An Energy-aware Service Composition Algorithm for Multiple Cloud-based IoT Applications

T. Baker[1], M. Asim[2], H. Tawfik[3], B. Aldawsari[1], and R. Buyya[4]
[1]Department of Computer Science, Liverpool John Moores University, UK
[2]Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan
[3]School of Computing, Creative Technologies and Engineering, Leeds Beckett University, UK
[4]Department of Computing and Information Systems, The University of Melbourne, Australia
{t.baker,b.aldawsari}@ljmu.ac.uk; muhammad.asim@nu.edu.pk;
H.Tawfik@leedsbeckett.ac.uk; rbuyya@unimelb.edu.au

Abstract

There has been a shift in research towards the convergence of the Internet-of-Things (IoT) and cloud computing paradigms motivated by the need for IoT applications to leverage the unique characteristics of the cloud. IoT acts as an enabler to interconnect intelligent and self-configurable nodes "things" to establish an efficient and dynamic platform for communication and collaboration. IoT is becoming a major source of big data, contributing huge amounts of streamed information from a large number of interconnected nodes, which have to be stored, processed, and presented in an efficient, and easily interpretable form. Cloud computing can enable IoT to have the privilege of a virtual resources utilization infrastructure, which integrates storage devices, visualization platforms, resource monitoring, analytical tools, and client delivery. Given the number of things connected and the amount of data generated, a key challenge is the energy efficient composition and interoperability of heterogeneous things integrated with cloud resources and scattered across the globe, in order to create an on-demand energy efficient cloud based IoT application. In many cases, when a single service is not enough to complete the business requirement; a composition of web services is carried out. These composed web services are expected to collaborate towards a common goal with large amount of data exchange and various other operations. Massive data sets have to be exchanged between several geographically distributed and scattered services. The movement of mass data between services influences the whole application process in terms of energy consumption. One way to significantly reduce this massive data exchange is to use fewer services for a composition, which need to be created to complete a business requirement. Integrating fewer services can result in a reduction in data interchange, which in return helps in reducing the energy consumption and carbon footprint.

This paper develops a novel multi-cloud IoT service composition algorithm called (E2C2) that aims at creating an energy-aware composition plan by searching for and integrating the least possible number of IoT services, in order to fulfil user requirements. A formal user requirements translation and transformation modelling and analysis is adopted for the proposed algorithm. The algorithm was evaluated against four established service composition algorithms in multiple cloud environments (All clouds, Base cloud, Smart cloud, and COM2), with the results demonstrating the superior performance of our approach.
Keywords: IoT; Multi-cloud; Service composition; Energy efficiency

1. Introduction

The Internet of Things (IoT) paradigm has rapidly evolved in the last few years and envisions the internet as a set of intelligent, self-configuring and interconnected objects in a dynamic and global infrastructure. Objects refer to uniquely addressable smart devices that are generally distributed endowed with sensing and actuation capabilities and equipped with limited computing resources such as CPU, memory, and network capabilities. IoT can be defined as the integration of several technologies and communication solutions such as Radio Frequency Identification (RFID) technology, sensors and actuators. The basic idea behind it is the pervasive presence of smart objects around people, able to understand, monitor, and even control the environment. IoT has the potential to play a key role in both home and work scenarios, such as assisted living, smart home and office, e-health, and smart transportation [1–5]. For instance, in healthcare applications, small wearable devices are deployed to capture patients health data such as blood pressure and heart rate. The collected data are then sent to doctors for healthcare advice and could also be made available remotely to medical experts for research. Smart Cities is another revolutionary development of IoT applications that aims to establish an efficient communication and collaboration platform exploiting various information sources to make the cities smarter [6–8]. Despite the recent advances towards making IoT a reality, there exist several open issues that require further research and development efforts in order to exploit its full potential [9–14]. One of these challenges refer to the huge amount of energy consumed at streaming information produced by a large number of resource constrained interconnected devices, which have to be stored, processed, and presented in an efficient, and easily interpretable form.

Recently, various research have advocated cloud computing as a promising solution to address some of the challenges offer by the IoT. NIST in [15] defines cloud computing as a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or

service provider interaction. Cloud computing has taken computing from desktop to the world of internet, offering virtually unlimited capabilities in terms of storage, and processing power. During the last decade, the use of cloud computing to run businesses has increased noticeably due to the cloud on-demand utility pricing model. This unprecedented proliferation of the cloud service industry is justified by the simple and promising cloud computing services provision model: providers offer services to end users on services pool, end-users search for and subscribe to use the services they need, and a high-speed network connection is established between users and providers to formulate the model. In general, IoT can take advantage of the virtually unlimited capabilities and resources of the cloud paradigm to compensate for its technological constraints (i.e., storage, processing and communication) [5, 6]. A report published by the International Data Corporation (IDC) predicts that the installed service provider datacenter capacity consumed by IoT workloads will increase nearly by 750% between 2014 and 2019 [16] and more than 90% of the data generated by IoT devices will be hosted on the cloud. The report also suggests that IoT will become the top driver of IT expansion in larger datacenters, speeding the transition to cloud-oriented infrastructure and data platform architectures.

Given that IoT can make use of cloud's unlimited resources, it can also take advantage of the cloud computing services provision model by offering IoT services to end users. The Service-oriented Architecture appears to be a promising solution and can be mapped to IoT resources in such a way by which each smart device should offer its functionalities as a standard service with a common communication interface. All involved services are classified as input/output components to become a "*thing*". The primary economic goal is to make these computational services available for users needs any time, based on a pay-as-you-go pricing model. Pay-per-use will encourage consumers to start relying heavily on IoT applications which will further allow them to easily and dynamically scale their services within the scope of their requirements [17]. In this regard, service providers offer a pool of IoT resources wrapped as web services and which can be composed by a broker to provide a single virtualized service to consumers. In an environment, where one provider is not enough to fulfil user demand, the service composition is a crucial. Essentially, a user request will traverse the route to the service provider, and get a reply traversed back from the provider to the user. This scenario increases in complexity as the number of required providers increases, to perform a user task that one service provider cannot process alone due to resources limitations, or when only a sub-part of the requested service is available. This is typically manifested in services composition and broker-based services models [17] that necessitate collaboration among a number of IoT services providers. This formulates what so-called Multi-clouds IoT Environment, whether explicitly or implicitly, to yield the service outcomes and

the end results to the user.

