Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 958241, 13 pages
http://dx.doi.org/10.1155/2013/958241

Research Article

Improving Performance through REST Open API Grouping for Wireless Sensor Network

Min Choi,[1] Young-Sik Jeong,[2] and Jong Hyuk Park[3]

[1] *Department of Information and Communication Engineering, Chungbuk National University, 52 Naesudong-ro, Heungdeok-gu, Cheongju, Chungbuk 361-763, Republic of Korea*
[2] *Department of Multimedia Engineering, Dongguk University, 30 Pildong-ro 1-gil, Jung-gu, Seoul 100-715, Republic of Korea*
[3] *Department of Computer Engineering, Seoul National University of Science and Technology, 232 Gongneung-ro, Nowon-gu, Seoul 139-743, Republic of Korea*

Correspondence should be addressed to Jong Hyuk Park; parkjonghyuk1@hotmail.com

Received 18 June 2013; Accepted 30 August 2013

Academic Editor: Naveen Chilamkurti

With this growth of the Internet, it is expected that every device, including computers, will be connected to the Internet, as it is called IoT. For example, smartphones and even refrigerators require an address to connect to the Internet. In this research, we design Internet of things architecture, especially for wireless sensor networks. The architecture consists of wireless sensor networks with a microcontroller at the very bottom level. They are connected to smart devices at the next level. However, the computing capability of the smart devices is generally less powerful than that of the conventional devices. Thus, it is necessary to offload the computation-intensive part by careful partitioning of application functions. In this research, we focus on designing the concept of MapReduce like approach through the web service grouping of several web services into one. We propose two methods: REST API grouping and REST API caching. First, the web service composition results in reducing energy consumption and communication latency by composing two or more REST web services into one. Second, the web service caching technique provides fast access that is recently accessed or frequently accessed. We conducted the experiments with Jersey REST web service server. Experimental result shows that our approach outperforms conventional approaches.

1. Introduction

With the fast development of the Internet technologies, web based architectures are becoming the major technologies for various fields of mobile computing. Nowadays, we are experiencing a major shift from traditional mobile applications to mobile cloud computing. The demand of Open API based development stems from the increasing use of smartphone applications [1, 2]. Community portal companies such as Google, Naver, and Yahoo are providing the Open API service for the access of their service. Before we go into more details, we briefly introduce the REST Open API based mobile application development approaches.

Within a few years, we can expect a major shift from traditional mobile application technology to mobile cloud computing [3]. It improves application performance and efficiency by offloading complex and time-consuming tasks onto powerful computing platforms. By running only simple tasks on mobile devices, we can achieve a longer battery lifetime and a greater processing efficiency. Not only is this offloading with the use of parallelism faster but it can also be used to solve problems related to large datasets of nonlocal resources. With a set of computers connected on a network, there is a vast pool of CPUs and resources, and you have the ability to access files on a cloud. In this paper, we propose a novel approach that realizes the mobile cloud convergence in a transparent and platform-independent way. Users need not know how their jobs are actually executed in distributed environment, and users need not take into account whether their mobile platforms are iPhone or Android.

To communicate with remote procedure call between client and server, interface should be defined at first. To this end, WSDL and RPC were used for the specification. However, these previous approaches are relatively complicated and highly overloaded. Recently, REST architecture is first introduced by Fielding. REST web service is becoming popular and explosively used in the field of application development of web and smartphone. Therefore, today's many Internet companies already provide their services by both traditional SOAP based web service and RESTful web services [4, 5]. The main difference between REST web service and SOAP/WSDL web service is as follows. Due to the complicated characteristics of SOAP based web services, REST web service is introduced. REST web service removes the overhead from encoding/decoding of header and body during message transfer. The REST web service enables users and developers to easily use the web services at remote or local sites. We need not add additional communication layer or protocols for REST web service, but we can easily achieve scalability and performance. This research evaluates the performance of mash-up architectures through RESTful Open API web services on smart mobile devices. It provides the analytical and experimental results for the performance evaluation of system models. We especially try to find an optimal number of parallel REST web server architectures under certain request arrival rates. And we show the performance of proposed architecture, especially the mean number of requests in the queue and the mean waiting time.

The area of REST web service composition is underexplored and most research efforts are still at their initial state [3, 6–8]. In this paper, we propose a new conversion method from web service execution result to object. REST web service execution results are usually provided in XML format. Previous composition method is required to analyze web service execution result with XML parser [9]. Other previous composition approaches were exploited to synthesize program code from linear logic or first-order logic [1, 2]. These papers are well organized theoretically and logically, but they have limited capability in terms of automatic synthesis. However, in order to provide an easy way for web service composition, we convert REST web service to objects. The conversion changes web service to a directly readable format (objects) with OOP language. The objects are programming primitives generally available for all types of OOP languages, such as C++ and Java. Since OOP languages are very popular for developers, they can easily utilize OOPs to compose web services.

REST web service is core technology for smartphone application development. This is because REST web service is the most appropriate way for accessing information through the Internet. Usually, a smartphone application needs information from several sources of (one or more) REST web services [1]. So, we need to utilize two or more REST web services composition to realize a target application [3, 6]. In this paper, we propose a server architecture for managing REST web services. This server is for managing web services so as to provide web server maintenance, especially on composition, deployment, and management of REST web services. It enables service developers to conveniently develop, deploy, upload, and run their composed web services with the use of general OOP languages.

The rest of this paper is organized as follows. Section 3 describes the necessity of WSN using REST web service grouping. Section 2 shows conversion of REST web service to objects. Section 4 depicts management server architecture for REST web service system. Then, Section 5 shows performance evaluation. Finally, we conclude and summarize our work in Section 6.

Wireless sensor network brings computing power to places and things that were previously not able to imagine to realize because they were cost prohibitive or physically impossible [10, 11]. This emerging wireless technology allows computing to go to places never before possible, everywhere of our physical world. Via the Internet, a variety of computing devices, including wireless sensor network devices, are connected into a worldwide computing network and becoming the next generation communication devices.

In this research, we make use of the Chipcon CC2420 RF Transceiver which is capable of 2.4 GHz communication by IEEE 802.15.4 standard [12]. It reduces the load on the host controller and allows CC2420 to interface low-cost microcontrollers. Figure 1 shows a block diagram of RF sensor module and its connection to ATmega128 microcontroller.

