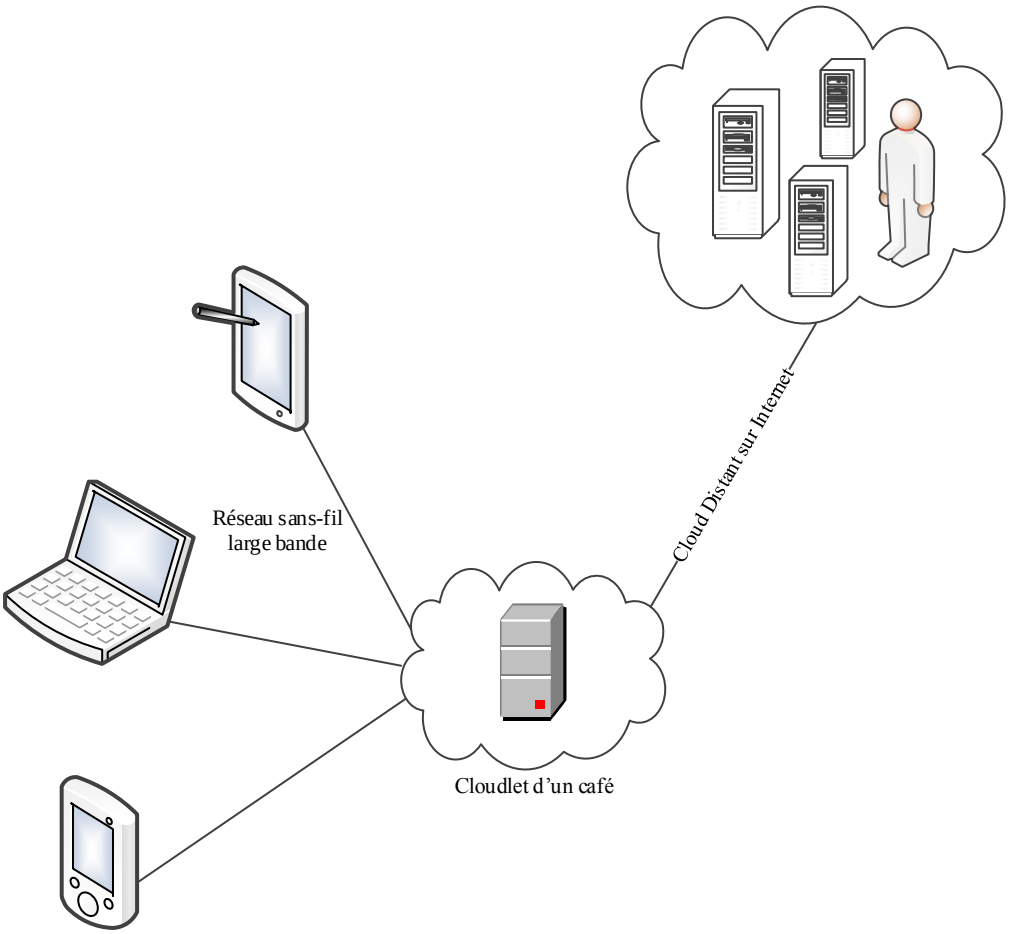

Figure 2.8 Concept du *Cloudlet* inspiré de [27]

2.4.7 MobiCloud

MobiCloud fait usage de l'informatique dans le nuage dans les réseaux mobiles ad hoc (*Manet*) de façon sécuritaire [28]. Les réseaux mobiles ad hoc deviennent des fournisseurs de services. Chaque nœud mobile joue le rôle d'un fournisseur de service dépendamment des ressources qui lui sont disponibles lors d'une éventuelle sollicitation.

2.4.8 Utilisation de la bande passante

L'utilisation de la bande passante pour prendre des décisions concernant la déportation de calcul a été analysée par Wolski et al. [29]. Ils considèrent que la déportation de calcul doit être accentuée sur trois prédictions :

1. une prédiction du temps d'exécution en local;
2. une prédiction du temps requis pour faire les traitements à distance une fois les données disponibles sur la machine à distance;
3. Une prédiction du temps requis pour transférer les données du téléphone mobile au nuage et pour la récupération des résultats sur le téléphone mobile.

La prédiction de la bande passante qui sera utilisée s'avère être une tache assez délicate. Pour ce faire, ils se sont basés sur des analyses statistiques de base.

2.4.9 *Ad-Hoc Mobile Cloud*

Un nuage mobile ad-hoc représente un réseau de terminaux mobiles qui servent de fournisseurs de services en exposant leurs ressources à d'autres terminaux mobiles comme c'est fait en général dans le *Cloud* [30]. Ce type de nuage peut être intéressant dans le cas où un utilisateur se trouverait isolé, n'ayant aucun accès internet et ayant dans son giron un ensemble de personnes munies de terminaux mobiles. Le cas où le client serait en voyage et ne voudrait pas payer une somme exagérée pour l'utilisation des données du roaming peut aussi être considérée. La déportation de calcul sur ce réseau ad-hoc mobile peut s'avérer intéressant dans le cas où l'utilisateur aurait besoin de se décharger de certaines tâches pour des raisons de performances ou de conservation de l'énergie.

Huerta-Canepa et Lee [30] présentent un canevas pour la création de ces fournisseurs virtuels de nuage informatique. Ce canevas imite à peu près la mise en place des nuages

traditionnels en utilisant des terminaux mobiles situés dans le voisinage de l'utilisateur. L'approche proposée permet d'éviter une connexion à un nuage à infrastructure fixe en permettant de profiter tant bien que mal de la même approche de déportation de calculs dans le nuage.

Ce travail mérite quand bien même de l'approfondissement car il peut être effectivement utilisé dans des cas particuliers où l'utilisateur n'aurait aucun grand fournisseur de services à sa disposition. Malgré la viabilité de l'approche, le principal problème reste et demeure encore et encore la limitation en termes de ressources des terminaux mobiles. Faire la délocalisation sur des ressources à peu près de même capacité peut se révéler pas trop intéressante. S'il est possible d'économiser de l'énergie sur le terminal en local, il sera assez rare les cas où l'on pourra avoir une plus grande performance.

2.5 Analyse des travaux

Dans ce chapitre, nous avons passé en revue quelques concepts clés liés au domaine de la santé liés à notre travail de recherche. Ensuite nous sommes passés à travers différents travaux qui sont fait dans le domaine du *Cloud Computing* de façon à améliorer la performance des applications sur les terminaux mobiles. Nous avons vu qu'il y a différentes approches qui ont été faites pour essayer de résoudre le problème. En premier lieu, nous avons parlé du partitionnement du code qui essaie de diviser les tâches en petits morceaux, certaines devant être exécutées en local et d'autres dans le *Cloud*. Cette méthode exploite la portabilité de certains langages pour faire exécuter le même code sur différentes plateformes. Le partitionnement peut être statique ou dynamique, ce dernier nécessitant une analyse à chaud sur chaque petit morceau pour la prise de décision de l'endroit d'exécution. Nous avons regardé en profondeur l'architecture du système MAUI, qui fait du partitionnement dynamique. Cette architecture est présentée par Eduardo dans [16]. L'auteur utilise le canevas .NET de Microsoft pour faire son travail. Ce canevas ne fonctionne que pour les plateformes de Microsoft. Les plus grands compétiteurs du marché mobile aujourd'hui tels qu'Android et iOS ne supportent pas ce canevas, on est obligé de passer à une autre implémentation. Il faut dire aussi qu'il y a un problème assez particulier sur le marché mobile aujourd'hui en ce qui a trait à la portabilité dans toute sa plénitude, chaque plateforme utilisant un langage de programmation qui lui est propre. C'est le cas de Java pour Android, Objective C pour iOS et surtout C# pour Windows Phone. Il n'est pas possible aujourd'hui de

prendre une application native, fonctionnant sur Android et la faire fonctionner sur iOS ou Windows Phone, ou fonctionnant sur iOS et la porter sur Windows Phone et ainsi de suite.

La deuxième approche qui a été proposée dans la littérature consiste à cloner un terminal mobile dans le *Cloud* pour en jouir de la toute-puissance de ce dernier. Cette approche utilise à la fois le partitionnement du code et la migration de machines virtuelles. Profitant du même environnement du téléphone dans le *Cloud*, le développeur n'a vraiment aucun changement à faire du point de vue de l'implémentation. Le même code fonctionnant sur le téléphone fonctionne tel quel sur son clone. Le problème avec cette approche est le temps de migration de ces machines virtuelles. Pour continuer avec l'exécution d'un programme, le système doit sauvegarder tout l'état du terminal mobile et l'envoyer au *Cloud*, qui doit utiliser ce même état pour exécuter une requête et ensuite envoyer la réponse au terminal mobile. À date, les résultats obtenus ne sont pas assez par rapport à cette proposition. En outre, il reste encore d'autres réponses à apporter par rapport au clonage du terminal mobile dans le *Cloud*. Cette étude est encore à ces débuts et nécessite encore de profondes analyses.

La troisième approche que nous avons identifiée est l'utilisation d'un réseau Wi-Fi situé dans le voisinage du téléphone au lieu d'envoyer la requête dans le *Cloud*. Dans ce cas, une entreprise quelconque peut mettre un centre de données à la disposition des usagers. On parle dans ce cas de *Cloudlet* pour dire que c'est un petit *Cloud*, un *Cloud* situé dans le voisinage et pas aussi puissant que le *Cloud* traditionnel. Cette approche nécessite aussi la migration de machines virtuelles et le partitionnement de code. L'approche d'exécution, une fois les services mis en place comme dans le cas du *Cloud* traditionnel, se fait de la même façon. L'évaluation de performance pour le temps d'exécution de cette approche n'est jusqu'à aujourd'hui pas concluante. Cette approche dépend du temps de migration de la machine virtuelle qui est assez élevé.

CHAPITRE 3
ARCHITECTURE D'UNE APPLICATION MOBILE BASÉE SUR LE CLOUD

Dans ce chapitre, nous présentons l'architecture générale d'une application mobile qui s'appuie sur le *Cloud* pour atteindre ses objectifs de performance. Cette architecture sera conçue pour les applications mobiles dans le domaine de la santé. Nous commencerons par présenter d'abord les requis fonctionnels et non fonctionnels. Un prototype de l'application suivra immédiatement après. Ensuite nous enchainerons avec l'architecture de l'application et ses différentes composantes.

3.1 Les requis de l'architecture proposée

Les principaux objectifs que nous poursuivons ici sont les suivants :

- Proposer une architecture qui tient compte de la limitation des ressources disponibles sur les terminaux mobiles pour les applications mobiles. Le contexte de développement de ces applications relève du domaine de la santé et le *Cloud* doit être utilisé pour améliorer la performance des applications.
- Proposer un système d'aide à la décision pouvant nous permettre de décider du meilleur endroit d'exécution de nos tâches, soit sur le terminal mobile ou sur le téléphone.

Une fois nos objectifs clairement définis, nous pouvons déterminer les différentes fonctionnalités du système. La réalisation de notre conception architecturale nous amène à considérer deux groupes de requis : les requis fonctionnels et les requis non fonctionnels. Ces derniers sont élaborés dans les sections suivantes.

3.1.1 Requis fonctionnels

Le calcul du niveau de stress par la méthode du *biofeedback* est la tâche retenue pour faire notre analyse par rapport à l'endroit d'exécution, soit sur le terminal mobile ou dans le *Cloud*.

3.1.2 Requis non fonctionnels

Les requis non fonctionnels du système peuvent être énumérés comme suit:

- la réduction du temps de réponse des tâches applicatives à l'aide du *Cloud* (*RNF. 1*);
- l'interopérabilité du système avec les entités se trouvant dans le *Cloud* (*RNF. 2*);
- la réutilisation de code provenant des systèmes hérités (*RNF. 3*);
- l'adaptation rapide au changement d'environnement (*RNF. 4*);
- la réduction de la taille des entêtes utilisés par le système (*RNF. 5*).

Le temps de réponse peut être réduit en utilisant des ressources externes situées dans le *Cloud*. Le système doit constamment chercher à améliorer la performance de l'application au niveau du temps de réponse. Si le système prévoit que le temps de réponse d'une requête envoyée dans le *Cloud* sera inférieur à celui qui sera obtenu en faisant exécuter la même tâche sur le terminal mobile, la requête sera envoyée dans le *Cloud* puisque nous cherchons constamment à améliorer la performance du système.

L'interopérabilité permet à l'application de pouvoir échanger des messages avec des entités externes gérant les services se trouvant dans le *Cloud*. Elle facilite l'accès aux services Web situés dans le *Cloud* par les terminaux mobiles indépendamment des plateformes utilisées. Pour réaliser ce requis, il importe de choisir un style architectural qui permet la communication entre l'application cliente située sur le terminal mobile et les entités situées dans le *Cloud*.

Le système doit pouvoir utiliser d'autres codes provenant des systèmes hérités et utilisés par d'autres plateformes telles que l'iOS d'Apple, BlackBerry et autres. Les librairies existantes des systèmes hérités permettent une uniformisation du fonctionnement des codes se trouvant au cœur de l'application. Elles permettent aussi la réutilisation des codes qui ont été développés dans le temps et qu'on ne soit pas obliger de les écrire dans un nouveau langage.

Le système doit pouvoir s'adapter rapidement aux changements d'environnement qui peuvent arriver à n'importe quel moment de l'exécution de l'application. Nous pouvons prendre en exemple le passage de l'usager d'un réseau A à un réseau B.

Le système doit maintenir des entêtes avec des tailles raisonnables pour minimiser le temps d'acheminement des paquets dans le *Cloud*.

3.2 Présentation du prototype

Le prototype que nous avons développé pour illustrer notre approche architecturale est une application mobile de gestion du stress par la méthode du *biofeedback*. Cette application est exécutée sur un terminal mobile. Elle reçoit des données d'un module Bluetooth attaché à des capteurs biologiques qui prélèvent des données provenant du corps humain. Un module a été développé sur le terminal mobile pour gérer la communication avec les capteurs biologiques. Une fois les données recueillies par le module d'acquisition de données, elles sont envoyées au module de décodage via l'interface *JNI* qui permet d'accéder au code écrit avec le langage de programmation C (les systèmes hérités).

Notre application a été conçue pour calculer le niveau de stress par la méthode du *biofeedback*. Notre objectif de recherche est de décider de l'endroit le plus performant pour faire exécuter cette tâche.

3.2.1 Hypothèses

1. Le temps d'exécution de l'algorithme de prise de décision est négligeable par rapport au temps d'exécution de la tâche à analyser.

