Comparaison de plateformes logicielles pour programmation de services Web dans un environnement aux ressources limitées

par

Marc Dagenais

Essai présenté au CeFTI

en vue de l'obtention du grade de maîtrise en génie logiciel incluant un cheminement de type cours en technologies de l'information

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, octobre 2016

Sommaire

L'Internet des objets est un système complexe intégrant les objets du monde physique au réseau Internet. Les objets branchés sont en interaction avec leur milieu, et échangent de l'information avec différents clients, machines ou utilisateurs à l'aide de service Web. Les ressources disponibles pour l'exécution des logiciels sur un tel objet sont restreintes. Ainsi, un service Web programmé selon le protocole SOAP et le format de notification XML utilise les ressources de façon différente d'un autre programmé avec le style d'architecture REST et le format d'échange de données JSON.

Cet essai vise à vérifier si la plateforme logicielle impacte l'efficacité d'un service Web exécuté dans un environnement aux ressources limitées. Si tel est le cas, la plateforme logicielle de programmation d'un service Web devrait être déterminante sur l'utilisation des ressources.

À cette fin, un service Web programmé en Java, un deuxième service Web programmé avec la plateforme Node.js, et un troisième programmés avec le langage Go serviront de comparatif. De ces trois services Web, le temps d'exécution des requêtes, l'utilisation du processeur et la consommation d'énergie sont utilisés aux fins de comparaison. Chacun de ces services est exécuté dans le même réseau IP et dans le même environnement aux ressources limitées.

Les résultats obtenus démontrent que la plateforme logicielle d'un service Web influence les résultats pour chaque élément de mesure utilisé. Les langages supportant un paradigme de programmation événementiel comme JavaScript avec la plateforme Node.js, ou encore un paradigme concurrentiel pour le langage Go obtiennent de meilleurs résultats que celui programmé avec un langage plus généraliste comme Java. Aussi, la consommation d'énergie du système observe la même tendance.

L'efficacité d'un service Web dans un environnement aux ressources limitées est influencée par la plateforme logicielle. Ainsi le choix de cette plateforme s'accorde avec l'idée de trouver des solutions logicielles correspondant bien aux problèmes soulevés.

Remerciements

Tout d'abord, je tiens à remercier mon directeur académique Pierre-Martin Tardif et mon directeur professionnel Patrice Roy qui ont accepté de me guider durant la rédaction de cet essai. Je voudrais également remercier Claude Cardinal, pour ce programme d'études formateur, ainsi que Vincent Echlard et Lyne Legault pour leurs précieux conseils et encadrements.

Aussi, je tiens à remercier Martin Lessage et Kaled Taibi, respectivement technicien électronique et ingénieur, leur expertise m'a permis de mesurer adéquatement l'alimentation électrique d'un circuit électronique. Je remercie également Raphael Nadeau et Karim Samouda pour leurs aides et conseils.

Finalement, je remercie tous les membres de ma famille qui m'ont encouragé et qui ont dû subir mes absences prolongées pendant mes études.

Et spécialement merci à Frédérique, Gabrielle et Marie-Laure!

Table des matières

Sommaire i

Remerciements ii

Table des matières iii

Liste des tableaux v

Liste des figures vi

Glossaire viii

Liste des sigles, des symboles et des acronymes ix

Introduction 1

Chapitre 1 Mise en contexte 3

1.1 Couches logicielles de l'Internet des objets 3

1.2 Concepts en jeu 6

1.3 Problème de recherche et contexte de réalisation 10

Chapitre 2 Revue de la littérature 12

2.1 L'Internet des objets 12

2.1.1 Architecture 13

2.1.2 De l'intergiciel au service Web 14

2.2 Structures d'un service Web 15

2.3 L'efficacité d'un service Web 16

Chapitre 3 Problématique 18

3.1 Description 18

3.1.1 Objectifs et hypothèses 19

3.1.2 Limites 20

Chapitre 4 Méthodologie et approche de recherche 22

4.1 Méthodologie proposée 22

4.2 Mise en œuvre 24

4.3 Description de l'approche 29
4.4 Résultats obtenus 30

Chapitre 5 Analyse des résultats 32
5.1 Taille du code exécutable et nombre de lignes de code source 32
5.2 Temps de réponse aux requêtes 34
5.3 Utilisation de l'unité centrale de traitement 38
5.3.1 Quantité de travail de l'unité centrale de traitement 40
5.3.2 Pourcentage d'utilisation de la mémoire vive 41
5.3.3 Temps d'attente en entrée/sortie 42
5.4 Consommation électrique 43
5.5 Analyse comparative des résultats 46
5.6 Retour sur les hypothèses 48

Conclusion 50

Liste des références 52

Bibliographie 55

Annexe I Code des services Web 60

Annexe 2 Résultats temps de réponse 68

Annexe 3 Résultats unité centrale de traitement 71

Annexe 4 Résultats consommation électrique 74

Liste des tableaux

Tableau 1 Comparaison SOAP/XML et REST/JSON 8

Tableau 2 Réponses JSON 28

Tableau 3 Poids requête/réponse en ko 28

Tableau 4 Modèle de tableau de taille en ligne de code 30

Tableau 5 Taille en lignes de code 33

Tableau 6 Résumé comparatif 47

Liste des figures

Figure 1 Architecture conceptuelle de l'IdO 4
Figure 2 Représentation d'un service Web 5
Figure 3 Fil d'exécution événementiel 9
Figure 4 Schéma conceptuel 20
Figure 5 Laboratoire de test 24
Figure 6 Circuit d'alimentation 25
Figure 7 Capteurs 25
Figure 8 Service Web 1 26
Figure 9 Service Web 2 27
Figure 10 Service Web 3 27
Figure 11 Modèle de graphique temps de réponse 31
Figure 12 Taille du code exécutable 32
Figure 13 Temps de réponse température 35
Figure 14 Temps de réponse température minimum - moyen - maximum 36
Figure 15 Temps de réponse moyen aux requêtes 37
Figure 16 Temps de réponse des vingt premières requêtes 38
Figure 17 Utilisation du processeur 39
Figure 18 Charge du processeur 40
Figure 19 Utilisation de la mémoire 41
Figure 20 Attente en entrée/sortie 43
Figure 21 Consommation au repos 44

Figure 22 Consommation en utilisation 45

Figure 23 Disponibilité 46

Glossaire

Platine d’expérimentation : Dispositif qui permet de réaliser le prototype d'un circuit électronique. 26

Liste des sigles, des symboles et des acronymes

6LoWPAN : IPv6 Low power Wireless Personal Area Networks 13

GPIO : General Purpose Input/Output 11

HTTP : HyperText Transfer Protocol 13

IdO : Internet des objets 3

IPv6 : Internet Protocol version 6 13

JSON : JavaScript Object Notation 16

mAh : milliampère-heure 17

REST : Representational State Transfer 6

RPC : Remote Procedure Call 7

SOA : Service Oriented Architecture 13

SOAP : Simple Object Access Protocol 7

W3C : World Wide Web Consortium 6

XML : eXtensible Markup Language 15

Introduction

Internet est en constante évolution; l'Internet des objets connectés est sans doute une nouvelle phase de cette évolution. L'intégration des objets au réseau Internet comporte des défis de programmation particuliers. Pour permettre l'intégration et la communication des objets avec Internet, il peut être utile de considérer l'utilisation de services Web dans une architecture orientée services. De plus, il faut s'attarder à l'efficacité du service Web s'exécutant dans cet environnement restreint en ressources.

Cet essai s'inspire de l'article « *Efficient Application Integration in IP-Based Sensor Networks* » de Yazar et Dunkels de l'institut suédois d'informatique [1], ayant pour objectif de démontrer l'efficacité d'un service Web utilisant l'approche REST et le format de données JSON pour échanger des données sur un réseau de capteurs branché dans un réseau IP.

Ainsi, pour aller un peu plus loin que cet article, l'objectif de cet essai est de démontrer que la plateforme logicielle d'un service Web est elle aussi d'une grande importance sur l'efficacité du service. Est-ce que la plateforme logicielle influence l'efficacité d'un service Web? Le temps de réponse aux requêtes, l'utilisation de l'unité centrale de traitement et la consommation d'énergie sont utilisés pour mesurer l'efficacité d'un service Web s'exécutant dans ce type d'environnement. Alors, dans un contexte où les ressources sont limitées, est-il possible d'obtenir d'un service Web une meilleure efficacité selon la plateforme logicielle utilisée?

Le premier chapitre fait une mise en contexte de l'Internet des objets et de son architecture. Les différents concepts d'échange de données, le style REST et le protocole SOAP y sont définis. L'intérêt de la programmation événementielle pour un service Web s'exécutant dans un environnement restreint en ressources y est présenté.

Le deuxième chapitre est consacré à la revue de la littérature. Les différents composants qui constituent l'architecture de ce système de systèmes qu'est l'Internet des objets y sont présentés au travers de divers articles scientifiques. Le service Web est positionné comme moteur d'exécution faisant le pont entre les capteurs et l'utilisateur final des données

transportées. Les caractéristiques des composants d'un service Web et les particularités et différences des formats d'échange des données XML et JSON y sont présentées. L'évaluation de l'efficacité d'un service Web est définie selon des mesures de temps de réponse, de charge du processeur, d'utilisation de la mémoire disponible, de temps passé en entrées/sorties et de la consommation d'énergie du système [1, 2].

Le troisième chapitre décrit la problématique de l'essai et situe le service Web au centre du questionnement. L'objectif est de comparer des services Web écrits avec différents langages de programmation sur différente plateforme logicielle, soit Java, JavaScript sur la plateforme Node.js, et la plateforme Go. Dans ce chapitre, deux hypothèses sont formulées et des limites sont établies pour encadrer la recherche. Il est donc pertinent de se demander si la plateforme logicielle influence l'efficacité d'un service Web dans un environnement aux ressources limitées.

Le quatrième chapitre décrit la méthodologie utilisée pour mesurer les services Web afin d'en faire une comparaison. Différents facteurs pour contrôler le cadre de l'expérience et assurer son succès y sont présentés. Ce chapitre explique de quelle façon le réseau de communication IP est mis en place pour l'expérience, et comment le circuit d'alimentation en électricité est modifié pour permettre de mesurer la consommation. La mise en place des capteurs ainsi que des services Web et leurs méthodes y sont décrits. Également, ce chapitre explique les détails de mise en œuvre de l'expérience et l'approche d'analyse des résultats.

Le cinquième chapitre présente l'analyse des résultats obtenus. L'analyse est effectuée en comparant les résultats des trois services Web selon différents angles soit la taille du code, le temps de réponse, les mesures obtenues auprès de l'unité centrale de traitement et la consommation d'énergie.

Chapitre 1

Mise en contexte

Il y a un long chemin à parcourir entre l'objet et l'Internet, et l'ensemble des éléments logiciels de communication à mettre en place pour y arriver est multiple. Peu importe la structure ou l'aménagement de ses éléments logiciels, une approche par service Web peut être efficace dans le cas où les ressources sont limitées.

L'évolution d'Internet propose d'établir le lien entre les objets, et de cette idée générale se dégage une foule de représentations de branchements. Ces branchements, dans une architecture orientée services, se réalisent avec des services Web, et chaque service est construit en utilisant différents protocoles et différents styles de programmation.

Ce chapitre a pour but de situer le service Web dans les différentes couches logicielles nécessaires aux objets branchés, et de présenter différentes approches de branchements possibles. Aussi d'exposer la situation de l'environnement d'exécution restreint en ressource qui impose des contraintes à la programmation.

1.1 Couches logicielles de l'Internet des objets

Même si l'expression n'est pas consensuelle, le Web 3.0 sera de toute vraisemblance la prochaine étape de transformation d'Internet. Ainsi le Web à venir sera celui des objets [3]. En 2020, il y aura plus de 26 milliards d'objets branchés selon Gartner [4]. Parler de l'Internet des objets (IdO), c'est avant tout parler d'un système de systèmes. Dans cette perspective, l'architecture conceptuelle de ce système de systèmes peut se présenter rapidement en quatre couches [5]. Ce modèle permet de définir le rôle de chaque couche logicielle et d'expliquer, à haut niveau, le fonctionnement des échanges entre chacune d'elles.

Le premier niveau, la couche « Présentation », sert d'interface de communication avec l'utilisateur. Ces échanges peuvent prendre différentes formes soit : l'affichage, la restitution des données, et les dialogues d'entrée/sortie. C'est cette couche qui est visible et qui fait la

liaison avec l'utilisateur. La couche « Réalisation », dernière couche du modèle, est chargée de l'encodage et du décodage des informations échangées avec les différents hôtes (application, service, appareil, contenu). Ces deux couches aux extrémités du modèle constituent les entrées et sorties du système.

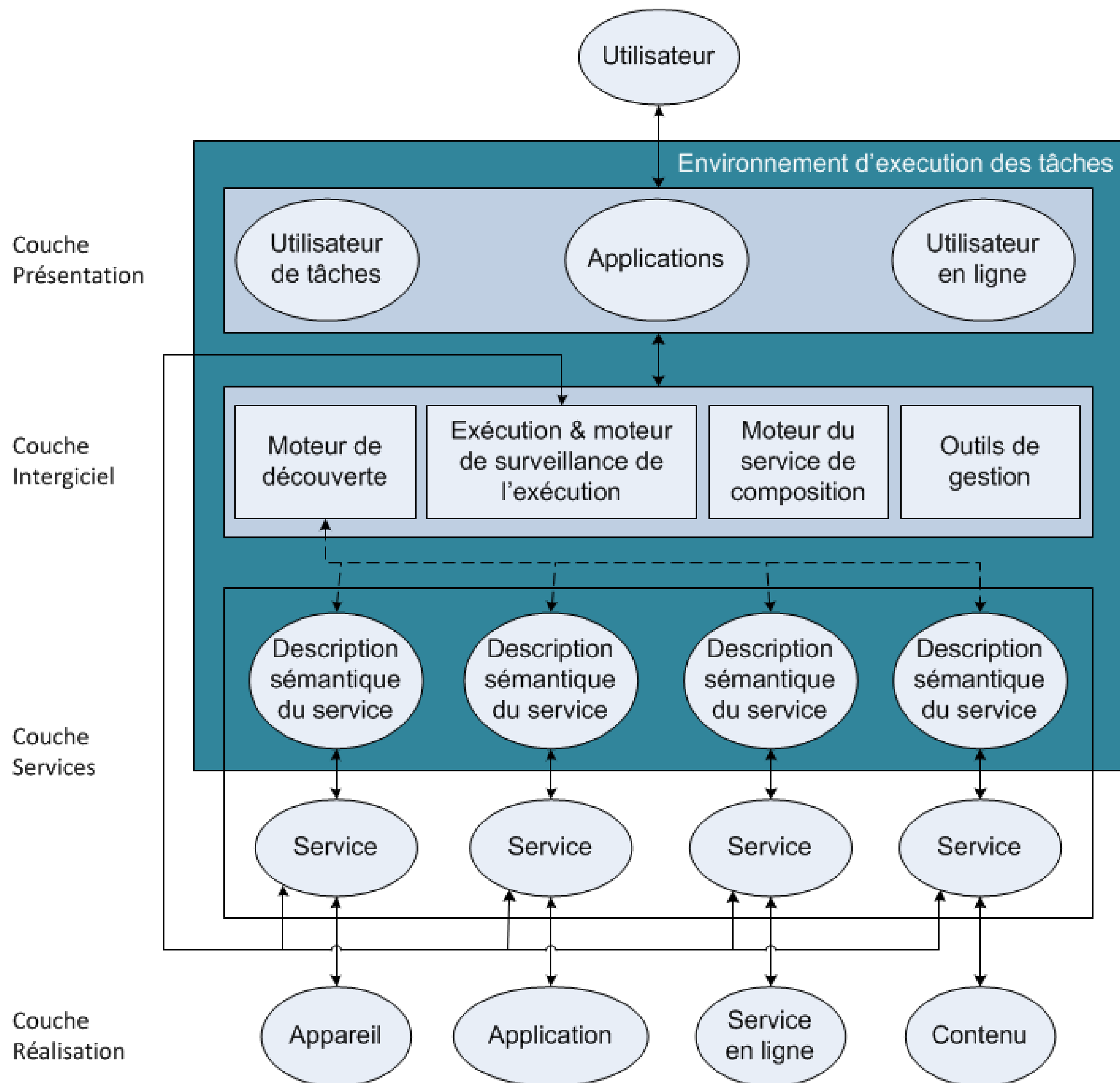


Figure 1 Architecture conceptuelle de l'IdO

Dans ce modèle, la description du service est présentée par la couche « Services », pour en permettre la découverte par d'autres systèmes. Chaque service consiste en une fonctionnalité ou une opération que le service peut réaliser.

La couche « Intergiciel » (*Middleware*) sert comme moteur d'exécution permettant la découverte et la communication. Cette couche implante la logique des opérations du système au travers de requêtes faites par la couche « Services ».

Ainsi, les couches « Présentation » et « Réalisation » correspondent aux applications concrètes offertes à l'utilisateur et aux entrées/sorties du système. La couche « intergiciel » offre une interface intermédiaire de découverte et de communication. C'est cette couche qui permet la reconnaissance et l'utilisation des services présentés par la couche « Services ». C'est aussi dans cette couche que se trouve le moteur de découverte et de communication.

Dans une architecture orientée services, cet intergiciel est la plupart du temps un service Web, et c'est l'optimisation de ce lien client/serveur qui nous intéresse.

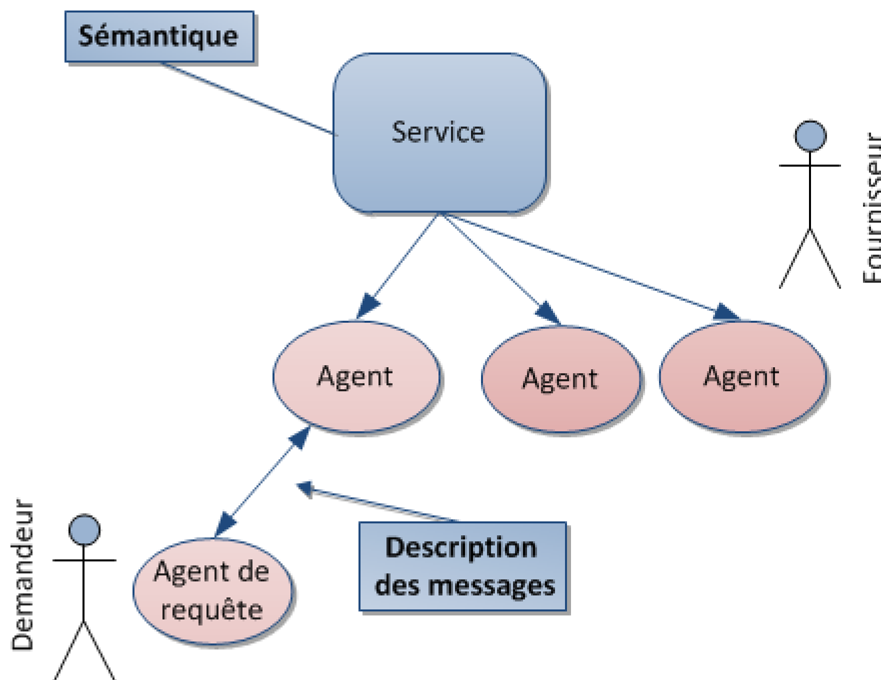


Figure 2 Représentation d'un service Web

Un service Web peut se constituer sous différentes formes, et se concrétiser avec différentes techniques. L'idée est avant tout de permettre la consommation d'un service par un client, le demandeur de la figure 2, tout en cachant la logique fonctionnelle aux utilisateurs [6].