The left side of Figure 2 shows a CC2420 RF module. The right side of Figure 2 represents an example of real connection between ATmega128 microcontroller and the CC2420 RF module. During transmission, the FIFO and FIFOP pins are only related to the RXFIFO. The SFD pin is active during transmission of a data frame. The SFD pin goes high when the SFD field has been completely transmitted. In receive mode, the FIFOP pin can be used to interrupt the microcontroller when a threshold has been exceeded or a complete frame has been received. This pin should be connected to an ATmega128's interrupt input port. The ATmega128 microcontroller communicates with smart devices to provide data which is applicable to arguments of REST Open APIs, for example, temperature, brightness, humidity, and any data from ADC. In 2004, the concept of MapReduce [13] was introduced as a novel programming model and implementation for a large set of computing devices. Map generates a set of intermediate key/value pairs and Reduce merges all intermediate values associated with the same intermediate key, so that programs with this are automatically parallelized and executed on a large cluster of computing devices.

This research focuses on designing the concept of MapReduce through the web service composition of several web services into one. This REST web service composition results in reducing energy consumption and communication latency. This is because the conventional approach generates several consecutive connection requests to remote servers through REST Open API. In this case, two or more REST web services execution should be carried out on smartphones. However, our REST web service composition eliminates such several consecutive connection requests since several REST web services are composed into one. Moreover, the REST web service caching technique in this research provides

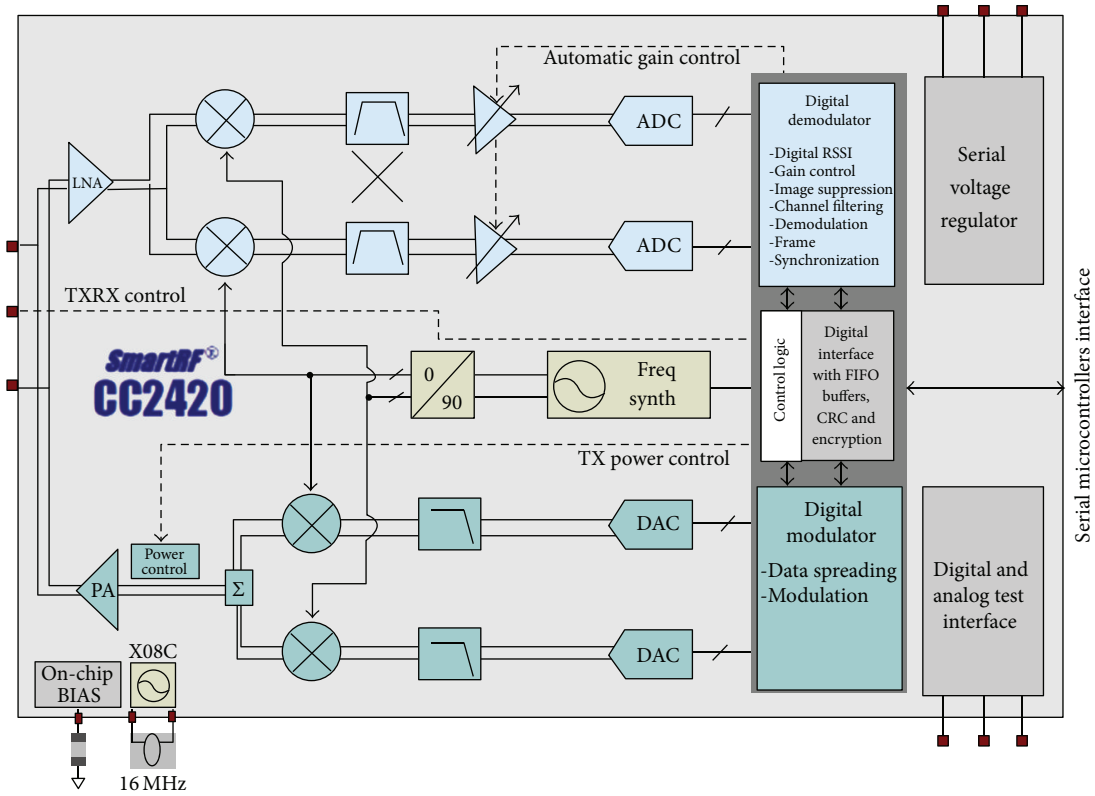
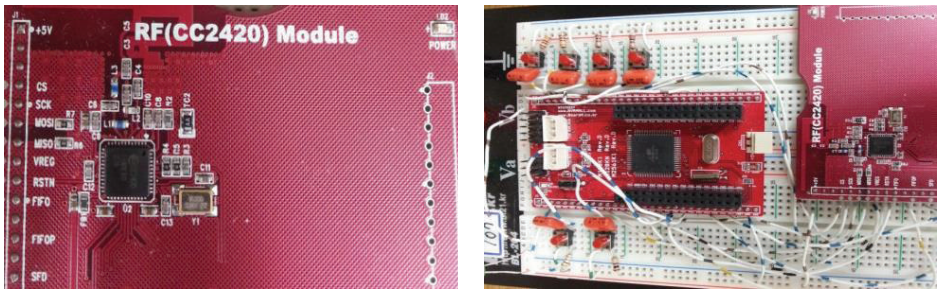


Figure 1: Chipcon CC2420 RF module block diagram [12].



(a) (b)

Figure 2: CC2420 RF module and ATmega128 board.

optimization through data caching that is recently accessed or frequently accessed.

REST web service is core technology for smartphone application development. This is because REST web service is the most appropriate way for information access through the Internet. Usually, a smartphone application needs information from several (one or more) REST web services. So, we need to utilize two or more REST web services for realizing target application. Algorithm 1 shows the Open API REST web service for the development of search applications on smartphones.

Algorithm 2 shows the example of open API REST web service for keyword search of web documents. Assuming developing a web search application, two-phase search task is necessary; the first step is to check the validity of search keyword. This step is to prevent persons who are under 19 years of age to access adult data through the search engine. The second step is to search the keyword actually from web database. This step is to get the content from search engine after checking keyword validity of Algorithm 1. Algorithm 3 describes the list of error messages when there is failure in Open API request, for example, invalid input parameters, network failure, and authentication failure. Like above, the necessity for several REST web service composition is obvious.

2. Conversion of REST Web Service to Objects

Web service composition requires a method to access data that are in XML format of web service execution result

Table 1: Real examples of the web service to object conversion.