2. Il y a assez d'énergie sur le téléphone pour exécuter les tâches considérées.

3.2.2 Période d'apprentissage

La période d'apprentissage est la période durant laquelle le système apprend à connaitre son environnement d'exécution en vue de décider du meilleur endroit pour faire exécuter les différentes tâches. Au lancement de l'application, le système ne sait rien de ce qui se passe sur le terminal mobile ni de ce qui se passe dans le *Cloud*. Il questionne pendant un certain temps ces deux environnements pour pourvoir prendre la meilleure décision à l'exécution des tâches dans le futur. Pour ce faire, le système lance une quantité de requêtes prédéfinie sur le terminal mobile et une quantité de requêtes prédéfinie dans le *Cloud* jusqu'à ce qu'il dispose d'un échantillon représentatif. Les différents paramètres nécessaires à la prise de décision sont sauvegardés dans une base de données et le moteur d'inférence est lancé pour décider du meilleur endroit

d’exécution à chaque fois qu’une tâche doit être exécutée. Dans le cas de notre prototype, la période d’apprentissage a été fixée à 20 requêtes dans le *Cloud* et 20 requêtes sur le téléphone suivant les analyses statistiques.

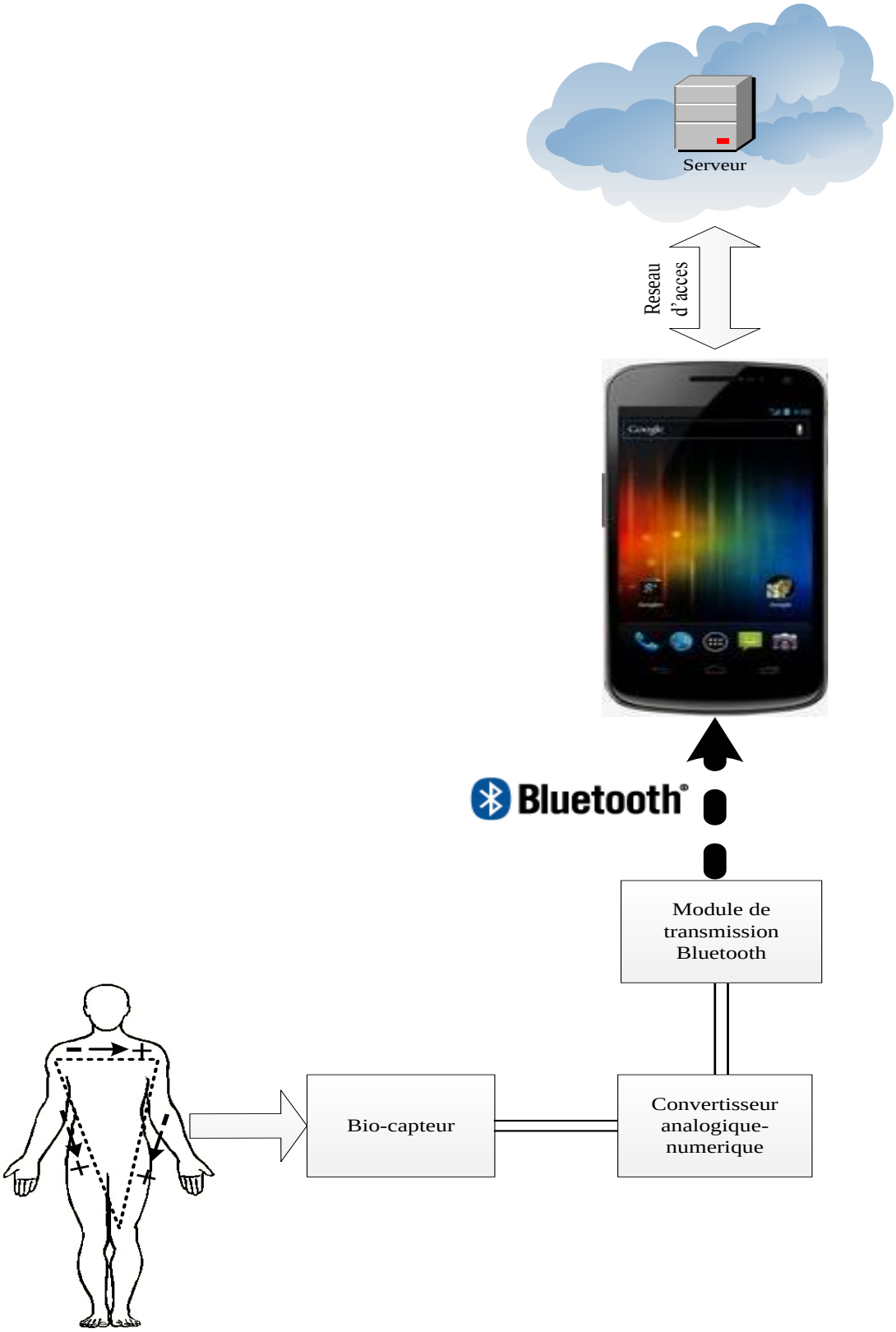

Figure 3.1. Vue globale du système

3.3 Architecture de l'application

La figure 3.2 présente l'architecture de l'application de façon générale. C'est une architecture orientée services.

Une architecture orientée services est une forme d'architecture qui consiste à développer les applications comme des composants logiciels indépendants mettant en œuvre des services à forte cohérence interne à la disposition des usagers. Les fonctionnalités du système sont décomposées en un ensemble de services encapsulés dans les composants logiciels [31]. Elles sont exposées aux utilisateurs qui peuvent y accéder au travers des interfaces développées à cette de communication.

Il existe deux types de méthodes de services Web : les services synchrones et les services asynchrones. Quand une requête est envoyée à un service dit synchrone, le terminal mobile attend que le serveur fasse le traitement demandé et lui envoie sa réponse avant de poursuivre les opérations. Dans le cas des services asynchrones, le thread pourrait continuer avec le code restant en attendant que le service Web envoie sa réponse. Étant donné la nature des opérations que nous traitons dans le cadre de notre application, l'utilisation des services synchrones s'impose, mais il est également possible d'utiliser les services asynchrones. La requête est envoyée dans le *Cloud*, le traitement désiré est effectué et le résultat est envoyé à l'application cliente.

Dans la section qui suit, nous allons présenter les différents éléments qui font partie de notre architecture. Cette dernière est divisée en plusieurs couches qui sont elles-mêmes constituées de diverses composantes.

3.3.1 L'interface utilisateur

La première composante de notre application est l'interface utilisateur qui permet à un usager de communiquer avec le système. Cette interface ouvre l'accès aux différentes fonctionnalités du système. La composante la plus importante est celle qui permet à l'utilisateur de gérer son état de stress. Les autres parties clés étant transparentes à l'utilisateur. Il n'en sait rien de l'endroit d'exécution des différentes tâches. Tout ce qui l'intéresse est l'affichage de l'évolution de son niveau de stress et comment faire pour conduire ses entrainements dans le but d'améliorer son état de relaxation.

3.3.2 Le module d'acquisition de données

Le module d'acquisition de données traite de la connexion Bluetooth avec les capteurs biologiques. Ce module s'assure d'abord que le module Bluetooth du téléphone est activé avant de se connecter au capteur. Une fois connecté au serveur Bluetooth implémenté sur le capteur, le système lui envoie une commande pour qu'il débute le transfert des données. Un thread est automatiquement lancé pour s'occuper de la réception des données. Elles sont reçues sous une forme brute et sont décodées en préparation à leur utilisation par le module de traitement.

3.3.3 Le module de traitement de données sur le terminal mobile

Le module de traitement des données est très important et représente une des composantes les plus importantes de notre application. Une fois les données reçues, le système fait appel à ce module pour exécuter la tâche requise. Dans le cadre de notre prototype, il s'agit du calcul du niveau de stress de l'utilisateur. Ce module est implémenté sur le terminal mobile aussi bien que dans le *Cloud*. Qu'il soit sur le terminal mobile ou dans le *Cloud*, le système fait appel au même code sans modification aucune. Ceci représente un avantage majeur car nous ne prenons pas en compte l'environnement d'exécution des différentes tâches.

3.3.4 Le module gérant les systèmes hérités

Dans le cas du calcul du niveau de stress, les paquets qui sont reçus du module Bluetooth doivent être décodés par une librairie écrite en C++. La plateforme que nous utilisons pour implémenter notre prototype est la plateforme Android dont le langage de prédilection pour le développement est le langage Java. La librairie existante en C++ ne doit pas être réécrite en Java pour des raisons de portabilité sur d'autres plateformes telles que l'iOS d'Apple. Ainsi, pour pouvoir utiliser ce code existant, nous utilisons l'interface JNI (Java Native Interface) de Sun Microsystems qui permet à partir du code Java implémenté sur Android d'appeler les méthodes implémentées en C++. Pour ce faire, nous avons développé des enveloppes pour les différentes fonctions existantes du langage C++, ce qui permet par la suite au code Java de communiquer avec le code natif.

3.3.5 Le module décisionnel

Le module décisionnel est le module le plus important par rapport à notre objectif de recherche. Il décide de l'endroit d'exécution d'un algorithme en se basant sur l'historique des temps d'exécution des tâches à la fois sur le téléphone et dans le *Cloud*. Au lancement d'une application, étant donné qu'aucun historique n'est disponible ou fiable, nous avons considéré une période d'apprentissage qui permet au module décisionnel de prendre ses décisions. A cet égard, le système construit un historique des paramètres d'exécution du téléphone et un historique des paramètres d'exécution du nuage informatique. Pour ce faire, le système lance un certain nombre de fois une des tâches à analyser pendant la période d'apprentissage et enregistre les différents temps d'exécution. L'enregistrement se poursuit durant toute la période d'exécution de l'application pour la mise à jour des données. Le système fait ensuite appel au système expert pour décider de l'endroit d'exécution à chaque fois qu'il doit exécuter une tâche sujette à la déportation.

3.3.6 Le module de traitement de données à distance

Le module exposant les services web dans le nuage informatique peut utiliser le style architectural REST que nous recommandons ou l'architecture SOAP que nous avons aussi implémentée dans un cadre de comparaison des deux approches. Ce module expose les fonctionnalités qui consomment beaucoup de ressources dans le *Cloud* dans le but d'améliorer la performance du système. La fonctionnalité de calcul du niveau de stress est offerte sous forme de service Web dans le *Cloud* et accessible par les applications à distance, sur le terminal mobile ou autres (RNF. 2).

3.3.7 Le moniteur de ressources

Ce module nous permet de surveiller les différentes ressources du système, qu'elles se trouvent en local sur le terminal mobile ou à distance dans le *Cloud*. Les différentes ressources qui sont surveillées à travers ce module sont le CPU, la mémoire interne et la mémoire RAM, le temps d'exécution. Ce module utilise à chaud une base de données pour sauvegarder les différents paramètres de ces ressources en tenant compte de l'évolution du système. A chaque fois qu'une tâche est exécutée, le système enregistre son temps d'exécution, la quantité de mémoire disponible, la quantité de CPU trouvée par rapport à son endroit d'exécution.

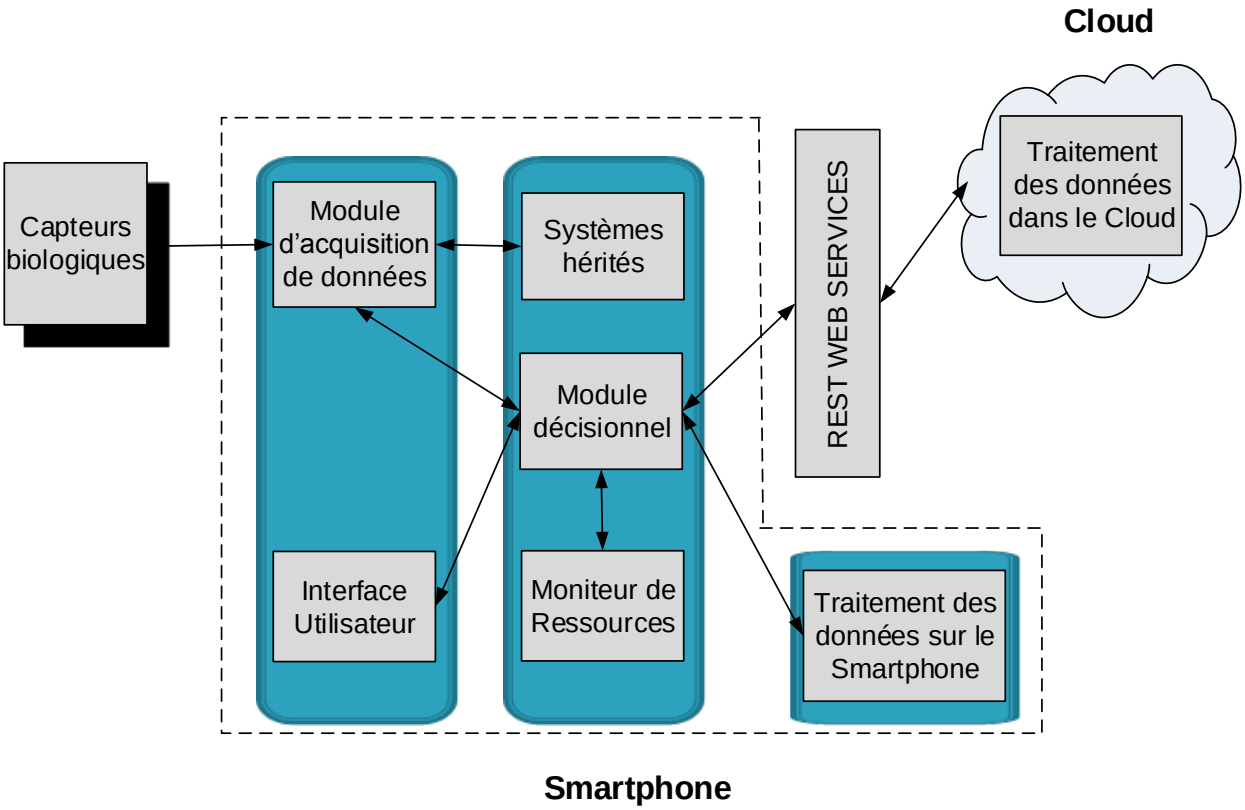

Figure 3.2. Architecture du système

3.4 Communication avec le *Cloud*

Pour communiquer avec les services Web situés dans le *Cloud*, nous utilisons le style architectural REST. Les serveurs sont implémentés dans le *Cloud* et les clients sur les terminaux mobiles. Les différentes tâches qui font objet de beaucoup de calculs sont implémentées dans le *Cloud* et peuvent être exécutées à travers les services Web implémentés que nous présentons ici. Dans le cadre de notre prototype, la tâche que nous implémentons dans le *Cloud* est celle qui nous permet de calculer le niveau de stress de l'utilisateur.

3.5 Utilisation du style architectural REST

La description fonctionnelle de la fonction qui calcule le niveau de stress d'un individu dans le *Cloud* est la suivante :

1. L'application cliente envoie une requête http utilisant la méthode GET. Les paramètres transmis sont une liste de données provenant du module Bluetooth et représentant les différents instants des battements de cœur de l'individu.
2. Le type de contenu utilisé pour l'acheminement des informations est JSON [32] qui réduit considérablement la taille des données à envoyer par rapport à XML.
3. Le serveur reçoit la demande et fait l'interprétation de la méthode GET qui lui demande la ressource. Arrivé à ce carrefour, le serveur Web passe la main au Framework de type REST pour traiter la requête. La tâche d'exécution du calcul du niveau de stress est effectuée et le résultat retourné au serveur Web.
4. Le serveur transforme la ressource reçue sous forme binaire dans le format spécifié par le client, notamment JSON dans notre cas. Ceci pourrait être aussi un autre format, bien que JSON soit notre meilleur choix par rapport à ce que nous sommes en train de faire (RNF 3).
5. Une fois la donnée convertie dans le bon format, ici JSON, le serveur Web envoie la réponse http avec le code numérique 200 avec pour charge la représentation de la ressource.
6. Le client reçoit la réponse du serveur, interprète les résultats et en fait le traitement nécessaire.