Generally, users focus on the speed of the service, response time and service cost with little or no consideration for the service location and number of composite services needed to meet the requirements or energy efficiency. The rapidly growing levels of energy consumption and carbon emission associated with using the cloud computing has become a key environmental concern. The increasing density in IoT applications users, providers, and datacenters will automatically lead to significant increases in network traffic and the associated energy consumed by the huge infrastructure (e.g., extra servers, switches) required to respond quickly and effectively to users requests. This power consumption issue is particularly significant when transferring data into a datacenter located somewhere in the world relatively far from the user geographical location; for example where the user is based in the UK and the datacenter is in Hong Kong. In addition, given the large number of service providers offering things and the huge amount of data generated, a higher bandwidth and network speed is required to cope with the cloud network traffic across the globe and to speed up data transformation process [18]. This results in an significant energy usage. Thus, a key challenge is to integrate fewer services from cross-continental and scattered providers in order to fulfil user demands, which means less data exchange and in return reduces the carbon footprint in the multi-cloud IoT environment. The movement of mass data between services influences the whole application process in terms of energy consumption. This paper presents and evaluates a novel Energy-aware multi-Cloud IoT service composition algorithm that generates energy efficient composition plans by integrating the least possible number of services from service providers that are scattered globally. Thus, this work aims at addressing a key and emerging IoT issue that is of adopting energy efficient approaches for cloud-based services and applications.

The rest of the paper is organised as follows: the next section defines the problem to be tackled. In section 3 we provide the main aim and objectives. The related works have been summarised in section 4. A detailed discussion of the proposed model and how it works is presented in section 5. The evaluation of the proposed model is detailed in section 6. Finally, the paper concludes the results and proposes future work in section 7.

2. Problem definition

In a traditional multi-cloud environment scenario, a user submits a request to a service broker stating the specifications of the required services. The broker should then find the appropriate service(s) and a service provider or set of providers that satisfy the request. Currently, locating the best-fit service that matches the

user needs, broker aims, and the environmental target presents the main challenging task for the multi-cloud IoT brokers for the following two reasons:

1. how can the broker compose multiple services to serve the request, when there is no one service that can match the request.
2. how can the broker compose the least possible number of services from a minimum number of providers to ensure that the energy efficiency target is met.

To further illustrate the above issues, consider the example of having the following five separate web services with the WSDL code along with a potential composition shown in Fig. 1 and Fig. 2 respectively:

- Given the street address/name as input, `geoCoding` type service, returns the associated geographical coordinates. This service denoted to in this paper as service `a`.
- Given the geographical coordinates, `pointOfInterest` type service returns the places that end users might be interested in. This service denoted to in this paper as service `b`.
- Given the geographical coordinates, `weatherForecast` returns the information about the weather observations at the station closest to the end user. This service denoted to in this paper as service `c`.
- Given the geographical coordinates, `map` type service returns a map showing the position of the end user. This service denoted to in this paper as service `d`.
- a `webPageInfoCollector` type service takes a set of information related to a location as input and returns a web page that shows it. This service denoted to in this paper as service `e`.

Next, consider the following requests received by the broker:

- Request 1: find the Geographical Coordinates of Byrom Street in Liverpool.
- Request 2: find the weather forecast near Big Ben in central London and show the directions from the current location to Big Ben on the map

The first request can be fulfilled simply by invoking service `a`, by passing the street name to get the geographical coordinates. However, the second request requires services composition since none of the five services can alone satisfy that request. Therefore, a new service called Information Service (IS), which provides a location-based information service (e.g., current weather or a map highlighting various points-of-interests (POI) based on the current GPS coordinates of a mobile device or after entering an address),

Service a

```
<xsd:element name='geoCoding_Request'>
   <xsd:element name='streetName' type="'xsd:string"/>
<xsd:element/>
<xsd:element name='geoCoding_Response'>
   <xsd:element name='geoCoordinates' type="'xsd:string"/>
<xsd:element/>
```

Service b

```
<xsd:element name='pointOfInterest_Request'>
   <xsd:element name='geoCoordinates' type="'xsd:string"/>
   <xsd:element name='typeOfInterest' type="'xsd:string"/>
<xsd:element/>
<xsd:element name='pointOfInterest_Response'>
   <xsd:element name='namesOfPoints' type="'xsd:string"/>
   <xsd:element name='directions' type="'xsd:string"/>
<xsd:element/>
```

Service c

```
<xsd:element name='weatherForecast_Request'>
   <xsd:element name='geoCoordinates' type="'xsd:string"/>
<xsd:element/>
<xsd:element name='weatherForecast_Response'>
   <xsd:element name='temprature' type="'xsd:string"/>
   <xsd:element name='windSpeed' type="'xsd:string"/>
   <xsd:element name='windDirection' type="'xsd:string"/>
<xsd:element/>
```

Service d

```
<xsd:element name='map_Request'>
   <xsd:element name='geoCoordinates' type="'xsd:string"/>
<xsd:element/>
<xsd:element name='map_Response'>
   <xsd:element name='position' type="'xsd:string"/>
<xsd:element/>
```

Service e

```
<xsd:element name='webPageInfoCollector_Request'>
   <xsd:element name='location' type="'xsd:string"/>
<xsd:element/>
<xsd:element name='webPageInfoCollector_Response'>
   <xsd:element name='webPage' type="'xsd:string"/>
<xsd:element/>
```

Figure 1: Five separate web services

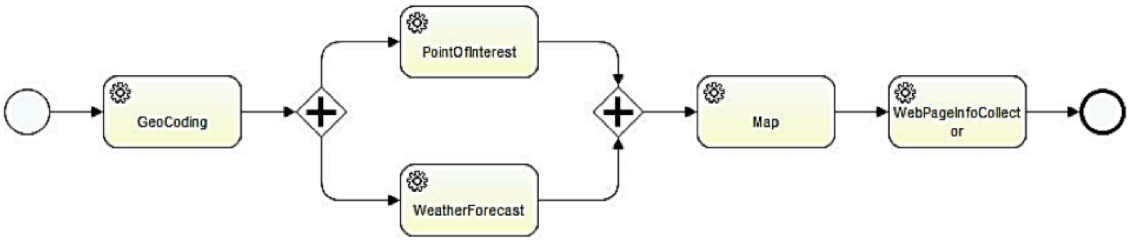

Figure 2: BPMN composition [19]

will be created. Each service in the IS composition is bound to a real remote Web Service running in the background and registered with a Marketplace. After providing the street address of the user as input the composite service returns a Web page with some information related to the users location. Hence, the second request can be achieved in one of the following two ways:

(i) invoking the `pointOfInterest` service, pass the geoCoordinates and the pointOfInterest as an input, to get Big Ben and directions to it; then invoke the `weatherForecast` to get the temperature and the other weather elements.

(ii) Compose the `weatherForecast` and `map` web services to fulfil the same request.

2.1. Problem statement

The aforementioned example illustrates the composition problem in a simple way so-called "web services composition". However, as the number of service providers and services increases, the composition of many services from different providers becomes more complicated in a real multi-cloud environment, and requires a massive amount of data interchange among all service participants, which consequently leads to high levels of energy consumption.

As stated above, typically the cloud user submits a job, in the form of a set of a functional and non-functional requirements, to a cloud service provider (datacenter) via a web interface of a cloud services broker. The broker, in turn, is responsible for finding the best-fit service to the user request based on the requirements submitted to satisfy the user Service Level Agreement (SLA). There have already been vast amount of research works in the area of cloud service discovery and composition, along with the resulting techniques and tools that are powerful enough for cloud service consumers to rely on to discover and use the services such as [20–23]. However, these research efforts are limited in the following ways:

(i) they assume that the requested services for composition are all contained within a single cloud;

(ii) they do not compare the energy required to compute and execute the similar services that can fulfil the user request, to choose the most energy efficient one.