Class variables	Num	Latitude	Longitude	Name	Street	City	County	State	Country
Object instance	1	37.511591238542	127.05944600764	Coex Mall		Samsung-dong	Kangnam-gu	Seoul	Republic of Korea
Object instance	2	37.512302929275	127.05963882578	Coex Square		Samsung-dong	Kangnam-gu	Seoul	Republic of Korea

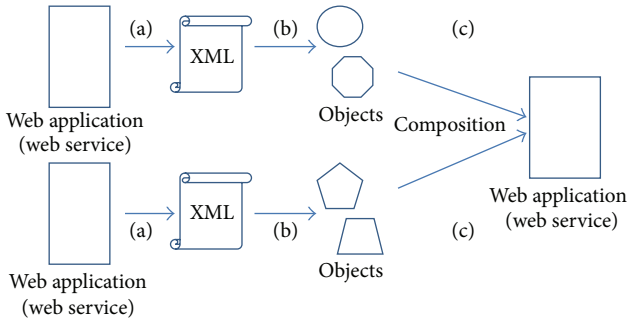


Figure 3: Conversion of web service execution result to objects.

from OOP languages. Because OOP languages are familiar to developers, they can easily utilize the OOPs to compose web services. In order to realize REST web service composition, we propose a web service composition method by conversion of web service to objects. The reason why we convert the REST web service to objects is to make an easy way for manipulating the web service result into composition. The objects mean that they are the programming primitives which are generally available for all types of OOP languages, such as C++ and Java.

Figure 3 depicts the extraction process from web service to objects. Step (a) represents that web service execution results are returned as XML format. Step (b) describes the process that XML is converted to object. The process is to derive objects that are available for object-oriented languages. Actually, our web service manager converts the XML result to create Java program objects that represent the result data of web service execution. step (c) is to compose several objects to make another new web service that utilizes one or more web services. The reason why we convert web service to object is that the object-oriented language is the most convenient tool for developers to manipulate and understand easily. This is because previous REST web service composition proposals of H. Zhao [4] and X. Zhao [5] are well organized theoretically and logically, but these are difficult for developers and users to easily understand and manipulate with a familiar programming language.

Algorithm 4 represents the result of REST web service execution. REST web service execution result is provided as XML data like above. After that, we proceed to the extraction process with the results of REST web service execution. In this example, we use web service from Yahoo. The REST web service is called "Open API." It returns execution result by a type of XML data. When clients receive the XML, they first parse the XML and finally they get the data that they wanted.

3. The Necessity of WSN Using REST Web Service Grouping for MapReduce

As shown in Algorithm 4, the above XML document contains a single element ResultSet, which has subelements for head, locations, and item. The locations element contains a collection of item elements. The item element has several attributes: num, latitude, longitude, name, street, city, county, state, and country. In Algorithm 5, we see the class definition for conversed object mapping. For convenient use of objects in OOP languages, the result of data conversion should be provided by a real object instance which can be directly referenced on OOP program source code. To do this, we need a skeleton class, shown as a sample in Algorithm 5, in which simple types are mapped to each property variable and service developers can access the property values using get and set methods as shown in Algorithm 5. The above example in Algorithm 5 of REST web service execution can be converted by the following set of object instances which have attributes of the following result in Table 1.

For the example of Algorithm 4, the locations element contains a collection of <item> tags which has subelements. The tag <item> is the *object identification element* in our conversion system, and it has several attributes: num, latitude, longitude, name, street, city, county, state, and country. It is easy for a human to make a decision that <item> tag is repeated per object. However, it is not easy for a machine to decide which tag is corresponding to the object separator. This conversion process is repeated until reaching the end tag. That is why our system requires the separation of tag name for object identification at the beginning of the conversion process.

4. Management Server Architecture for REST Web Service

In this section, we propose a server architecture for managing REST web services. This server is for managing web services instead of web so as to provide web server maintenance service, especially composition, deployment, and management. Figure 4 shows the architecture of our REST web service management system. The main role of the system is composition/deployment/management of REST web services. It enables service developers to conveniently develop, deploy,

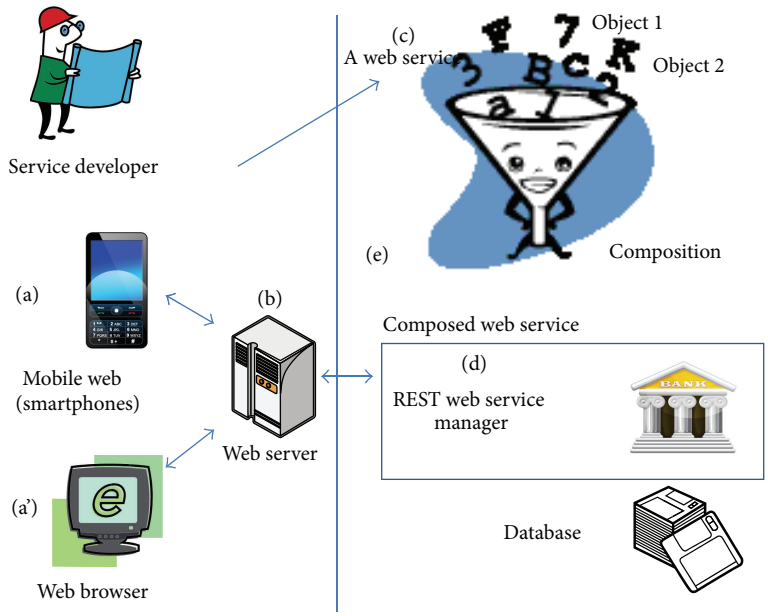


Figure 4: REST web service management system.

(1) Request URL
http://openapi.naver.com/search

(2) Request parameter
key string (mandatory): key string for authentication
target string (mandatory): adult
query string (mandatory): search keyword as UTF-8 encoding

- Sample URL
http://openapi.naver.com/search?key=test&query=girl&target=adult

(3) Response field
adult integer: 0, 1 (0—non adult, 1—adult)

Algorithm 1: Open API of REST web service for checking search keyword validity.

upload, and run the composed web service by general OOP languages. Web browsers and mobile web browsers shown in (a) of Figure 4 are very popular on desktop and smartphones, respectively. They commonly utilize the HTTP to communicate with web server through port number 80. The web server in (b) of Figure 4 is an application daemon which receives request from web browser and provides the requested documents and data. Module (c) in Figure 4 represents web service or composed web service. It can be provided by platform-independent packaging technology, such as COM/COM+ and JavaBeans. This package can include directory structure that has a restriction on which directory should have a configuration file for our web service management system. Module (d) in Figure 4 is REST web service manager. It manages REST web services which can be either a native REST web service or a composed REST web service. It provides service to requests from clients. The service developers (e) in Figure 4 can upload their web service or composed web service onto our REST web service management system, so that web services can be launched and serviced on demand. This is quite useful for smartphone application developers. This is because the computing power of smartphones is generally less than that of other mobile computing devices, such as laptop computers and mobile tablets. Therefore, it is necessary to offload the computation-intensive part by the careful partitioning of application functions across the cloud computing platform. To this end, we make use of RESTful web service to realize distributed computing environment.