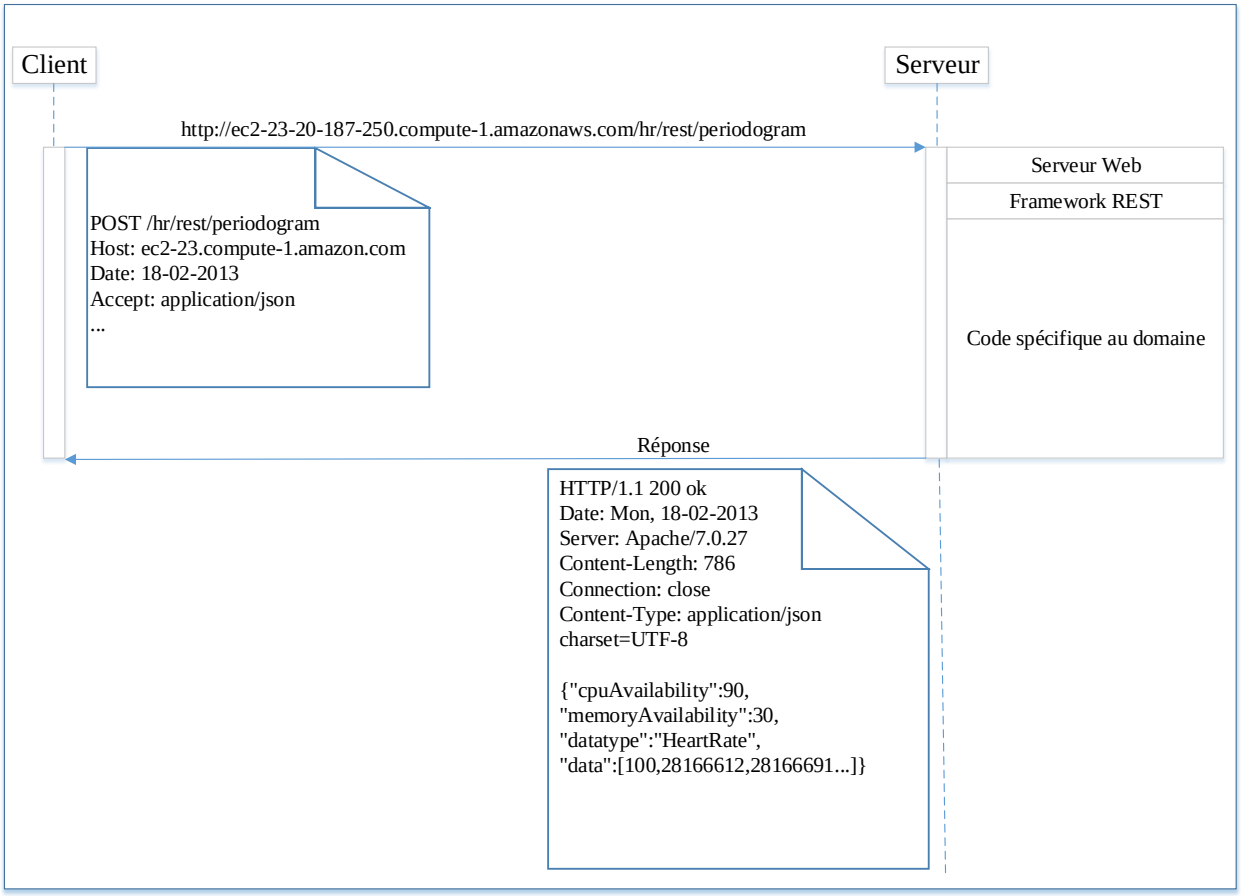

Figure 3.3. Diagramme de séquence d'une requête dans le *Cloud*

3.6 Partitionnement du programme

Toute application qui veut déporter une partie de son code à distance doit être divisée en deux parties. Une première partie du code qui sera seulement exécutée sur le terminal mobile et une deuxième partie qui peut être exécutée aux deux endroits, soit le terminal mobile soit le *Cloud*. Dans le cas de notre prototype pour mettre en valeur notre architecture, nous avons une partie de notre code qui demande beaucoup de temps de calcul à être exécutée. Cette partie est celle qui doit calculer le niveau de stress d'un individu et fait appel à l'algorithme de Lomb-Scargle [33] qui consomme beaucoup de ressources. En local, le système fait appel à une méthode qui fait tout le calcul pour donner le niveau de stress. Quand l'exécution est faite à distance, le système fait appel à un service web REST implémenté dans le *Cloud*. L'algorithme de Lomb-Scargle a été implémenté à la fois dans le *Cloud* et sur le téléphone.

3.7 Algorithme de prise de décision

L'algorithme de prise de décision est une vue assez restreinte de notre système expert. Il permet de comprendre l'analyse qui est faite par l'expert afin de décider de l'endroit d'exécution des différentes tâches. Pour arriver à cette prédiction, le système essaie de déterminer l'endroit exact d'exécution qui prendra le moins de temps. Si nous désignons par C_l le coût d'exécution d'une tâche sur le téléphone, et par C_c le coût d'exécution d'une tâche dans le nuage informatique, si $C_l < C_c$ le système fait exécuter l'algorithme sur le téléphone, dans le cas contraire il le fait exécuter dans le nuage informatique. La figure 3.4 présente l'algorithme de prise de décision.

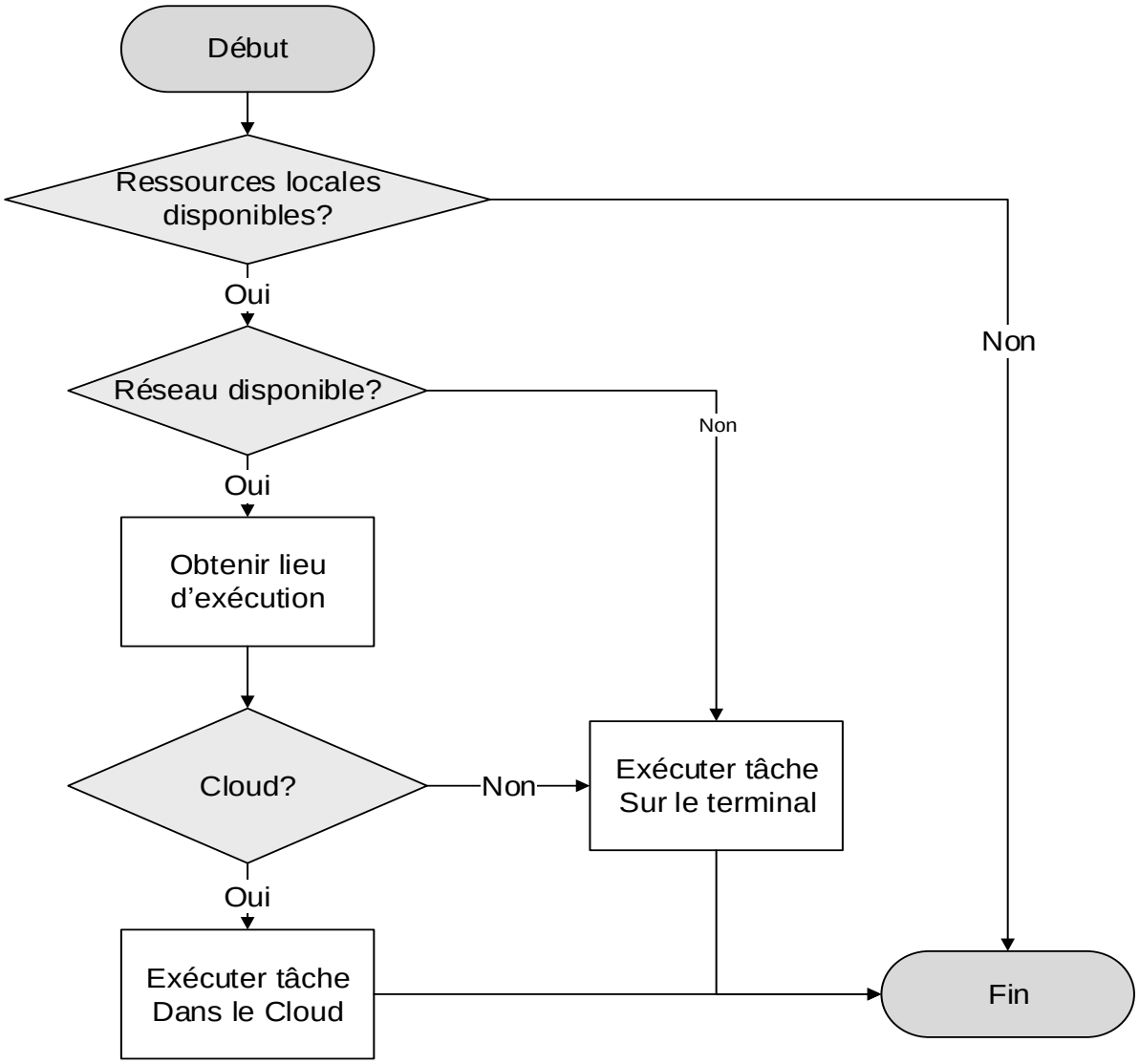

Figure 3.4. Algorithme de prise de décision

3.8 Prédiction du temps d'exécution

Pour prédire les temps d'exécution, différentes méthodes ont été analysées. Celle qui a été retenue est la moyenne glissante mobile que nous présentons dans la section suivante.

3.8.1 Moyenne mobile glissante

La moyenne mobile est utilisée pour prédire le temps d'exécution. La moyenne mobile permet de faire la moyenne des derniers temps d'exécution dans un passé récent afin de pouvoir prédire le temps d'exécution de la plus prochaine tâche. Dans nos différents tests, nous avons considéré le cas de la moyenne mobile pure (lorsque nous n'utilisons pas de temps de garde), et le cas de la moyenne mobile avec un temps de garde. Ce temps de garde est utilisé pour empêcher l'utilisation de valeurs aberrantes dans notre modèle de prédiction.

3.9 Authentification et autorisation

L'objectif de la classe d'authentification et d'autorisation est de permettre l'identification d'un utilisateur. S'il franchit cette étape, il est autorisé à utiliser les objets du système. Dans cette classe, nous implémentons une méthode qui s'appelle verifierIdentite qui reçoit en paramètre l'identificateur et le mot de passe de l'utilisateur. Cette méthode vérifie l'existence de l'utilisateur et son mot de passe dans la base de données. En cas de succès, l'utilisateur est autorisé à utiliser les services Web qui sont implémentés dans le *Cloud*. En cas d'échec, il ne pourra pas utiliser les services.

Les réseaux d'accès

Il y a deux façons d'accéder au réseau en utilisant un téléphone intelligent. L'utilisateur peut utiliser le réseau Wi-Fi offrant aujourd'hui un débit assez intéressant pouvant aller jusqu'à 54 Mbps tandis que le débit du réseau 3G est très limité.

3.10 Présentation de quelques classes du système

Nous présentons ci-dessous une ébauche de quelques classes faisant partie de notre diagramme de classe :

La classe **ResourceMonitor :**

- exeTask : le nom de la tâche qui est exécutée;

- CPULevel : le niveau du CPU sur le terminal mobile;
- MemoryLevel : la quantité de mémoire disponible sur le terminal mobile;
- exeTime : le temps d'exécution de la tâche considérée;
- exePlace : lieu d'exécution de la tâche considérée.

La classe **Rule**

- ruleNumber : le numéro de la règle;
- factList : la liste des prémisses ou conditions de réalisation;
- resultList : la liste des conclusions résultant de l'ensemble des prémisses;
- nextRule : la prochaine règle à considérer.

La classe **Fact**

- factNumber : le numéro d'un fait;
- factDesc : la description d'un fait;
- nextFact : le prochain fait à considérer suivant l'ordre de considération des faits dans la liste.

La classe **Lomb**

- LombRatio : Niveau de relaxation de l'utilisateur;
- getLombRatio : Obtenir le niveau de relaxation de l'utilisateur.

La classe **InferenceEngine**

- Rules : ensemble des règles du système;
- getRules : Obtenir l'ensemble des règles du système;
- Facts : ensemble des faits du système;
- getFacts : obtenir l'ensemble des faits du système;
- sendResponse : envoyer une réponse après consultation de la base des connaissances.

3.11 Système Expert

Un système expert est un outil informatique d'intelligence informatique qui nous aide à prendre des décisions à la place d'un humain qualifié d'expert dans un domaine particulier [34].

Il est composé d'une base de connaissances et d'un moteur d'inférence comme illustré à la figure 3.5. La base de connaissance est constituée d'un ensemble de faits et de règles.

Au cours de l'exécution de notre application, il revient assez souvent de prendre des décisions par rapport à l'endroit d'exécution de certaines tâches suivant un certain nombre de critères. Nous avons élaboré à partir de nos tests présentés au chapitre 4 un ensemble de règles sur lesquelles se base notre moteur d'inférence afin de prendre les décisions appropriées à chaque fois que pareille situation se présente.

Notre système est basé sur les règles contrairement à d'autres qui sont orientés objets.

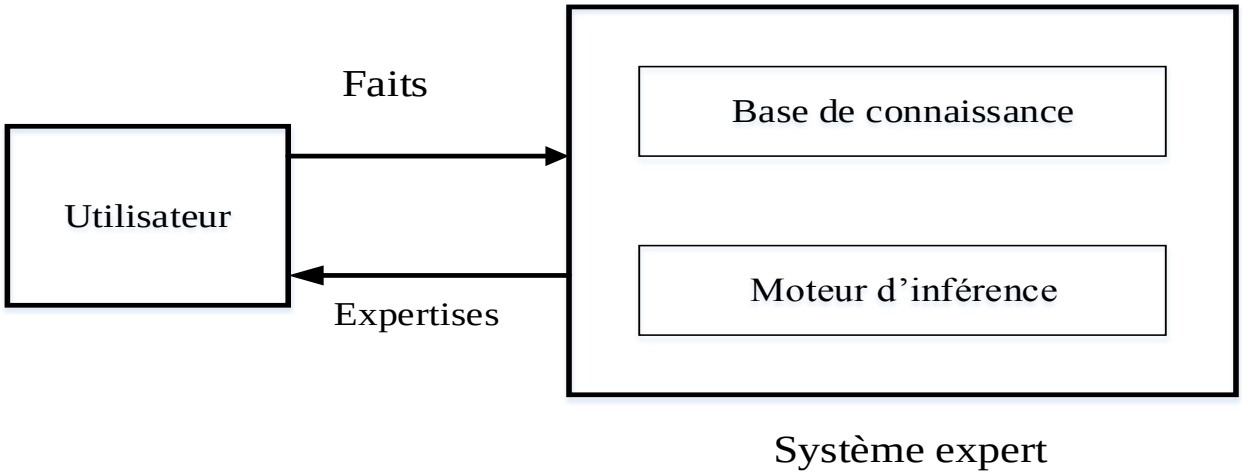

Figure 3.5. Concept de base d'un système expert

3.11.1Moteur d'inférence

Le moteur d'inférence est aussi appelé interpréteur de règles ou moniteur d'inférence. C'est la composante active du système expert. C'est elle qui déclenche les règles. Le système expert utilise les règles implantées dans la base des règles pour inférer un résultat. Ces règles infèrent sur des données en entrée, qui sont une base de donnée appelée la base de faits. La décision de l'endroit d'exécution d'une tâche est produite en sortie. Quand plusieurs faits arrivent simultanément, plusieurs règles peuvent être candidates au déclenchement, les stratégies de résolution de conflit sont utilisées pour résoudre ce problème. Pour décider de l'endroit d'exécution des tâches, il est nécessaire d'avoir des faits et des règles. Mais il faut aussi une instance de décision qui, sur la base des faits et règles établis, aura à prendre cette décision. Cette instance de décision s'appelle le moteur d'inférence et le mode de raisonnement utilisé est le

chainage avant dans le cadre de notre projet. Nous partons des prémisses pour aboutir aux conclusions.