(iii) they do not consider the number of services involved in the composition, which has a direct impact on the energy consumption.

The brokers and service providers tend to care about QoS metrics, such as service security [24, 25], availability, response time, as these factors attract clients. The communication cost, and sending and receiving data among the composite web services from different cloud providers would be expensive, time and energy

consuming. Hence, what continues to be a challenging and an under-investigated issue is finding a composition plan with the least possible number of composite services that fulfils the user request from minimum number of cloud service providers. This implies arriving at the most energy efficient service composition plan, which becomes more complicated as the number of the cloud service providers increase.

3. Aim of the proposed algorithm

Generally, there are three main pillars for energy consumption in multi-cloud computing environment that should be dealt with equally to achieve the full green cloud computing model:

(i) the amount of energy consumed by the datacenter equipments (e.g., computer air conditioning unit (CRAC)),

(ii) the amount of energy consumed on transporting the data between the user and the cloud datacenter.

(iii) the overall amount of energy required by the appropriate composite services.

The current state-of-the-art solutions focus primarily on points (i and ii) above, as is detailed in the section 4, whereas the primary aim of this paper is to propose and evaluate a high-end energy efficient service composition algorithm to address the overall amount of energy required by the appropriate composite services.

The new algorithm acts as an intermediary bridge between the user and the subscribed datacenters. It creates an energy efficient composition plan that contains the least possible number of services, from minimum possible number of cloud services' providers in a multi-cloud environment; while making sure the user's needs are met. To accomplish this aim, we first formalise the user-broker model to illustrate how a user submits a service request to the broker, and what the user request is composed of. This will then be followed by formalising the datacenter-broker communication model, which shows the kind of data that each datacenter is required to send to the broker to facilitate how the broker/algorithm finds the best-fit energy efficient composition at a later step. A formalised optimal service composition plan algorithm is presented thereafter, along with its experimentation and evaluation.

4. Related Work

Recent work have shown the potential to encapsulate IoT devices in well-defined APIs in order to provide an efficient and unified way on accessing and operating cloud based IoT applications [26–29]. The Web of

Things in terms of the convergence of cloud and IoT was established in [30], which proposed a set of methods to access smart devices through existing web based technologies such as web services and RESTful interfaces. It offered a solution to manage and use IoT resources in a service-oriented paradigm. The authors in [31] proposed a sensor-cloud infrastructure to virtualize and manage physical sensors on the cloud. The work in [32, 33] focused mainly on integrating IoT devices with enterprise applications, by taking advantage of the cloud and service-oriented paradigm. Nastic et al. [29] presented an approach that encapsulates fine-grained IoT resources and IoT capabilities in software-defined APIs in order to create IoT cloud system. In [34] authors addressed the problem of integrating sensor-based services with traditional IT systems by adopting the concepts of service orchestration and choreography. The orchestration process facilitates the integration of results obtained from several sensors or smart devices while the choreography is used to invoke several other services in order to extend the cooperation between virtual and physical worlds. TinySOA [35] is a service-oriented architecture that can be used for the development of Wireless Sensor Networks (WSN) applications and its integration with the Internet application in order to collect information from the remote sensors using APIs. In addition, it has mechanisms for WSN infrastructure registry and node discovery. Li et al. [26] proposed a framework that provides essential platform services for IoT solution providers to deliver and continuously extend their services. It enables IoT solution providers to efficiently deliver new resources by leveraging computing resources and platform services.

Automating service composition aims to overcome the problem where no single service can satisfy the functionality required by the user; it allows multiple services to be combined in to a large application in order to fulfil the request. A number of different approaches have been proposed such as, Matchmaking-based [36], Logic-based [37], Graph-Theory-based [38], Petri net-based [39], and AI-Planning based [40]. There are several other open source tools for service composition and execution, for example SWORD [41], ZenFlow [42], and Flow Editor [43]. SWORD offers a set of tools for building composite web services using rule-based plan generation. In SWORD, a service is modelled by a rule that expresses that for a given inputs, the service is capable of producing particular outputs. ZenFlow is a visual composition builder for web services written in BPEL4WS. It provides a number of visual tools to ease the definition of a business process such as multiple views of process, filtering, logical zooming capabilities and hierarchical representations. FlowEditor is a visual semantic web service composition tool based on OWL-S specification. It helps users solve business problems by creating and exporting new on-demand services. FlowEditor offers several advantages, such as visual service composition by drag-and-drop and visual control construct supports.

While there are number of approaches that can be employed for configuring, operating and composing

cloud-IoT resources, only a few approaches have addressed the optimization of the resulting compositions. Optimization of service composition aims at selecting appropriate services and service components to optimize the overall quality of the composition according to a set of predefined metrics, such as cost, performance, trust, energy efficiency and security [25, 44–48]. Energy efficiency has been receiving relatively less attention when it comes to the optimization of service composition. Several approaches exist in the literature, which focuses on the IoT devices energy conserving issues at the hardware level [49–52]. For example, sensor nodes can switch their radios off to save power when not in use and can wake up only when they are required to operate. These techniques are not useful when the IoT devices are exposed as software components and are deployed in the cloud environment for its users to access them. The work presented in [53] discusses the challenges and problems of measuring the power consumption of an individual web service. Park et al. [54] presented a SOA-based middleware to support quality-of-service control of mobile applications and to configure an energy-efficient service composition graph. Luo at el. [55] proposed a technique to select a composite service by using the path with the best QoS and lowest cost. The technique is based on the Dijkstras search path, which assumes that QoS attributes such as duration and throughput are additive. Elshaaf et al. [46] presented an approach to collaboratively infer the trustworthiness of shared component services of a service composition in distributed services environment. The service composition trustworthiness is based on consumers feedback on the reliability and their satisfaction with the services.

The construction of the optimal QoS aware service composition using the optimal set of service components often leads to inefficient compositions with redundant services or functionalities. The number of services involved in a composition has a direct impact on a number of QoA measures. For example, the work in [56, 57] suggest that by minimizing the number of services in a composition will help minimizing the total response time and maximize the throughput. Wang et al. [58] proposed a greedy algorithm to minimize the number of required service composition during a persistent querys life time such that the involved routing update cost and transmission cost is minimized. In [59, 60] the authors extended greedy algorithm by searching for and integrating the most energy efficient service composition route to the datacenter. According to [57], the main drawback of existing service composition approaches is their low performance such as, when the requirement of minimizing the number of services is not a priority, that is, when the selected number of services and the input/output interaction among them is high. There exist a number of approaches in the Literature [61–64], which addressed the optimization of the resulting compositions according to a set of QoS parameters. However, these approaches do not find optimal compositions in terms of number of services.