1.2 Concepts en jeu

Dans un mode de communication client/serveur, le service Web permet l'échange des données entre les différents systèmes. Cet échange de données s'effectue selon des règles préétablies. Pour établir ces règles, il est possible d'utiliser un protocole de communication ou une architecture préétablis comme REST [7]. Le standard de base, normalisé par le W3C, est le protocole SOAP; le style d'architecture REST est aussi largement utilisé [8]. Le protocole de communication SOAP est typiquement utilisé pour faire appel à des procédures situées sur d'autres ordinateurs distants. Le style d'architecture REST est employé par les applications Web pour communiquer de l'information. Ces deux façons d'établir les règles d'échange de données entre deux systèmes sont fort différentes, il faut les envisager dans le contexte de l'IdO et dans un environnement où les ressources sont limitées.

Pour communiquer les données entre client et serveur, il existe des applications Web et des services Web. Les applications Web sont programmées pour une utilisation précise, et employées le plus souvent par un navigateur Web. Quant aux services Web, ils sont programmés pour avoir comme client d'autres programmes. Dans les applications Web, le style architectural REST est fortement utilisé. Ce style respecte certains principes; il impose des contraintes de découplage aux composants logiciels serveur et clients du système. Ainsi, l'évolution du logiciel client et du logiciel serveur est séparée et indépendante, ce qui a pour effet de réduire la dépendance entre le serveur et ses clients. Aussi, dans ce style architectural le système est hiérarchisé par couches, chaque couche étant indépendante des autres. Cela permet une évolution plus flexible de chacune des couches : le client peut ne pas connaître les différentes couches du serveur à certains moments dans le temps sans que cela n'entrave son fonctionnement, dans la mesure où les couches avec lesquelles il est directement en contact demeurent intactes.

Dans une architecture REST, le serveur ne dépend pas d'un contexte, c'est-à-dire qu'il ne conserve pas l'état de ses interactions avec le client : on le dit alors « sans états », ce qui laisse une plus grande indépendance entre le client et le serveur. Ainsi, le couplage entre clients et serveur demeure faible, un changement dans l'un ne demande pas nécessairement un changement dans l'autre. Aussi, certaines réponses du serveur peuvent être mises en cache, ce qui permet au client de se décharger des requêtes répétitives, parce qu'équivalentes. Le serveur offre une interface uniforme aux clients en identifiant chacune de ses ressources de façon unique, et en offrant une représentation définie, chaque message expliquant sa propre nature. Il ne devient pas nécessaire de faire une analyse syntaxique du message selon des règles préétablies [1].

En ce qui concerne l'approche SOAP, elle ne constitue pas un style d'architecture en soi, mais se veut plutôt un protocole de communication permettant d'appeler des procédures sur un ordinateur distant (*Remote Procedure Call*, RPC). Ce protocole se compose d'une enveloppe et d'un modèle de données. L'enveloppe contient la méta-information sur le message et sur le traitement à effectuer. Le modèle de données fournit la définition et la signification des informations échangées dans le message.

Avec SOAP, la définition de chacune des parties (enveloppe et modèle de données) s'élabore avec un langage de balisage XML. Ce langage facilite la création et l'échange de documents entre les systèmes d'information; il se structure par l'usage de chevrons encadrant des balises et validé par un schéma. L'approche SOAP implique un certain couplage entre le client et le serveur : en ce qui concerne l'interface, si les paramètres se trouvent à changer, un changement à l'interface du serveur signifie une modification du côté client. Le client reste captif de ce raccordement avec le serveur, il doit savoir effectuer l'analyse syntaxique du message.

Les techniques SOAP ou REST s'implantent différemment côté serveur, le client étant uniquement utilisé pour placer des demandes. Le serveur SOAP implique un analyseur syntaxique qui devra déchiffrer le XML de l'enveloppe et du message; cet analyseur alourdit son fonctionnement et le complexifie, son usage peut-être plus coûteux en ressources. Le serveur REST peut s'affranchir de ce besoin et s'implanter sur différents supports, car il ne participe pas au déchiffrage des données échangées. Le serveur REST peut échanger des

messages de différents formats, et le client doit quand même décoder la réponse servie par un serveur REST. Bien que la génération de messages en format JSON puisse s'effectuer avec n'importe quel langage de programmation, l'utilisation de JavaScript est légitime puisque le format JSON peut être naturellement interprété comme un objet JavaScript.

Tableau 1 Comparaison SOAP/XML et REST/JSON

SOAP/XML	REST/JSON
Protocole	Style d'architecture
Fort couplage	Faible couplage
XML implique un analyseur syntaxique	JSON est un sous-ensemble de JavaScript

Dans leur article, Yazar et Dunkels démontrent les avantages d'un service Web de type REST pour obtenir un meilleur temps de réponse et une plus petite consommation d'énergie qu'avec un service Web SOAP. Pour leur service Web, ils utilisent la plateforme logicielle Java. Dans un contexte aux ressources limitées, la plateforme Node.js constitue un candidat contrastant pour un service Web. Cette plateforme écrite en JavaScript s'exécute côté serveur. Le langage JavaScript reste communément utilisé aussi du côté client. Le format JSON demeure privilégié pour l'échange de messages entre un client et serveur écrit avec ce langage. Aussi, l'approche de programmation événementielle de cette plateforme permet à un service Web une meilleure utilisation du processeur parce qu'elle favorise les entrées/sorties non bloquantes [9].

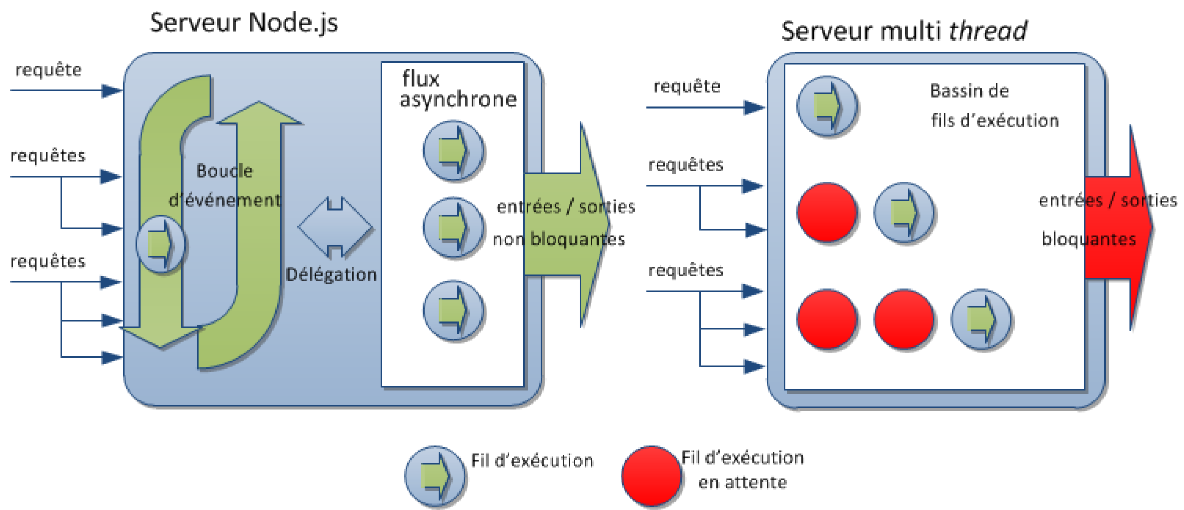


Figure 3 Fil d'exécution événementiel

Dans un système avec un processeur à un seul cœur, celui-ci peut être bloqué par l'attente d'une opération d'entrée/sortie fréquemment utilisée par les services Web. L'approche événementielle permet de signaler la disponibilité de cette opération qui était bloquée en attente. Cette approche préconise l'utilisation de fonctions de rappel. Une fonction de rappel est une fonction qui est passée en paramètre à une autre fonction à l'appel de celle-ci. Avec une fonction de rappel, les opérations d'entrée/sortie sont effectuées sur un autre fil d'exécution. Cela facilite les transitions lors des moments d'attente, et laisse du temps de traitement à d'autres opérations. Cette approche est plus efficace pour balancer la charge de travail : elle offre plus de possibilités de développement au programmeur en termes de contrôle sur la solution au problème de programmation qu'il doit résoudre. [9]

La plateforme Go est aussi une candidate intéressante pour un service Web dans un environnement aux ressources limitées. Elle présente un paradigme de programmation concurrentielle pour répondre au défi de communication d'un service Web.

Ainsi, la plateforme logicielle utilisée pour implanter un service Web peut impacter son efficacité. Cette plateforme logicielle ce compose d'une machine virtuelle pour Java ou d'un interpréteur pour Node.js et d'un moteur d'exécution pour Go, des bibliothèques utilisées pour communiquer au niveau inférieur avec les capteurs, et du serveur incluant son format de messages échangés avec le client. Il est possible de comparer l'efficacité d'un service Web

sur une plateforme logicielle Java, Node.js ou Go. Le concept d'efficacité se définit par le temps de réponse plus court du service aux requêtes client, une meilleure utilisation des ressources du processeur et une plus petite consommation d'énergie par l'objet.

1.3 Problème de recherche et contexte de réalisation

Le but de cette recherche est d'évaluer l'efficacité de la plateforme logicielle d'un service Web, dans le contexte de l'IdO et dans le cas où les ressources sont limitées. Dans ce contexte, les logiciels sont exécutés dans un environnement faible en mémoire (au maximum 512 Mo de mémoire vive) et avec un processeur limité en vitesse d'exécution (au maximum 700 MHz). Il est donc nécessaire d'accorder de l'importance à cet environnement pour fournir une solution logicielle qui lui convienne.

Ainsi, comment améliorer un intergiciel dans un tel contexte? Une architecture simple et évolutive de style REST, par opposition à un service Web SOAP plus complexe parce que nécessitant un analyseur syntaxique, devrait améliorer l'efficacité du service Web dans un milieu restreint en ressources. Bien que le style REST ne soit pas dépourvu de complexité et que la gestion des états revient aux clients, cette approche est plus avantageuse, car les réponses aux requêtes sont plus rapides et la consommation énergétique est plus faible qu'un service SOAP. Yazar et Dunkels en ont fait l'expérience et ont démontré la faisabilité et l'efficacité du remplacement d'un service Web SOAP par un service de style REST pour un service Web embarqué aux ressources limitées. Ils ont obtenu de meilleurs temps d'exécution, une plus faible consommation d'énergie par le serveur, et cela avec un code source moins volumineux [1].

Dans un service Web sans distinction de type REST ou SOAP, les messages échangés entre le client et le serveur sont structurés pour faciliter leur interprétation. La façon de formater les données échangées permet d'optimiser ces échanges en profitant des relations créées entre le contenant et le contenu. L'utilisation d'un format de notation tel que JSON, à titre d'alternative à XML pour structurer l'information, délivre le service Web d'une surcharge due à l'analyse syntaxique nécessaire à XML; en effet, JSON est interprété par le client ou le serveur comme un objet JavaScript, ce qui permet un accès direct aux données sans

transformations (description des messages Figure 2). Aussi, l'utilisation du côté serveur de la plateforme Node.js, par son approche de programmation événementielle, facilite le traitement des requêtes envoyées au serveur en évitant les entrées/sorties bloquantes, ce qui optimise le service dans son temps de réponse et diminue la consommation d'énergie de la carte embarquée.

Pour vérifier la faisabilité de cette approche et le gain en efficacité du système (taille du code, temps de réponse, utilisation du processeur, consommation électrique), un service Web est créé avec la plateforme Node.js et ces résultats comparés avec ceux d'un service Web programmé avec le langage Java comme celui utilisé par Yazar et Dunkels.

Le service Web est déployé sur une carte embarquée Raspberry Pi [10]. Cette carte permet de contrôler l'environnement à ressources limitées. La polyvalence de la connectique de cette carte, Ethernet et électronique, en fait un outil d'expérimentation intéressant. Il est facile de rouler un système d'exploitation sur cette carte et de la relier à Internet. Les périphériques sont facilement accessibles par le système d'exploitation, et il est possible de faire le pont avec d'autres circuits électroniques ou microcontrôleurs. En bref, avec cette carte, il devient simple de passer par Internet pour rejoindre un capteur par son contrôleur électronique GPIO, et ainsi démontrer les possibilités d'utilisation de ce système par d'autres domaines, tel la domotique.

Dans ce contexte expérimental, des mesures sont prises sur la carte Raspberry Pi lors des demandes aux services Web. Le temps de réponse à une requête du client, la charge de l'unité centrale de traitement, le temps du processeur passé en entrée/sortie, le pourcentage d'utilisation de la mémoire vive, de même que la consommation électrique du côté serveur sont mesurés et journalisés afin de comparaison.

Chapitre 2

Revue de la littérature

L'IdO est un système de systèmes vaste et complexe. Ce système forme un domaine qui peut être abordé sous plusieurs angles. En effet, l'architecture des systèmes d'objets branchés à Internet contient plusieurs couches. L'intergiciel est dans une position stratégique entre le capteur et l'utilisateur des données. Dans une architecture orientée services, cet intergiciel peut-être un service Web. Ce service Web est un programme client/serveur.

Le serveur qui offert le service est formé de différents composants. Dans la partie frontale, le format d'échange de données implanté par le serveur influence sa relation avec le client. Dans l'arrière-plan, la plateforme logicielle permet l'application concrète du service. Elle est constituée de plusieurs programmes qui peuvent être réalisés en différents langages de programmation. Alors l'efficacité du serveur est affectée, dans sa capacité à produire le travail attendu, par le nœud sur lequel il s'exécute parce que c'est un environnement restreint en ressources. Cette efficacité est mesurable du point de vue du client, en temps de réponse; du côté serveur, elle se mesure en consommation de mémoire, utilisation de processeur et consommation d'énergie.

2.1 L'Internet des objets

L'IdO est un sujet à la mode; la discussion sur ce sujet est ouverte, et ce sera vraisemblablement la prochaine ère informatique [11]. Ses impacts sont multiples : en informatique, en design, en information et sur les différents marchés d'affaires [12].

Il y a plusieurs articles scientifiques qui traitent de l'IdO, allant de l'architecture des solutions jusqu'aux langages de programmation susceptibles d'être utilisés.

Toutefois, deux articles scientifiques concernant directement les services Web traitent de l'efficacité des applications en contexte de ressources limitées. Le premier, concerne l'utilisation de l'approche SOAP pour concevoir un système de capteurs, et mesure l'efficacité

en termes d'utilisation de bande passante et de consommation d'énergie [2]. Le second, en réponse au premier, propose une solution basée sur l'approche REST et compare ses résultats en termes de temps d'achèvement des requêtes et de consommation d'énergie. Il affirme aussi que l'approche REST est un mécanisme plus simple [1].

2.1.1 Architecture

Plusieurs articles concernent l'architecture, non pas uniquement l'architecture logicielle, mais aussi l'architecture de solutions complètes pour l'implantation de réseaux d'objets branchés. SOCRADES [13] est une solution intégrée qui tente de couvrir l'ensemble de la modélisation manufacturière de l'objet, des requis de programmation, du design et de l'implantation. Le marché et la technologie y sont intégrés dans une vision d'ensemble. Également, l'approche d'architecture orientée services (*Service-Oriented Architecture*, SOA) est présentée dans l'optique de réutilisation future des solutions [14]. Ces points de vue sont de haut niveau, et présentent des solutions intéressantes parce que le raisonnement couvre l'intégralité du système, mais cette vision est beaucoup trop globale et laisse dans l'ombre des parties importantes tels que le côté développement et efficacité de ce genre de solution.

Au sujet d'architecture logicielle, l'approche Web et le protocole HTTP sont considérés comme prometteurs pour accéder à l'information d'objets. Une version légère des protocoles HTTP et TCP est proposée pour les environnements à ressources limitées [15].

Les protocoles de connexion sont souvent présentés comme des solutions aux nouveaux défis de l'IdO. Que ce soit IPv6 ou 6LoWPAN, l'utilisation de ces protocoles facilite la communication des capteurs et leur intégration avec le Web [16]. Les protocoles utilisés pour lier les objets entre eux ou à Internet sont nombreux et variés. Ils s'appliquent selon différentes situations, selon l'utilisation envisagée pour un système de capteurs. Il faut envisager le système dans son ensemble pour faire un bon choix de protocole. On associe souvent l'infonuagique à l'IdO; dans ce cas, plusieurs protocoles différents pourront être envisagés.

L'accessibilité des objets par l'infonuagique est proposée comme solution pour stocker les informations inhérentes aux objets. Cette architecture couple l'objet à une grille dans un système conçu pour capter ses informations [17].

De ce point de vue, il est important de bien comprendre la place de l'intergiciel dans ce système qui transporte l'information du capteur vers l'infonuagique ou l'inverse. Tel que discuté au premier chapitre, son architecture est présentée par couches et son rôle est primordial pour assurer l'interopérabilité selon différents scénarios d'utilisation des objets branchés [18]. Un scénario prometteur est d'intégrer les capteurs directement avec les systèmes d'entreprise, par exemple dans le progiciel de gestion intégré de l'entreprise, en utilisant les principes REST [19].

La programmation pour l'IdO est un défi, car les designs d'intégration nécessitent la connaissance de différentes techniques provenant de différents domaines. Des systèmes embarqués jusqu'à Internet en passant par l'infonuagique, l'intégration des différentes techniques peut être facilitée par de nouvelles façons de concevoir l'architecture et de nouveaux logiciels [20].

2.1.2 De l'intergiciel au service Web

L'interconnexion des objets est un défi important pour l'intergiciel : comme il est placé entre l'infrastructure globale du système et l'application qui reçoit l'information, son design est primordial. Il doit répondre aux défis d'interopérabilité, d'évolutivité, d'abstraction, d'intégration, de découverte automatique, de multiplicité et de sécurité [21]. L'intergiciel est un point central dans ce système de systèmes, et son développement doit être facilité pour les environnements faibles en ressources.

Ainsi, pour couvrir le terrain entre l'utilisateur final et le réseau de capteurs, les auteurs de l'article « *A middleware architecture for sensor networks applied to industry solutions of internet of things* » proposent un intergiciel composé de modules implantés selon différents designs [18]. Plusieurs intergiciels existants sont évalués et comparés, chacun utilisant des protocoles de communication différents. Plusieurs propositions de design s'y trouvent, mais peu d'évaluations de fonctionnement sont réalisées. Il n'y a pas d'évaluation d'efficacité de ces propositions pour voir le rapport entre le résultat attendu et le résultat obtenu. Pour Bandyopadhyay et coll. les d'intergiciels existes dans différents domaines et se trouvent construits selon différentes approches, à la fois génériques et particulières propres aux

contextes [22]. Les intergiciels sont présentés selon l'utilisation que le marché pourrait vouloir en faire sans vraiment montrer leurs composants ni leurs interactions.

L'intégration d'un intergiciel constitué par un service Web avec l'infonuagique est complexe [23]. Plusieurs éléments logiciels ou matériels de nature différente doivent être mis en interactions. Un service Web utilisant le protocole SOAP présente entre autres l'avantage de l'interopérabilité, et facilite la programmation d'un système de capteurs tout en le connectant librement à Internet [2]. Il est possible d'améliorer les temps de réponse en compressant les messages XML échangés par le système [24].