Les sections qui suivent présentent notre base de faits et de règles. Les règles sont présentées ici de façon à ce qu'elles soient très compréhensibles à l'humain.

3.11.2 Base de faits

Notre base de faits contient tout ce qui peut influer sur la prise de décision dans le système par rapport à l'endroit d'exécution des tâches considérées. Par exemple, le réseau peut ne pas être accessible durant l'exécution de l'application. Cette connaissance entraine que le système ne doit envoyer aucune tâche à être exécutée dans le *Cloud* durant cette période. Elle doit être insérée dans notre base de faits, ce qui permettra à notre moteur d'inférence de pouvoir prendre une décision appropriée lorsque cette situation se présente dans le système.

3.11.3 Base de règles

Notre système expert se compose d'un ensemble de règles d'inférence qui se présentent sous la forme (prémisse) -> (conclusion) où prémisse désigne un ensemble de propositions et conclusion fait référence à une proposition. À noter qu'une proposition est une expression quelconque qui peut être évaluée à vrai ou faux.

Nous considérons deux temps d'exécution pour faire nos analyses. Un premier temps d'exécution où le système fait l'apprentissage des environnements d'exécution. Au cours de cette période d'apprentissage, il fait exécuter les tâches sur le terminal mobile et aussi dans le *Cloud* pour pouvoir décider sur un échantillon raisonnable. Dans le cas typique de notre application, nos analyses statistiques ont abouti à 20 exécutions pour la période d'apprentissage. Le deuxième temps d'exécution est celui considéré après la période d'apprentissage du système.

Étant donné que nous voulons avoir une certaine flexibilité par rapport à l'évolution des technologies, un fichier de configuration a été conçu pour permettre aux développeurs et utilisateurs d'avoir la possibilité de modifier certaines valeurs par rapport aux contraintes environnementales. Les valeur-seuils pour le processeur, la mémoire, la taille des données sont des valeurs qui peuvent être modifiées par les développeurs ou utilisateurs suite à leurs analyses du temps d'exécution des tâches ou applications qui seront considérées.

(R1) SI 1) Aucun réseau (Wi-Fi ou 3G) n’est accessible

2) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

3) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche est exécutée sur le terminal mobile.

(R2) SI 1) Aucun réseau n’est accessible (Wi-Fi ou 3G)

2) La quantité de mémoire disponible sur le terminal mobile est inférieure au seuil minimal

3) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche n’est pas exécutée.

(R3) SI 1) Aucun réseau n’est accessible (Wi-Fi ou 3G)

2) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

3) La quantité de CPU disponible sur le terminal mobile est inférieure au seuil minimal

ALORS la tâche n’est pas exécutée.

(R4) SI 1) Un réseau est accessible (Wi-Fi ou 3G)

2) La quantité de mémoire disponible sur le terminal mobile est inférieure au seuil minimal

3) La quantité de CPU disponible sur le terminal mobile est inférieure au seuil minimal

4) La prédiction du temps d'exécution sur le terminal mobile est inférieure au délai maximal

ALORS la tâche est exécutée sur le terminal mobile.

(R5) SI 1) L'apprentissage a déjà été fait sur le terminal mobile et dans le *Cloud*

2) Un réseau est accessible (Wi-Fi ou 3G) et il y a un problème quelconque au niveau du *Cloud* de répondre aux requêtes qui lui sont envoyées

2) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

3) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche est exécutée sur le terminal mobile.

(R6) SI 1) Aucun apprentissage n'a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* n'est pas accessible

3) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

4) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche est exécutée sur le terminal mobile (Aucun apprentissage n'est fait).

(R7) SI 1) Aucun apprentissage n'a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* est accessible

3) La quantité de mémoire disponible sur le terminal mobile est supérieur au seuil minimal

4) La quantité de CPU disponible sur le terminal mobile est supérieur au seuil minimal

ALORS l'apprentissage se fait en faisant exécuter la tâche concernée 20 fois dans le *Cloud* et 20 fois sur le terminal mobile.

(R8) SI 1) Aucun apprentissage n'a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* est accessible

3) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

4) Le temps CPU disponible sur le terminal mobile est supérieur au seuil minimal

5) Le temps d'exécution prédit pour le *Cloud* est inférieur à celui du terminal mobile

ALORS la tâche est exécutée dans le *Cloud*.

(R9) SI 1) Aucun apprentissage n'a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* est accessible

3) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

4) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

5) Au moins 4 délais de garde sur les 20 dernières requêtes lancées dans le *Cloud* ont été repérés

6) Aucun changement de réseau n'a été enregistré sur les 20 dernières requêtes

7) Le temps d'exécution prédit sur le terminal mobile est inférieur au délai de garde de base

ALORS la tâche est exécutée sur le terminal mobile.

(R10) SI 1) Aucun apprentissage n’a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* est accessible

3) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

4) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

5) Au moins 4 délais de garde sur les 20 dernières requêtes lancées dans le *Cloud* ont été repérés

6) Le temps d’exécution prédit sur le terminal mobile est supérieur au délai de garde de base

ALORS le délai de garde de base est modifié à la valeur moyenne du temps d’exécution sur le terminal mobile et le système continue à envoyer les requêtes dans le *Cloud*.

(R11) SI 1) Aucun apprentissage n’a été fait sur le terminal mobile ni dans le *Cloud*

2) Le *Cloud* est accessible

3) Le terminal mobile change de réseau (Il sort d’un réseau A pour aller à un réseau B)

3) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

4) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS l’apprentissage est repris seulement sur le nouveau réseau avec 20 nouvelles requêtes.

(R12) SI 1) Aucun apprentissage n’a été fait sur le terminal ni dans le *Cloud*

2) Le *Cloud* est accessible

3) Les tâches de l'application doivent être exécutées en temps réel

4) Le temps de prédiction en local est inférieur au délai maximal fixé pour le temps réel

5) Deux délais de garde sur moins de 20 requêtes lancées dans le *Cloud* sont enregistrés

6) La quantité de mémoire sur le terminal mobile est supérieure au seuil minimal

7) La quantité de CPU sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche est exécutée sur le terminal mobile.

(R13) SI 1) Aucun apprentissage n'a été fait sur le terminal ni dans le *Cloud*

2) Le *Cloud* est accessible

3) Le temps de prédiction en local est supérieur au délai de garde de base

4) Le délai de garde de base n'est plus utilisé pour le *Cloud*

5) Au moins 4 délais de garde sur les 20 dernières requêtes lancées dans le *Cloud* sont enregistrés

ALORS la tâche est exécutée sur le terminal mobile.

(R14) SI 1) L'apprentissage a déjà été fait sur le terminal et dans le *Cloud*

2) Le *Cloud* est accessible

3) La taille des données à envoyer dans le *Cloud* est supérieure à la valeur-seuil fixée

4) La quantité de mémoire disponible sur le terminal mobile est supérieure au seuil minimal

5) La quantité de CPU disponible sur le terminal mobile est supérieure au seuil minimal

ALORS la tâche est exécutée sur le terminal mobile.

Il y a deux raisons qui entrainent la mise en place de la règle 14. La première raison est que vu le type de réseau utilisé, l'utilisateur peut être contraint à utiliser le terminal mobile si la quantité de données à envoyer dans le *Cloud* est trop élevé. Cette condition est particulièrement intéressante dans le cas d'utilisation des réseaux 3G où le prix consenti pour la bande passante est relativement élevé. La seconde est liée au temps d'exécution qui peut être plus élevé pour une quantité de données à traiter dans le *Cloud* que sur le téléphone, ce qui peut être vu comme un cas classique d'utilisation du terminal mobile.

L'avantage avec le système expert que nous avons présenté ici est que ce système est assez dynamique et peut tenir compte d'autres recommandations dans le futur. Celles-ci peuvent être déduites par rapport à d'autres analyses du système expert. Les différentes règles qui sont élaborées ici sont le fruit de plusieurs scénarios que nous avons mis en place tout au long de notre projet. Il est toujours possible de s'imaginer d'autres scénarios et de conduire d'autres tests qui pourront rendre le système encore plus fiable et plus robuste. Ces tests peuvent engendrer d'autres règles qui seront insérées dans le système.

CHAPITRE 4
ÉVALUATION DE L'ARCHITECTURE ET RÉSULTATS

Un prototype de l'application a été développé pour nous permettre d'évaluer l'architecture proposée au chapitre 3. Ce prototype est réalisé dans le cadre des applications mobiles dans le domaine de la santé. Cependant, moyennant quelques analyses supplémentaires, il peut être adapté à d'autres types d'applications mobiles. L'objectif est de montrer que la déportation des calculs dans le *Cloud* permet d'améliorer la performance de nos applications sur les terminaux mobiles. Les résultats obtenus sur le terminal mobile sont comparés à ceux qui sont obtenus dans le *Cloud* en tenant compte des contraintes qui influencent l'évolution du système. Ce chapitre présente l'évaluation des requis fonctionnels de l'application, l'implémentation de notre architecture et les résultats obtenus.

4.1 Évaluation des requis fonctionnels

La métrique utilisée comme indice de performance pour évaluer les requis fonctionnels (calcul du niveau de stress par la méthode du *biofeedback*) est le temps. Il s'agit du temps que prend une tâche pour être exécutée dans le *Cloud* ou sur le téléphone mobile. Ce temps de réponse peut être décomposé en deux parties quand les tâches sont exécutées dans le *Cloud*, un temps aller-retour (Round Trip Time ou RTT) pour acheminer les paquets dans le *Cloud* et un temps d'exécution de la tâche sur la machine virtuelle située dans le *Cloud*. Le temps aller-retour ou RTT (Round Trip Time) est le temps que prend un paquet pour arriver dans le *Cloud* et le temps pris par le serveur pour envoyer la réponse au terminal mobile. Une tâche pouvant être exécutée sur le terminal mobile ou dans le *Cloud*, le temps d'exécution peut être le temps d'exécution d'une tâche dans le *Cloud* ou son temps d'exécution sur le terminal mobile. Ce temps sera toujours considéré par rapport à l'endroit d'exécution de la tâche. Dans les sections qui suivent, nous présentons d'abord les paramètres de configuration de l'environnement de développement et ensuite les différents scénarios nous permettant d'aboutir aux résultats qui en découlent.

4.2 Environnement de développement

Pour évaluer la performance de notre système, nous avons mis en place un serveur situé dans le *Cloud* aux États-Unis en Virginia du Nord, un serveur situé au centre-ville de Montréal et un autre serveur situé à la maison. Pour le prélèvement des données sur le corps humain, nous avons utilisé des capteurs biologiques d'une entreprise située à Montréal. Et, comme terminal mobile nous utilisons à chaque fois un téléphone intelligent. Ce dernier utilise la plateforme Android de Google sur lequel nous avons installé un prototype de notre application qui a été présenté au chapitre 3.

Le principal terminal mobile utilisé pour tester notre application est le Nexus S de Google muni d'un processeur ARM (Advanced RISC Machine) Cortex-A8a de 1 GHz. Sa mémoire RAM est de 512 Mégaoctets et sa capacité de stockage interne de 16 giga-octets. La version Android utilisée sur le téléphone est la version 4.1.2. Le téléphone communique avec le serveur dans le *Cloud* en utilisant les réseaux 3G ou Wi-Fi. Pour communiquer avec les périphériques biologiques pour la cueillette des données provenant du corps humain, le module Bluetooth version 2.1 est utilisé étant donné que le terminal mobile Nexus S que nous utilisons dispose déjà de cette version. Il serait préférable d'utiliser d'autres appareils plus récents équipés de la version 4.0 qui nous garantit une plus grande portée, une meilleure économie d'énergie et une plus grande fiabilité du point de vue de la connexion. Dans le cas de notre projet, la version 2.1 nous permet de tester nos scénarios sans aucun problème.

Dans le *Cloud* situé aux États-Unis, nous utilisons la plateforme EC2 proposée par Amazon, mettant à notre disposition une instance aux caractéristiques suivantes :

- instance (machine virtuelle) de type t1.micro (la plus petite qui soit disponible chez Amazon);
- capacité mémoire de 613 Mégaoctets;
- 2 unités de calcul pour les pics périodiques de courte durée;
- système d'exploitation linux, distribution *Red Hat Enterprise*.

Le serveur situé au centre-ville de Montréal est HP ProLiant DL320 G2. Il est doté d'un processeur de 3.06 GHz, d'une mémoire RAM de 2 giga-octets. Le serveur situé à la maison est un HP P6153F. Il est doté d'un processeur de 2.5 GHz, d'une mémoire RAM de 8 giga-octets.

La solution EC2 d'Amazon utilise la virtualisation Xen qui permet de faire fonctionner plusieurs systèmes d'exploitation virtuels de manière isolée sur une même machine physique. Notre système fonctionne sur le système d'exploitation *Red Hat Entreprise Linux*, qui est un système très populaire et très stable.

L'application est développée en utilisant le langage de programmation Java et l'environnement de développement Éclipse version Juno. Le serveur d'application *Tomcat 7.0* d'Apache a été utilisé. Nous avons aussi utilisé le langage C pour développer les enveloppes qui nous permettent de communiquer avec les systèmes hérités. Le fait que notre système utilise à la fois le téléphone et le *Cloud* pour faire exécuter ses différentes tâches, ceci entraine une implémentation de certaines méthodes à la fois sur le *Cloud* et sur le terminal mobile.

Nous avons évalué deux approches architecturales pour la mise en place de nos applications, REST et SOAP. Pour ce faire, nous avons développé un serveur pour chacune de ces architectures. Pour le serveur REST, nous avons utilisé Jersey fournie par *SUN Microsystems* (Oracle). Jersey est une implémentation de référence de JAX-RS (Java API for Restful Web Services) qui fait partie intégrante de la spécification Java EE. JAX-RS est une standardisation de l'implémentation de REST en Java.