During loading and running composed REST web service in (d) of Figure 4, dynamic binding is required for composed web service to use objects which are converted from web service. Likewise, the management system has to instantiate and dynamically bind the composed objects. This is because our system provides service concurrent users at the same time. At the time, it is not possible for all web services to be loaded onto memory. Some of them might be garbage collected during the runtime. Thus, we need to reload and bind the object when it is about to be referenced. So, our web service management system supports dynamic loading and binding for converted objects from web service.

Likewise, we propose a REST web service management system that provides REST web service for clients such as

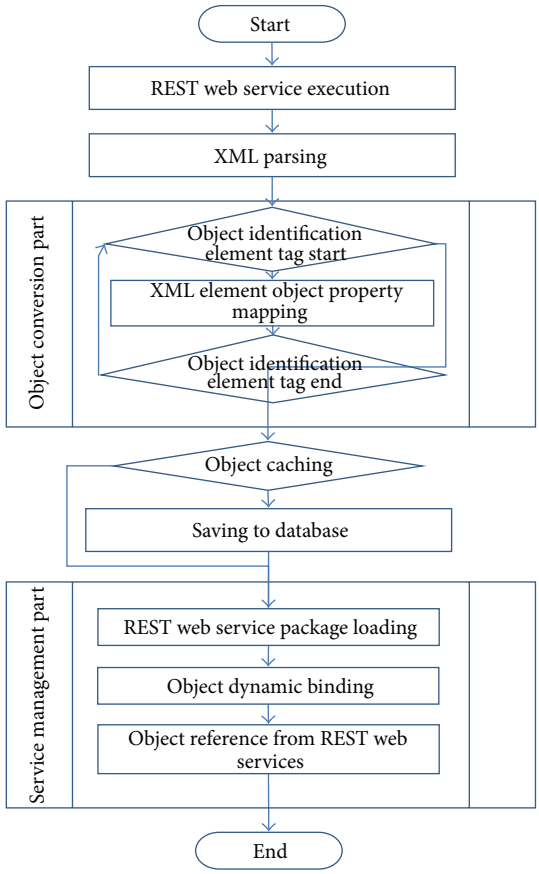


Figure 5: REST web service composition procedure.

smartphone applications. Simply by uploading their web service package onto our system, web service developers can operate their service without a physical server. There is only a restriction that binary format of uploaded package should be compatible to our service management operating system and configuration file should be located onto a specified directory on our system. The configuration file could be of various formats such as xml and cfg. The contents can be as follows.

Algorithm 6 represents the sample configuration file for the description of REST web service package location. The configuration file includes a single top-level element <XML element> which has subelements <name> for package and <value> element for package path (directory). By putting the configuration file onto a specific path of REST web service management system, our management system locates the configuration file and launches developer's web services.

Figure 5 depicts the flowchart of our web service composition and management system. There are two big parts; one is the object conversion module and the other one is the service management module. The object conversion module is to convert the web service execution result to object. The service management module is to deploy and run the composed web service. In Figure 5, there is a stage in the middle of those two above modules for checking the option of saving the converted object into the database temporarily. This option is to improve the performance of our web service composition and management system. This is similar to object caching for consecutive requests. Assuming that we enable the option for object caching, then we can remove the upper part of Algorithm 6. This is because object 1 in Algorithm 6 is already stored in the database and the lower part of Algorithm 6 can reuse the stored object. At this time, we have to be careful about checking that the stored object is up to date. If the stored value is not up to date, the cached object should be invalidated. To this end, our management system periodically checks the object validity by a preset threshold which is called OVL. Usually, we set this threshold to the web service characteristics, because for some web services the result does not change when input value is the same, but for some web services the result often changes depending on time.

Object caching significantly improves the performance of our web service composition and management system, especially on sudden batch requests within a short period of time, because the object caching technique does not reexecute web service that results in the same result at every request, for example, in Algorithm 7, the web service execution for checking adult keyword. In this case, for the same keyword, the result of web service execution is always the same.

If result of REST web service execution contains several result sets, the object conversion module separates the result sets into several objects. This is one of the major differences between REST web service and SOAP/WSDL based web service. Object conversion from SOAP/WSDL web service is relatively easy to decide which part is corresponding to a specific object, since WSDL describes specifically and formally the details of the web service so that the system automatically understands the web service result and manipulates the result. However, object conversion from REST web service is not so easy because it does not support WSDL-like description and there is no formal description for the REST web service. That is why we introduced the concept of *object identification element*. As shown in the example of Algorithm 4 in the previous research [1], the tag <item> is selected as object identification element by REST web service developer (human). It is easy for a human to make the decision that <item> tag is repeated per object. However, it is not easy for a machine to decide which tag is corresponding to the object separator. So, the conversion process is repeated until reaching the end tag. To this end, our system requires the separation tag name for object identification at the beginning of the conversion process.

After conversion, we can develop various applications on desktop, mobile, and smartphones. Then, we need not access REST web service through Open API interface but we make use of converted objects. Algorithm 7 shows an example source code for actual REST web service composition. The upper parts of Algorithm 7 represent REST web service execution for checking search keyword validity. When we get the result, we make use of XmlObjectConversionFactory and XmlObjectConverter to convert the result to object. These tools enable programmers to access and modify XML documents via a general OOP language, not via XML parsers. Actually, the lower parts of Algorithm 7 manipulate the web service execution result with XML parsers, especially XMLPullParser and XMLPullParserFactory. This research

(1) Request URL
http://openapi.naver.com/search

(2) Request parameter
key string (mandatory): key string for authentication.
target string (mandatory): webkr
query string (mandatory): search keyword as UTF-8 encoding
domain string: site domain for searching
display integer: search result numbers per a page (10 as default, 100 as max)
start integer: search start position (1 as min, 1000 as max)

- Sample URL
http://openapi.naver.com/search?key=test&query=keyword&display=10&start=1&target=webkr

(3) Response field
rss: for debugging
channel: container for search results
lastBuildDate datetime: date of result generated
total integer: total number of documents
start integer: start value of search result documents
display integer: number of search result
item: search result including title, link, description
title string: search result document title
link string: hypertext link for search result document.
description string: summary for search result document

Algorithm 2: Open API of REST web service for searching from web database.