Un service Web appliquant les principes REST en utilisant des messages en format JSON est viable et avantageux dans un environnement aux ressources limitées, car il permet une plus grande vitesse d'exécution et une plus faible consommation d'énergie selon l'expérience menée par Yazar et Dunkels [1].

2.2 Structures d'un service Web

Bien que le service Web puisse être conçu avec différentes technologies, la structure de sa partie frontale peut-être généralement de deux types : REST ou SOAP [6]. Un service Web SOAP et un service Web REST sont différents de nature; les différences intrinsèques de leurs composants peuvent influencer leur efficacité. Ainsi, il convient de comprendre comment le service Web fournit l'accès aux données des capteurs, et comment cela impacte l'interopérabilité du réseau selon le type utilisé [25].

L'utilisation de XML par SOAP nécessite un analyseur syntaxique pour la lecture des données, voir Tableau 1. Il devient donc important, dans un environnement où les ressources sont limitées, de comparer son coût d'utilisation (processeur, mémoire vive) en opposition à un autre format d'échange de données comme JSON. Pour le Web 2.0, qui est un Internet plus riche en interactivité, les avantages de JSON sur XML à titre de format d'échange de données sont démontrés par son intégration plus facile avec le DOM des fureteurs [26, 27]. La notation JSON a l'avantage d'être plus rapide et plus légère, donc plus facilement interprété par le client. Par contre, la notation XML est standardisée et plus évolutive [26, 28].

Dans la structure d'arrière-plan, la plateforme logicielle destinée à soutenir le service Web peut aussi influencer l'efficacité d'utilisation des ressources du système par le service. La plateforme Java est fortement implantée pour le développement de services Web; son approche de programmation orientée objet facilite la création service Web de type SOAP en offrant l'accès à des bibliothèques intégrant un ensemble d'outils, de classe et d'API pour le développement de service Web de type SOAP. Dans le contexte de SOAP, l'analyse syntaxique de la description des services par le standard WSDL, *Web Service Definition Language*, simplifie le développement en décrivant les procédures offertes par le service [1].

Aussi, il peut être avantageux d'utiliser une plateforme en JavaScript comme Node.js s'exécutant du côté serveur. Le service Web peut profiter du gain en temps d'utilisation du processeur obtenu par sa programmation événementielle [9]. Selon Karagoz et coll. cette plateforme permet d'implanter un service Web REST pour un réseau de capteurs sans fil dans une architecture infonuagique en diminuant le temps de développement [29]. JavaScript est reconnu comme un langage permissif, et des problèmes liés à l'utilisation du langage sont principalement reliés au DOM, ce qui ne pose problème que du côté client [30].

Également, pour la partie serveur en arrière-plan, le langage Go est intéressant. Go est un langage de programmation avec un modèle de programmation orientée objet un peu différent sans notion de classe et d'héritage, mais avec un modèle objet basé sur l'interface. Ce langage est construit pour faciliter le passage à l'échelle et simplifier la programmation concurrentielle [31].

2.3 L'efficacité d'un service Web

L'efficacité d'un service Web se mesure souvent du point de vue du client par le temps d'accès aux ressources. Mais pour l'IdO, il faut aussi considérer la consommation de mémoire et d'énergie disponible, ainsi que la charge et la mémoire utilisées par le processeur, car les ressources sont restreintes. Dans certains cas, les capteurs se réveillent pour communiquer, et retournent en mode de veille pour économiser leur énergie. Il est possible de comparer l'utilisation des ressources disponibles par ces capteurs [23].

Les articles concernant les intergiciels traitent de différentes approches en termes de protocoles réseautiques, et aussi d'approche SOA incluant les services Web. Deux articles proposent des solutions logicielles différentes concernant l'efficacité de service Web pour un réseau de capteurs dans un contexte où les ressources sont limitées.

Le premier article mesure l'efficacité d'un service Web utilisant SOAP pour un réseau de capteurs [2]. Dans leur expérience, les auteurs mesurent la taille du code en octets des binaires en mémoire morte et la taille du programme lorsqu'il s'exécute en mémoire vive. Ils distinguent ensuite le temps d'exécution du service Web par le processeur du serveur et le temps de communication du message sur le réseau par les paquets TCP. En effectuant une requête selon trois modes, soit toutes les minutes, ensuite toutes les dix minutes, puis aux cent minutes, ils mesurent la consommation énergétique du système. En extrapolant les données de consommation énergétique du système sur l'énergie fournie par deux piles AA de 2200 mAh, les chercheurs obtiennent un comparatif des durées de vie énergétique du système selon le transfert de données, et cela pour chaque mode.

Le deuxième article, *Efficient Application Integration in IP-Based Sensor Networks*, mesure la taille du code source en octets et l'empreinte en mémoire vive pour chaque système [1]. Un premier service Web, ayant comme composant le serveur HTTP et un moteur REST, est comparé au deuxième ayant un serveur HTTP, un analyseur syntaxique XML, et un moteur SOAP. Chaque service Web (REST ou SOAP) offre les requêtes de température, de lumière et d'occupation. Sont aussi offerts un service de contrôle d'une diode électroluminescente (DEL) et un service « vide » qui ne retourne aucune donnée. Le temps d'achèvement des requêtes est mesuré selon deux protocoles qui contrôlent le courant des capteurs de deux façons différentes pour épargner de l'énergie. Le premier protocole permet aux capteurs de garder l'information de session du réseau, ce qui minimise le temps de requête, mais augmente l'utilisation d'énergie. L'expérience permet de comparer les résultats de consommation d'énergie et de temps de réponse d'un service Web REST et d'un service Web SOAP. Les résultats de consommation énergétique sont ensuite utilisés pour extrapoler la durée de vie de deux piles AA de 2500 mAh selon le nombre de requêtes par heure. Les résultats démontrent que la durée de vie des piles diminue selon le nombre requête par heure.

Chapitre 3

Problématique

Quoique l'étendue de son influence soit encore mal comprise, l'IdO aura un impact considérable sur le monde des affaires. 40 % des entreprises sondées par Gartner considèrent que l'IdO aura un impact à court terme sur leur entreprise, 60 % des entreprises sondées y voient plutôt un impact à long terme [28]. Comme lors de l'arrivée du réseau Internet, l'entreprise doit se préparer à cet impact, et il est donc important d'évaluer les solutions offertes. La présence dans Internet de l'information fournie par les objets prendra de plus en plus de place, et il devient donc à propos d'étudier cette mécanique de remontée d'information. Le modèle de communication client/serveur est idéal pour ce genre de transmission entre programmes. Dans cette optique, le client envoie des requêtes vers un serveur qui retourne en réponse l'information recueillie par l'objet capteur. Les technologies changent rapidement, il est donc approprié de mesurer l'efficacité d'un service Web développé sur différentes plateformes en différents langages de programmation.

3.1 Description

Dans le contexte où les ressources sont limitées, un service Web peut-il être plus efficace selon la plateforme logicielle utilisée? Une approche de programmation événementielle pourrait-elle permettre un gain sur le temps du processeur passé en entrée/sortie? Qu'en est-il de l'utilisation de la mémoire, du temps de réponse?

En répondant à ces questions, les développeurs sont en mesure de décider, de façon éclairée, quelle plateforme logicielle utilisée pour la programmation de service Web selon le contexte d'utilisation.

3.1.1 Objectifs et hypothèses

L'objectif est de vérifier si la plateforme logicielle utilisée dans la conception d'un service Web influence l'obtention de meilleurs temps d'achèvement pour les requêtes, d'une meilleure consommation d'énergie et d'une meilleure utilisation du processeur en ce qui concerne la charge de travail, l'utilisation de la mémoire et du temps passé en entrée/sortie. Ce test est réalisé sur une carte Raspberry Pi qui tient lieu de système aux ressources limitées.

En résumé, un service efficace offre le meilleur temps de réponse aux requêtes client, c'est-à-dire avec un intervalle de temps le plus court. Un service efficace utilise les ressources du processeur de façon à minimiser sa charge et à minimiser son temps d'attente lors des entrées/sorties, et cela tout en faisant une moins grande utilisation de la mémoire disponible. Pour terminer, un service efficace a une plus petite consommation d'électricité.

Il faut alors se demander : dans un contexte où les ressources sont limitées, est-il possible d'obtenir d'un service Web une meilleure efficacité selon la plateforme logicielle utilisée?

Formuler ainsi une première hypothèse : un service Web sur une plateforme logicielle Node.js apporte un gain en efficacité en comparaison à un service Web sur une plateforme logicielle Java, puisqu'un bénéfice est observable en ce qui concerne le temps d'achèvement des réponses aux requêtes. Un avantage est visible en pourcentage d'utilisation de la mémoire. Une plus petite charge de travail du processeur et une réduction du temps passé en entrée/sortie sont mesurables. Aussi la consommation d'énergie de la carte est plus faible parce que le processeur est moins sollicité.

Il faut aussi se demander : est-ce que c'est la plateforme logicielle utilisée qui influence l'efficacité du service Web? Ainsi, un service sur une plateforme différente dans un autre langage de programmation tel que le langage Go, utilisera les ressources d'une manière différente qui pourrait être ou non, plus efficace.

Formuler ainsi une seconde hypothèse : un service Web programmé dans un autre langage tel que Go, apporte un gain significatif en efficacité puisqu'un bénéfice est observable en ce qui concerne la taille du code, l'empreinte en mémoire vive, la charge de travail du processeur, le temps d'achèvement des requêtes/réponses et de la consommation d'énergie.

Dans ce cas, il sera avantageux de privilégier cette plateforme de programmation pour la conception des services Web qui assurent la communication avec les objets.

3.1.1.1 Schéma conceptuel de la recherche

Le schéma conceptuel de cette recherche est présenté dans la Figure 4 Schéma conceptuel :

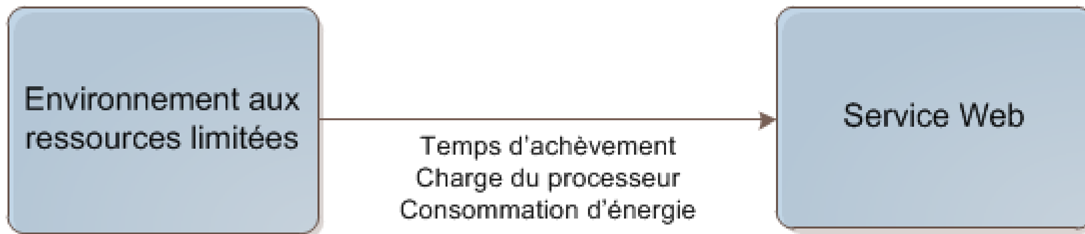


Figure 4 Schéma conceptuel

La variable indépendante, celle qui est manipulée, est le service Web. C'est cette variable qui sera isolée et modifiée pour que l'on puisse en voir les effets. Trois types de services Web sont utilisés pour faire l'expérimentation. Le premier est conçu à l'aide du langage Java de manière à reproduire celui utilisé dans l'expérience de Yazar et Dunkels [1], le second est conçu en JavaScript et repose sur Node.js, et le troisième a été écrit avec le langage Go. Les trois services Web échangent des messages dans un même format, soit JSON. L'expérience menée par Yazar et Dunkels démontre que le format d'échange de données JSON est moins coûteux que XML tel qu'utilisé dans l'expérience de Priyantha et coll. [2]. Pour chacun, les temps d'exécution et de consommation d'énergie d'une série de requêtes sont mesurés, ainsi que la charge de travail de l'unité centrale de traitement. Les services Web sont mis en œuvre sur une carte Raspberry Pi comme environnement aux ressources limitées, ce qui est la variable dépendante. En comparant les temps d'exécution, les mesures de charge de travail, d'utilisation de la mémoire, des attentes en entrée/sortie et la consommation d'énergie pour chacun des services, il est possible de savoir s'il est plus efficace d'utiliser un service Web plutôt que l'autre dans un tel contexte.

3.1.2 Limites

Les couches « Présentation » et « Réalisation » de la Figure 1 Architecture conceptuelle de l'IdOqui correspondent aux applications concrètes offertes à l'utilisateur et aux entrées/sorties

du système, ne seront pas considérées par ce travail. Ces deux couches sont à l'extrémité du modèle, et leurs interactions avec d'autres composants peuvent être multiples, ce qui augmenterait l'acquisition de données et cela sans apporter de crédibilité à la théorie.

Les différents protocoles de communication tels IPv4 et IPv6 ne seront pas explorés dans cette recherche. Un protocole sans fil de connexion avec les capteurs ne sera pas utilisé. La compression des données et la mise en cache seront contrôlées de façon équivalente d'un service Web à l'autre; l'efficacité de la mise en cache est largement documentée. Dans ce cas, la cache ne peut avantager un service Web plutôt qu'un autre. Pour contrôler le travail du service Web de façon équitable, chaque requête et réponse aura exactement la même taille d'un service à l'autre. La taille des messages envoyés est gardée intentionnellement petite pour centrer les mesures sur le travail du serveur. Dans l'architecture de test, l'objet tient le rôle de serveur bien qu'un objet pourrait aussi tenir le rôle de client.

Le prochain chapitre concerne la méthodologie de l'expérience et définit, de façon détaillée, l'approche utilisée pour répondre à la question de recherche.

Chapitre 4

Méthodologie et approche de recherche

Dans le contexte où les ressources sont limitées, un service Web peut-il être plus efficace selon la plateforme logicielle utilisée? Ce chapitre présente la façon de procéder afin de répondre à cette question. Le point de départ est inspiré de l'expérience menée par Yazar et Dunkels [1], dans laquelle un service Web de type REST codé en Java est comparé à un service de type SOAP utilisé dans l'expérience de Priyantha et coll. [2]. Dans cet essai, il s'agit d'une comparaison de plateforme logicielle pour trois services Web de type REST. Les prochaines sections présentent la méthodologie utilisée et les particularités de la mise en oeuvre de l'expérience, de même qu'une description de l'approche de collecte de données nécessaire à la comparaison. Une brève explication de la présentation des résultats obtenus termine le chapitre.

4.1 Méthodologie proposée

Cette étude est une recherche quantitative corrélationnelle prédictive. Elle consiste à faire une comparaison des données de temps d'achèvement des requêtes, d'utilisation de l'unité centrale de traitement, de mesure de taille des services et de la consommation d'énergie pour chaque service Web.

L'expérience s'est déroulée chaque fois avec le même système de capteurs, selon les mêmes règles de collecte de données, c'est-à-dire que chaque service Web est mesuré exactement dans le même milieu de travail. Ainsi, pour chaque service Web, la partie serveur du système est exécutée sur une carte Raspberry Pi, et la partie client lance les requêtes à partir d'un même ordinateur membre du même réseau. Le périmètre du réseau est contrôlé pour isoler le client et le serveur de l'encombrement des messages provenant des autres réseaux. Ce réseau est coupé d'Internet, afin d'éviter de charger inutilement le réseau par de tierces parties et de ralentir le temps de réponse du service Web.

La même série de requêtes est faite en répétition sur chaque service Web sur un laps de temps de 180 secondes. Cette série de requêtes est effectuée sur le service Web par le même client, en utilisant la même façon pour mesurer le temps d'achèvement et pour récupérer les données. Cette façon de collecter les réponses permet de contrôler les bruits externes qui pourraient influencer l'efficacité du serveur. Les réponses retournées par les services Web sont toutes du même format JSON et de la même taille. Elles imposent donc la même charge de travail sur le service.

Les caractéristiques matérielles de l'environnement d'exécution du côté serveur, comme la puissance de l'unité centrale de traitement, ou *Central Processing Unit* (CPU), la quantité de mémoire vive et la réseautique ont tous une influence sur le temps de réponse d'un service Web. La carte Raspberry Pi offre les mêmes caractéristiques côté serveur pour chaque service Web. Le fait d'avoir les mêmes milieux d'exécution du côté client et serveur pour chaque service Web permet d'isoler le service et d'assurer que rien ne vient biaiser les résultats.

Les services Web évalués ne sont différents que par leur serveur, un premier programmé en Java et un autre en JavaScript sur Node.js, puis un dernier en Go. Du côté client, sont mesurés la validité de la réponse obtenue, soit la bonne chaîne de caractères au format JSON, et le temps d'achèvement des requêtes c'est-à-dire le temps écoulé entre l'envoi du premier octet de la requête et la réception complète des entêtes de la réponse.

Pour l'unité centrale de traitement, la moyenne de la charge système, le nombre de secondes passées par le processeur en attente d'entrée/sortie et l'utilisation de la mémoire sont mesurés. Les mesures sont prises au repos, pendant l'exécution du service Web et pendant la période de requête par le client. La consommation d'énergie par la carte électronique est aussi mesurée pendant une même période de temps de 180 secondes. La mesure est prise lors des requêtes au service, sans requête au service, et au repos sans service Web en attente de requête. Le réseau est configuré de la même façon pour chaque service et isolé d'Internet. Chaque service Web s'exécute sur un Raspberry Pi modèle B à un seul cœur ayant une vitesse de processeur de 700 MHz, une capacité de mémoire de 512 Mo de mémoire vive, et d'une consommation maximale d'électricité de 480 mA et de 360 mA au repos. Les données obtenues dans ce contexte peuvent facilement se comparer, et ainsi

faire en sorte qu'aucune variable étrangère n'influence les résultats. Ainsi la comparaison des mesures permet de déterminer le serveur ayant obtenu les meilleurs résultats et dans les circonstances de favoriser une approche de programmation.

4.2 Mise en œuvre

Le laboratoire de test est formé d'un routeur fournissant un réseau local entre un ordinateur et une carte électronique, d'un capteur de température et d'une DEL, tel qu'illustré dans la Figure 5 Laboratoire de test.

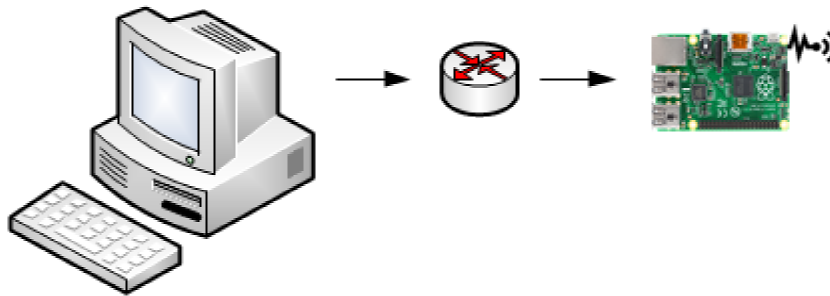


Figure 5 Laboratoire de test

La carte électronique sert de support pour l'intergiciel. Les capteurs sont reliés à cette carte, et cette carte est alimentée en énergie par une alimentation micro USB de façon contrôlée telle qu'illustrée dans la figure 6.