Pour tester certains facteurs comme la mémoire ou le CPU sujets à modifier le comportement de ces types d'application, nous utilisons les émulateurs d'Android. Ces derniers sont fournis avec la plate-forme Android et nous permettent de faire varier la quantité de mémoire du terminal mobile et de changer le type de processeur utilisé.

Facteurs influençant le lieu d'exécution

Les différents facteurs considérés par rapport à la prise de décision du lieu d'exécution de nos tâches sont :

- la mémoire (avant de faire exécuter une tâche sur le terminal mobile, le système vérifie la quantité de mémoire disponible);
- le CPU (avant de faire exécuter nos tâches sur le terminal mobile, le système vérifie le processeur);
- la taille des données (avant d'envoyer une requête dans le *Cloud*, le système vérifie la taille de la requête) ;

- le type de réseau (3G, Wi-Fi sont les plus courants aujourd'hui, mais le Wi-Fi reste toujours un choix de prédilection en général quand les deux réseaux sont disponibles);
- le temps d'exécution des tâches dans un passé récent ou au cours de la période d'apprentissage.

4.3 Évaluation des requis fonctionnels

La tâche que nous utilisons dans notre prototype pour évaluer nos requis fonctionnels est le calcul du niveau de stress par la méthode du *biofeedback*. Ces tests ont été effectués en milieu d'entreprise et se sont révélés concluants. La figure 4.1 présente les résultats qui ont été obtenus en faisant interagir l'utilisateur avec le système. Deux schémas illustratifs sont fournis sur la figure. Ces résultats illustrent les calculs de niveau de stress en utilisant l'algorithme de *Lomb-Scargle*.

Le premier schéma est le cas typique d'utilisateur stressé et pour le deuxième schéma l'utilisateur agit sur le système pour corriger son niveau de stress en utilisant la méthode de *biofeedback*. Le résultat apparait sur la première figure dans le cas où l'utilisateur est stressé, le niveau de relaxation est de 40.45% et la courbe des battements cardiaques affiche un comportement irrégulier, ce qui traduit le fait que l'utilisateur est effectivement stressé. Le premier schéma représente des états émotionnels liés au stress, à la colère, à la frustration où l'onde de la variabilité du rythme cardiaque présente des pics déchiquetés. Nous avons affaire à un modèle cardiaque incohérent qui expose l'état de stress de l'individu à ce moment donné.

Dans le deuxième cas, le niveau de relaxation est de 64.84% et l'utilisateur n'est pas stressé, ce qui s'explique par l'allure régulière de la courbe. Cette courbe traduit la cohérence cardiaque de l'individu et traduit des émotions comme l'appréciation, la joie, l'amour. La courbe est lisse et formée d'ondes harmonieuses. La cohérence se caractérise par sa régularité et une onde sinusoïdale. Pour obtenir cet état de cohérence cardiaque, il est demandé à l'utilisateur de s'exercer en utilisant la méthode respiratoire pour améliorer sa cohérence cardiaque. Ceci constitue à faire successivement cinq expirations et cinq inspirations. Cette méthode est très appliquée par les psychologues pour améliorer le niveau de stress de leurs

patients. Le deuxième schéma présente un schéma très différent et traduit un autre état de l'individu où il n'est pas stressé.

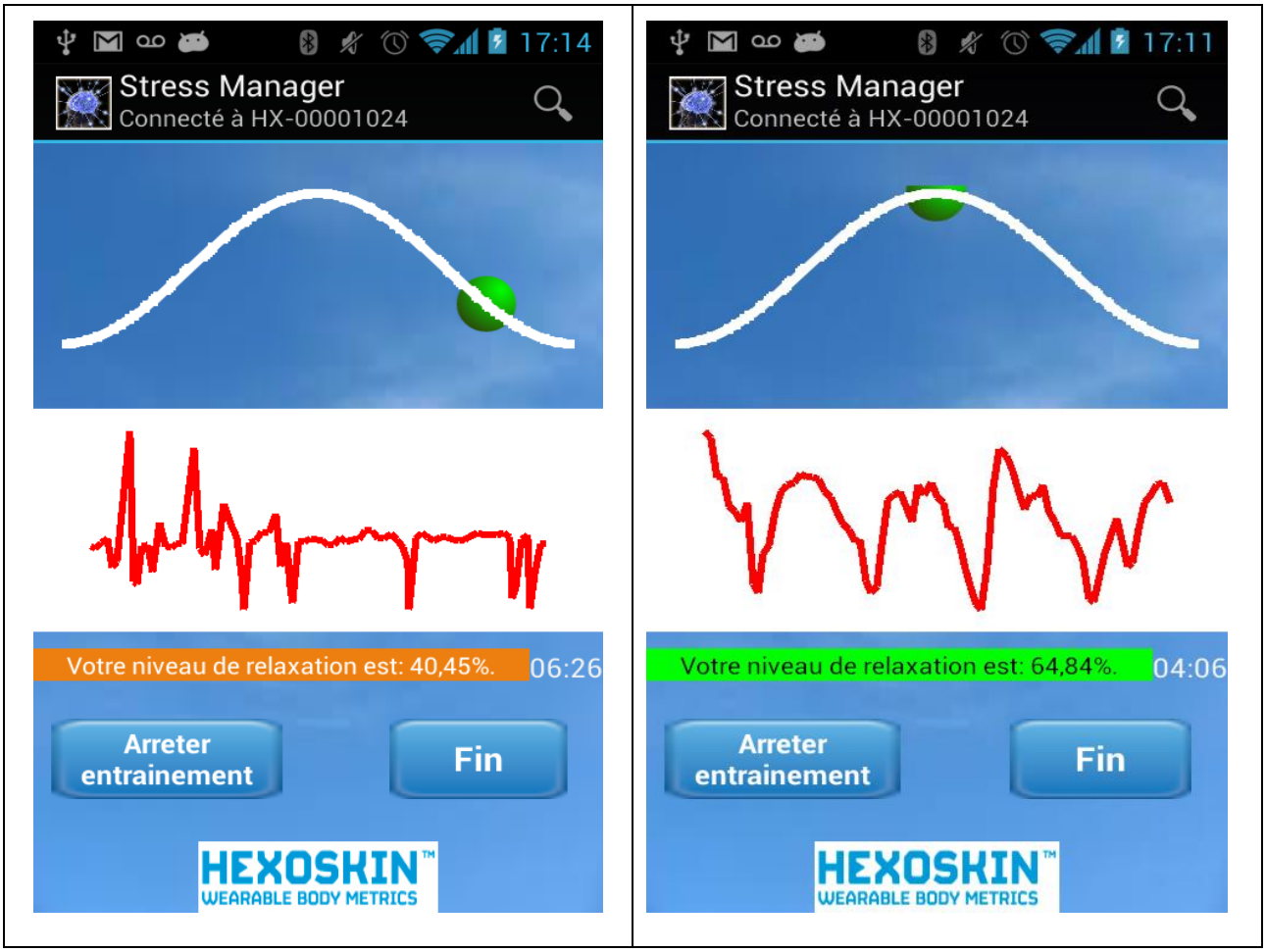

Figure 4.1. Niveau de stress de l'utilisateur (Stressé et non stressé)

4.4 Scénario 1 : Exécution sur le terminal mobile

Ce scénario nous permet de mesurer le temps d'exécution sur un terminal mobile. Ces données sont recueillies tout de suite après le démarrage du système d'exploitation pour s'assurer que toutes les ressources sont disponibles pour l'application. Au cours de la réalisation de ce scénario, nous avons fait exécuter seulement l'application sur le téléphone. Nous avons vérifié également qu'il ne se lance aucune tâche qui consomme beaucoup de ressources en background au démarrage du système d'exploitation. Donc le système est le mieux préparé pour l'exécution de l'application. Les résultats obtenus sont présentés à la figure 4.2. Dans ce scénario, les tâches à analyser sont exécutées en local, c'est-à-dire sur le terminal mobile. Un échantillon assez

représentatif des temps d'exécution est prélevé sur une période de 15 minutes afin d'avoir une analyse assez réaliste. Nous avons trouvé un temps d'exécution moyen de 347.3 millisecondes avec un minimum de 295 millisecondes et un maximum de 1743 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 36.84 millisecondes.

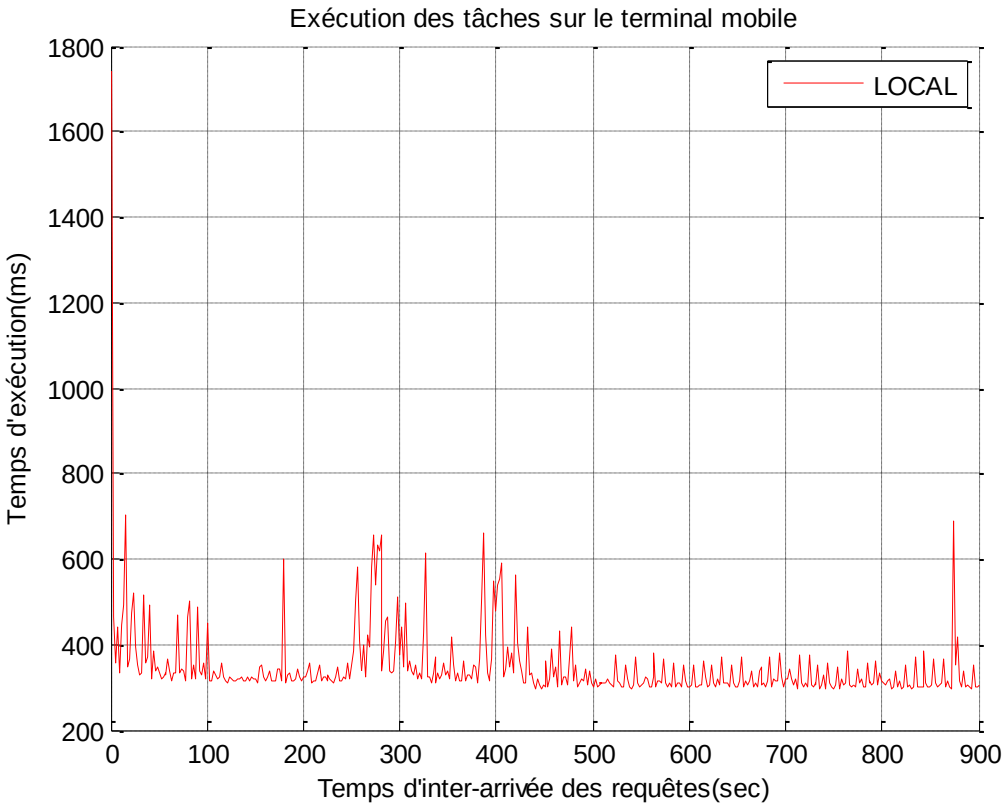

Figure 4.2. Exécution des tâches sur le terminal mobile

4.5 Scénario 2 : Exécution avec REST, réseau Wi-Fi, sans délai de garde

Ce scénario nous permet de mesurer le temps d'exécution sur un terminal mobile qui utilise le réseau Wi-Fi pour envoyer ses requêtes dans le *Cloud*. Le style architectural REST est utilisé. Le système envoie un ensemble de requêtes dans le *Cloud* pour faire exécuter les mêmes tâches qui ont été exécutées sur le terminal mobile pendant environ 15 minutes. Une fois la réponse obtenue, nous mesurons le temps de réponse de la requête dans le *Cloud* (qui est la somme du temps d'exécution dans le *Cloud* et du temps aller-retour pour trouver la réponse). Le temps de réponse moyen obtenu est de 147 millisecondes avec un minimum de 61 millisecondes et un maximum de

7581 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 358.8 millisecondes.

Le temps aller-retour que nous appelons ici RTT ou Round Trip Time est le temps mis pour envoyer la requête et recevoir la réponse, hormis le temps d'exécution sur le serveur situé dans le *Cloud*. La figure 4.3 fait état d'un ensemble de valeurs aberrantes par rapport à la moyenne obtenue, ce qui explique la valeur élevée obtenue pour l'écart-type. Ces valeurs aberrantes nuisent grandement au temps de réponse que pourrait espérer un utilisateur par rapport à la valeur moyenne. Des corrections doivent être apportées afin de résoudre ce problème. Le scénario 5 tient compte de cette réalité et apporte une réponse appropriée.

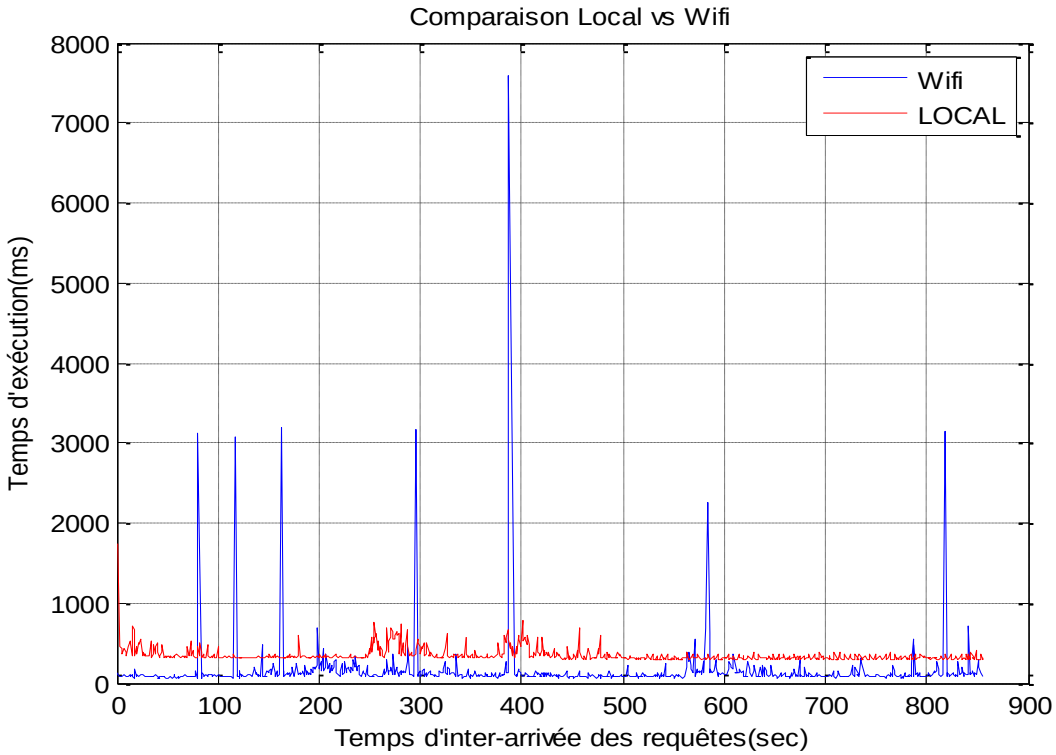

Figure 4.3. Comparaison des temps d'exécution en utilisant le Wi-Fi

Sur la figure 4.3, nous avons relevé un ensemble de valeurs aberrantes. Cette situation peut nous conduire à des décisions erronées en ce qui a trait à l'endroit d'exécution de nos tâches. Nous avons constaté que dans plus de 99% des cas, le temps d'exécution de la tâche considérée dans le *Cloud* est inférieur aux temps d'exécution trouvé sur le téléphone. Ce qui normalement doit nous entrainer à faire exécuter les tâches dans le *Cloud*. Les valeurs aberrantes peuvent nous amener à prendre de mauvaises décisions sans une bonne politique décisionnelle.