Overall, although the leverage of IoT services in the cloud service delivery model have been established,

research on energy efficient composition of IoT resources is an emerging IoT area that is still relatively under-explored. To the best of our knowledge, there exists no previous work, which focuses on the optimization of the resulting composition by taking into account the energy efficiency as the primary metric for IoT resources. Given the large number of cloud IoT resources available, we achieve energy efficiency by integrating least possible services to satisfy the user requirements.

5. Formal Model Design

In order to formulate the problem and the proposed solution in this paper, we need to identify the main parties of the multi-cloud environment as: users, a broker, and services providers. The next subsections formalise the interrelationship among those parties, and show how the new Energy Efficient multiple Cloud Computing service composition algorithm works (named E2C2) to identify and select the most energy efficient service composition plan.

Table 1: Summary of notations used

Notation	Meaning
s	a web Service
s^i	service input
s^o	service output
s^{ec}	energy of service computation
I	request interface
G	goal interface
S	a set of candidate ws
π_B	broker composition plan
π_{CP}	a cloud provider composition plan
π'_B	a optimal composition plan
MCP	multiple cloud providers
CP	a cloud provider
EC	energy consumption

5.1. Formal user-broker model

User-broker model is the actual service request model that users submit to a broker(s) to find the appropriate composition plan. A web service s is syntactically described by two sets of parameters as depicted

in Figure 1: $s^i = \{I_1, I_2, ...\}$ as input gained from the service request, and $s^o = \{O_1, O_2, ...\}$ as output for the service response; as such, the user should submit a service request in a 2-tuple format $\langle I, G\rangle$ where, I is the initial interface indicating the request such that $I \supseteq s^i$; and G is a goal interface, which indicates the ultimate response the user wants to get, such that $G \subseteq s^o$. This makes it simple to find a single web service that match the user needs, if and only if $I = s^i$ and $G = s^o$. However, in a complex user request where there is not a single web service that satisfies the request, a composition plan must be in place. Therefore, we will focus in this paper on building up a composition plan for multiple web services in a multi-cloud providers environment. Upon receiving the user request, the service composition will be defined by the broker using four-tuple model $\langle I, G, S, \pi_B\rangle$, where I and G already defined by the user as mentioned above, S is a set of candidate web services that will be identified by the broker to match the outcome based on G; and π_B is a composition plan that is a sequence of ordered web services such that $\pi_B \subseteq S$. By applying each service in π_B, the resulting interface is a superset of G.

5.2. Formal datacenter-broker model

In a multi-cloud environment, the requested services may come from different commercial cloud providers so they can be integrated and used together via mutual communication protocols to satisfy a complex service request. The Multiple Cloud service Providers (MCP) is a set of cloud providers, such that $MCP = \{CP_{i.m}, CP_{i+1.m}, ..., CP_{n.m}\}$. It can be seen that each CP in MCP is identified by 2 numbers i and m such that $(1 \leq i \leq n)$ represents a CP unique identification number; and m is the number of pre-defined/developed composition plan provided by each specific CP. For illustration purposes, the $CP_{3.6}$ denotes to the Cloud Provider 3 that provides 6 composition plans, in addition to the atomic services. Moreover, $\pi_j(CP_{n.m})$ denotes to the pre-defined composition plan (j) by $(CP_{n.m})$.

Since the energy required for service computation is a significant factor in our proposed algorithm, it is deemed essential that services' providers provide the broker with the energy consumption variable of each service to enable the broker to decide the most energy efficient one. As such, a service s is described by its provider in a 3-tuple format $\langle s^i, s^o, s^{ec}\rangle$, where s^{ec} is the energy required for the service computation at the hosting datacenter. In order to ensure that our aims (section 3) are met, the proposed algorithm makes the following assumptions:

1. The broker will create a composition plan, denoted to as (π_B) that includes services from either the same provider or from different providers.

2. Each service provider will essentially send the number of its pre-defined composition plans (m), the actual composition plans $\pi(CP)$, and a list of atomic services in the form of $\langle s^i, s^o, s^{ec} \rangle$.
3. The proposed algorithm lists the cloud providers in a descending order based on (m). Such that, the cloud provider that has the largest number of composition plans will be listed first. This procedure will help in creating a final composition plan containing the minimum possible number of services and service providers.
4. The broker starts examining the pre-defined composition plans of all providers first to try and find a one that matches the user request, as shown in (Algorithm 1).
5. If none is found to be matching the user request, then individual services, which might be a subset of a pre-defined composition plan, will be checked then as shown in (Algorithm 2).

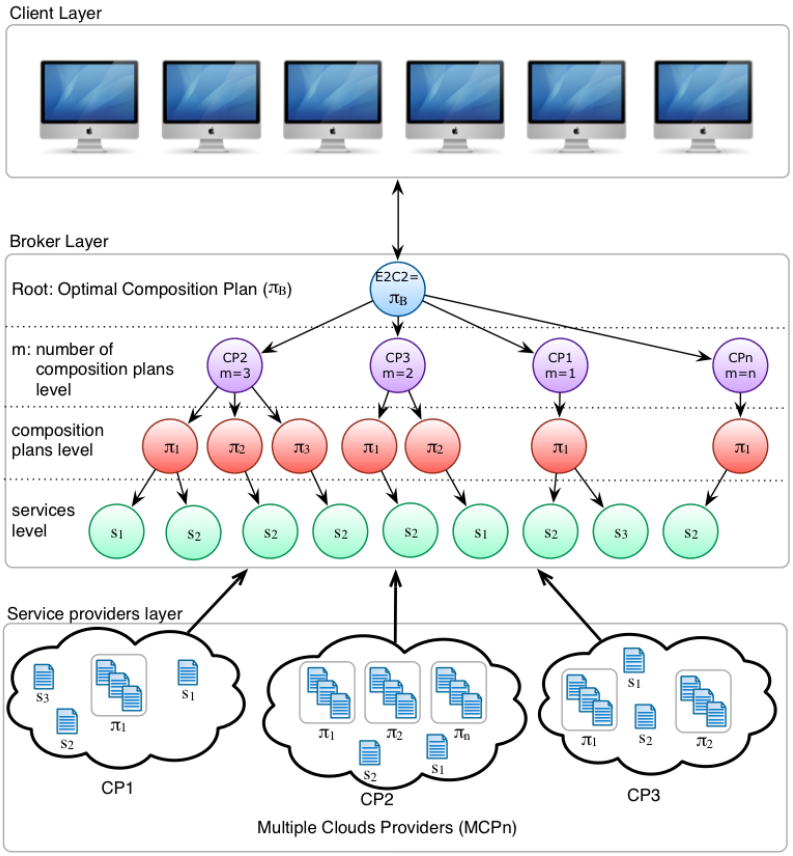

Figure 3: A conceptual representation of the proposed approach

A high-level conceptual representation of how the proposed algorithm works is shown in Fig. 3, in which

the root of Broker Layer (π_B) is the final, optimal, web service composition plan. If $\pi_B = \pi_j(CP_{n.m})$, then π_B is the optimal composition plan that contains services that all come from the same provider, which should also be the most energy efficient one amongst all available composition plans. Otherwise, a composition plan will be created by the broker as per the following section.