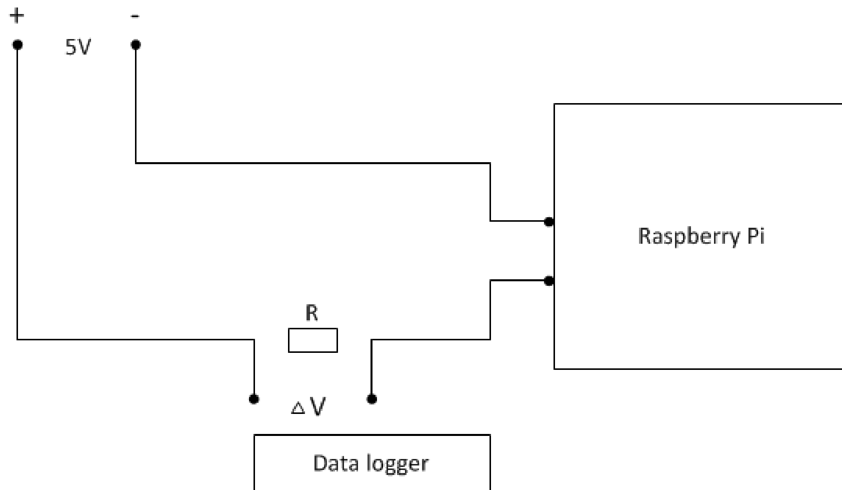


Figure 6 Circuit d'alimentation

Afin de calculer la consommation énergétique de la carte électronique pendant l'expérience, le circuit d'alimentation est modifié pour y intégrer un capteur de courant. Le circuit d'alimentation de la carte électronique est surveillé par ce capteur, qui délivre en sortie une tension correspondant à celle qu'il voit passer. Cette tension est enregistrée dans un journal sur une carte SD pendant l'utilisation du service Web. Le voltage retourné par le capteur varie de façon linéaire avec l'intensité détectée; il suffit de calibrer le capteur pour en déduire la consommation correspondante. Une fois la calibration effectuée, chaque tension enregistrée dans le journal correspond à une consommation en milliampère. Cette corrélation permet de construire une échelle de comparaison indiquant la consommation de la carte pendant l'expérience.

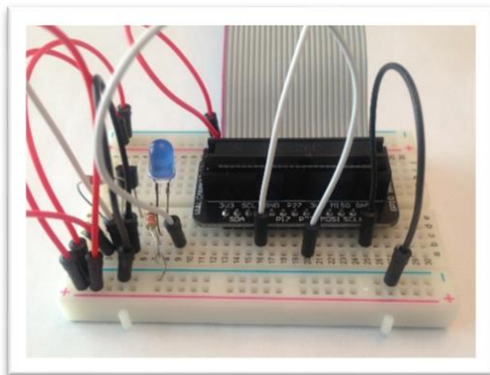


Figure 7 Capteurs

Deux capteurs sont branchés sur une platine d'expérimentation et reliées à la carte électronique par les ports d'entrées/sorties GPIO : une DEL bleue qui change d'état, étant tour à tour allumée puis éteinte, et un capteur qui mesure la température ambiante. Le lecteur des informations récupérées par les capteurs est fourni par une bibliothèque codée par une tierce partie et distribué libre de droits. Il s'agit de la même bibliothèque pour tous les services, mais la mise en application diffère selon le langage de programmation. Ce code permet une intégration rapide et simple entre les GPIO de la carte électronique Raspberry Pi, la platine d'expérimentation et le logiciel du service Web.

Une plateforme logicielle et un langage de programmation différent sont utilisés pour chaque service Web. Le premier service est codé en Java. Ce service est une interface de connexion représentée dans la Figure 8 Service Web 1par le serveur et le moteur REST. Cette interface suit le style d'une architecture REST et offre les réponses aux requêtes dans un format JSON.

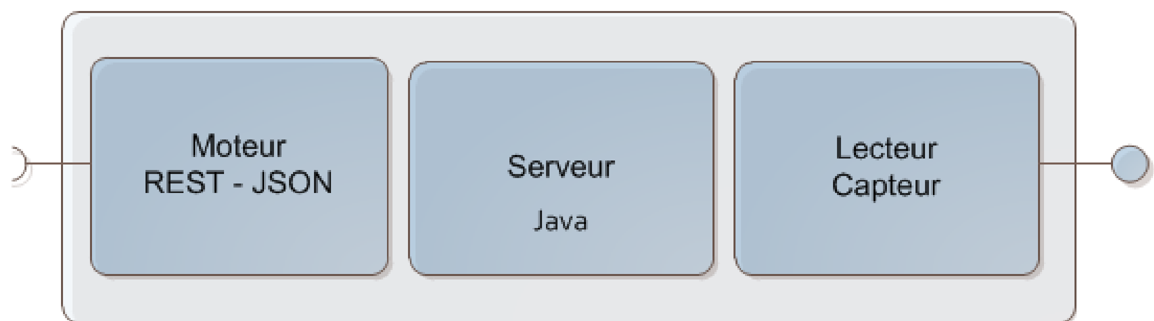


Figure 8 Service Web 1

Le service Web 2 est en tout point semblable au premier service sauf pour le serveur, comme illustré par la Figure 9 Service Web 2

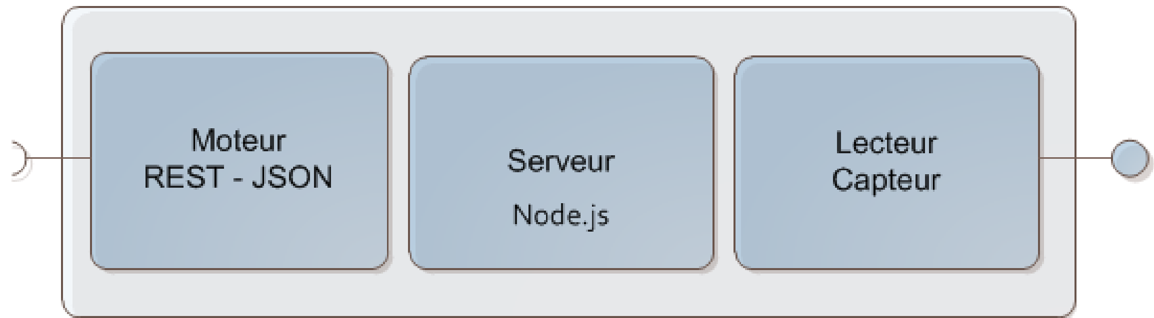


Figure 9 Service Web 2

La seule différence entre les services se situe au point de vue du serveur qui est conçu avec la plateforme Node.js. Cette différence constitue la variable indépendante qui est manipulée pour les besoins de l'expérimentation. Le troisième utilise le langage Go.

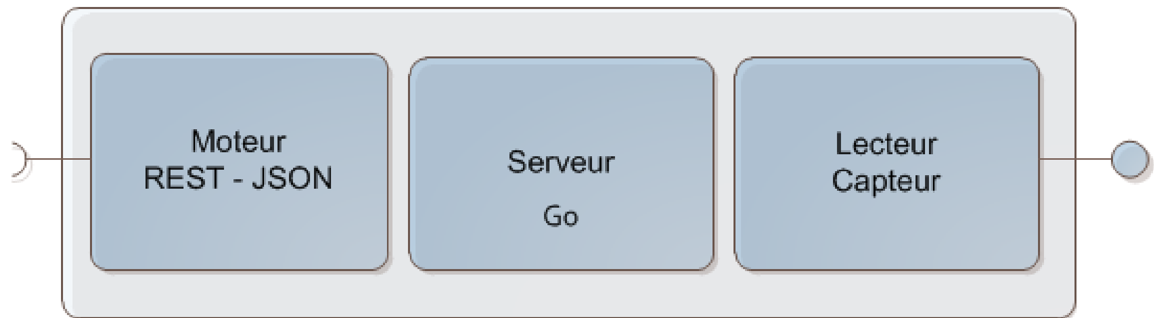


Figure 10 Service Web 3

Le code produit est conçu par un programmeur expérimenté, et validé par une revue de code effectuée par un tiers parti afin d'en assurer la fiabilité et la qualité. Tel qu'illustré par le Tableau 2 Réponses JSON, les trois services offrent les mêmes cinq requêtes au client, soit quatre avec le protocole HTTP GET et une avec HTTP POST. Ces requêtes retournent la même réponse en format JSON, ce qui assure que les trois services Web fassent le même travail pour retourner les mêmes réponses sans imposer un trop grand coût sur le système.

Tableau 2 Réponses JSON

Nom	Méthode	Requête	Réponse JSON
Bidon	GET	/	Allo le monde!
Température	GET	/temp	{"temperature":19.6}
DEL on	GET	/allume	{"DEL" :"on"}
DEL off	GET	/ferme	{"DEL" :"off"}
Bonjour	POST	/bonjour?nom=bob	{"nom" :"bob"}

Chaque service Web retourne exactement le même message vers le client, et la charge utile est chaque fois exactement du même poids.

Tableau 3 Poids requête/réponse en ko

	Requête	Réponse
Bidon	82 ko	14 ko
Température	86 ko	12 ko
DEL on	88 ko	13 ko
DEL off	87 ko	20 ko
Bonjour	93 ko	13 ko

4.3 Description de l’approche

Ce laboratoire informatique permet de procéder à la mesure d’efficacité de chacun des services Web. Ces mesures sont effectuées de trois façons.

En premier lieu, un script programmé avec le langage Python utilisant la bibliothèque « request » est utilisé pour lancer les requêtes au service Web [32]. Ce script mesure le temps d’achèvement de la requête et contrôle son succès. Des requêtes sont envoyées à chacun des services Web pendant 180 secondes, ce qui donne dans chaque cas plus ou moins 180 réponses attendues, soit au moins une réponse par seconde sur 180 secondes. Dans tous les cas, le service a retourné plus de deux réponses par seconde. Ce nombre de réponses est contrôlé par un marqueur de temps récolté au moment de l’envoi de la requête. Ce marqueur de temps permet d’ajuster la récolte des données entre les services Web programmés avec Node.js, Java et Go. À partir d’un script qui sert de lanceur de requêtes, le temps d’exécution et le moment de la demande au service sont conservés dans un fichier pour servir de trace. La réponse retournée par le service est conservée et atteste du succès de la demande. Cette méthodologie est appliquée pour chaque service.

En deuxième lieu, l’état de l’unité centrale de traitement est surveillé par un autre script programmé avec le langage Python en utilisant la bibliothèque « psutil » [33]. Cette bibliothèque permet de récolter plusieurs marqueurs de fonctionnement, comme la moyenne de la charge système, le temps de travail en espace utilisateur et en espace noyau, le temps d’attente en seconde des entrées/sorties, et la quantité de mémoire utilisée par le service Web, au repos et en attente. Chacun de ces marqueurs permet d’évaluer l’état de l’unité centrale de traitement lors de l’activation d’un service Web.

En troisième lieu, la consommation d’énergie de la carte est mesurée, au repos et pendant les requêtes aux différents services Web. Pour ce faire, un capteur de courant et un enregistreur de données sont utilisés. Un petit montage est nécessaire pour bien mesurer l’énergie consommée par la carte. La consommation d’énergie de la carte au repos est validée à partir des spécifications fournies par le fabricant de la carte. L’enregistreur de données permet de marquer le moment de la capture de consommation; cette information est primordiale pour croiser les résultats de mesure d’énergie avec le moment précis des

demandes au service Web. Il est aussi possible de rapporter la consommation d'énergie sur une base commune de piles AA, permettant une comparaison plus facile avec d'autres systèmes.

Afin de comparer les services, la taille du code exécutable est mesurée en octets pour chacun des services, et il en va de même pour la taille de chacun des composants. La taille en octets des composants est importante pour les systèmes embarqués, car l'espace mémoire est souvent restreint; un exécutable plus petit est alors à privilégier. La dimension en ligne de code est utilisée pour quantifier l'effort de développement.

4.4 Résultats obtenus

Les résultats obtenus sont exposés sous forme de tableaux et graphiques. Il est possible de comparer les performances en croisant les données obtenues, par exemple la taille du code et le temps réponse.

Tableau 4 Modèle de tableau de taille en ligne de code

Module	Taille du code (octets)	Empreinte RAM (octets)
Serveur HTTP	3976	72
Moteur REST	692	4

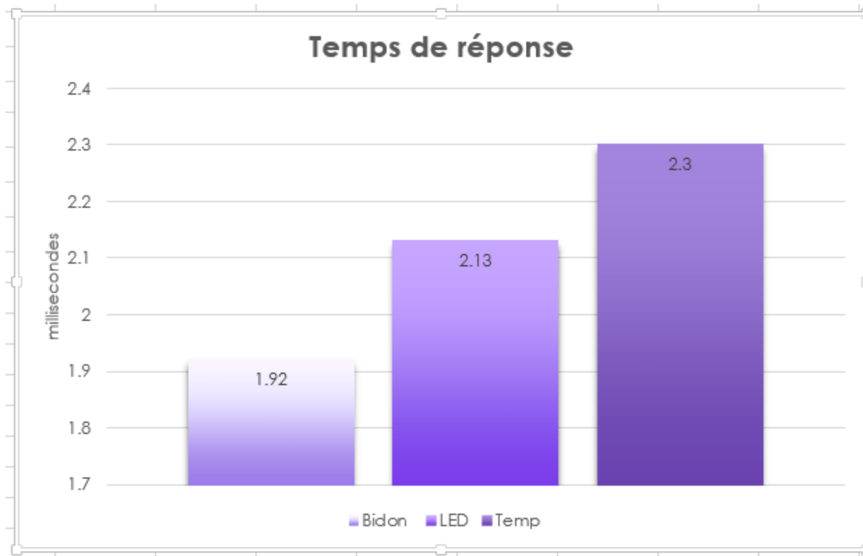


Figure 11 Modèle de graphique temps de réponse

Les résultats attendus offrent différentes possibilités. Dans le cas où les résultats des temps de réponse ne différaient pas de plus d'une ou deux millisecondes, les deux approches sont considérées comme similaires ou égales. Dans la situation où les résultats des temps de réponse diffèrent, le serveur avec le temps de réponse le plus rapide est déterminé comme ayant l'avantage. Dans le cas où les temps de réponse sont similaires de une ou deux millisecondes de différence, l'impact sur la charge de travail du processeur et l'utilisation de la mémoire est considéré comme identique.

De plus, il est nécessaire d'examiner les résultats en les croisant avec la consommation d'énergie. Encore une fois, des résultats similaires ne permettent pas de rejeter une des approches, mais une différence distincte de consommation d'énergie signifie un avantage.

Le chapitre suivant présente les résultats de l'étude.

Chapitre 5

Analyse des résultats

Ce chapitre présente l’analyse des résultats obtenus pour les trois services Web. Le premier comparatif est la taille du code exécutable selon chaque service; à titre indicatif, le nombre de lignes de code utilisé (avec ou sans tenir compte des bibliothèques) est aussi rapporté pour fournir une idée de l’effort de programmation en écho à l’expérience de Yazar et Dunkel [1]. Le deuxième élément de comparaison est le temps de réponse du service Web à chacune des cinq requêtes possibles. Le troisième élément de comparaison regroupe les mesures de l’unité centrale de traitement. La quatrième mesure est la moyenne de charge de travail du processeur, la sixième l’utilisation de la mémoire, la septième le temps d’attente en entrée/sortie, et la huitième la consommation électrique de la carte Raspberry Pi.

5.1 Taille du code exécutable et nombre de lignes de code source

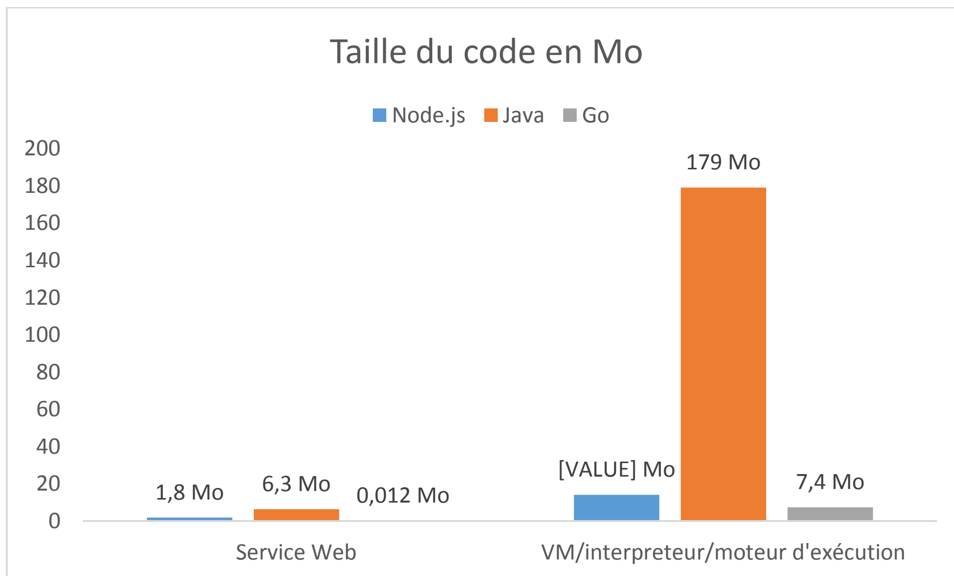


Figure 12 Taille du code exécutable

D’abord, la taille du code exécutable pour les serveurs se divise en deux catégories : le code serveur et le code de la machine virtuelle « hotspot » dans le cas de Java et de l’interpréteur pour Node.js et du moteur d’exécution pour Go. La première catégorie, le code serveur, est le code exécuté par le serveur pour offrir le service. Cette catégorie se divise en deux parties : le code du serveur lui-même et le code utilisé pour lire les capteurs (le code importé de bibliothèques tierces), et le code de la machine virtuelle ou de l’interpréteur ou moteur d’exécution selon le cas, qui est l’environnement de travail du programme permettant au code serveur d’être exécuté.

Dans le cas du code serveur, le code exécutable Node.js occupe 29 % de l’espace en mémoire que celui occupé par le code exécutable Java. La taille de la machine virtuelle « hotspot » de Java SE 8 est plus de dix fois plus grande pour 179 Mo. Également, le code des bibliothèques importées pour faire la lecture des capteurs est beaucoup plus volumineux en Java, même s’il s’agit à la base d’un emballage de la même bibliothèque pour les trois services. À titre indicatif, bien que ce ne soit pas une mesure de l’efficacité du service Web, mais plutôt un indicatif de l’effort de programmation requis, le nombre de lignes de code utilisé pour le code serveur est doublé en Java, 66 lignes en Node.js pour 142 lignes en Java. Voir Tableau 5 Taille en lignes de code.

Tableau 5 Taille en lignes de code

	Node.js		**Java**		**Go**	
Lignes de code	serveur	import	serveur	import	serveur	Import
	66	249	142	1386	79	339
Total	315		1528		418	

Donc le code utilisé pour faire un service Web en Java est plus volumineux en ligne de code et en taille qu’en Node.js. Cela est principalement dû aux bibliothèques importées, car le nombre de lignes de code est sensiblement le même pour le serveur lui-même. Ainsi la plus

grande différence repose sur les bibliothèques utilisées pour faire la gestion des capteurs et sur la volumineuse machine virtuelle de Java.

Le contraste de taille avec le serveur Web en Go est marqué. La taille du code serveur est beaucoup plus petite, seulement 12 kilo-octets pour le code serveur, incluant la bibliothèque pour faire la lecture des capteurs. La taille du moteur d'exécution Go est de 7,4 Mo, 50 % plus petite que celui de Node.js, et près de 4 % de la taille de la machine virtuelle de Java.

Dans l'ensemble, le code en Go est plus petit. Le code exécutable en Java est le plus volumineux et le code en langage JavaScript de Node.js se retrouve au milieu. Finalement, la comparaison est difficile, car la machine virtuelle en Java est très volumineuse et difficilement comparable à l'interpréteur de Node.js ou moteur d'exécution de Go. De plus, plutôt que d'utiliser le moteur d'exécution, le code en Go pourrait être compilé pour gagner un peu plus d'espace sur le disque. Alors en définitive, en ce qui concerne la taille du code exécuté côté serveur, le langage Go est avantagé.