Le problème que nous avons soulevé au scénario 2 est dû au fait qu'il n'y a aucun temps de garde pour limiter le temps d'exécution d'une tâche. Une fois la requête lancée, l'utilisateur attend une réponse qui ne lui parvient jamais. Nous proposons d'utiliser un temps de garde pour limiter le temps d'exécution dans le *Cloud*. Ceci nous permet d'avoir un contrôle assez intéressant sur le temps d'exécution dans le *Cloud*.

Nous proposons également que ce temps de garde évolue avec le temps. Il varie par rapport à l'expérience de l'usager. Étant donné la mobilité des usagers et la dynamique de changement de l'environnement du terminal mobile, le système fait une mise à jour des différents paramètres dans sa base de données au fur et à mesure que le système évolue. Nous utilisons comme premier temps de garde la moyenne des temps d'exécution sur le terminal mobile calculé après la période d'apprentissage. Le temps de garde durant la période d'apprentissage est laissé à la discrétion du développeur ou peut être configurable par l'utilisateur.

4.6 Scénario 3 : Exécution avec REST, sur un réseau 3G, sans délai de garde

Ce scénario est présenté pour nous permettre d'apprécier le temps d'exécution des requêtes en utilisant un réseau 3G étant donné que ces derniers sont plus lents que les réseaux Wi-Fi. Le système envoie un ensemble de requêtes dans le *Cloud* pour exécuter les mêmes tâches qui ont été exécutées sur le terminal mobile pendant environ 15 minutes. Les temps d'exécution et RTT sont mesurés par la suite. Dans le *Cloud*, les paramètres tels que temps d'exécution moyen de la tâche, maximum et minimum sont pratiquement les mêmes pour le réseau Wi-Fi et pour le réseau 3G. La figure 4.4 fait état d'une différence assez considérable étant donné la lenteur des réseaux cellulaires par rapport aux réseaux Wi-Fi. Cette différence s'explique par le fait que le temps aller-retour des requêtes envoyées dans le Cloud est en général plus élevé dans un réseau cellulaire.

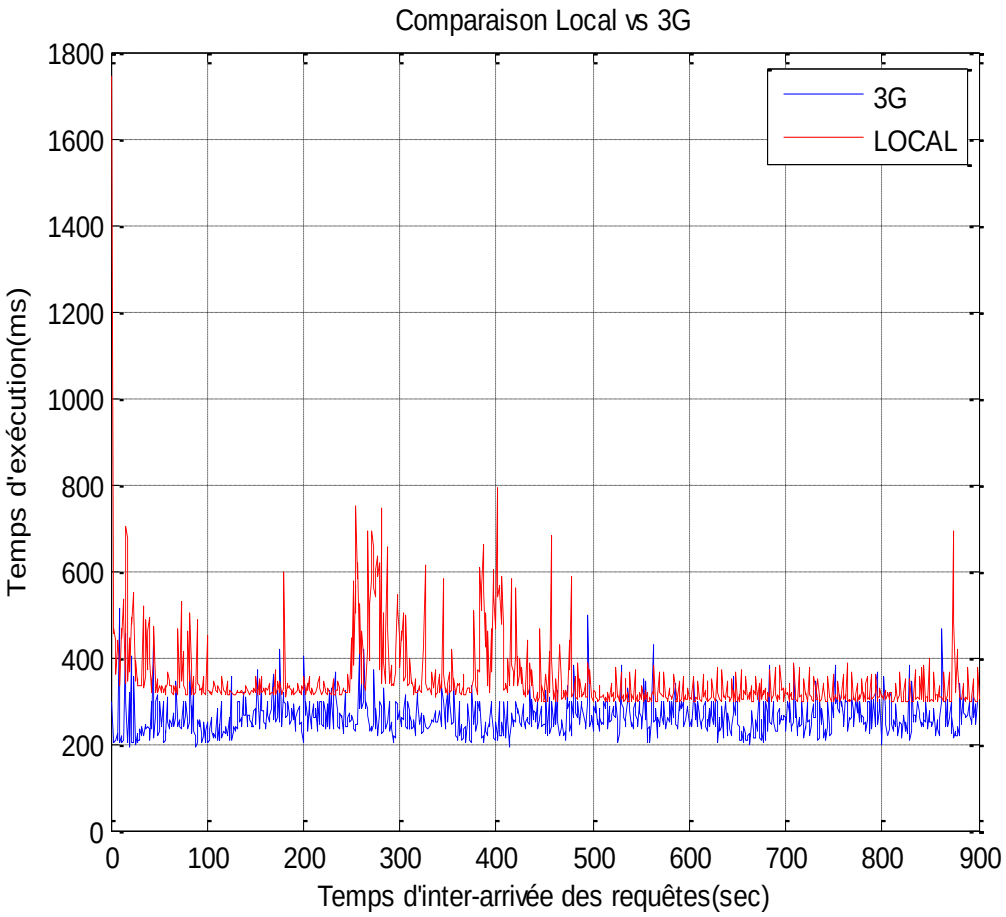

Figure 4.4. Comparaison des temps d'exécution en utilisant le réseau 3G

4.7 Scénario 4 : Comparaison REST et SOAP

L'architecture SOAP est très utilisée de nos jours même si le style architectural REST est en train de lui damer le pion. Pour s'assurer de la meilleure technologie à utiliser, nous avons testé les deux approches SOAP et REST. La figue 4.5 présente les résultats obtenus. Le système envoie des requêtes au serveur SOAP et au serveur REST situé dans le *Cloud* en les alternant pour s'assurer que l'environnement reste le même. Ces deux serveurs sont implémentés sur la même machine physique. Nous avons à la fin enregistré une quantité de requêtes REST et SOAP s'étalant sur une période de 30 minutes environ. Ces tests ont été réalisés avec le serveur situé au centre-ville de Montréal.

Pour l'utilisation de SOAP, nous avons trouvé un temps d'exécution moyen de 233.5 millisecondes, un minimum de 154 millisecondes et un maximum de 3252 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 195.3 millisecondes.

Pour le style architectural REST, nous avons trouvé un temps d'exécution moyen de 173.4 millisecondes, un maximum de 780 millisecondes et un minimum de 128 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 53.14 millisecondes.

Ce scénario nous permet de déduire que le style architectural REST nous permet d'avoir une meilleure performance dans le *Cloud* par rapport à SOAP. Il est à noter que pour SOAP nous avons utilisé Axis de Apache qui est la référence en termes de performance comparativement aux autres moteurs de services Web SOAP.

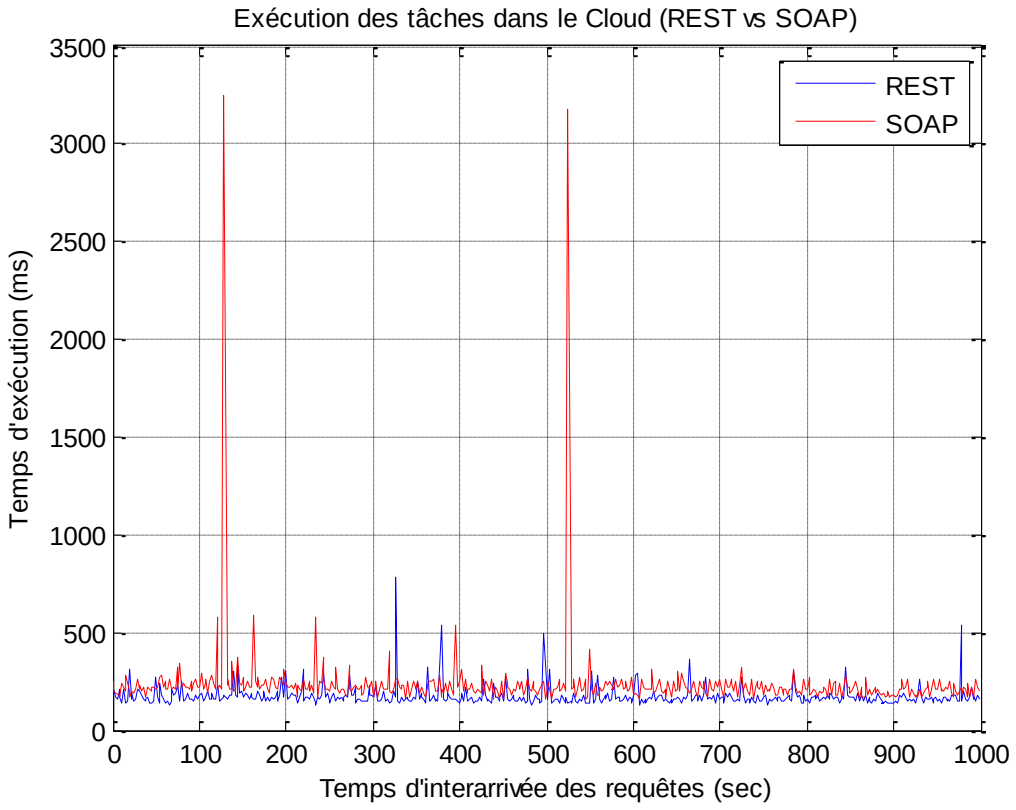

Figure 4.5. Comparaison des temps d'exécution (REST vs SOAP)

4.8 Scénario 5 : Exécution avec REST, sur un réseau 3G, avec temps de garde

Ce scénario nous permet de mesurer le temps moyen d'exécution d'une tâche dans le *Cloud* en utilisant le style architectural REST. Pour ce faire, le réseau 3G est utilisé et un ensemble de requêtes est envoyé dans le *Cloud* pendant 15 minutes. Les requêtes sont envoyées à chaque seconde si le temps de réponse de la dernière requête est inférieur à 1 seconde. Sinon le système envoie la prochaine requête immédiatement après. Nous avons trouvé un temps de réponse moyen de 210.10 millisecondes, avec un minimum de 138 millisecondes et un maximum de 350 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 56.07 millisecondes. Lorsque nous faisons la comparaison de ces résultats par rapport aux valeurs trouvées sur le terminal mobile, nous constatons que le temps d'exécution dans le *Cloud* avec un réseau 3G est beaucoup plus avantageux que le temps d'exécution sur le terminal mobile. Nous avons obtenu en moyenne une différence de 138.3 millisecondes. Ceci nous entraine à considérer les systèmes qui utilisent aussi les réseaux 3G pour faire exécuter leurs tâches dans le *Cloud*. Le réseau 3G leur permet de bénéficier d'une vitesse d'exécution beaucoup plus intéressante que sur le terminal mobile. Néanmoins, nous n'avons pas la même rapidité enregistrée avec les réseaux Wi-Fi.

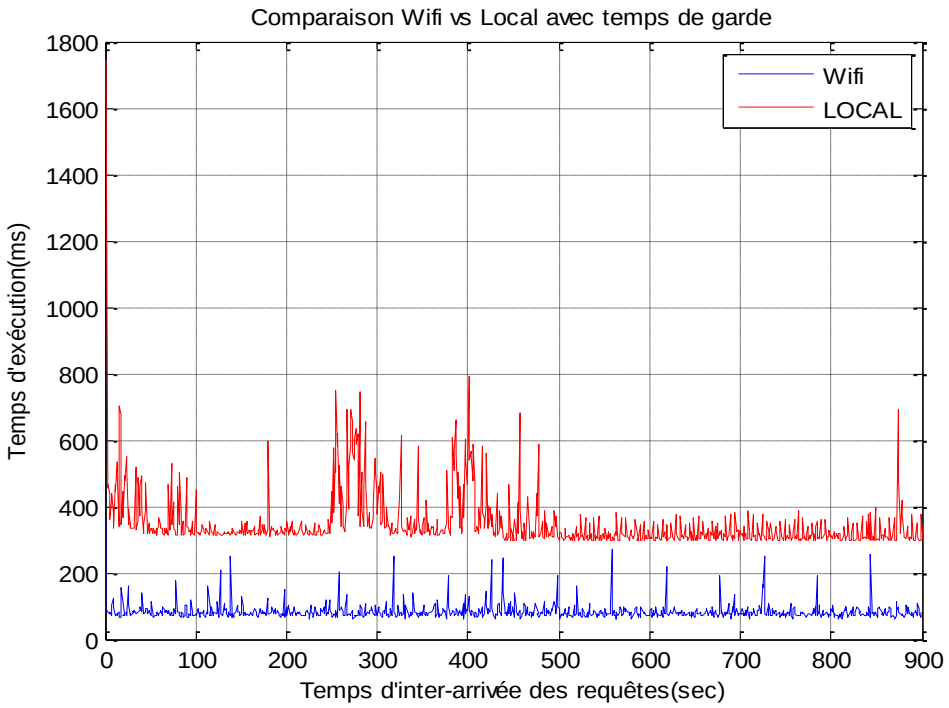

Figure 4.6. Comparaison des temps d'exécution en utilisant un temps de garde

Décomposition du temps de réponse dans le *Cloud*

La figure 4.7 nous permet d'apprécier la décomposition du temps de réponse d'une requête envoyée dans le *Cloud*. En effet, le temps de réponse total est décomposé en deux parties, le temps d'exécution sur le serveur situé dans le *Cloud* et le temps aller-retour. La figure 4.7 nous présente cette décomposition du temps de réponse total. Le temps d'exécution obtenu au niveau du *Cloud* est assez stable, ce qui reflète en général ce qui se passe au niveau de ces serveurs assez robustes et rapides. En effet, l'écart-type est de 1.472 millisecondes. Ceci nous permet de déduire que dans la majeure partie des cas, les décisions dépendent surtout du temps aller-retour de nos paquets. Ceci nous amène à décider de l'endroit d'exécution de nos tâches seulement par rapport au seul temps aller-retour lors d'un changement de réseau étant donné que le temps d'exécution dans le *Cloud* est déjà connu et que ce dernier n'aurait pas changé. Étant donné que le système dispose déjà du temps d'exécution dans le *Cloud* qui est assez stable, il suffit de connaitre le temps d'aller-retour du nouveau réseau, il est additionné au temps moyen qui avait été trouvé durant la période d'apprentissage pour trouver le temps de réponse total. Il est ensuite comparé au temps d'exécution sur le terminal mobile pour décider de l'endroit d'exécution. Pour décider ainsi, le développeur doit s'assurer que l'écart-type ne dépasse pas une valeur qu'il aura fixée. Cette valeur seuil a été arbitrairement fixée à 10 millisecondes dans le cas de notre application.