5.2.1. Optimal service composition plan

As stated above, S is a set of candidate web services that will initially be identified and named by the broker to satisfy the user needs. Therefore, S can be seen as $S = \{\langle s_i, CP_p\rangle, \langle s_j, CP_q\rangle, ..., \langle s_k, CP_r\rangle\}$ such that applying the sequence of web services results in an interface R, where $G \subseteq R$. The desired output o of each service s in S together will result in $(s_1^o \cup s_2^o \cup ... \cup s_k^o) = R \supseteq G$. By taking into account the minimum energy efficiency condition of the proposed algorithm, the broker optimal composition will be a sequence of: $\langle s', CP', EC(s')\rangle$, such that:

$$\pi'_B = \begin{cases} min\{\langle s'_i, CP'_p, EC(s'_i)\rangle, \langle s'_j, CP'_q, EC(s'_j)\rangle, ..., \langle s'_k, CP'_r, EC(s'_k)\rangle\} \subseteq S, & \textit{subject to (1) below:} \\ \pi_j(CP_{n.m}) \end{cases}$$

$$\text{minCons}|\sum_{i=1}^{k}\{EC(s'_i)\} \mid s' \in \pi'_B| \tag{1}$$

Thus, the optimal π'_B composition plan must be either a pre-defined composition plan by one of the subscribed cloud providers, or a set of atomic services from the same or different cloud providers that guarantee the minimum possible number of services involved with the least energy required to compute each service selected. Back to the example in Section 2, consider a multi-cloud environment where a broker deals with four cloud providers $\{CP_1, CP_2, CP_3, CP_4\}$. Each of these providers provides a set of atomic services, which is a subset of the services $\{a, b, c, d, e\}$, and a set of pre-defied composition plans $\pi(CP)$, as shown in Table 2 below.

Table 2: Multiple-cloud providers and services

Cloud providers	**CP**$_{4.4}$	**CP**$_{1.3}$	**CP**$_{2.1}$	**CP**$_{3.1}$
Atomic services	a, b, c, e	a, b, c	c, d, e	c, d
EC (kW)	0.52, 0.8, 0.721, 0.56	0.65, 0.5, 1.2	0.72, 0.32	1.2, 0.45
$\pi(CP)$	$\{a, e\}$, $\{b, c, e\}$,$\{c, e\}$, $\{b, e\}$	$\{a, b\}$, $\{a, c\}$, $\{b, c\}$	$\{d, e\}$	$\{c, d\}$

Upon receiving the user request (USR=$\langle I, G\rangle$), the broker starts examining the composition plans of the

four subscribed service providers. It starts first with the one that has the largest number of composition plans, which is $CP_{4.4}$ in Table 2 above. For instance, if the user asks for services b, c, e, then the providers will be checked in this order: $CP_{4.4}$, $CP_{1.3}$, $CP_{2.1}$, $CP_{3.1}$.

Hence, given that $CP_{4.4}$ has the requested composition plan already defined, then this satisfies the minimum possible number of providers involved in the composition. The broker will continue checking the other compositions of the other cloud providers to find out if there is any other pre-defined composition plan that satisfies USR, with a lower energy consumption. The proposed E2C2 algorithm structures the multi-cloud environment, the services and the available pre-defined composition plans as a four-level tree format, as shown in Fig. 3, such that:

(i) Level 1: the Root, which is the optimal service composition plan by E2C2.

(ii) Level 2: lists the *number* of the pre-defined composition plans by each subscribed CP.

(iii) Level 3: lists the actual services composition plan(s) by each subscribed provider, along with the total energy consumed by each composition plan.

(iv) Level 4: is the atomic web services, in each composition plan (e.g., a, b, etc.)

In this case, E2C2 algorithm starts (USR=$\langle I, G \rangle$) in line 2 of Algorithm 1. The algorithm then selects (in line 3) and checks (in line 6-24) the first CP that contains the largest number of (m) pre-defined composition plans. This kind of sorting helps to quickly identify the cloud provider with the greater number of composition plans, which can possibly have a composition plan that fulfils the received request, as demonstrated in (line 14-22). However, it should be noted that clouds are not necessarily organised in such an order in real-life scenarios. When all clouds are listed and arranged, all composition plans are then examined. If a composition plan is found, then the energy required by that plan will be stored in the ($minCons$) buffer. The value stored inside *minCons* will be used every time a matching composition plan is found, to compare the consumption of the new composition plan with the one saved in the *minCons*. Alternatively, if no composition plan is identified from the first provider, the next provider will be checked (as shown in line 35), until an appropriate cloud provider is obtained.

It can also happen that the identified predefined composition plan becomes dynamic at runtime (e.g., a failed or unavailable service(s) needs hot-plugging); in such case Algorithm 1 recommences the work to find an alternative predefined composition (line 35), otherwise Algorithm 2 starts in line 40 to create an atomic service based dynamic composite plan.

input : user service request USR, multiple cloud providers MCP, largest number of composition plan m
output : Optimal Composition Plan π'_B
assumption: Cloud providers are sorted in decreasing order based on the number of composition plans

```
1  beginalgorithmic[1] USR ← Ø, π'_B ← NULL, minCons ← NULL, m ← largest number of
   composition plan;                                                    ▷ Initialise
2  Get USR⟨I, G⟩
3  Select (CP_m)                          ▷ CP that contains the largest number of composition plans m
4  i ← m
5  if (i is True) then
6      foreach j ← 1 to j ≤ i step 1 do
7          if (π_j(CP_i) ∩ USR) == Ø then
8              if (j = i) then
9                  go to 35
10             else
11                 go to 6
12             end
13         else
14             if (j = 1) then
15                 minCons ← EC(π_j(CP_i))
16             else
17                 if (EC(π_j(CP_i)) < minCons) then
18                     minCons ← EC(π_j(CP_i))
19                 else
20                     go to 6
21                 end
22             end
23         end
24     end
25     return minCons
26     if (i = m) then
27         π'_B ← minCons
28     else
29         if (minCons < π'_B) then
30             π'_B ← minCons
31         end
32     end
33     return π'_B                                          ▷ Optimal composition plan
34 end
35 i=i-1
36 if (i ≥ 1)then
37     Select(CP_i)
38     go to 5                  ▷ So that other CPs will be checked in a decreasing order
39 else
40     Invoke Algorithm 2
41 end
42
```

Algorithm 1: Step 1: Finding a predefined optimal composition plan π'_B from a single provider