5.2 Temps de réponse aux requêtes

Le temps de réponse est la différence entre le temps écoulé (en seconde) entre l'envoi de la requête par le client et l'arrivée de la réponse à celui-ci. Cela mesure précisément le temps écoulé entre l'émission du premier octet de la requête et la fin de l'analyse des entêtes de la réponse. Ce temps est affecté par la consommation du contenu de la réponse, et c'est pourquoi chaque contenu de réponse est précisément le même pour chacun des services Web. Du côté serveur, lors de la création de la réponse, l'impact des bibliothèques qui font les entrées/sorties est négligeable parce que le temps de lecture ou d'écriture des capteurs en local est le même d'une plateforme à l'autre.

Pour chaque requête effectuée au service Web, une réponse avec le bon contenu est reçue. Il n'y a aucune requête manquée, donc aucun code d'erreur reçu. Dans chaque cas le service Web a répondu dans la même proportion de 20 % à chacune des requêtes. Des requêtes sont envoyées pendant 180 secondes pour chacun des services Web, ce qui donne dans chaque cas au moins 180 réponses pour les cinq types de requêtes en continu. La

moyenne du temps de réponse, pour chaque type de requête, est comparée et présentée dans les tableaux Figure 13 Temps de réponse température et Figure 15 Temps de réponse.

La première requête analysée est celle de température. La lecture du capteur de température implique l'ouverture et la lecture d'un fichier par le serveur. Dans des tests faits en amont du service Web, sur la carte embarquée, le temps d'ouverture et de lecture du fichier du capteur de température pour chaque bibliothèque fournissant la température au service suit la même tendance que le temps de réponse par le service Web. Ces opérations occupent plus l'unité centrale de contrôle et peuvent être bloquantes pour le service Web parce qu'elles impliquent des entrées/sorties, elles aussi plus coûteuses en temps. Ceci explique le temps de réponse plus long pour cette requête au service.

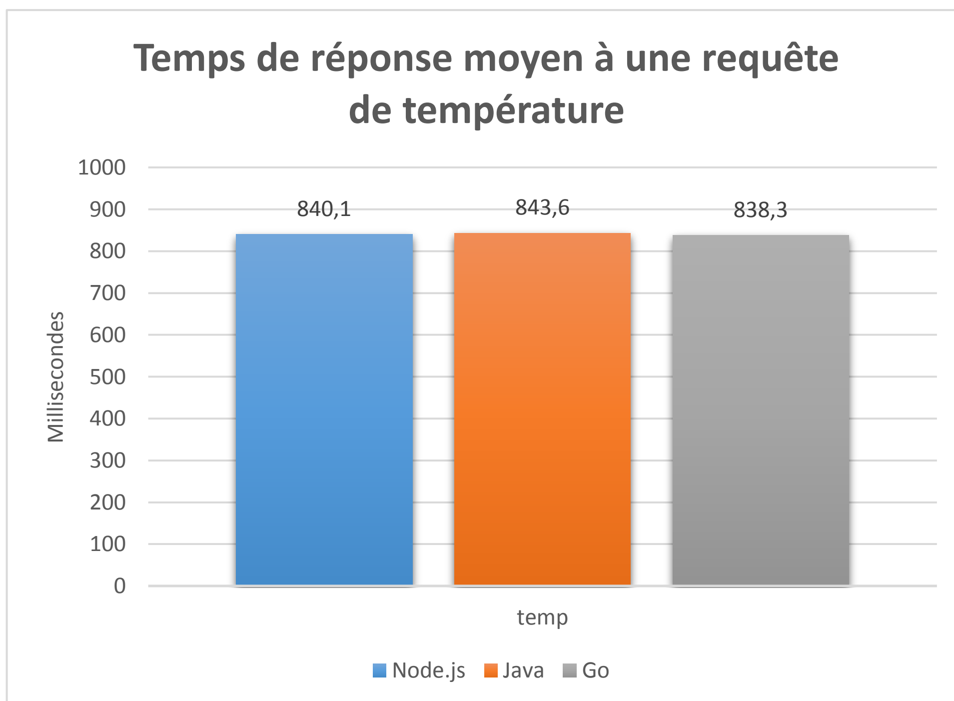
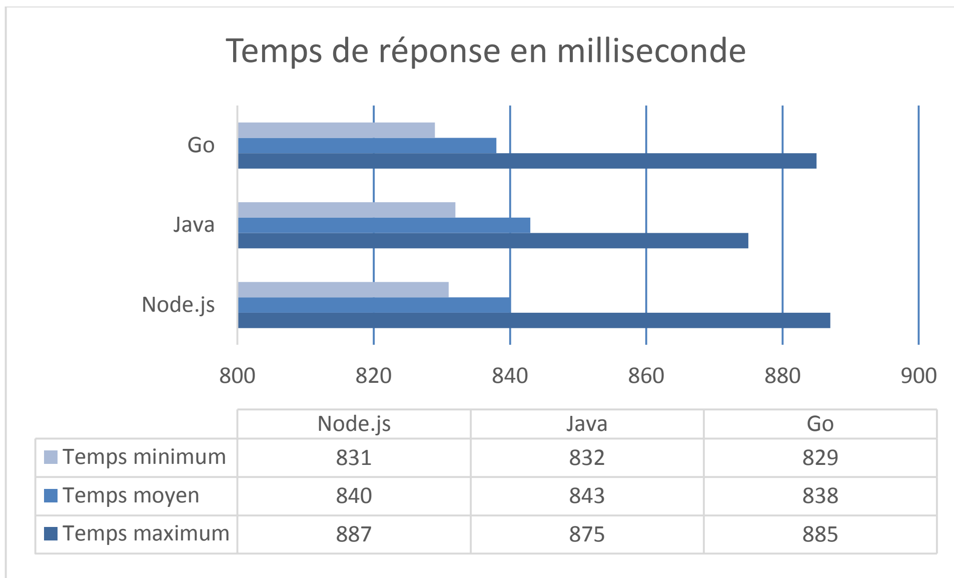


Figure 13 Temps de réponse température

Le temps de réponse moyen entre les différents services est très semblable, il diffère uniquement de cinq millisecondes. Le temps de réponse pour la température diffère de deux millisecondes en faveur de service Web en Go sur celui en Node.js.



	Node.js	Java	Go
Temps minimum	831	832	829
Temps moyen	840	843	838
Temps maximum	887	875	885

Figure 14 Temps de réponse température minimum - moyen - maximum

Les temps de réponse minimum et maximum sont également très proches pour les trois services. En somme, lorsqu'il s'agit de lire la température, malgré une bonne variation entre le meilleur et le pire cas, les trois services se ressemblent beaucoup. Pour les services Node.js et Go, le mode est le même, soit 836 millisecondes. Le service Java n'a pas de mode, c'est-à-dire qu'aucune valeur n'apparait plus qu'une autre; les temps de réponses aux requêtes de température ne sont donc pas constantes pour le service en Java. Les autres requêtes sont beaucoup plus rapides et beaucoup plus significatives.

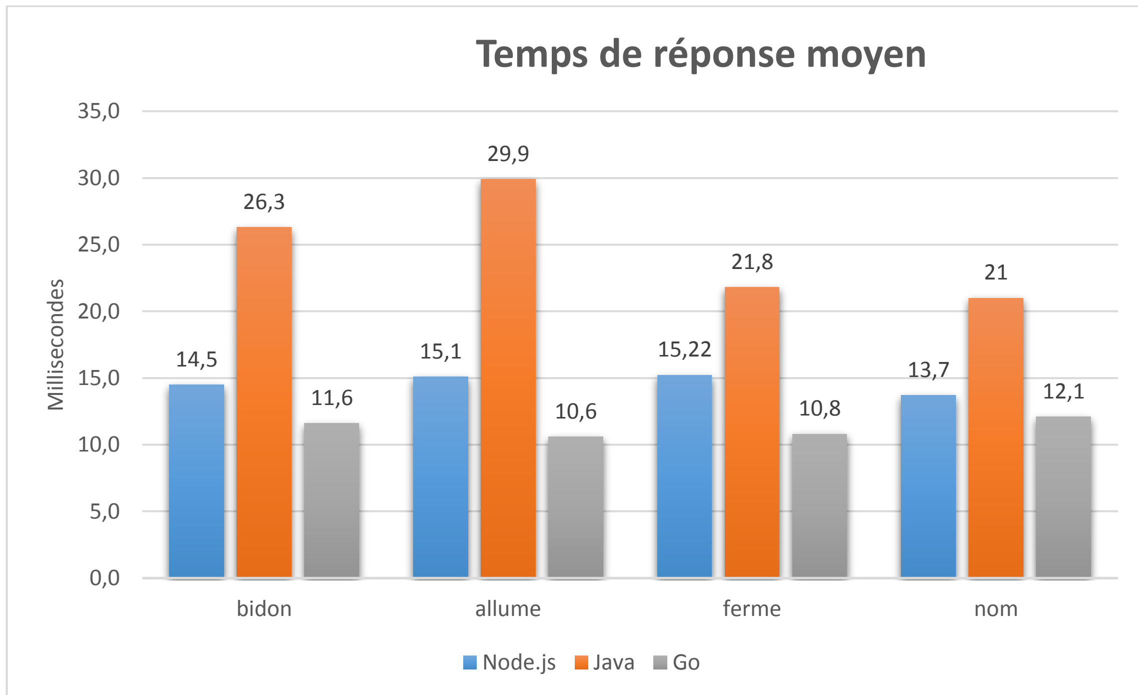


Figure 15 Temps de réponse moyen aux requêtes

Une simple requête au service « bidon », qui retourne une réponse vide, montre un écart de plus de dix millisecondes entre les services Web programmés avec Node.js et celui avec Java. La requête au service vide est la première de la série de requêtes ; comme le temps de réponse aux requêtes est une moyenne, celle-ci souffre d'un temps de réponse plus long lors de la première requête. Le temps de réponse est plus long à la première requête parce que le programme n'est pas encore complètement initialisé. Le temps de réponse à la première requête au service « bidon » en Java est de 192 millisecondes. En ne tenant pas compte de la première requête pour chaque plateforme logicielle, le temps de réponse à une requête au service « bidon » est de 12,2 millisecondes pour Node.js et 16 millisecondes pour Java. La moyenne pour le service en Go ne change pas. Pour chaque type de requête, Java affiche un retard moyen de presque dix millisecondes sur Node.js. Le service Web en Go fait chaque fois un peu mieux que Node.js avec des temps de réponse très stables dans l'ensemble. Pour Java il y a un écart de 8,1 ms entre le temps de réponse pour la requête pour allumer la DEL et celui de la requête pour la fermer. Cette requête utilise presque le même code pour ouvrir ou fermer le GPIO qui contrôle la DEL. Un si grand écart est difficile à comprendre,

puisque si la DEL est actionnée à partir de la carte elle-même et non par le service Web, le temps requis pour ouvrir et fermer le GPIO qui contrôle la DEL est pratiquement le même pour Java et Node.js. En somme, Java est surclassé et Node.js fait très bien, mais le service programmé avec le langage Go est dans l'ensemble plus rapide.

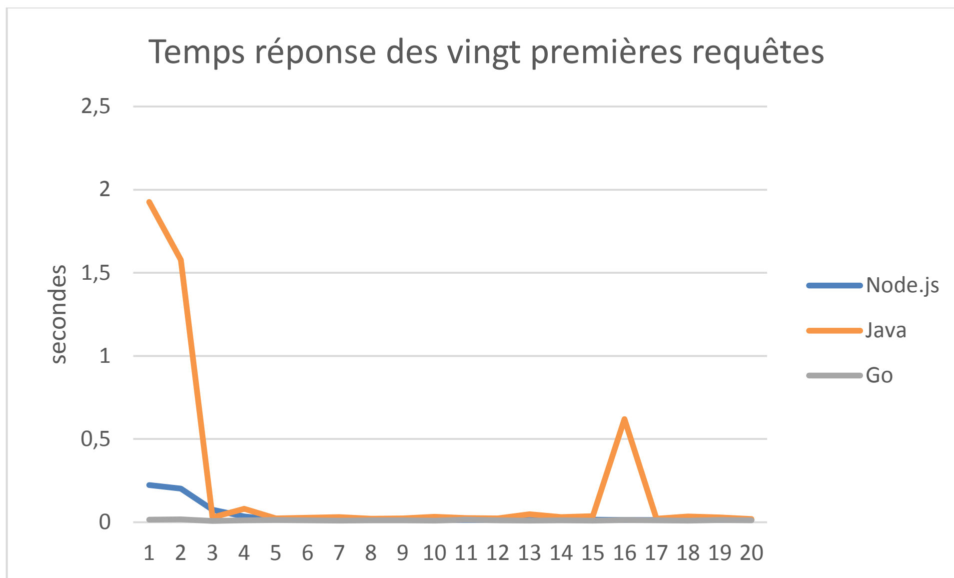


Figure 16 Temps de réponse des vingt premières requêtes

La figure 16 présente le temps de réponse cumulé de vingt premières requêtes, sans tenir compte des requêtes de type température qui ralentissent considérablement le service. Le service Web en Java prend beaucoup plus de temps à démarrer; les premières requêtes sont considérablement plus longues, et presque deux secondes s'écoulent avant d'avoir des résultats comparables à ceux des autres services. Un deuxième ralentissement est aussi constaté après une quinzaine de requêtes, peut-être dû au passage du ramasse-miettes.

5.3 Utilisation de l'unité centrale de traitement

L'unité centrale de traitement compose différemment avec les différents services Web. Il est intéressant d'examiner le comportement du processeur pendant l'utilisation du service Web,

mais aussi sans service, lorsque le processeur ne reçoit aucune requête, et de comparer avec l'utilisation du processeur à sec, complètement au repos.

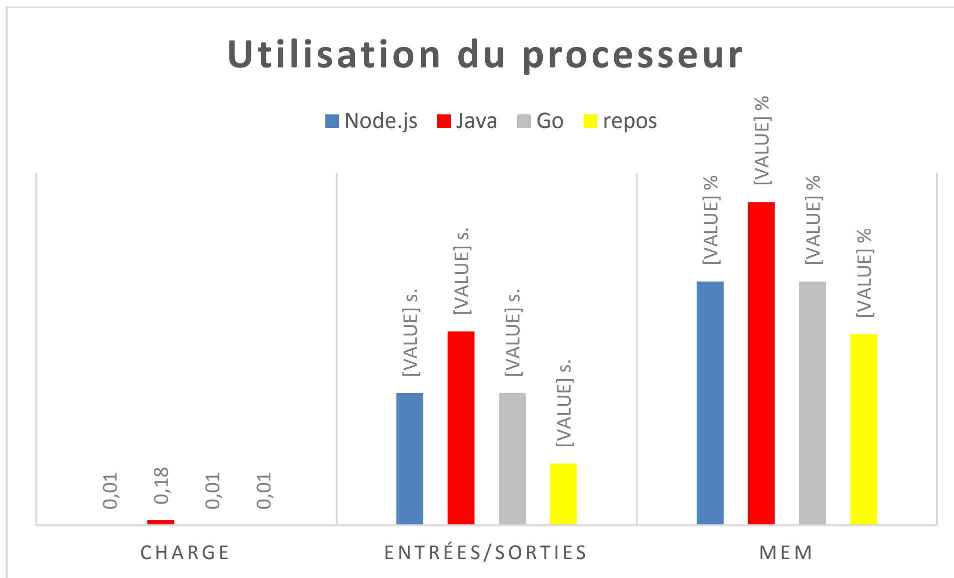


Figure 17 Utilisation du processeur

Le niveau de charge du processeur est au plus bas lorsque le système est au repos complet, alors que le processeur ne fait rien. Lorsque le processeur se met au travail, le niveau de charge monte. La moyenne de la charge est pareille pour les services Web programmés avec Node.js et Go : lorsqu'ils sont en attente de requêtes, ils ne sollicitent pas le processeur, et le système ne fait donc rien, il y a à peine une charge de 0,01 processus. Il est donc possible de déduire que le processeur ne travaille pas lorsque les services Web programmés avec Node.js et Go sont en attente de requêtes. Par contre, les données montrent une charge de 0,18 processus pour le service Web programmé avec Java lorsqu'il est en attente de requêtes, ce qui mène à conclure que le processeur est à la tâche dans ce cas.

L'unité centrale de traitement passe du temps en attente des activités d'entrée/sortie. Le processeur au repos sans travail attend en moyenne 2,1 secondes. Ce temps en attente est calculé globalement pour le système. Pour cette mesure, un langage ayant une meilleure gestion des entrées/sorties est favorisé, mais comme cette mesure n'est pas limitée au programme en cours, mais à l'ensemble de ce qui passe dans le processeur, il est difficile de

relier l'attente en entrée/sortie à une cause exacte. Par contre, lorsque les services Web programmés avec Node.js et avec Go sont en attente de requêtes, l'attente moyenne en entrée/sortie passe à 4,5 secondes et à 6,6 secondes pour le service Web programmé en Java.

L'indicateur d'utilisation de la mémoire est représenté par un pourcentage de la mémoire utilisée. Lorsque le système tourne à vide, il utilise 6,5 % de la mémoire disponible. Ainsi, le service Web en JavaScript sur Node.js et celui en Go, lorsqu'ils sont en attente de requêtes, utilisent tous deux 8,3 % de la mémoire disponible. Java est le plus grand consommateur, utilisant 11 % de la mémoire disponible au repos.

5.3.1 Quantité de travail de l'unité centrale de traitement

La mesure de charge représente l'état du système par la quantité de travail que fait l'unité centrale de traitement pendant la durée de l'expérience. Cette charge, calculée sur une minute, représente le nombre moyen des processus en utilisation, en attente du processeur ou bloqués. Une charge de 1 signifie qu'il y'a à tout moment un processus en travail. Une charge plus petite que 1 signifie que l'unité centrale de traitement n'est pas occupée totalement. Une charge plus grande que 1 indique un engorgement. Lorsque les services Web fournissent des réponses aux requêtes, la charge du processeur augmente.

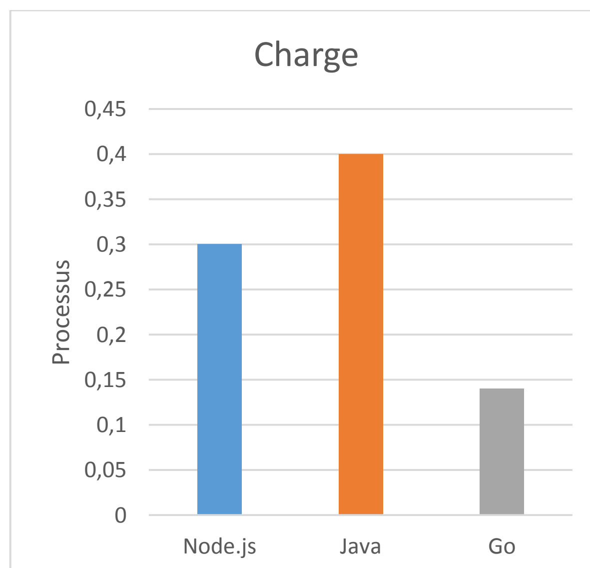


Figure 18 Charge du processeur

Dans tous les cas, la charge est plus petite que 1, et cela indique qu'il n'y a pas assez de processus pour occuper entièrement le processeur. Bien qu'il n'y ait pas de danger de surutilisation du processeur, le service Web en langage Go met 50 % moins de charges sur le système que celui en Node.js, tandis que le serveur en Java met 75 % plus de charges que celui en Node.js. Le service Web programmé en langage Go serait donc un avantage pour un système qui voudrait utiliser le processeur à d'autres tâches.