Pour une moyenne de 1.284 millisecondes et un écart-type de 1.472 millisecondes, le temps d'exécution dans le *Cloud* est vraiment insignifiant par rapport au temps aller-retour. Il n'en est pas toujours ainsi pour toutes les applications. Mais ce qu'il faut surtout noter ici est la valeur trouvée pour l'écart-type, ce qui nous amène à déduire que le temps d'exécution dans le *Cloud* varie très peu. Il est à rappeler aussi que ces exécutions ont été effectuées sur la plus lente des machines virtuelles mises en place sur le marché par Amazon, soit une micro instance avec un processeur d'une vitesse relativement faible et ne disposant pas de beaucoup de mémoire RAM. D'autres instances beaucoup plus rapides disponibles chez Amazon peuvent aussi être utilisées pour d'autres types d'applications encore plus gourmandes en ressources, ce qui garantirait aussi de bien meilleurs résultats. Pour notre scénario, cette instance répond bien à ce que nous voulons expérimenter. Nous avons obtenu un gain énorme comparativement au temps d'exécution trouvé sur le téléphone. Pour un temps d'exécution de plus de 300 secondes sur le

terminal mobile, nous avons obtenu un temps d'exécution d'environ 1 seconde dans le *Cloud*, le temps aller-retour n'étant pas considéré.

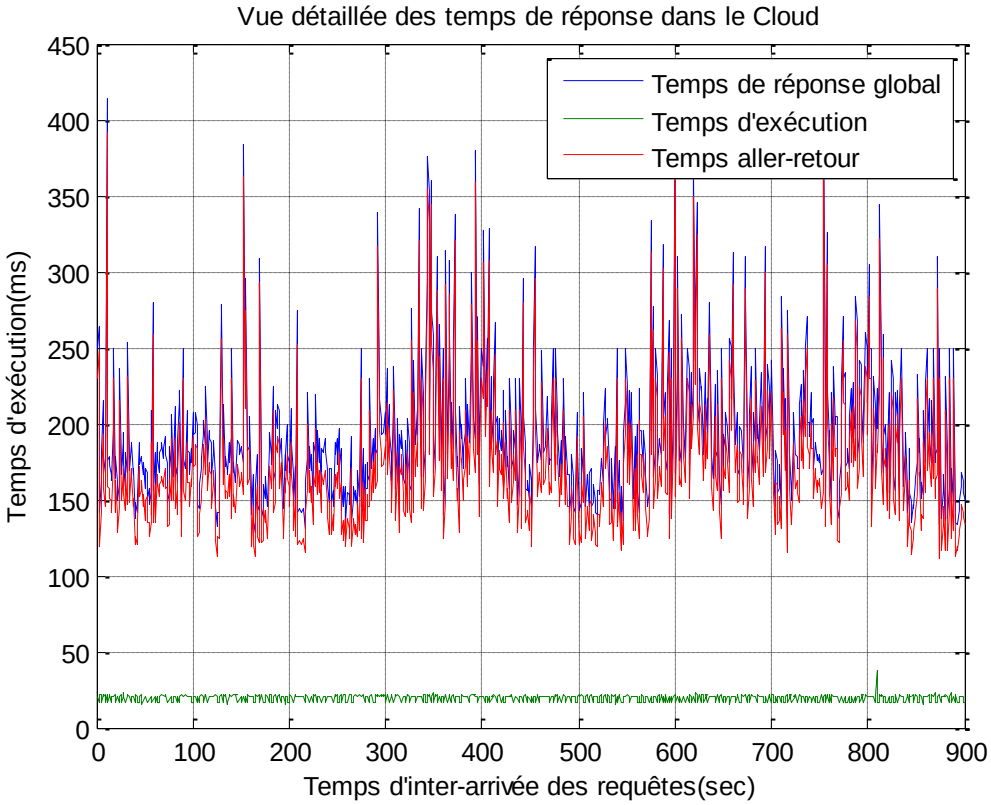

Figure 4.7. Décomposition des temps de réponse dans le *Cloud*

4.9 Scénario 6 : Exécution avec REST, sur un réseau Wi-Fi, avec un temps de garde

Ce scénario nous permet de mesurer le temps moyen d'exécution d'une tâche dans le *Cloud* en utilisant le style architectural REST. Dans ce cas, le réseau Wi-Fi est utilisé et le système envoie un ensemble de requêtes dans le *Cloud* pendant 15 minutes. Les requêtes sont envoyées à chaque seconde si le temps de réponse de la dernière requête est inférieur à 1 seconde. Sinon la prochaine requête est envoyée immédiatement après. Nous avons trouvé un temps de réponse moyen de 83.21 millisecondes, avec un minimum de 62 millisecondes et un maximum de 350 millisecondes. Les mesures effectuées ont une dispersion ou un écart-type de 25.05 millisecondes.

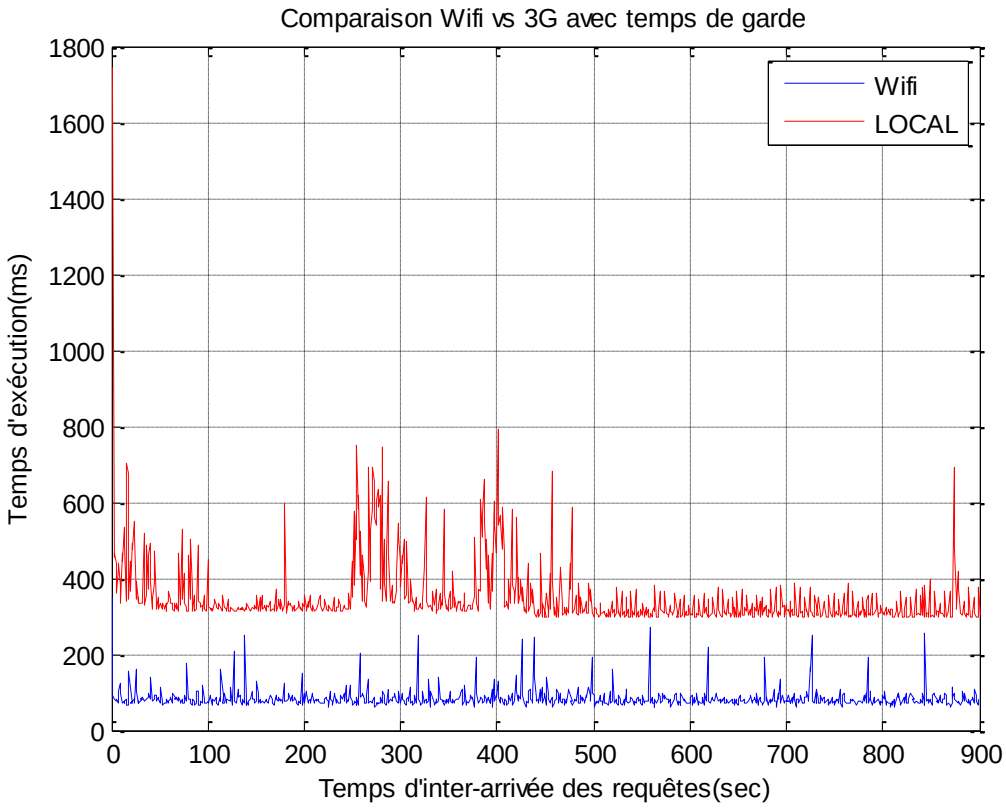

Figure 4.8. Comparaison des temps d'exécution avec temps de garde (Wi-Fi vs Local)

4.10 Scénario 8 : Influence de la taille des données

Ce scénario nous permet de mesurer le temps d'exécution sur un terminal mobile en tenant compte de la variabilité de la taille des données. Plus la taille des données est grande, plus le système est susceptible d'avoir un temps de réponse élevé pour une requête envoyée dans le *Cloud*. Pour tester ce scénario, nous avons fait exécuter les requêtes dans le *Cloud* en faisant varier la taille des données. Pour présenter nos résultats, nous utilisons deux tailles différentes pour les données qui sont envoyées, une taille de 20 octets et une autre de 500 octets. La figure 4.9 nous permet d'apprécier la différence des temps de réponse entrainée par la taille des requêtes. Ces exécutions ont été faites dans le même environnement réseau. Les requêtes sont envoyées à un même serveur durant 15 minutes à raison d'une requête par seconde. Pour s'assurer que l'environnement reste le même pour les différentes tailles des requêtes, nous les faisons exécuter en alternance l'une à la suite de l'autre, soit une requête de 20 octets suivi d'une requête de 500 octets.

Le temps d'exécution moyen pour les requêtes de 20 octets est de 170.9 millisecondes, le maximum de 384 millisecondes et le minimum de 129 millisecondes. La dispersion des données ou l'écart-type est de 37.92 millisecondes.

Pour les requêtes à 500 octets, le temps d'exécution moyen est de de 187.5 millisecondes, le maximum de 578 millisecondes et le minimum de 149 millisecondes. L'écart-type est de 36.82 millisecondes.

La figure 4.9 nous permet de voir la différence qui existe pour les temps d'exécution par rapport à la taille des données. Plus celle-ci est petite, plus le temps de réponse est petit. Il s'avère nécessaire de décider de l'endroit d'exécution des tâches par rapport à la taille des données car celle-ci peut faire varier considérablement le temps de réponse obtenu quand une requête est envoyée dans le *Cloud*.

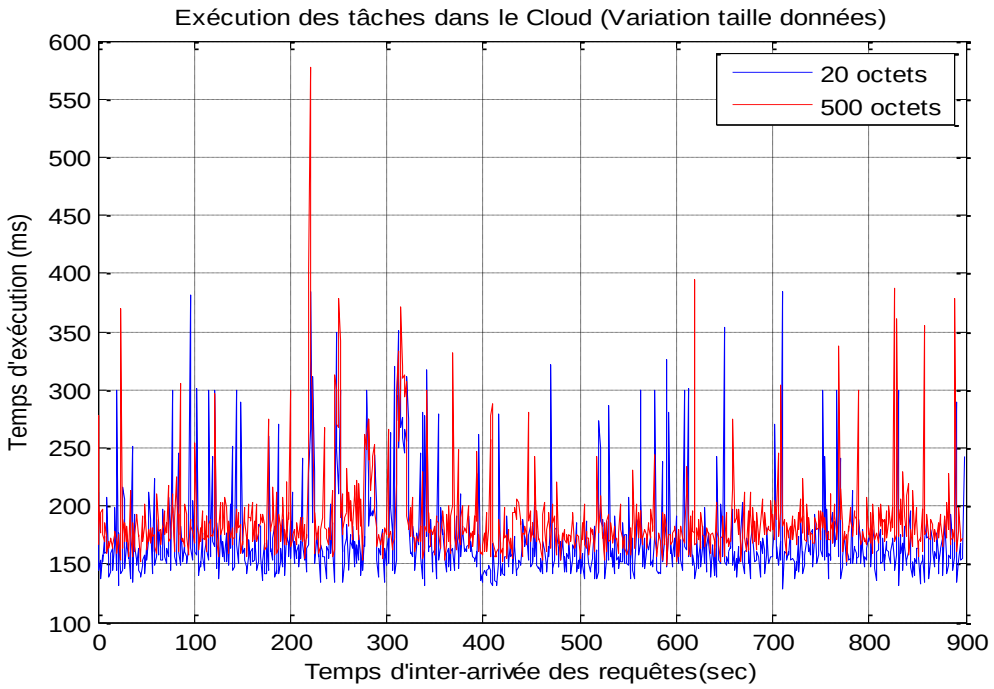

Figure 4.9. Comparaison des temps d'exécution avec différentes tailles de données

4.11 Scénario 7 : Exécution avec retour immédiat sur le terminal mobile dans le cas où le délai de garde est dépassé

Ce scénario nous permet de mesurer le temps d'exécution sur un terminal mobile en utilisant un délai de garde. Nous essayons de trouver la réponse à deux questions par rapport à ce

scénario. La première grande question est de savoir comment décider de la valeur à donner au délai de garde. La deuxième question est de savoir quand décider de revenir en local étant donné un certain nombre de délais de garde. Lorsque nous analysons les valeurs recueillies à la figure 4.2, décider de revenir en local sur la base d'un délai de garde ne tient pas. Étant donné qu'une valeur aberrante peut poindre à l'horizon sans pour autant indiquer la vraie tendance des temps de réponse dans le *Cloud*.

La première réponse à cette question serait de fixer le temps de garde par rapport aux caractéristiques du téléphone. Ces tests se sont avérés très difficile étant donné la variété des téléphones existant sur le marché. Les émulateurs d'Android ne donnent pas le même temps de réponse que les téléphones dans la réalité, ce qui ne nous permet pas non plus de conclure. Des tests ont été effectués en faisant varier les caractéristiques des émulateurs et les résultats n'ont pas été les mêmes que dans la réalité.

Une meilleure réponse par rapport à la première question est de fixer le temps de garde par rapport au temps d'exécution sur le terminal mobile. Cette moyenne étant une moyenne mobile, le système aura aussi un temps de garde variable et qui tient compte du fait que l'environnement dans lequel évolue le terminal mobile change constamment.

La deuxième question est quand l'utilisateur doit-il revenir en local advenant un certain nombre de temps de garde par rapport aux requêtes qui sont envoyées dans le *Cloud*. Le choix de revenir en local immédiatement après avoir enregistré un délai de garde ne se révèle pas intéressant. La figure 4.8 révèle qu'il arrive souvent d'atteindre le délai de garde sans pour autant avoir une meilleure situation en local. L'utilisateur peut avoir un train d'exécution dans le *Cloud* avec des durées intéressantes et subitement passer à une valeur aberrante. Nous devons prendre des mesures qui nous permettront de revenir en local au cas où le nombre de temps de garde enregistré serait trop élevé, ce qui se traduirait par des cas de réexécution des requêtes. Dans le cas où le temps d'exécution de la tâche dans le *Cloud* dépasse le délai de garde, le système lance à nouveau la requête dans le *Cloud*. L'utilisateur revient en local après un certain nombre de délais de garde fixé à travers le système expert. Ce paramètre est configurable et varie d'une application à une autre.

4.12 Scénario 9 : Passage d'un réseau Wi-Fi 1 à un réseau Wi-Fi 2 (Mobilité)

Ce scénario nous permet de mesurer l'impact de la mobilité des usagers. Quand une personne passe d'un réseau 1 à un réseau 2, la connexion au niveau réseau prend un certain temps. Si une requête était déjà envoyée sur le réseau 1 et que l'utilisateur n'avait pas encore reçu de réponse avant de quitter ce réseau, ce dernier risque d'attendre indéfiniment cette réponse du réseau 1 qui ne lui parviendra jamais dans le réseau 2. Pour résoudre ce problème, dès qu'un utilisateur se connecte à un nouveau réseau, le système revient instantanément en local un nouveau processus d'apprentissage est lancé car il faut tenir compte des paramètres de ce nouveau réseau qui sont en général différents de l'ancien.

Les paramètres de l'ancien réseau sont sauvegardés et il est permis à l'utilisateur de ne pas refaire l'apprentissage au cas où il y reviendrait avant une durée variable que nous nous sommes fixés arbitrairement à trois minutes dans notre cas.

Les tests qui ont été réalisés révèlent un trou considérable dans le cas où cette politique ne serait pas appliquée. Les cas de passage au nouveau réseau sont traduits par des trous assez considérables étant donné le temps de connexion au nouveau réseau. Pour simuler ce comportement, nous avons utilisé un réseau Wi-Fi. Nous nous déconnectons du réseau à un intervalle de 3 minutes pour passer d'un réseau à un autre et ceci sur une durée de 60 minutes. Chaque seconde, le système envoie une requête dans le *Cloud*.