(4) Error messages
000: System error
010: Your query request count is over the limit
011: Incorrect query request
020: Unregistered key
021: Your key is temporary unavailable
100: Invalid target value
101: Invalid display value
102: Invalid start value
110: Undefined sort value
200: Reserved
900: Undefined error occurred

Algorithm 3: Error messages of Open API REST web service.

focuses on the composition stage, which is shown in the upper part of Algorithm 7, and the stage only deals with conversion from web service to object. It does not care about the implementation details after conversion. Therefore, the lower parts of Algorithm 7 simply make use of conventional XML parsing as implementation techniques for another web service. This approach makes it possible for a developer who knows little about XML to process with a simple method and they immediately make use of the classes that are automatically converted. To work with web service execution result in their program, we map the result directly to a set of objects that reflects the XML result according to its structure. We can achieve this by converting the web service execution result into a set of derived content classes.

5. Performance Evaluation

In this section, we introduce system architecture and elementary technologies for implementing the mobile application. REST web service is core technology for smartphone/smart TV application development. This is because REST web service is the most appropriate way for accessing information through the Internet. Usually, a smartphone application needs information from several sources of (one or more) REST web services. First, we exploited our web service conversion technique [8] between REST Open API and objects. We have the REST web service conversion technology that includes the extraction process from web service to objects. When web service execution results are returned in XML format, we can convert the XML result to object. The process is to derive objects that are available for object-oriented languages. Actually, our web service manager converts the XML result to create Java program objects that represent the result data of web service execution. Finally, we compose several objects to make another new web service that utilizes one or more web services. The reason why we convert web service to object is that the object-oriented language is the most convenient tool for developers to manipulate and understand easily. Second, we make use of OAuth [6] based authentication mechanism in order to authenticate our REST web service users. OAuth is an open standard for authorization. It allows users to share their private resources stored on one site with another site without having to hand out their credentials, typically supplying username and password tokens instead. Each token grants access to a specific site for specific resources and for a defined duration. This allows a user to grant a third party site access to their information stored with another service

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet>
<head><publisher>YahooKorea
GUGI</publisher><Error>0</Error><ErrorMessage>No
Error</ErrorMessage><Found>8</Found>
</head>
<locations>
<item>
        <num>1</num>
        <latitude>37.511591238542</latitude>
        <longitude>127.05944600764</longitude>
        <name>Coex Mall</name>
        <street></street>
        <city>Samsung-dong</city>
        <county>Kangnam-gu</county>
        <state>Seoul</state>
        <country>South Korea</country>
</item>
<item>
        <num>2</num>
        <latitude>37.512302929275</latitude>
        <longitude>127.05963882578</longitude>
        <name>Coex Square</name>
        <street></street>
        <city>Samsung-dong</city>
        <county>Kangnam-gu</county>
        <state>Seoul </state>
        <country>South Korea</country>
</item>
</locations>
</ResultSet>
<!-- openapi1.local.maps.kr3.yahoo.com uncompressed/chunked Fri Jan 6 23:55:53 UTC 2012 -- >
```

Algorithm 4: REST web service execution example.

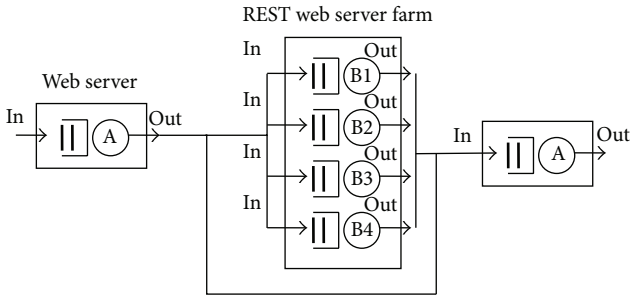


Figure 6: System architecture.

provider, without sharing their access permissions or the full extent of their data. Third, we adopt the Apache Tomcat 7.0 as a web application server and Jersey 1.8 for REST Open API service provider. Apache Tomcat is an open software with Java Servlet and JavaServer Pages technologies. Apache Tomcat powers numerous large-scale web applications across a diverse range of industries and organizations. Jersey is the open source JAX-RS (JSR 311) Reference Implementation [7] for building RESTful web services. Jersey provides an API so that developers may extend Jersey to suit their needs. We make use of both Tomcat and Jersey in order to implement our systems.

For evaluating system performance of modeled web server traffic (requests), we present a model of queuing network [8] as shown in Figure 6, incorporating M/M/1 and M/M/s models. The model of our REST Open API web service architecture is presented in Figure 1. REST Open API web service is composed of 3 components comprising the following: (1) a web server, (2) REST web server farms, and (3) mash-up applications. As shown in Figure 1, there are a number of components (nodes) that consist of several queues. A request may receive service at one or more queues before exiting from the system. A model shows the system architecture in which jobs departing from A arrive at another queue (i.e., the REST web server farm from B1 to B4).

Requests arrive at the web server A with frequency FreqIn. The initialization process for the request is done at node A. Then, the request proceeds to the component, either "REST web server farm" network or others depending on the type of the request; if the request is for the REST web server, then it goes to the REST web server farm. If the request is for just web server, then it goes to the web server. The requests traverse via the Internet users and are received by the client's browser, represented by the components at the bottom of

```
class definition {
        access_modifier variable_type property_variable1 (argument);
        access_modifier variable_type property_variable2 (argument);
        return_type get_property_variable1() {
                return property_variable1;
        }
        return_type set_property_variable1(variable_type value1) {
                property_variable1 = value1;
        }
        return_type get_property_variable2() {
                return property_variable2;
        }
        return_type set_property_variable2(variable_type value2) {
                property_value2 = value2;
        }
}
```

Algorithm 5: Class definition for web service to object conversion.