5.3.2 Pourcentage d'utilisation de la mémoire vive

L'utilisation de la mémoire est la proportion de la mémoire vive utilisée par le processeur pendant l'utilisation du service Web, exprimée en pourcentage. Cette valeur est obtenue en soustrayant la mémoire totale de la mémoire disponible. La mémoire disponible est celle qui pourrait être donnée instantanément au processus. On obtient le pourcentage avec la formule :

$$(total - disponible) / total * 100 \qquad (1)$$

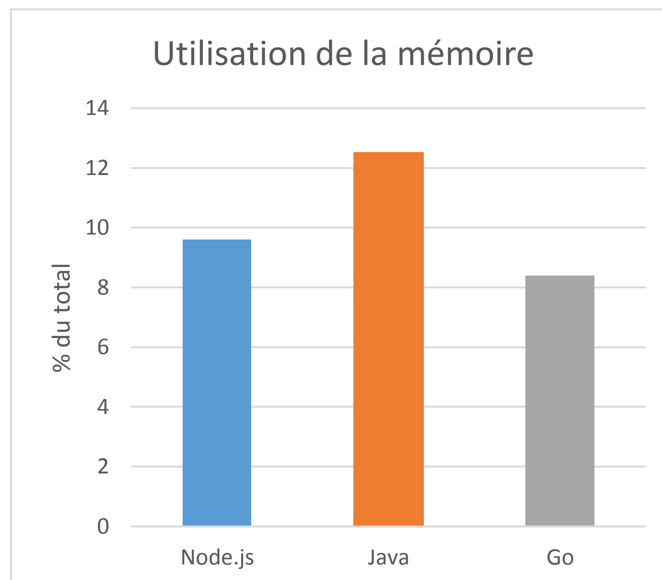


Figure 19 Utilisation de la mémoire

Bien que la gestion de la mémoire soit effectuée de façon différente par chaque langage de programmation, il est intéressant ici de constater que le service Web en Go fait un usage inférieur de la mémoire en comparaison à Node.js et Java, et que Java utilise 148 % de la mémoire utilisé par Go. Pourtant, le temps de réponse est ici inversement proportionnel au pourcentage d'utilisation de la mémoire : le service Web programmé en Go a un meilleur temps de réponse, tout en utilisant moins de mémoire.

5.3.3 Temps d'attente en entrée/sortie

L'attente en entrée/sortie est évaluée en secondes. C'est le temps total d'attente par le processeur qu'une opération d'entrée/sortie se termine. Ce calcul est pris globalement par le système sans évaluer spécifiquement quel processus bloque en attente d'entrées/sorties. Sur les 180 secondes de l'expérience, le processeur passe 2,1 secondes en attente d'entrées/sorties au repos sans requête au service. Lorsque les services Web sont sollicités, le temps d'attente en entrée/sortie augmente selon le type de service Web et cela pour les 475 requêtes en 180 secondes pour le service en Node.js, 935 requêtes en 180 secondes pour le service en Java et 1001 requêtes en 180 secondes pour le service écrit en Go. Le service Web en Node.js retourne 2,6 réponses par seconde pour une attente moyenne de 9,7 secondes en entrée/sortie, tandis que le service en Go retourne 5,7 réponses par seconde pour une attente moyenne de 13,2 secondes en entrée/sortie. Le service en Java, quant à lui, retourne 5,2 réponses pour une attente moyenne de 12,4 secondes.

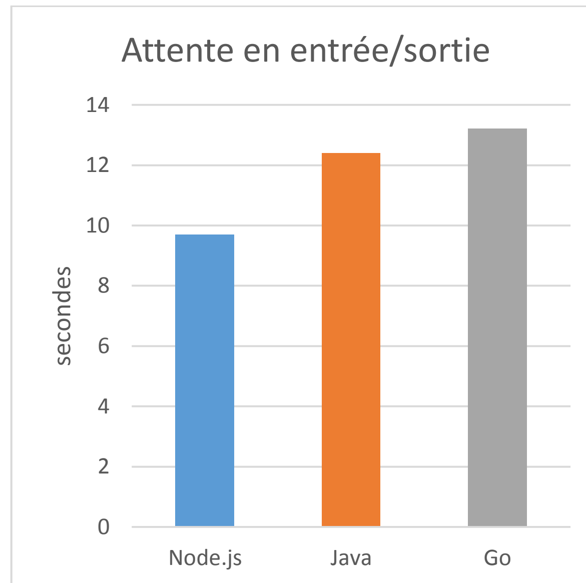


Figure 20 Attente en entrée/sortie

Il faut noter que la programmation événementielle de Node.js est ici plus efficace, au sens où le processeur passe moins de temps en attente d'une opération d'entrée/sortie pour passer à la prochaine : sur les 180 secondes d'expérimentation, il passe presque 9,7 secondes en entrée/sortie. Le service Web Go passe plus de temps que Java en attente d'entrée/sortie, soit 13,2 secondes contrairement à 12,4 secondes pour Java. Le calcul des attentes entrées/sorties se faisant par le processeur, il est impossible de déterminer exactement quelles requêtes ou opérations causent ce temps d'attente. En définitive, cette métrique avantage le service Web programmé avec Node.js malgré qu'il fournisse moins de réponses sur le même laps de temps.

5.4 Consommation électrique

La consommation électrique du système est calculée selon une table de référence pour passer de la tension de l'alimentation, mesurée en volt, à la consommation mesurée en milliampère. Cette table est construite en contrôlant la consommation d'un appareil branché

sur le circuit d'alimentation à la place de la carte électronique. Pour plus de détails, voir la section 4.2.

Selon les résultats obtenus, au repos, la carte consomme 362 mA en moyenne sur une période de temps de 180 s, ce qui est très près des spécifications du fabricant qui indique une moyenne de 360 mA.

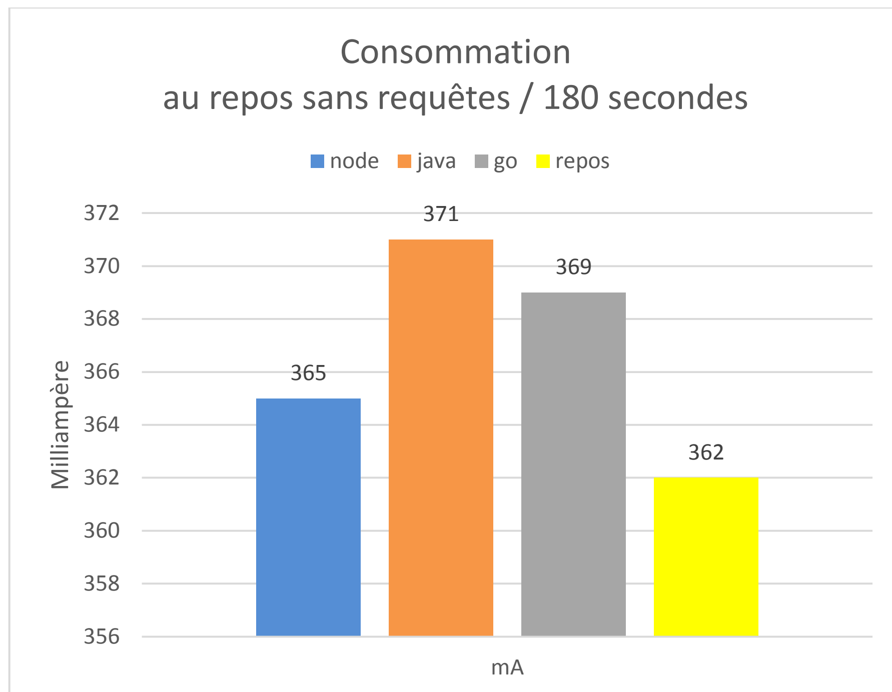


Figure 21 Consommation au repos

Au repos, sans requête, lorsqu'un serveur Web est en attente de demandes, il augmente la consommation énergétique de la carte électronique. Dans cette situation, le service Web Java est le plus énergivore, et le service Web programmé avec Node.js est le moins grand consommateur d'énergie. Le service Web programmé avec le langage Go consomme presque autant que celui en Java.

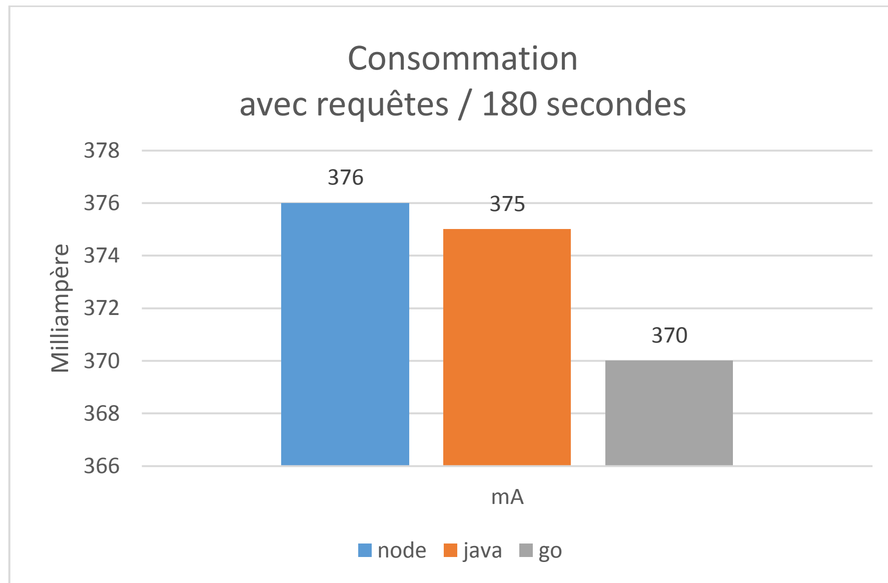


Figure 22 Consommation en utilisation

La consommation électrique du service Web programmé avec Node.js est la plus grande lorsqu'il est en activité. D'une différence à peine perceptible avec le service Web Java. On peut dire que lors de requêtes, les services en Java et en Node.js consomment presque la même énergie.

Le service Web en Go est le moins grand consommateur d'énergie lorsqu'il répond aux demandes. Aussi, il a la consommation d'énergie la plus stable : entre les moments d'attente de demande et les moments de réponse aux requêtes, la consommation est pratiquement la même au repos qu'au travail, par opposition au service Web en Node.js dont la consommation fluctue considérablement entre ces deux situations.

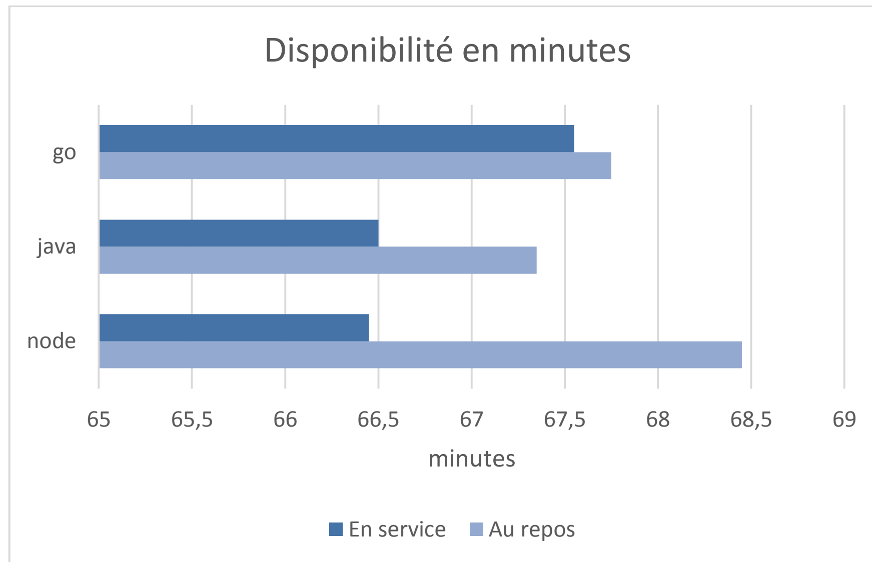


Figure 23 Disponibilité

Ainsi, alimenter le système avec deux piles AA de 2500 mAh chacune permet d'obtenir une disponibilité d'opération plus longue avec un service Web programmé avec le langage Go. Le service en Node.js sera plus longtemps disponible en attente parce qu'il consomme moins d'énergie au repos. Cette disponibilité dépend donc du modèle l'utilisation du service et peu varier selon la situation. Le service programmé en Go obtient l'avantage parce qu'il a la meilleure consommation d'énergie au travail et la plus grande disponibilité. C'est pour le temps passé au travail que les services sont comparés, et non pour le temps en attente.

5.5 Analyse comparative des résultats

Chaque service Web n'est différent que par le code serveur et les bibliothèques importées ainsi que leurs machines virtuelles pour Java et moteur d'exécution pour Node.js. Chaque service Web, un en Node.js, un en Java et un autre en Go, est exécuté dans le même réseau sur la même carte Raspberry PI en mesurant sa consommation électrique avec le même circuit. Les mêmes scripts de mesure sont utilisés pour accumuler les traces d'exécution dans chaque cas, et le même script de demande de requêtes reçoit les mêmes réponses de chaque service. Chaque service Web retourne exactement le même message vers le client.

Dans ce cadre, il est possible de répondre oui à la question posée dès l'introduction: « dans un contexte où les ressources sont limitées est-il possible d'obtenir d'un service Web une meilleure efficacité selon la plateforme logicielle utilisée? », puisque les résultats obtenus pour chaque élément de comparaison donnent le service Web en langage Go comme le plus efficace. Pour les mêmes raisons, l'on peut également répondre oui à la question : est-ce que le langage utilisé influence l'efficacité du service Web?

Pour les sept éléments de comparaison, le service en langage Go a obtenu l'avantage dans cinq cas et une égalité dans l'un des deux autres. Le service Web sur la plateforme Node.js a obtenu un avantage et une égalité. Pour sa part, le service en Java n'a obtenu en aucun cas l'avantage sur les deux autres.

Tableau 6 Résumé comparatif

	Node.js	Java	Go
Taille du code			X
Temps de réponse			X
Utilisation de l'unité centrale de traitement	=		=
Charge de travail			X
Utilisation de la mémoire			X
Attente en entrée/sortie	X		
Consommation électrique			X

5.6 Retour sur les hypothèses

Les résultats obtenus entre le serveur programmé en Java et celui avec la plateforme Node.js ne sont pas similaires, et la comparaison en temps de réponse est révélatrice. En moyenne, sur chaque différente requête au service, le service Web en Node.js est plus rapide de 9.75 ms. Cet écart est révélateur et lui donne un avantage sur Java. Pour sa part, le service en langage Go est plus rapide que Node.js dans chaque situation.

Pour ce qui est de la taille du code utilisé, la taille des binaires de la machine virtuelle qui supporte l'environnement d'exécution du service Web en Node.js est plus petite que celle en Java, qui est considérablement plus volumineuse. Pour le service en langage Go, le code exécutable est beaucoup plus petit.

Pour les indicateurs de l'unité centrale de traitement, rien n'est révélateur d'une part ou de l'autre. L'indicateur de charge est un peu plus haut avec le service Web Java, mais cela reste dans un seuil tout à fait acceptable, parce que le système n'est jamais sous une trop grande quantité de travail. En comparant la charge du système au repos avec la charge du système lors de l'utilisation des services Web, il n'y a pas de hausse de charge révélatrice d'une congestion des processus dus à l'utilisation des services Web. Par contre, l'utilisation de la mémoire par Java est un peu plus grande.

En définitive, l'approche de programmation événementielle de Node.js montre une plus grande efficacité que Java dans un système aux ressources limitées, confirmant la première hypothèse.

Pour les résultats obtenus avec la seconde hypothèse, soit avec le service Web codé en utilisant le langage de programmation Go, les résultats sont différents de ceux avec JavaScript sur Node.js. Tout d'abord, en ce qui concerne la taille du code, le code du serveur est plus succinct, autant pour les modules à importer que pour le code à écrire. Le moteur d'exécution du code est 50 % plus petit que l'interpréteur en Node.js.

Ensuite, les temps de réponse sont meilleurs : le service Web en Go est plus rapide sur chaque requête, au dixième de seconde dans certains cas. Pour les indicateurs de l'unité centrale de traitement, l'indicateur de charge est plus bas que pour les autres services Web;

l'utilisation de la mémoire est également plus basse, et il y a nettement moins d'attente en lecture/écriture. La consommation en électricité est plus stable que les autres services.

En conséquence, la seconde hypothèse est également confirmée. La plateforme logicielle utilisée pour programmer un service Web influence son efficacité. Donc la plateforme logicielle utilisée agit de manière importante sur l'efficacité d'un service Web.

Conclusion

L'IdO est un système de systèmes formé de logiciels divers. L'architecture de ces multiples systèmes est variée. Un intergiciel agissant entre les capteurs de données et les utilisateurs du service est un logiciel stratégique de cet ensemble de systèmes. En situation où les ressources sont limitées, cet intergiciel doit être conçu pour utiliser les ressources de la meilleure façon possible. Dans ce cas, l'efficacité d'un service Web doit être mise en évidence. Car dans un environnement faible en ressources, la consommation en électricité est déterminante.

Un service Web peut être conçu selon différentes approches avec différentes techniques, SOAP, REST ou autres. Il peut retourner des données en divers formats, le plus souvent XML ou JSON. Selon les études, l'efficacité d'un service Web SOAP n'est pas établie dans un environnement faible en ressources. Selon Yazar et Dunkels, l'utilisation d'un service de style REST est plus simple et rapide [1]. Il faut alors considérer la plateforme logicielle utilisée pour comprendre si elle va influencer l'efficacité du service Web.

Ainsi, le but de cet essai est de démontrer quelle plateforme logicielle permet d'obtenir des services Web plus efficaces dans un environnement où les ressources sont limitées, à partir de trois services Web écrits dans des langages de programmation différents, soit Java, JavaScript sur Node.js et Go. Chaque service Web récupère des données d'un capteur de température et d'une DEL à la demande d'un client pendant une période de temps de 180 secondes.

Après avoir pris des mesures de taille du code, de temps de réponse, de l'utilisation de l'unité centrale de traitement, de consommation électrique et à la lumière des résultats obtenus, la compilation des résultats permet d'établir que la programmation d'un service Web en Java est plus coûteuse qu'avec les autres plateformes examinées, que ce soit en termes de ressources pour le processeur ou de consommation d'énergie, tout en offrant des temps de réponse plus longs. Le service Web programmé avec la plateforme JavaScript Node.js obtient de meilleurs résultats que celui programmé en Java selon toutes les métriques

retenues, sauf pour la disponibilité en service du système. Les résultats du service Web programmé avec la plateforme logicielle Go sont aussi meilleurs que ceux obtenus avec Node.js, et par conséquent meilleurs que ceux obtenus par Java.

Seuls la consommation d'énergie et le temps de réponse distinguent fortement les services JavaScript sur Node.js et Go. Pour donner l'avantage à l'un ou l'autre des services, il faudrait considérer de réels besoins d'affaires, et évaluer dans quel milieu ils évoluent. À titre d'exemple, si le temps de réponse est un besoin plus important cela favorisera l'utilisation du langage Go, mais si une faible consommation électrique est nécessaire et que le système n'est pas souvent sollicité cela favorisera l'utilisation de Node.js. En définitive, pour un service Web dans un environnement où les ressources sont limitées, l'approche de programmation événementielle de la plateforme Node.js et l'approche concurrentielle de la plateforme Go sont plus efficaces que Java.