The alternative scenario our algorithm handles is where there is not a single pre-defined composition plan that can satisfy the user need, Algorithm 2 starts checking the services in the available composition plans in case that any of them can match a subset of the user's requested services defined in USR, as shown in line 6 of Algorithm 2. In addition, it checks all atomic services from available providers and combines the ones that fulfil the user request, as demonstrated in line 16. It also counts the number of providers collaborated in the final composition (line 15), and calculates the total energy consumption of the final composition, as shown in line 14.

```
1  Get USR⟨I, G⟩
2  Select (CP_m)                          ▷ that contains the largest number of composition plans m
3  i ← m
4  if (i is True) then
5      foreach j ← 1 to j ≤ i step 1 do
6          if (π_j(CP_i) ⋂ USR) ⊂ USR then
7              if (i = m) then
8                  eneCons ← EC(π_j(CP_i))
9              else
10                 if (EC(π_j(CP_i)) < eneCons) then
11                     eneCons ← EC(π_j(CP_i))
12                 end
13             end
14             minCons = minCons + eneCons
15             proList = proList + CP_i
16             serList=serList ⋃((π_j(CP_i) ⋂ USR))
17         else
18             if (π_j(CP_i) ⋂ USR) == Ø then
19                 if (j = i) then
20                     go to 27
21                 else
22                     go to 5
23                 end
24             end
25         end
26     end
27 else
28     i=i-1
29     if (i ≥ 1) then
30         Select(CP_i)
31         go to 4
32     else
33         exit
34     end
35 end
```

Algorithm 2: Step 2: Creating a dynamic optimal composition plan π'_B from multiple providers

The IoT service registry describes and publishes the offered functionality of the IoT resources, which are wrapped up as web services, to potential consumers. A service broker can use our proposed algorithm to create an optimal composition plan by searching and locating the most energy efficient services from the service registry. Service discovery is not only based on matching user functional requirements but also takes into account the energy efficiency. A cloud carrier [15] ensure seamless provisioning by providing connectivity among cloud entities. In our case, Internet is the major way to access and use cloud services. The role of a cloud auditor [15] can be incorporated in our proposed framework to conduct a design time verification of the generated energy efficient composition and assess its adaptability towards runtime changes. At this stage, the role of the cloud auditor is considered beyond the scope of this paper.

6. Evaluation

6.1. Experimental Settings

In order to evaluate the performance and efficiency of the proposed algorithm, it is important to compare the results against well established benchmark algorithms. Four different algorithms for selecting the cloud services combination were adopted for our comparative evaluation purposes: (All Clouds, Base Cloud, Smart Cloud [65], and COM2 [66]). The first algorithm, All Clouds, considers all clouds as inputs for the composition and determines all possible solutions. The algorithm helps to locate a service composition sequence with a minimal execution time, but does not to minimize the number of clouds in the final composition. The Base Cloud algorithm recursively enumerates all cloud combination possibilities in increasing order until an optimal solution is identified. It begins by analyzing all singleton sets of clouds and stop searching if the required combination can be found utilizing a single cloud, otherwise, it extends its search to cloud sets of size two, then three until the required combination is found. It generates an optimal composition solution with a small number of clouds. The Smart Cloud algorithm is designed to locate a near optimal composition plan based on an approximation algorithm. It considers a multiple cloud environment as a tree and then identifies a minimum demand set from searching the tree. The Smart Cloud locates a sub-optimal solution at a reduced cost while using a reduced cloud set. Heba kurdi et al [66] proposed a novel combinatorial optimization algorithm that considers multiple clouds and performs service composition with a short execution time and a minimal number of clouds, thereby reducing communication costs.

In E2C2 simulation we use identical simulation parameters of the aforementioned algorithms in order to perform a comparative evaluation. The experimental data were based on the default web service test-set

provided in the OWL-S XPlan package [67, 68], along with high-level semantic and syntactic declarative description of properties of the services [69] in terms of $\langle I, G \rangle$ to determine whether desired services are composable. The experiments were conducted on an Apple iMac (Retina 5K display, 3.2 GHz Intel Core i5, and 8 GB 1867 MHz DDR3). We have used NetBeans 8.1 as the prototype development platform and the simulation running environment, and Java EE 8 as the programming language to implement the proposed algorithm.

The broker in our simulation deals with four cloud providers $\{CP_1, CP_2, CP_3, CP_4\}$. Each of these providers provides a set of pre-defined composition plans, which are subset of $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$, and are based on the Multi-cloud providers (MCP) environment as shown in Table 3.

Table 3: Cloud providers composition set per multiple-cloud providers environment

MCPs	**CP**$_1$	**CP**$_2$	**CP**$_3$	**CP**$_4$
MCP1	π_1, π_2, π_3	π_4, π_5	π_3, π_4	$\pi_1, \pi_2, \pi_3, \pi_5$
MCP2	π_1, π_2	π_3	π_2, π_5	π_1, π_4, π_5
MCP3	π_1, π_3, π_5	π_5	π_1, π_2	π_3, π_4
MCP4	π_2, π_3, π_5	π_3, π_4	π_1, π_2, π_3	π_4, π_5
MCP5	π_1, π_2	π_2, π_3	π_3	π_1, π_4, π_5

In addition, $\{2, 3, 8, 3, 3\}$ represents the number of web services involved in each of the aforementioned composition plan respectively, as shown in Table 4.

Table 4: Number of services per composition plans

Composition plan	π_1	π_2	π_3	π_4	π_5
Number of services	2	3	8	3	3

The cloud providers are listed in a descending order based on the total number of the pre-defined composition plans provided, which helps in arriving at a final composition plan with the least possible number of services and providers involved if there is not a single pre-defined composition plan that can satisfy the request. It is a noteworthy to say that this order will be different in each MCP, for the same provider, depending upon the number of composition plans as shown in Table 5.b. For example, CP_4 comes first in MCP1 as it has four composition plans, whereas it is last in MCP4 with only 2 composition plans.

Table 5: CPs and number of (π) composition plans per MCPs

(a) Before descending order of CPs

MCPs	**CP**$_1$	**CP**$_2$	**CP**$_3$	**CP**$_4$
MCP1	$CP_{1.3}$	$CP_{2.2}$	$CP_{3.2}$	$CP_{4.4}$
MCP2	$CP_{1.2}$	$CP_{2.1}$	$CP_{3.2}$	$CP_{4.3}$
MCP3	$CP_{1.3}$	$CP_{2.1}$	$CP_{3.2}$	$CP_{4.2}$
MCP4	$CP_{1.3}$	$CP_{2.2}$	$CP_{3.3}$	$CP_{4.2}$
MCP5	$CP_{1.2}$	$CP_{2.2}$	$CP_{3.1}$	$CP_{4.3}$

(b) After descending order based on number of π

MCPs	Sorting order of CPs			
MCP1	$CP_{4.4}$	$CP_{1.3}$	$CP_{2.2}$	$CP_{3.2}$
MCP2	$CP_{4.3}$	$CP_{1.2}$	$CP_{3.2}$	$CP_{2.1}$
MCP3	$CP_{1.3}$	$CP_{3.2}$	$CP_{4.2}$	$CP_{2.1}$
MCP4	$CP_{1.3}$	$CP_{3.3}$	$CP_{2.2}$	$CP_{4.2}$
MCP5	$CP_{4.3}$	$CP_{1.2}$	$CP_{2.2}$	$CP_{3.1}$

6.2. Experimental results and analysis

The results for the four benchmark algorithms, the All Clouds in Table 6.a, the Base Cloud in Table 6.b, the Smart Cloud in Table 6.c and COM2 in Table 6.d have all confirmed previously published results in [65, 66]. We list the results of the E2C2 performance in Table 6.e such that it can be compared against the aforementioned approaches. We adopt the same cloud simulation environment and simulation parameters used by previous work. The three performance measures we considered in the conducted experiments are:

(i) The number of cloud providers that are involved in the final composition $|CP|$.