```
<XML element>
    <name>web service package</name>
    <value>web service package path(directory)</value>
</XML element>
```

Algorithm 6: Configuration requirement for uploading REST web service.

Figure 5. Our system model is a sort of open queuing network that has external arrivals and departures. The requests enter the system at "IN" and exit at "OUT." The number of requests in the system varies with time. In analyzing an open system, we assume that the throughput is known (to be equal to the arrival rate), and we also assume that there is no probability of incomplete transfer in this system, so there is no retrial path to go back to node A. The REST web server farm can have more than one computing server; we especially present 4 computing servers in Figure 5.

At each node, let us consider an M/M/1 queue with a processor sharing service discipline. The interarrival times of requests are according to a Poisson process with rate λ and the service times are exponentially distributed and there is only one server. There are no buffer or population size limitations and the service discipline is FCFS. In this research, experimental results were conducted by both real experiments and performance analysis of M/M/1 model. In our system in Figure 1, we do not depend only on the M/M/1 queue as shown but we also make use of M/M/s model [9]. This is because the REST web service in our system is provided by clustered/clouded systems consisting of multiple numbers of servers. So, it is necessary to analyze this component separately using M/M/s model. The M/M/s model can be used to model parallel systems that have several identical servers and all jobs waiting for these servers are kept in one system. It is assumed that there are m servers each with a service rate of μ requests per unit time. The arrival rate is λ jobs per unit time. If any m servers are busy, the arriving requests wait in a queue. The state of the system is represented by the number of jobs n in the system. We can get the mean waiting time as follows:

$$E[w] = \frac{E[n_q]}{\lambda} = \frac{\varrho}{[m\mu(1-\rho)]}. \quad (1)$$

The expressions are used for performance analysis of M/M/s model in this research. Requests arrive from outside following a Poisson process with a certain arrival rate $\lambda > 0$. Hence, we have the overall arrival rate to node i, λ_i, including both external arrivals and internal transitions:

$$\lambda\lambda_i = \alpha p_{0i} + \sum_{j=1}^{J} \lambda_j p_{ji}, \quad i = 1, \ldots, J; \quad (2)$$

then

$$\lambda = (-P)^{-1} a. \quad (3)$$

All requests submitted must first pass through the web server for providing HTTP service before moving on to the REST web servers, Jersey. Requests arrive at the web server at an average rate of 1000/sec to 15000/sec. To handle the load, the REST web server components may have several parallel clouded or clustered architectures.

Figure 7 represents the mean number of requests in the queue and traffic intensity at component A. Traffic intensity is calculated by the arrival rate over the service rate that means how fast the incoming traffic is serviced on the server. The traffic intensity is a sort of constant on M/M/1 queue (component A is M/M/1 queue). Since the service rate of the Apache web server is 16000 requests/sec, the mean number of requests in the queue reaches up to maximum on the total arrival rate is increasing to 15000. Figure 8 shows the mean number of requests in the queue as increasing total arrival rate. And this is the similar to the case of Figure 7.