Cependant l'approche plus généraliste de Java peut permettre de programmer un serveur Java traitant les requêtes de manières asynchrones, mais cela n'a pas été abordé dans cet essai. Encore, il faudrait considérer les différentes machines virtuelles Java offertes. En effet, une machine virtuelle spécialisée pour un environnement faible en ressources pourrait bien obtenir de meilleurs résultats. Par contre, l'idée ici n'était pas d'explorer toutes les façons possibles d'optimiser le logiciel afin d'avoir les meilleurs résultats possibles.

Aussi, il serait pertinent d'explorer les résultats obtenus lors de montée en charge du système. Ces résultats permettraient d'orienter davantage le choix d'un langage de programmation. Bien entendu, il faudrait orienter ce choix d'abord selon les requis du système.

En somme, dans le cadre de l'IdO, il est indispensable de faire une meilleure utilisation des ressources fournies par l'environnement, et pour ce faire, il est essentiel de considérer la plateforme logicielle.

Liste des références

[1] D. Yazar and A. Dunkels, "Proceedings of the first ACM workshop on embedded sensing systems for energy-efficiency in buildings - BuildSys '09; efficient application integration in IP-based sensor networks," in 2009, pp. 43.

[2] N. B. Priyantha, A. Kansal, M. Goraczko and F. Zhao, "Proceedings of the 6th ACM conference on embedded network sensor systems - SenSys '08; tiny web services," in 2008, pp. 253.

[3] (). *Web3.0*. Available: https://fr.wikipedia.org/wiki/Web_3.0.

[4] (). *The Internet of Things Will Demand New Application Architectures, Skills and Tools*.

[5] Z. Song, A. A. Cardenas and R. Masuoka, "2010 internet of things (IOT); semantic middleware for the internet of things," in 2010, pp. 1 8.

[6] (). *Service web*. Available: https://fr.wikipedia.org/wiki/Service_web.

[7] (). *Representational State Transfer (REST)*. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[8] (). *SOAP Specifications*. Available: https://www.w3.org/TR/soap12/.

[9] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Comput.,* vol. 14, pp. 80 83, 2010.

[10] (). *Raspberry Pi*. Available: https://fr.wikipedia.org/wiki/Raspberry_Pi.

[11] (). *Informatique ubiquitaire*. Available: https://fr.wikipedia.org/wiki/Informatique_ubiquitaire.

[12] P. Benghozi, S. Bureau and F. Massit-Folléa, *L'Internet Des Objets : Quels Enjeux Pour L'Europe?* Paris: Éditions de la Maison des sciences de l'homme., 2009.

[13] A. Cannata, M. Gerosa and M. Taisch, "2008 IEEE international conference on industrial engineering and engineering management; SOCRADES: A framework for developing intelligent systems in manufacturing," in 2008, pp. 1904 1908.

[14] A. Cannata, S. Karnouskos and M. Taisch, "2010 8th IEEE international conference on industrial informatics; evaluating the potential of a service oriented infrastructure for the factory of the future," in 2010, pp. 592 597.

[15] S. Bae, D. Kim, M. Ha and S. H. Kim, "2011 IEEE 17th international conference on parallel and distributed systems; browsing architecture with presentation metadata for the internet of things," in 2011, pp. 721 728.

[16] A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby and M. Zorzi, "2010 8th IEEE international conference on pervasive computing and communications workshops (PERCOM workshops); architecture and protocols for the internet of things: A case study," in 2010, pp. 678 683.

[17] G. C. Fox, S. Kamburugamuve and R. D. Hartman, "2012 international conference on collaboration technologies and systems (CTS); architecture and measured characteristics of a cloud based internet of things," in 2012, pp. 6 12.

[18] Y. Wen, Z. Li, X. Peng and H. Zhao, "2011 second international conference on digital manufacturing & automation; A middleware architecture for sensor networks applied to industry solutions of internet of things," in 2011, pp. 50 54.

[19] D. Guinard, V. Trifa, T. Pham and O. Liechti, "2009 sixth international conference on networked sensing systems (INSS); towards physical mashups in the web of things," in 2009, pp. 1 4.

[20] M. Kovatsch, M. Lanter and S. Duquennoy, "2012 3rd IEEE international conference on the internet of things; actinium: A RESTful runtime container for scriptable internet of things applications," in 2012, pp. 135 142.

[21] M. A. Chaqfeh and N. Mohamed, "2012 international conference on collaboration technologies and systems (CTS); challenges in middleware solutions for the internet of things," in 2012, pp. 21 26.

[22] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti and Subhajit Dutta, "Role Of Middleware For Internet Of Things: A Study," *International Journal of Computer Science & Engineering Survey,* vol. 2, pp. 94 105, 2011.

[23] J. Vasseur, *Interconnecting Smart Objects with IP : The Next Internet.* Morgan Kaufmann Publishers/Elsevier, 2010.

[24] N. Gligoric, I. Dejanovic and S. Krco, "2011 international conference on distributed computing in sensor systems and workshops (DCOSS); performance evaluation of compact binary XML representation for constrained devices," in 2011, pp. 1 5.

[25] M. Thoma, T. Kakantousis and T. Braun, "2014 11th annual conference on wireless on-demand network systems and services (WONS); rest-based sensor networks with OData," in 2014, pp. 33 40.

[26] B. Lin, Y. Chen, X. Chen and Y. Yu, "2012 international conference on computer science and service system; comparison between JSON and XML in applications based on AJAX," in 2012, pp. 1174 1177.

[27] (). *Web 2.0*. Available: https://fr.wikipedia.org/wiki/Web_2.0.

[28] (). *Extensible Markup Language (XML)*.

[29] M. F. Karagoz and C. Turgut, "IEEE Xplore Abstract - Design and Implementation of RESTful Wireless Sensor Network Gateways Using Node.js Framework," 2014.

[30] F. Ocariza, K. Bajaj, K. Pattabiraman and A. Mesbah, "2013 ACM / IEEE international symposium on empirical software engineering and measurement; an empirical study of client-side JavaScript bugs," in 2013, pp. 55 64.

[31] J. Meyerson, "The Go Programming Language," *IEEE Software,* vol. 31, pp. 104 104, 2014.

[32] (). *Requests: HTTP for Humans — Requests 2.10.0 documentation*. Available: http://docs.python-requests.org/.

[33] (). *psutil 4.3.0 : Python Package Index*. Available: https://pypi.python.org/pypi/psutil.

Bibliographie

P. Anantharam, P. Barnaghi and A. Sheth, "Proceedings of the 3rd international conference on web intelligence, mining and semantics - WIMS '13; data processing and semantics for advanced internet of things (IoT) applications " in 2013, pp. 1.

S. Bae, D. Kim, M. Ha and S. H. Kim, "2011 IEEE 17th international conference on parallel and distributed systems; browsing architecture with presentation metadata for the internet of things " in 2011, pp. 721 <last_page> 728.

P. Benghozi, S. Bureau and F. Massit-Folle´a, L'Internet Des Objets : Quels Enjeux Pour l'Europe? Paris: E´ditions de la Maison des sciences de l'homme., 2009.

A. Cannata, M. Gerosa and M. Taisch, "2008 IEEE international conference on industrial engineering and engineering management; SOCRADES: A framework for developing intelligent systems in manufacturing " in 2008, pp. 1904 <last_page> 1908.

A. Cannata, S. Karnouskos and M. Taisch, "2010 8th IEEE international conference on industrial informatics; evaluating the potential of a service oriented infrastructure for the factory of the future " in 2010, pp. 592 <last_page> 597.

A. P. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby and M. Zorzi, "2010 8th IEEE international conference on pervasive computing and communications workshops (PERCOM workshops); architecture and protocols for the internet of things: A case study " in 2010, pp. 678 <last_page> 683.

M. A. Chaqfeh and N. Mohamed, "2012 international conference on collaboration technologies and systems (CTS); challenges in middleware solutions for the internet of things " in 2012, pp. 21 <last_page> 26.

C. Diedrich, M. Muhlhause, M. Riedl and T. Bangemann, "2008 IEEE international workshop on factory communication systems; mapping of smart field device profiles to web services " in 2008, pp. 375 <last_page> 381.

G. C. Fox, S. Kamburugamuve and R. D. Hartman, "2012 international conference on collaboration technologies and systems (CTS); architecture and measured characteristics of a cloud based internet of things " in 2012, pp. 6 <last_page> 12.

V. Gazis, K. Sasloglou, A. Merentitis and K. Mathioudakis, "2014 28th international conference on advanced information networking and applications workshops; on the role of semantic descriptions for adaptable protocol stacks in the internet of things " in 2014, pp. 437 <last_page> 443.

N. Gligoric, I. Dejanovic and S. Krco, "2011 international conference on distributed computing in sensor systems and workshops (DCOSS); performance evaluation of compact binary XML representation for constrained devices " in 2011, pp. 1 <last_page> 5.

D. Guinard, V. Trifa, T. Pham and O. Liechti, "2009 sixth international conference on networked sensing systems (INSS); towards physical mashups in the web of things " in 2009, pp. 1 <last_page> 4.

S. Hachem, T. Teixeira and V. Issarny, "Proceedings of the 8th middleware doctoral symposium on - MDS '11; ontologies for the internet of things " in 2011, pp. 1 <last_page> 6.

(). The Internet of Things Will Demand New Application Architectures, Skills and Tools .

(). Survey Analysis: The Internet of Things Is a Revolution Waiting to Happen .

A. Kamilaris, V. Trifa and A. Pitsillides, "2011 18th international conference on telecommunications; HomeWeb: An application framework for web-based smart homes " in 2011, pp. 134 <last_page> 139.

M. F. Karagoz and C. Turgut, "IEEE Xplore Abstract - Design and Implementation of RESTful Wireless Sensor Network Gateways Using Node.js Framework " 2014.

O. Kleine, "2013 IEEE 6th international conference on service-oriented computing and applications; integrating the physical world with the internet -- A concept evaluation " in 2013, pp. 323 <last_page> 327.

M. Kovatsch, M. Lanter and S. Duquennoy, "2012 3rd IEEE international conference on the internet of things; actinium: A RESTful runtime container for scriptable internet of things applications " in 2012, pp. 135 <last_page> 142.

B. Lin, Y. Chen, X. Chen and Y. Yu, "2012 international conference on computer science and service system; comparison between JSON and XML in applications based on AJAX " in 2012, pp. 1174 <last_page> 1177.

P. Malo, M. Mateus, B. Almeida and T. Teixeira, "2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing; measuring data transfer in heterogeneous IoT environments " in 2013, pp. 518 <last_page> 526.

J. Meyerson, "The Go Programming Language " IEEE Software, vol. 31, pp. 104 <last_page> 104, 2014.

Z. Ming, F. Hong and M. Yan, "2013 fourth international conference on intelligent control and information processing (ICICIP); semantic annotation method of IOT middleware " in 2013, pp. 495 <last_page> 498.

S. N. A. U. Nambi, C. Sarkar, R. V. Prasad and A. Rahim, "2014 IEEE world forum on internet of things (WF-IoT); A unified semantic knowledge base for IoT " in 2014, pp. 575 <last_page> 580.

F. Ocariza, K. Bajaj, K. Pattabiraman and A. Mesbah, "2013 ACM / IEEE international symposium on empirical software engineering and measurement; an empirical study of client-side JavaScript bugs " in 2013, pp. 55 <last_page> 64.

D. Park, H. Bang, C. S. Pyo and S. Kang, "2014 IEEE world forum on internet of things (WF-IoT); semantic open IoT service platform technology " in 2014, pp. 85 <last_page> 88.

Z. Peng, Z. Jingling and L. Qing, "Proceedings of 2012 2nd international conference on computer science and network technology; message oriented middleware data processing model in internet of things " in 2012, pp. 94 <last_page> 97.

N. B. Priyantha, A. Kansal, M. Goraczko and F. Zhao, "Proceedings of the 6th ACM conference on embedded network sensor systems - SenSys '08; tiny web services " in 2008, pp. 253.

S. Bandyopadhyay, M. Sengupta, S. Maiti and S. Dutta, "Role Of Middleware For Internet Of Things: A Study " International Journal of Computer Science & Engineering Survey, vol. 2, pp. 94 <last_page> 105, 2011.

H. Son, S. Han and D. Lee, "2012 international symposium on ubiquitous virtual reality; contextual information provision on augmented reality with IoT-based semantic communication " in 2012, pp. 46 <last_page> 49.

Z. Song, A. A. Cardenas and R. Masuoka, "2010 internet of things (IOT); semantic middleware for the internet of things " in 2010, pp. 1 <last_page> 8.

M. Thoma, T. Braun, C. Magerkurth and A. Antonescu, "2014 IEEE network operations and management symposium (NOMS); managing things and services with semantics: A survey " in 2014, pp. 1 <last_page> 5.

M. Thoma, T. Kakantousis and T. Braun, "2014 11th annual conference on wireless on-demand network systems and services (WONS); rest-based sensor networks with OData " in 2014, pp. 33 <last_page> 40.

S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs " IEEE Internet Comput., vol. 14, pp. 80 <last_page> 83, 2010.

J. Vasseur, Interconnecting Smart Objects with IP : The Next Internet. Morgan Kaufmann Publishers/Elsevier, 2010.

Y. Wen, Z. Li, X. Peng and H. Zhao, "2011 second international conference on digital manufacturing & automation; A middleware architecture for sensor networks applied to industry solutions of internet of things " in 2011, pp. 50 <last_page> 54.

D. Yazar and A. Dunkels, "Proceedings of the first ACM workshop on embedded sensing systems for energy-efficiency in buildings - BuildSys '09; efficient application integration in IP-based sensor networks " in 2009, pp. 43.().

Extensible Markup Language (XML) Available: https://www.w3.org/XML/.

Requests: HTTP for Humans — Requests 2.10.0 documentation . Available: http://docs.python-requests.org/.

Service web . Available: https://fr.wikipedia.org/wiki/Service_web.

ProgrammationÉvénementielle.Available:https://fr.wikipedia.org/wiki/Programmation_evenementielle.

psutil 4.3.0 : Python Package Index . Available: https://pypi.python.org/pypi/psutil.

RepresentationalStateTransfer.Available:https://fr.wikipedia.org/wiki/Representational_State_Transfer.

SOAP . Available: https://fr.wikipedia.org/wiki/SOAP.

SOAP Specifications . Available: https://www.w3.org/TR/soap12/.

Informatique ubiquitaire . Available: https://fr.wikipedia.org/wiki/Informatique_ubiquitaire.

Raspberry Pi . Available: https://fr.wikipedia.org/wiki/Raspberry_Pi.

Requests: HTTP for Humans — Requests 2.10.0 documentation . Available: http://docs.python-requests.org/.

Web 2.0. Available: https://fr.wikipedia.org/wiki/Web_2.0.

Web 3.0. Available: https://fr.wikipedia.org/wiki/Web_3.0.

WiringPi . Available: http://wiringpi.com/.

Annexe I

Code des services Web

Node.js

```
var http = require(« http »);
var url = require(« url »);
var gpio = require("rpi-gpio");
var tsensor = require(« ds18x20 »);
var qs = require(« querystring »);

gpio.setMode(gpio.MODE_BCM);

var server;
server = http.createServer(function(req, res){
    var path = url.parse(req.url).pathname;
    switch (path){
        case '/' :
            res.writeHead(200, {'Content-Type' : 'text/html'});
            res.write("Allo le monde!");
            res.end();
            break;
        case '/temp' :
            res.writeHead(200, {'Content-Type' : 'application/json'});
            temperature = tsensor.get('28-00000604c8ed');
            res.write(JSON.stringify({'temperature':temperature}));
            res.end();
            break;
        case '/allume' :
            res.writeHead(200, {'Content-Type' : 'application/json'});
            gpio.setup(22, gpio.DIR_OUT, allume);
            res.write(JSON.stringify({'DEL :' :'on'}));
            res.end();
            break;
        case '/ferme' :
            res.writeHead(200, {'Content-Type' : 'application/json'});
            gpio.setup(22, gpio.DIR_OUT, ferme);
            res.write(JSON.stringify({'DEL' :'off'}));
            res.end();
            break;
        case '/bonjour' :
            var url_parts = url.parse(req.url,true)
            var nom = url_parts.query['nom']
```

```
            res.writeHead(200, {'Content-Type' : 'application/json'});
            res.write('{"nom" :"'+nom+'"}');
            res.end();
            break;
        default : send404(res);
    }
}),

send404 = function(res){
    res.writeHead(404);
    res.write('404');
    res.end();
};

function ferme() {
    gpio.write(22, false, function(err) {
        if (err) throw err;
    });
}

function allume() {
    gpio.write(22, true, function(err) {
        if (err) throw err;
    });
}

server.listen(8001);
console.log("Server listening...");
```

Java serveur

```
package tjws;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.Request;
import org.eclipse.jetty.server.handler.AbstractHandler;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;

public class Server extends AbstractHandler {
    public void handle(String target,
                       Request baseRequest,
                       HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html;charset=utf-8");
        response.setStatus(HttpServletResponse.SC_OK);
        baseRequest.setHandled(true);
        response.getWriter().println("Allo le monde!");
    }

    public static void main(String[] args) throws Exception {
        Server server = new Server(8088);
        ServletContextHandler context = new
ServletContextHandler(ServletContextHandler.SESSIONS);
        context.setContextPath("/");
        server.setHandler(context);
        context.addServlet(new ServletHolder(new WsServlet()),"/");
        server.start();
        server.join();
    }
}
```

Java Servlet

```
package tjws;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.util.List;
import java.util.Objects;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

@SuppressWarnings("serial")
public class WsServlet  extends HttpServlet {

    private final String greeting="Allo le monde!";

    public WsServlet(){}

    public WsServlet(String greeting) {
            this.greeting=greeting;
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            response.setCharacterEncoding("UTF-8");
            response.setStatus(HttpServletResponse.SC_OK);
            String serve = request.getRequestURI();

            if(Objects.equals(serve, new String("/"))){
                response.setContentType("text/html");
                response.getWriter().print(greeting);
            }

            if(Objects.equals(serve, new String("/temp"))){
                Double tempD = readTempFromFile();
                String temp = Double.toString(tempD).substring(0,
Double.toString(tempD).length()-2);
                response.setContentType("application/json");
                response.getWriter().print("{\"temperature\":"+temp+"}");
            }
```

```
            if(Objects.equals(serve, new String("/allume"))){
                this.ledOn();
                response.setContentType("application/json");
                response.getWriter().print("{\"DEL\":\"on\"}");
            }

            if(Objects.equals(serve, new String("/ferme"))){
                this.ledOff();
                response.setContentType("application/json");
                response.getWriter().print("{\"DEL\":\"off\"}");
            }

            if(Objects.equals(serve, new String("/bonjour"))){
                String nom = request.getParameter("nom");
                response.setContentType("application/json");
                response.getWriter().print("{\"nom\":\""+nom+"\"}");
            }
    }

    private static double readTempFromFile() {
        int iniPos, endPos;
        String strTemp, strTempIdentifier = "t=";
        double tvalue = 0;
        List<String> lines;
        try {
            lines =
Files.readAllLines(FileSystems.getDefault().getPath("/sys/bus/w1/devices/28
-00000604c8ed/w1_slave"), Charset.defaultCharset());
            for(String line : lines){
                if(line.contains(strTempIdentifier)){
                    iniPos = line.indexOf(strTempIdentifier)+2;
                    endPos = line.length();
                    strTemp = line.substring(iniPos, endPos);
                    tvalue = Double.parseDouble(strTemp)/1000;
                }
            }
        } catch (IOException ex) {
            //logger.error("Error while reading temp sensor", ex);
        }
        return tvalue;
    }

    private void ledOn(){
        final GpioController gpio = GpioFactory.getInstance();
        final GpioPinDigitalOutput pin =
gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "MyLED", PinState.LOW);
        pin.high();
        gpio.shutdown();
        gpio.unprovisionPin(pin);
    }
```

```
    private void ledOff(){
        final GpioController gpio = GpioFactory.getInstance();
        final GpioPinDigitalOutput pind =
gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "MyLED", PinState.LOW);
        pind.low();
        gpio.shutdown();
        gpio.unprovisionPin(pind);
    }
}
```