(ii) The number of services checked before reaching into the final composition $|S|$.

(iii) The running time, measured in seconds, for running the algorithm until an appropriate composition is reached.

The results (Table 6) indicate that E2C2 algorithm successfully surpassed the other algorithms in maintaining a low number of examined services and composite clouds. This can also be directly related to the execution time which is much less than the execution time of the best of the four algorithms, which is COM2. The number of atomic services examined $|S|$ did not exceed 38. The best result, in terms of the difference in the number of examined services in E2C2 and the best algorithm in each of the five environments, have been achieved in MCP4 and MCP5. There was 6 less atomic services examined by E2C2 compared to All Clouds in MCP4, and the same between E2C2 and COM2 in MCP5. Fig. 4 shows a comparison of the % reduction in the number of examined atomic services (relative to the baseline case for each MCP) by E2C2, All Clouds, Base Cloud, Smart Cloud, and COM2. The baseline for each MCP refers to the case of the maximum number of atomic services examined for that particular MCP. This is then used as the reference point for calculating the percentage reduction. Fig. 5 shows a comparison of the average number of examined atomic services in E2C2 across all MCPs is 30 services, compared to the average number of examined services of all other algorithms.

Table 6: CPs and number of (π) composition plans per MCPs

(a) All Clouds Algorithm

Performance	CP involved	|CP|	|S|
MCP1	CP1 CP2 CP4	3	46
MCP2	CP1 CP2 CP3 CP4	4	27
MCP3	CP1 CP3 CP4	3	32
MCP4	CP1 CP2 CP3 CP4	4	44
MCP5	CP1 CP2 CP3 CP4	4	32
Total		18	181

(b) Based Cloud Algorithm

Performance	CP involved	|CP|	|S|
MCP1	CP1 CP2	2	65
MCP2	CP1 CP2 CP4	3	148
MCP3	CP3 CP4	2	128
MCP4	CP2 CP3	2	68
MCP5	CP2 CP4	2	112
Total		11	521

(c) Smart Cloud Algorithm

Performance	CP involved	|CP|	|S|
MCP1	CP1 CP3	2	70
MCP2	CP1 CP2 CP4	3	48
MCP3	CP3 CP4	2	48
MCP4	CP2 CP3	2	140
MCP5	CP1 CP2 CP4	3	56
Total		12	362

(d) COM2 Algorithm

Performance	CP involved	|CP|	|S|
MCP1	CP4 CP2	2	35
MCP2	CP4 CP2 CP3	3	45
MCP3	CP1 CP4 CP3	3	50
MCP4	CP1 CP3 CP2	3	49
MCP5	CP2 CP4	2	30
Total		13	209

(e) E2C2 Algorithm

Performance	CP involved	|CP|	|S|
MCP1	CP4 CP2	2	35
MCP2	CP4 CP1 CP2	3	26
MCP3	CP1 CP3 CP4	3	29
MCP4	CP1 CP3 CP2	3	38
MCP5	CP4 CP2	2	24
Total		12	152

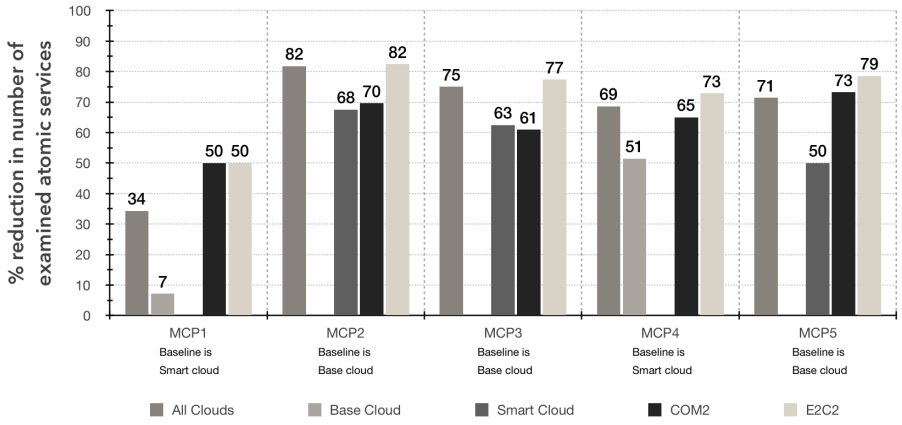

Figure 4: % reduction in the number of examined atomic services

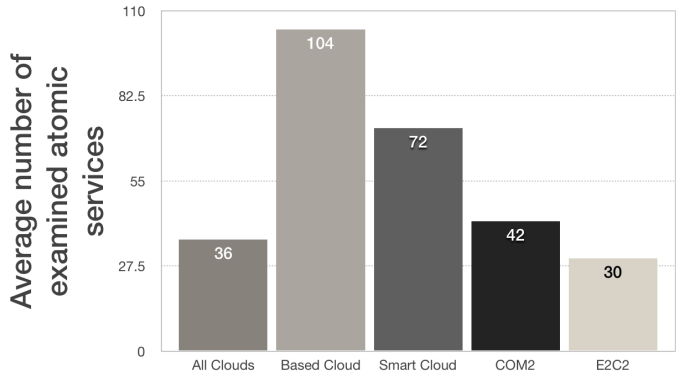

Figure 5: The average number of examined services

The number of combined clouds in the case of our E2C2 algorithm was as low as two clouds in some cases and never exceeded three in the worst, as shown in Fig. 6. Although Table 6.b shows that Base Cloud algorithm is the best in regards to the number of combined clouds, its weakness is associated with the high cost in execution time and number of examined services, which reaches more than 3 times the number of services in proposed E2C2 algorithm. The results are obtained from a pre-sorted list of cloud providers in a decreasing order based on the number of composition plans. Fig. 7 shows substantial performance improvement in E2C2 execution time compared to COM2, which is the best of the four examined algorithms, in each MCP. The results in Fig. 7 show that the execution time of E2C2 did not exceed 298 second in the worst scenario (MCP4), and 173 second in the best scenario.