4.13 Scénario 10 : Passage du Wi-Fi au 3G (Mobilité : Relève verticale)

Ce scénario nous permet de mesurer l'impact de la relève verticale par rapport à la mobilité des usagers. Un terminal mobile qui quitte un réseau Wi-Fi pour aller dans un réseau 3G ou vice-versa change de technologie. Quand le système a accès aux deux types de réseau Wi-Fi et 3G, la priorité est en général accordée aux réseaux Wi-Fi étant donné que la capacité de transmission des réseaux Wi-Fi est beaucoup plus élevée que celle des réseaux 3G.

Normalement quand l'utilisateur change de réseau, l'utilisateur devrait faire un nouvel apprentissage. Étant donné que le système connait le temps d'exécution des tâches dans le nuage par rapport à la période d'apprentissage, il suffit de connaitre le temps d'aller-retour d'un paquet pour connaitre le temps de réponse d'une requête envoyée dans le *Cloud*. Le temps de réponse d'une requête étant le temps aller-retour dans le *Cloud* additionné au temps d'exécution de la

tâche considérée. Cette approche nous impose de sauvegarder à des fins de prise de décision le temps d'exécution des tâches dans le *Cloud* à côté du temps d'aller-retour des requêtes. Pour éviter de relancer à nouveau des requêtes dans le *Cloud* sous peine de les voir prendre trop de temps, étant donné que le système ne connait pas encore le temps de réponse par rapport au nouveau réseau, des requêtes vides sont lancées dans le *Cloud* afin de trouver le temps aller-retour par rapport au nouveau réseau, ce qui nous permet de décider du nouvel endroit d'exécution des tâches considérées. L'apprentissage dans ce cas est fait dans le seul objectif de trouver le temps aller-retour des paquets. Ensuite ce temps est additionné au temps d'exécution dans le *Cloud* pour trouver le temps de réponse des requêtes envoyées dans le *Cloud*.

4.14 Scénario 11 : Test sur un réseau à Montréal

Pour asseoir nos analyses, nous avons aussi implémenté nos services sur un serveur situé ici au centre-ville de Montréal, étant donné que le *Cloud* d'Amazon se trouve aux États-Unis. Nous avons réalisé que la différence n'est pas notoire pour les temps de réponse obtenus. Nous avons trouvé un temps de réponse supérieur pour le *Cloudlet* situé ici à Montréal. Ce qui est le plus important que nous avons déduit est qu'on peut toujours gagner en temps d'exécution que nos requêtes soient envoyées dans le *Cloudlet* ici à Montréal ou dans le *Cloud* public d'Amazon aux États-Unis. Amazon EC2 est aujourd'hui disponible dans neuf régions du monde dont quatre aux États-Unis. Notre serveur se trouve en Virginie du Nord aux États-Unis, le plus rapide par rapport à notre positionnement géographique. Amazon garantit une disponibilité de 99.95% de ses instances peu importe la zone géographique. Ce qui fait une différence dans ce cas est la distance par rapport aux centres de données. Les instances virtuelles définies par Amazon étant standardisées, ils répondent aux mêmes critères de performance quelle que soit leur position géographique.

4.15 Mise au point autour des scénarios

Après cette analyse, nous nous permettons de faire une mise au point à travers ces trois situations. Dans la première situation, le téléphone fonctionne normalement, le terminal mobile accède au *Cloud* facilement et obtient un temps de réponse inférieur au temps d'exécution de la requête sur le téléphone. Dans cette situation, un gain considérable a été obtenu en faisant exécuter les tâches dans le *Cloud* et le choix pour l'endroit d'exécution de nos tâches n'est plus le

téléphone. Le temps d'exécution dans le *Cloud* est de loin supérieur à celui trouvé sur le téléphone. Nous avons bien fait la séparation des deux temps d'exécution en enlevant le temps aller-retour et le constat a été une différence en moyenne de plus de 200% du gain obtenu en temps d'exécution dans le cas de notre application. Si un utilisateur peut avoir un réseau en tout temps disponible avec une bonne connexion, l'exécution des tâches dans le *Cloud* aurait été le meilleur des choix car il n'aurait presque rien à faire exécuter sur le téléphone. Les résultats, une fois obtenus, pourraient être utilisés par le terminal mobile.

Dans le deuxième type de situation, l'exécution des tâches dans le *Cloud* est impossible ou très lent car le temps de réponse obtenu est nettement supérieur au temps d'exécution des tâches sur le téléphone. Dans ce cas, l'endroit d'exécution des différentes tâches ne présente aucune difficulté étant donné que le téléphone s'impose par sa vitesse d'exécution ou que les conditions d'exécution dans le *Cloud* sont trop difficiles.

Le troisième type de situation est celui qui pose le plus de problème. C'est le cas où l'on enregistre beaucoup de chevauchement dans la plage des temps d'exécution dans le *Cloud* et celle du téléphone. Cette situation nous amène à beaucoup plus de gymnastiques (ou beaucoup plus de considérations) pour trouver le meilleur endroit pour faire exécuter nos tâches.

Étant donné que l'environnement d'exécution des différentes tâches peut varier n'importe quand, (la connexion avec le *Cloud*, beaucoup plus rapide avec le Wi-Fi qu'avec le réseau cellulaire qui coûte beaucoup aujourd'hui, la quantité de mémoire disponible sur le *Cloud* et sur le téléphone, la quantité de CPU disponible sur le *Cloud* et sur le téléphone, la taille des données qui sont envoyée dans le *Cloud*, la capacité du disque dur disponible sur le téléphone), le système se fait vigilant par rapport à certains changements drastiques de situation. Ceci entraine un coût pour suivre l'évolution des temps d'exécution des tâches par rapport aux différents endroits mais nous permet de gagner en performance en nous évitant un arrêt complet du programme ou un prolongement de situation d'attente dans des cas où les temps d'exécution sortent de l'ordinaire.

CHAPITRE 5
CONCLUSION

Dans ce chapitre, nous faisons la révision des principaux points qui ont été abordés durant notre travail de recherche. Dans la revue de littérature, nous avons exploré les différents aspects liés à la déportation des tâches qui sont gourmandes en calcul dans le nuage informatique. Nous avons exposé certains concepts du domaine médical vu le contexte d'évaluation de notre prototype. Dans le troisième chapitre, nous avons présenté notre solution architecturale tandis que au chapitre quatre, nous avons présenté les différents scénarios qui valident notre architecture.

Ce chapitre clôture notre travail de recherche. Il s'en suit une synthèse des travaux qui ont été réalisés et nous présenterons par la suite les limitations de notre travail. Pour finir, nous donnerons un aperçu des améliorations possibles qui pourraient bien faire l'objet d'autres travaux dans le futur.

5.1 Synthèse des travaux

L'architecture du système que nous avons présenté ici est une proposition de résolution du problème de performance rencontré aujourd'hui par les applications qui sont implémentées sur les téléphones mobiles. Étant donné la limitation des ressources disponibles sur les terminaux mobiles, nous avons utilisé le *Cloud* pour mettre à la disposition de ces derniers une capacité de calcul et une capacité de stockage qui dépassent dans une très large mesure celle dont ils disposent en local. Les tests qui ont été réalisés en ce sens se sont révélés très concluants. Un utilisateur se trouvant ici au Canada et qui déporte ces calculs dans un nuage se trouvant aux États-Unis obtient une réponse beaucoup plus rapide par rapport à celle enregistrée quand la même tâche est exécutée sur le terminal mobile. Le temps d'exécution d'une tâche dans le *Cloud* peut être divisé en deux parties, une première partie qui est le temps d'exécution de la tâche sur un serveur situé dans le *Cloud* et une seconde qui est le temps pris pour envoyer la requête sur le serveur et obtenir la réponse sur le téléphone. Ce temps aller-retour ou RTT (Round Trip Time) additionné au temps d'exécution sur le serveur dans le *Cloud* est en général, dans les bonnes conditions de fonctionnement du réseau et du *Cloud*, plus petit que le temps d'exécution sur le téléphone. Pour pouvoir gérer toutes les situations que l'utilisateur peut rencontrer dans

l'exécution des applications par rapport à la dynamique de changement d'environnement du terminal mobile (changement au niveau du réseau, capacité mémoire, processeur, type de réseau, accès disque, etc.), nous avons proposé un système expert, qui sur la base d'un ensemble de connaissances mises en place par un ensemble de règles et de faits repérés à partir des différents tests qui ont eu lieu, permet de décider du meilleur endroit d'exécution de nos tâches.

5.2 Limitations des travaux

Les différents résultats qui ont été obtenus dans le cadre de notre travail montrent bien qu'en déportant certaines tâches dans le *Cloud* dans des situations idéales de fonctionnement de réseau et du serveur à distance, nous avons considérablement amélioré la performance de nos applications sur les terminaux mobiles. Cependant notre travail comporte bien des limitations. L'un des problèmes confrontés par les terminaux mobiles est celui de la conservation de l'énergie. Ce problème n'a pas été pris en compte par notre travail qui a seulement cherché à améliorer la performance de nos applications.

Une autre limitation de notre travail est que nous n'avons pas tenu compte des paramètres pouvant influencer notre décision côté serveur. S'il arrive un problème dans le *Cloud*, il serait possible de l'anticiper et ne pas seulement tenir compte du délai de garde, ce qui pourrait peut-être améliorer la solution.

Dans le cadre de notre travail, nous avons considéré un seul *Cloud* mais nous pourrions aussi l'améliorer en utilisant plusieurs *Cloud*. Le serveur situé dans le *Cloud* peut devenir indisponible ou le réseau y conduisant peut être congestionné. Ces problèmes nous imposent de faire exécuter nos tâches sur le terminal mobile alors qu'il serait possible de les faire exécuter dans un autre *Cloud*. Ce sont autant d'avenues à exploiter qui peuvent faire l'objet d'autres travaux dans le futur.

5.3 Travaux futurs

Vu les limitations de notre système, d'autres travaux peuvent être réalisés dans le but de l'améliorer. Le temps aller-retour dans le *Cloud* reste un problème assez considérable qui pourrait être amélioré si l'utilisateur puisse utiliser d'autres réseaux situé dans son voisinage immédiat. En ce sens, d'autres études peuvent être menées par rapport à la migration des machines virtuelles qui exécuteraient les tâches sur les serveurs situés dans l'environnement immédiat du téléphone.

La réduction du temps de migration pourrait bien aider à réduire le problème du temps aller-retour s'il est possible de le faire dans une plage de temps assez satisfaisant.

Ça peut aussi arrivé, pour une raison ou pour une autre, qu'un *Cloud* ne peut plus répondre à l'application cliente. L'utilisation de plusieurs *Cloud* peut permettre de résoudre ce problème en permettant de passer d'un *Cloud* à un autre qui soit fonctionnel à ce moment. Plusieurs *Cloud* peuvent aussi être analysés tout au long de l'exécution d'une application, ce qui permettrait de changer de *Cloud* par rapport à l'évolution des différents temps de réponse, étant donné la dynamique de changement des différents environnements.

Étant donné que le domaine médical est un domaine très sensible, il est impératif de considérer le volet de la sécurité afin de pouvoir rassurer les patients. Sans une fiabilité du système, nous ne saurions utiliser le *Cloud* pour des applications hyper sensibles. Déporter les tâches dans le *Cloud* peut présenter une menace réelle pour les patients si tout n'est pas minutieusement analysé. D'où l'intérêt d'une étude beaucoup plus approfondie pour garantir l'intégrité des données des patients et gagner leur confiance par rapport à la fiabilité du système.

BIBLIOGRAPHIE

[1] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301-314.

[2] P. M. Byung-Gon Chun, "Dynamically Partitioning Applications between Weak Devices and Clouds," 2010.

[3] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 2012, pp. 29-36.

[4] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," *Sensors*, vol. 12, pp. 11734-11753, 2012.

[5] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," *Middleware 2009*, pp. 83-102, 2009.

[6] M. Nkosi and F. Mekuria, "Cloud computing for enhanced mobile health applications," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 629-633.

[7] E. Lagerspetz and S. Tarkoma, "Mobile search and the cloud: The benefits of offloading," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, 2011, pp. 117-122.

[8] J. Lakshmi and S. S. Vadhiyar, "Cloud Computing: A Bird's Eye View."

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50-58, 2010.

[10] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," *Cloud Computing*, pp. 626-631, 2009.

[11] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, 2012.

[12] W. Jiang, X. Wang, Z. Yang, P. Peng, Y. Lu, and H. Ren, "The present situation and application of Biosensor," in *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, 2011, pp. 1743-1746.

[13] M. Malik, J. T. Bigger, A. J. Camm, R. E. Kleiger, A. Malliani, A. J. Moss, *et al.*, "Heart rate variability Standards of measurement, physiological interpretation, and clinical use," *European Heart Journal*, vol. 17, pp. 354-381, 1996.

[14] A. M. H. Kuo, "Opportunities and Challenges of Cloud Computing to Improve Health Care Services," *Journal of Medical Internet Research*, vol. 13, 2011.

[15] R. Padhy, M. R. Patra, and S. C. Satapathy, "Design and Implementation of a Cloud based Rural Healthcare Information System Model," ed: UNIASCIT, 2012.

[16] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, *et al.*, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49-62.

[17] B. G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland*, 2009.

[18] B. G. Chun, S. Ihm, P. Maniatis, and M. Naik, "Clonecloud: boosting mobile device applications through cloud clone execution," *arXiv preprint arXiv:1009.3088*, 2010.

[19] D. Gombarska and M. Horicka, "Evaluation of heart rate variability in time — Frequency domain," in *ELEKTRO, 2012*, 2012, pp. 415-418.

[20] P. Mell and T. Grance, "Draft nist working definition of cloud computing," *Referenced on June. 3rd*, 2009.

[21] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, *et al.*, "NIST Cloud Computing Reference Architecture," *NIST Special Publication*, vol. 500, p. 292, 2011.

[22] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2011.

[23] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, 2000.

[24] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 2012, pp. 21-28.

[25] V. Galetic, I. Benc, S. Desic, J. Krizanic, M. Mosmondor, A. Grguric, *et al.*, "Ericsson mobile health solution overview," in *MIPRO, 2010 Proceedings of the 33rd International Convention*, 2010, pp. 350-354.

[26] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of Cloudlets on Interactive Mobile Cloud Applications," in *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International*, 2012, pp. 123-132.

[27] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14-23, 2009.

[28] D. Huang, X. Zhang, M. Kang, and J. Luo, "MobiCloud: building secure cloud framework for mobile computing and communication," in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, 2010, pp. 27-34.

[29] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1-8.

[30] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010, p. 6.

[31] A. W. Brown, *Large-scale, component-based development* vol. 1: Prentice Hall PTR Englewood Cliffs, 2000.

[32] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[33] W. H. Press and G. B. Rybicki, "Fast algorithm for spectral analysis of unevenly sampled data," *The Astrophysical Journal*, vol. 338, pp. 277-280, 1989.

[34] F. Hayes-Roth, D. Waterman, and D. Lenat, *Building expert systems*, 1984.