```
Object1 obj1;
String strSearchKeyword = getParameter(STR_PARAM_SEARCHKEYWORD);
String strWebSvcQuery = "http://openapi.naver.com/search?key=test&query=";
strWebSvcQuery += strSearchKeyword + "&target=adult";
URL text = new URL(strWebSvcQuery);
XmlObjectConversionFactory objCreator = XmlObjectConversionFactory.newInstance();
XmlObjectConverter xoConverter = objCreator.newConverter();
obj1 = xoConverter.setInput (text.openStream(), null);
if (obj1.getbAdult()) {
   return;
}
else
{
      try{
            strWebSvcQuery = "http://openapi.naver.com/search?key=test&query=";
            strWebSvcQuery += strSearchKeyword +
            "&display=10&start=1&target=webkr";
            URL text = new URL(strWebSvcQuery);
            String test = text.toString();
            XmlPullParserFactory parserCreator =
            XmlPullParserFactory.newInstance();
            XmlPullParser parser = parserCreator.newPullParser();
            parser.setInput(text.openStream(), null);
            String tag;
            int parserEvent = parser.getEventType();
            while (parserEvent != XmlPullParser.END_DOCUMENT){
                  switch(parserEvent){
                  case XmlPullParser.TEXT:
                  tag = parser.getName();
                  break;
                  case XmlPullParser.END_TAG:
                  tag = parser.getName();
                  break;
                  case XmlPullParser.START_TAG:
                  tag = parser.getName();
                  break;
            }
      }catch(Exception e){
            Log.e("dd", "Error in network call" + e);
      }
}
```

Algorithm 7: Composition of multiple web services.

Figures 9 and 10 present the mean waiting time and the mean number of requests by the number of REST web servers. Until now, we just make use of four REST web servers without considering the optimal number of parallelism. So, we carried out the experiment as shown in Figures 9 and 10. To do this, we modeled component B as the M/M/m queue. The value m is larger than 1. This means the REST web servers are comprised of multiple Jersey 1.6 servers. These figures show the fact that the mean number of requests in the queue is decreasing as the number of REST web servers increases. Actually, we carried out this experiment with the four multiple servers of Jersey 1.6 REST web service providers. From these experiments, we see the fact that 4 or 5 numbers of REST web servers are enough for the current level of workloads.

Figures 11 and 12 present the mean waiting time and the mean number of requests in the queue with increasing of total arrival rate at component B. We model component B as the M/M/m queue. The value m is larger than 1. This means the REST web servers are comprised of multiple Jersey 1.6 servers. These figures show the fact that the mean number of requests and the mean waiting time are increasing as the total arrival

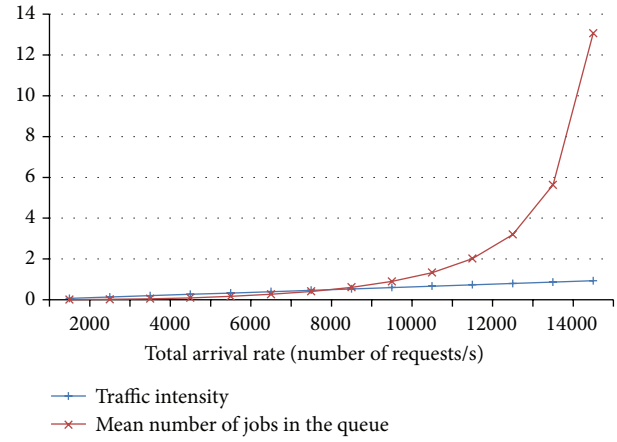


Figure 7: Traffic intensity and mean number of requests in the queue as increasing total arrival rate.

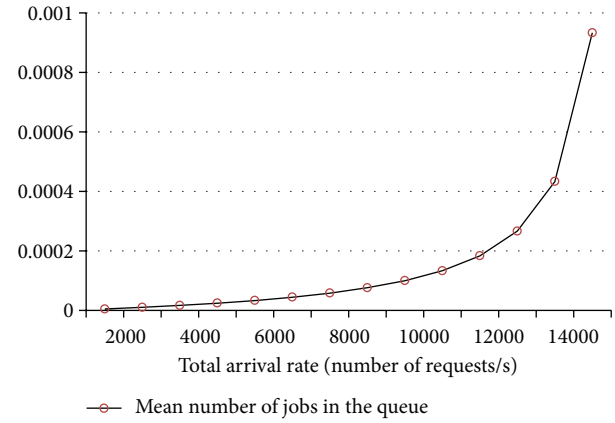


Figure 8: Mean number of requests in the queue as increasing total arrival rate.

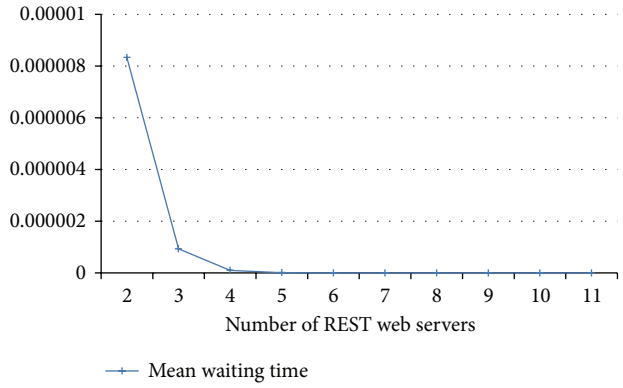


Figure 9: Mean waiting time as increasing number of REST web servers.

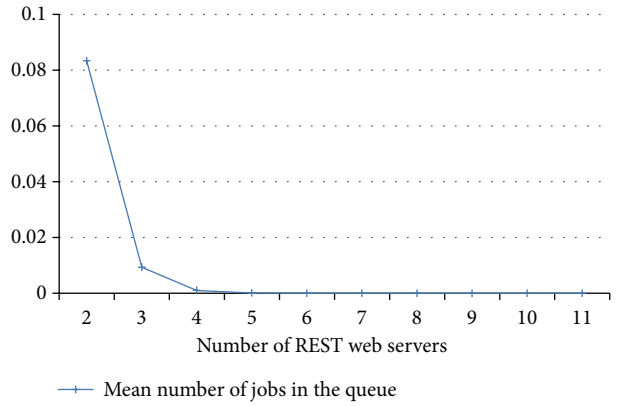


Figure 10: Mean number of requests in the queue as increasing number of REST web servers.

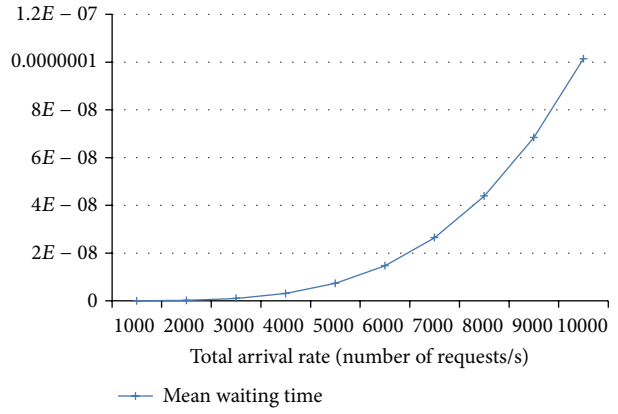


Figure 11: Mean waiting time as increasing total arrival rate.

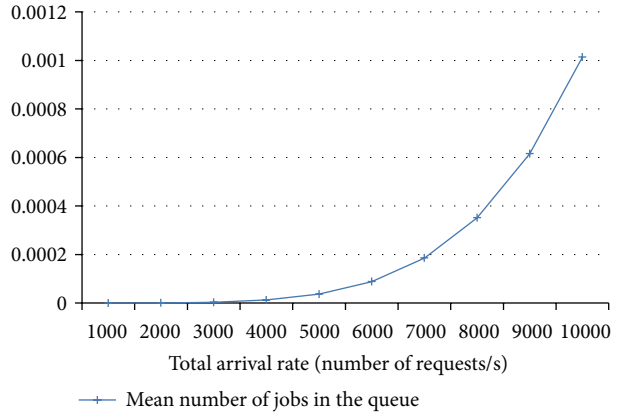


Figure 12: Mean number of requests in the queue as increasing total arrival rate.

rate. The reason why the waiting time and queue length are not reaching the maximum even though the total arrival rate approaches the maximum value (15000) is the multiple REST web servers. Actually, we carried out this experiment with the four multiple servers of Jersey 1.6 REST web service providers.

We also provide the experimental result to represent the efficiency of our management server for REST web service. We established the architecture described in Section 4, consisting of web service manager and web server, and we implemented and deployed the composed REST web service as shown in example code of Table 2. We performed our experiment on Intel Pentium Core I5, 4G platform. We used Tomcat

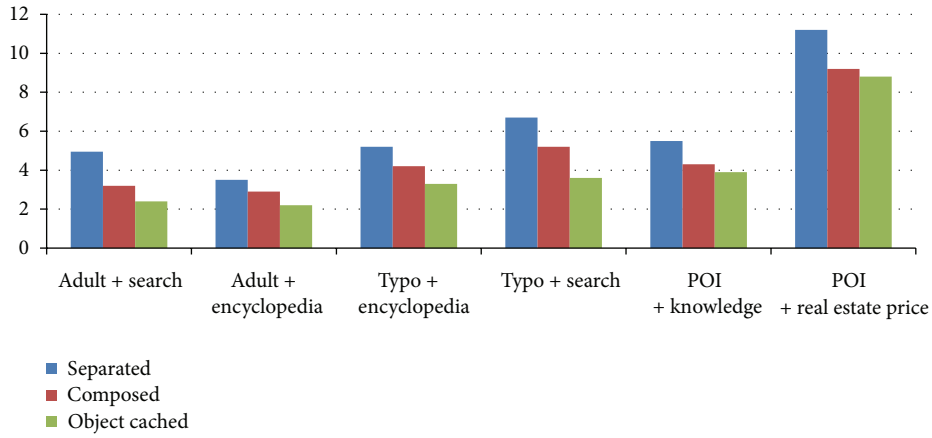


Figure 13: Experimental result of REST web service management server.

Table 2: Configuration parameters.

System measure	Component A	Component B
Total arrival rate, λ_i	From 1000/sec to 15000/sec	From 1000/sec to 15000/sec
Service rate, μ_i	15000/sec	15000/sec
Multiple number of servers, s_i	1	From 2 to 11
Traffic intensity, ρ_i	λ/μ	$\lambda/s\mu$

7.0 as web server and web application server. We implemented web service management and composition server onto the same server, but they might be separated depending on architecture design for performance improvement. Figure 13 shows our experimental result. The "separated" represents the conventional approach in which two or more REST web services execution should be carried out on smartphones. The "composed" represents our approach that several REST web services are composed into one. The "object cached" represents optimized approach in which object caching is exploited on composed REST web service.

As shown in Figure 13, REST web service composition method outperforms conventional approach from 17.15% up to 35.35%. The object caching optimization improves our REST web service composition technique from 4.34% to 25%. In cases of POI + knowledge and POI + real estate price of Figure 13, the performance improvement from the optimization is not significant because the amount of time to POI access is relatively lower than access time to real estate price and knowledge search open APIs.