Go

```go
package main

import (
    "./go-rpio-master"
    "net/http"
    "bufio"
    "log"
    "os"
    "strings"
    "strconv"
)

func handler(w http.ResponseWriter, r *http.Request) {
    q := r.URL.Path[1:]

    if q == "temp" {
        temp := readTempFromFile()
        w.Header().Set("Content-Type", "application/json")
        w.Write([]byte("{\"temperature\":"+temp+"}"))
    }

    if q == "allume" {
        rpio.Open()
        pin := rpio.Pin(22)
        pin.High()
        rpio.Close()
        w.Header().Set("Content-Type", "application/json")
        w.Write([]byte("{\"DEL\":\"on\"}"))
    }

    if q == "ferme" {
        rpio.Open()
        pin := rpio.Pin(22)
        pin.Low()
        rpio.Close()
        w.Header().Set("Content-Type", "application/json")
        w.Write([]byte("{\"DEL\":\"off\"}"))
    }

    if q == "bonjour" {
        nom := r.URL.Query().Get("nom")
        w.Header().Set("Content-Type", "application/json")
        w.Write([]byte("{\"nom\":\""+nom+"\"}"))
    }

    if q == "" {
        w.Header().Set("Content-Type", "text/html")
        w.Write([]byte("Allo le monde!"))
```

```
    }
}

func  readTempFromFile() (temp string) {
  tempId := "t="
  if file, err := os.Open("/sys/bus/w1/devices/28-00000604c8ed/w1_slave");
err == nil {
    defer file.Close()
    scanner := bufio.NewScanner(file)
    for scanner.Scan() {
        if strings.Contains(scanner.Text(), tempId){
           iniPos := strings.Index(scanner.Text(), tempId)+2
           strTemp := scanner.Text()[iniPos:len(scanner.Text())]
           tvalue, _ := strconv.ParseFloat(strTemp, 32)
           tfloatValue := tvalue / 1000
           temp = strconv.FormatFloat(tfloatValue, 'f', 1, 32)
        }
    }
    if err = scanner.Err(); err != nil {
      log.Fatal(err)
    }
  } else {
    log.Fatal(err)
  }

  return
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Annexe 2

Résultats temps de réponse

Temps de réponse service Web en Node.js

2016-01-30 14:09:20.115947	{"nom":"bob"}	0.015252	
2016-01-30 14:09:20.329958	Allo le monde!	0.012274	
2016-01-30 14:09:20.539479	{"DEL:":"on"}	0.013198	
2016-01-30 14:09:20.760039	{"DEL":"off"}	0.013278	
2016-01-30 14:09:20.969645	{"temperature":20.8}	0.831994	

Temps de réponse service Web en Java

2016-01-30 14:22:45.969805	Allo le monde!	1.926713	
2016-01-30 14:22:47.911970	{"DEL":"on"}	1.5767	
2016-01-30 14:22:49.490000	{"DEL":"off"}	0.028505	
2016-01-30 14:22:49.522493	{"temperature":20.6}	0.868777	
2016-01-30 14:22:50.395794	{"nom":"bob"}	0.080051	
2016-01-30 14:22:50.479349	Allo le monde!	0.021603	
2016-01-30 14:22:50.504475	{"DEL":"on"}	0.025712	
2016-01-30 14:22:50.533962	{"DEL":"off"}	0.030899	
2016-01-30 14:22:50.568260	{"temperature":20.}	0.868597	
2016-01-30 14:22:51.440018	{"nom":"bob"}	0.021391	
2016-01-30 14:22:51.464478	Allo le monde!	0.023169	
2016-01-30 14:22:51.491612	{"DEL":"on"}	0.0314	
2016-01-30 14:22:51.525284	{"DEL":"off"}	0.024173	
2016-01-30 14:22:51.551915	{"temperature":20.}	0.843739	
2016-01-30 14:22:52.397949	{"nom":"bob"}	0.021771	
2016-01-30 14:22:52.421003	Allo le monde!	0.047273	
2016-01-30 14:22:52.471373	{"DEL":"on"}	0.029464	
2016-01-30 14:22:52.504134	{"DEL":"off"}	0.035465	
2016-01-30 14:22:52.541757	{"temperature":20.}	0.85278	
2016-01-30 14:22:53.396923	{"nom":"bob"}	0.620457	
2016-01-30 14:22:54.020120	Allo le monde!	0.020159	
2016-01-30 14:22:54.045362	{"DEL":"on"}	0.034417	
2016-01-30 14:22:54.083487	{"DEL":"off"}	0.02807	
2016-01-30 14:22:54.114928	{"temperature":20.8}	0.851199	
2016-01-30 14:22:54.970036	{"nom":"bob"}	0.019205	
2016-01-30 14:22:54.993475	Allo le monde!	0.016558	
2016-01-30 14:22:55.014212	{"DEL":"on"}	0.031558	
2016-01-30 14:22:55.048172	{"DEL":"off"}	0.02386	
2016-01-30 14:22:55.075714	{"temperature":20.8}	0.840368	
2016-01-30 14:22:55.919047	{"nom":"bob"}	0.018005	
2016-01-30 14:22:55.942185	Allo le monde!	0.021864	
2016-01-30 14:22:55.967462	{"DEL":"on"}	0.025802	
2016-01-30 14:22:55.995703	{"DEL":"off"}	0.02661	
2016-01-30 14:22:56.025102	{"temperature":20.8}	0.851869	

Temps de réponse service Web en Go

2016-01-30 14:30:26.477276	{"temperature":20.8}	0.839002	
2016-01-30 14:30:27.318434	{"nom":"bob"}	0.010931	
2016-01-30 14:30:27.333951	Allo le monde!	0.012051	
2016-01-30 14:30:27.349090	{"DEL":"on"}	0.009989	
2016-01-30 14:30:27.361759	{"DEL":"off"}	0.00866	
2016-01-30 14:30:27.373477	{"temperature":20.8}	0.842028	
2016-01-30 14:30:28.217985	{"nom":"bob"}	0.010249	
2016-01-30 14:30:28.232646	Allo le monde!	0.011406	
2016-01-30 14:30:28.247183	{"DEL":"on"}	0.009681	
2016-01-30 14:30:28.259701	{"DEL":"off"}	0.014888	
2016-01-30 14:30:28.276609	{"temperature":20.8}	0.838094	
2016-01-30 14:30:29.116839	{"nom":"bob"}	0.011814	
2016-01-30 14:30:29.132625	Allo le monde!	0.009601	
2016-01-30 14:30:29.145237	{"DEL":"on"}	0.010507	
2016-01-30 14:30:29.157991	{"DEL":"off"}	0.008794	
2016-01-30 14:30:29.168721	{"temperature":20.8}	0.839996	
2016-01-30 14:30:30.011936	{"nom":"bob"}	0.013134	
2016-01-30 14:30:30.028991	Allo le monde!	0.011027	
2016-01-30 14:30:30.044966	{"DEL":"on"}	0.009473	
2016-01-30 14:30:30.057229	{"DEL":"off"}	0.011928	
2016-01-30 14:30:30.070691	{"temperature":20.8}	0.834563	
2016-01-30 14:30:30.907408	{"nom":"bob"}	0.011329	
2016-01-30 14:30:30.923708	Allo le monde!	0.009317	
2016-01-30 14:30:30.937336	{"DEL":"on"}	0.008504	
2016-01-30 14:30:30.947781	{"DEL":"off"}	0.008553	
2016-01-30 14:30:30.959974	{"temperature":20.9}	0.835792	
2016-01-30 14:30:31.797735	{"nom":"bob"}	0.011424	
2016-01-30 14:30:31.813743	Allo le monde!	0.019375	
2016-01-30 14:30:31.837513	{"DEL":"on"}	0.013041	
2016-01-30 14:30:31.854130	{"DEL":"off"}	0.010042	
2016-01-30 14:30:31.866735	{"temperature":20.9}	0.838311	
2016-01-30 14:30:32.709152	{"nom":"bob"}	0.021031	
2016-01-30 14:30:32.733539	Allo le monde!	0.010616	

Annexe 3

Résultats unité centrale de traitement

Charge, attente en entrées/sorties et utilisation de la mémoire pour Node.js

Time	Load_1	Load_5	Load_15	iowait	user	system	Mem %	Mem usage
2016-01-3(	0.08	0.04	0.05	9.72	68.28	48.36	9	146378752
2016-01-3(	0.08	0.04	0.05	9.72	68.38	48.37	9	146403328
2016-01-3(	0.08	0.04	0.05	9.72	68.55	48.72	9	146632704
2016-01-3(	0.08	0.04	0.05	9.72	68.62	48.84	9	146681856
2016-01-3(	0.08	0.04	0.05	9.72	68.68	49.14	9	146759680
2016-01-3(	0.08	0.04	0.05	9.72	68.8	49.34	9.1	146841600
2016-01-3(	0.08	0.04	0.05	9.72	68.91	49.48	9.1	146886656
2016-01-3(	0.08	0.04	0.05	9.72	69.13	49.76	9.1	146903040
2016-01-3(	0.08	0.04	0.05	9.72	69.19	49.8	9.1	146886656
2016-01-3(	0.15	0.06	0.06	9.72	69.42	50.1	9.1	147013632
2016-01-3(	0.15	0.06	0.06	9.72	69.46	50.1	9.1	147013632
2016-01-3(	0.15	0.06	0.06	9.72	69.67	50.43	9.1	147140608
2016-01-3(	0.15	0.06	0.06	9.72	69.73	50.44	9.2	147648512
2016-01-3(	0.15	0.06	0.06	9.72	69.9	50.79	9.3	147775488
2016-01-3(	0.14	0.05	0.05	9.72	69.96	50.85	9.3	147824640
2016-01-3(	0.14	0.05	0.05	9.72	70.09	51.16	9.3	147775488
2016-01-3(	0.14	0.05	0.05	9.72	70.16	51.38	9.3	147865600
2016-01-3(	0.14	0.05	0.05	9.72	70.27	51.54	9.3	147808256
2016-01-3(	0.14	0.05	0.05	9.72	70.42	51.86	9.3	147865600
2016-01-3(	0.13	0.05	0.05	9.72	70.45	51.92	9.3	147808256
2016-01-3(	0.13	0.05	0.05	9.72	70.58	52.3	9.3	147808256
2016-01-3(	0.13	0.05	0.05	9.72	70.59	52.31	9.3	147808256
2016-01-3(	0.13	0.05	0.05	9.72	70.78	52.63	9.3	147808256
2016-01-3(	0.13	0.05	0.05	9.72	70.8	52.63	9.3	147808256
2016-01-3(	0.2	0.07	0.06	9.72	70.95	53	9.3	147841024
2016-01-3(	0.2	0.07	0.06	9.72	70.99	53.02	9.3	147865600
2016-01-3(	0.2	0.07	0.06	9.72	71.16	53.35	9.3	147841024
2016-01-3(	0.2	0.07	0.06	9.72	71.29	53.49	9.3	147992576

Charge, attente en entrées/sorties et utilisation de la mémoire pour Java

Time	Load_1	Load_5	Load_15	iowait	user	system	Mem %	Mem usage
2016-01-3(	0.17	0.11	0.09	10.79	116.54	94.57	9.4	152096768
2016-01-3(	0.17	0.11	0.09	10.79	117.28	94.87	9.3	152985600
2016-01-3(	0.17	0.11	0.09	10.79	118.16	95.03	9.5	156667904
2016-01-3(	0.24	0.12	0.1	10.79	119.15	95.08	10	159207424
2016-01-3(	0.24	0.12	0.1	10.79	120.14	95.13	10.7	162381824
2016-01-3(	0.24	0.12	0.1	10.79	120.88	95.42	10.7	163651584
2016-01-3(	0.24	0.12	0.1	10.79	121.53	95.8	10.8	162889728
2016-01-3(	0.24	0.12	0.1	10.79	122.21	96.15	10.8	162889728
2016-01-3(	0.38	0.16	0.11	10.79	122.94	96.45	10.8	164540416
2016-01-3(	0.38	0.16	0.11	10.8	123.64	96.77	10.9	163278848
2016-01-3(	0.38	0.16	0.11	10.8	124.46	96.98	11.1	164225024
2016-01-3(	0.38	0.16	0.11	10.8	125.2	97.27	11.1	164605952
2016-01-3(	0.38	0.16	0.11	10.8	125.91	97.59	11.2	164986880
2016-01-3(	0.43	0.17	0.11	10.8	126.58	97.95	11.4	167907328
2016-01-3(	0.43	0.17	0.11	10.81	127.35	98.2	11.4	165875712
2016-01-3(	0.43	0.17	0.11	10.81	128.08	98.5	11.4	166002688
2016-01-3(	0.43	0.17	0.11	10.81	128.79	98.82	11.5	165875712
2016-01-3(	0.43	0.17	0.11	10.81	129.52	99.12	11.6	166264832
2016-01-3(	0.4	0.17	0.11	10.81	129.84	99.13	11.6	166645760
2016-01-3(	0.4	0.17	0.11	10.81	129.86	99.13	11.7	166678528
2016-01-3(	0.4	0.17	0.11	10.83	129.88	99.13	11.7	166678528
2016-01-3(	0.4	0.17	0.11	10.83	130.11	99.15	11.8	167153664
2016-01-3(	0.4	0.17	0.11	10.83	131.1	99.21	12.1	168939520
2016-01-3(	0.44	0.18	0.12	10.83	132.06	99.29	12.4	169955328
2016-01-3(	0.44	0.18	0.12	10.83	132.93	99.46	12.5	170590208
2016-01-3(	0.44	0.18	0.12	10.83	133.12	99.48	12.5	170590208
2016-01-3(	0.44	0.18	0.12	10.83	133.3	99.5	12.5	170590208
2016-01-3(	0.44	0.18	0.12	10.83	133.42	99.57	12.5	170590208
2016-01-3(	0.57	0.21	0.13	11.21	133.59	99.64	12.6	171003904
2016-01-3(	0.57	0.21	0.13	11.71	133.71	99.65	12.6	170971136
2016-01-3(	0.57	0.21	0.13	11.71	133.79	99.68	12.6	170971136
2016-01-3(	0.57	0.21	0.13	11.71	133.93	99.69	12.6	170971136
2016-01-3(	0.57	0.21	0.13	11.71	134.01	99.72	12.6	171098112

Charge, attente en entrées/sorties et utilisation de la mémoire pour Go

Time	Load_1	Load_5	Load_15	iowait	user	system	Mem %	Mem usage
2016-01-3(	0.08	0.12	0.13	12.99	155.28	106.41	9.4	149098496
2016-01-3(	0.08	0.12	0.13	12.99	156.24	106.49	13	165605376
2016-01-3(	0.08	0.12	0.13	12.99	157.19	106.58	16.6	181985280
2016-01-3(	0.16	0.14	0.13	12.99	158.1	106.71	19.3	194301952
2016-01-3(	0.16	0.14	0.13	12.99	159.06	106.79	19.9	200396800
2016-01-3(	0.16	0.14	0.13	12.99	159.21	106.88	8.1	146956288
2016-01-3(	0.16	0.14	0.13	12.99	159.22	106.9	8.1	146956288
2016-01-3(	0.14	0.13	0.13	12.99	159.23	106.91	8.1	146956288
2016-01-3(	0.14	0.13	0.13	12.99	159.24	106.92	8.1	146989056
2016-01-3(	0.14	0.13	0.13	12.99	159.26	106.92	8.1	146989056
2016-01-3(	0.14	0.13	0.13	12.99	159.29	106.94	8.1	147091456
2016-01-3(	0.14	0.13	0.13	12.99	159.32	106.96	8.1	147091456
2016-01-3(	0.13	0.13	0.13	12.99	159.38	106.97	8.1	147091456
2016-01-3(	0.13	0.13	0.13	12.99	159.42	106.98	8.1	147099648
2016-01-3(	0.13	0.13	0.13	12.99	159.48	106.98	8.1	147099648
2016-01-3(	0.13	0.13	0.13	13	159.53	106.99	8.1	147132416
2016-01-3(	0.13	0.13	0.13	13	159.57	107.03	8.1	147132416
2016-01-3(	0.12	0.13	0.13	13	159.62	107.05	8.1	147099648
2016-01-3(	0.12	0.13	0.13	13	159.67	107.05	8.2	147226624
2016-01-3(	0.12	0.13	0.13	13	159.71	107.07	8.2	147226624
2016-01-3(	0.12	0.13	0.13	13	159.74	107.09	8.2	147226624
2016-01-3(	0.12	0.13	0.13	13	159.79	107.1	8.2	147226624
2016-01-3(	0.11	0.13	0.13	13	159.83	107.12	8.2	147226624
2016-01-3(	0.11	0.13	0.13	13	159.88	107.14	8.2	147226624
2016-01-3(	0.11	0.13	0.13	13	159.91	107.16	8.2	147226624
2016-01-3(	0.11	0.13	0.13	13	159.97	107.16	8.2	147226624
2016-01-3(	0.11	0.13	0.13	13.01	160	107.18	8.2	147259392
2016-01-3(	0.18	0.14	0.13	13.01	160.05	107.19	8.2	147226624
2016-01-3(	0.18	0.14	0.13	13.01	160.08	107.21	8.2	147226624
2016-01-3(	0.18	0.14	0.13	13.01	160.11	107.23	8.2	147226624
2016-01-3(	0.18	0.14	0.13	13.01	160.17	107.28	8.2	147226624
2016-01-3(	0.18	0.14	0.13	13.02	160.21	107.3	8.2	147226624
2016-01-3(	0.17	0.14	0.13	13.02	160.23	107.32	8.2	147226624
2016-01-3(	0.17	0.14	0.13	13.02	160.27	107.34	8.2	147226624

Annexe 4

Résultats consommation électrique

idle	idle-node	idle-java	idle-go	node	java	go	
310	309	303	299	298	312	303	
314	299	298	305	300	313	309	
306	294	311	296	311	295	309	
312	308	296	303	305	303	307	
303	296	318	302	304	296	310	
309	304	305	307	310	315	293	
305	299	301	303	297	309	294	
304	304	306	300	312	309	297	
302	311	298	305	299	307	305	
303	305	300	308	309	308	300	
314	298	313	304	301	298	299	
295	295	304	305	289	293	309	
314	309	313	309	303	294	301	
304	305	315	299	306	311	295	
308	295	303	301	311	307	299	
301	290	304	310	314	305	301	
308	309	302	310	304	302	305	
307	300	305	305	304	297	302	
302	305	296	303	301	309	299	
297	305	304	310	308	306	305	
294	305	294	300	307	298	299	
294	302	302	313	301	306	303	
305	305	302	309	310	293	306	
303	309	305	302	298	300	297	
297	310	315	299	293	305	302	
308	301	305	301	300	293	303	
303	303	300	302	306	306	296	
314	304	305	299	307	309	300	
302	305	303	306	308	303	296	
299	310	295	303	306	309	306	