6. Concluding Remarks

In this paper, REST web service execution results are usually provided in XML format. So, conventional methods need to analyze the result with XML parser. However, since OOP languages are familiar to developers, they can easily utilize the OOPs to compose web services. In order to realize REST web service composition, we proposed a new conversion method from web service execution result to object. The reason why we convert the REST web service to objects is to make an easy way for manipulating the web service result into composition. Actually, this paper showed Java objects conversion from the result of web service execution. This is very useful for smartphone application development, because REST web service is becoming popular and explosively used in the field of smartphone and web application development.

In addition, we proposed RESTful web service composition method and system which enable developers to easily deploy their web service through HTTP protocol. In this research, we proposed a REST Open API based mobile cloud application development architecture. This paper presents two case studies for the actual mobile cloud convergence information system. First, researchers are struggling to get information about research conferences, seminars, invited lectures, and so on. They try to search such activities through web search engines, but it takes a lot of time. This research is to provide a service for the Open APIs including the following: getting/putting conference information, searching research papers, PDF viewers, and mobile review systems. Future information systems will be working on smart devices such as smartphones or smart devices. Such smart devices are especially appropriate for the alerting conferences/seminars/workshops and reviewing research papers.

We also proposed a REST Open API based mobile cloud application development architecture. This research evaluates the performance of mash-up architectures through RESTful Open API web services on smart mobile devices. It provides the analytical and experimental results for the performance evaluation of system models. In particular, we found an optimal number of parallel REST web servers architectures under certain request arrival rates. And we showed the performance of proposed architecture, especially the mean number of requests in the queue and the mean waiting time.

As a future work, we will make this service available from MapReduce parallel computing platform in which job submission begins by the invocation of the REST Open API through web browsers and the job execution is done

by parallel computing platform with phases of mapping, synchronization, and reducing. One of the most important benefits of this approach is the fact that the interface of job submission is very familiar to users because it is REST Open API, not the MapReduce interface. Likewise, we will continue research on design and implementation of this platform, called REST-MapReduce.

Acronyms

REST: REpresentational state transfer
WSDL: Web service description language
RPC: Remote procedure call
IoT: Internet of things
M2M: Machine to machine
SOAP: Simple object access protocol
OOP: Object-oriented programming
ADC: Analog-digital converter
HTTP: Hyper text transfer protocol
OVL: Object validity lifetime.

Acknowledgments

This research was jointly supported by the Basic Science Research Program through the National Research Foundation of Korea (2013055028) and the MSIP (Ministry of Science, ICT & Future Planning), Republic of Korea, under the R&D program supervised by the KCA (Korea Communications Agency) (KCA-2013-12-912-03-001).

References

[1] M. S. Islam, M. R. Rahman, A. Roy, M. I. Islam, and M. R. Amin, "Performance evaluation of finite queue switching under two-dimensional M/G/1(m) traffic," *Journal of Information Processing Systems*, vol. 7, no. 4, pp. 679–690, 2011.

[2] T. Saba, "Implications of E-learning systems and self-efficiency on students outcomes: a model approach," *Human-Centric Computing and Information Sciences*, vol. 2, no. 6, pp. 1–12, 2012.

[3] R. Pan, G. Xu, B. Fu, P. Dolog, Z. Wang, and M. Leginus, "Improving recommendations by the clustering of tag neighbours," *Journal of Convergence*, vol. 3, no. 1, pp. 13–20, 2012.

[4] H. Zhao and P. Doshi, "Towards automated RESTful Web service composition," in *Proceedings of the IEEE International Conference on Web Services (ICWS '09)*, pp. 189–196, Los Angeles, Calif, USA, July 2009.

[5] X. Zhao, E. Liu, G. J. Clapworthy, N. Ye, and Y. Lu, "RESTful web service composition: extracting a process model from linear logic theorem proving," in *Proceedings of the 7th IEEE International Conference on Next Generation Web Services Practices (NWeSP '11)*, pp. 398–403, October 2011.

[6] "Mobile Public Information in Seoul Korea," http://mobile.openapi.seoul.go.kr/.

[7] "Shared resource portal in Korea," http://www.data.go.kr/.

[8] T. Saha, M. Abu Shufean, M. Alam, and M. I. Islam, "Performance evaluation of the WiMAX network based on combining the 2D markov chain and MMPP traffic model," *Journal of Information Processing Systems*, vol. 7, no. 4, pp. 653–678, 2011.

[9] R. Jain, *The Art of Computer System Performance Analysis, Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley & Sons, New York, NY, USA, 1991.

[10] "The internet of things: in action, the next web," http://thenextweb.com/insider/2013/05/19/the-internet-of-things-in-action/.

[11] M. Yoon, Y. K. Kim, and J. W. Jang, "An energy-efficient routing protocol using message success rate in wireless sensor networks," *Journal of Convergence*, vol. 4, no. 1, 2013.

[12] Chipcon CC2420 data sheet-2.4 GHz, IEEE 802.15.4 RF Transceiver.

[13] J. Dean and S. Ghemawa, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th USENIX Symposium on Operating System Design and Implementation*, 2004.