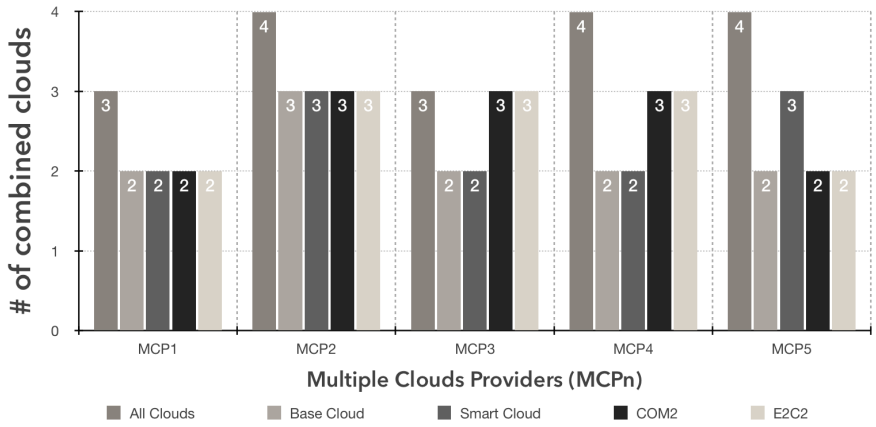

Figure 6: Number of combined clouds in multi-cloud based service composition plan π

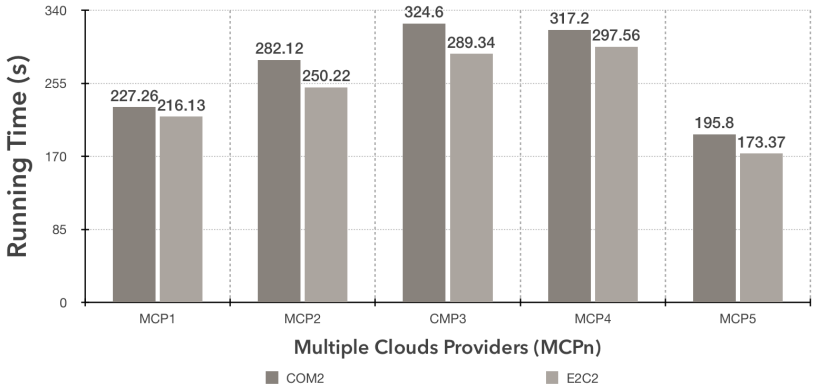

Figure 7: Running Time to achieve service composition plan π

7. Conclusions and Future Work

A novel multi-cloud IoT service composition algorithm, named E2C2, has been developed to emphasise energy awareness when searching for optimum composition plans to meet specified user requirements. Our algorithm searches for and integrates the minimum number of IoT services that satisfies the user request in an energy efficient manner. Evaluation of our algorithm was based on the systematic performance comparison with existing alternative algorithmic solutions, namely All Cloud, Base Cloud, Smart Cloud and COM2, using a benchmark example. The simulation results demonstrated a favourable performance of our algorithm in terms of achieving the least number of services searched to arrive at an optimum and energy efficient composition. Our work endeavours to contribute to the emerging IoT issue of energy efficient approaches, services and systems for cloud-based applications, by focusing on the energy implications of composite services.

Future work will explore modifications to the E2C2 algorithm for example by sorting the cloud providers based on the aggregate energy consumption of their corresponding services, and testing the algorithmic performance using more elaborate topologies. Other avenues for future research include evaluating the potential energy efficiency gains in various application domains such as mobile commerce, smart government and disaster recovery scenarios, and examining the potential benefit of heuristic optimisation search techniques for selecting service compositions.

References

[1] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, P. Doody, Internet of things strategic research roadmap, in: Cluster of European Research Projects on the Internet of Things, CERP-IoT, European Commission - Information Society and Media DG, 2011.

[2] S. Chen, S. Member, H. Xu, D. Liu, B. Hu, H. Wang, A vision of iot: Applications, challenges, and opportunities with china perspective, IEEE Internet of Things Journal 1 (4) (2014) 349–259.

[3] A. Whitmore, A. Agarwal, L. D. Xu, The internet of things—a survey of topics and trends, Information Systems Frontiers 17 (2) (2015) 261.

[4] L. D. Xu, W. He, S. Li, Internet of things in industries: A survey, IEEE Transactions on Industrial Informatics 10 (4) (2014) 2233–2243.

[5] A. Botta, W. de Donato, V. Persico, A. Pescapé, On the integration of cloud computing and internet of things, in: 2014 International Conference on Future Internet of Things and Cloud (FiCloud), IEEE, 2014, pp. 23–30.

[6] H. Yue, L. Guo, R. Li, H. Asaeda, Y. Fang, Dataclouds: Enabling community-based data-centric services over internet of things, IEEE Internet of Things Journal 1 (5) (2014) 472–482.

[7] K. Tei, L. Gurgen, Clout : Cloud of things for empowering the citizen clout in smart cities, in: 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 369–370.

[8] M. Victoria, F. Terroso-Sáenz, A. González-Vidal, M. Valdés-Vela, A. F. Skarmeta, M. A. Zamora, V. Chang, Applicability of big data techniques to smart cities deployments, IEEE Transactions on Industrial Informatics PP (99) (2016) 1–10.

[9] E. Cavalcante, J. Pereiraa, M. P. Alvesc, P. Maiaa, R. Mouraa, T. Batistaa, F. C. Delicatoc, P. F. Piresc, On the interplay of internet of things and cloud computing: A systematic mapping study, Computer Communications 89 (90) (2016) 17–33.

[10] J. A. Stankovic, Research directions for the internet of things, IEEE Internet of Things Journal 1 (1) (2014) 3–9.

[11] E. Borgia, The internet of things vision: Key features, applications and open issues, Computer Communications 54 (2014) 1–31.

[12] R. Romana, J. Zhoua, J. Lopezb, On the features and challenges of security and privacy in distributed internet of things, Computer Networks 57 (10) (2013) 2266–2279.

[13] M. Mital, V. Chang, P. Choudhary, A. Pani, Z. Sun, Adoption of cloud based internet of things in india: A multiple theory perspective., International Journal of Information Management.

[14] A. M. Alberti, E. S. dos Reis, R. da R. Righi, V. M. Munoz, V. Chang, Converging future internet, "things", and big data: A specification following novagenesis model, in: The first international conference on Internet of Things and Big Data,, 2016.

[15] P. Mell, T. Grance, The nist definition of cloud computing, National Institute of Standards and Technology (NIST).

[16] R. L. Villars, J. Cooke, C. MacGillivray, Impact of internet of things on datacenter demand and operations, Special Study 255397, IDC Analyze the Future (2015).

[17] B. Aldawsari, T. Baker, D. England, Trusted energy efficient cloud-based services brokerage platform, International Journal of Intelligent Computing Research (IJICR) 6 (3) (2016) 630–639.

[18] K. Rose, S. Eldridge, L. Chapin, The internet of things: An overview understanding the issues and challenges of a more connected world, Report report-InternetofThings-20151015-en, Internet Society (2015).

[19] Object management group, business process model and notation (bpmn) version 2.0 [online] (May 2016) [cited March 2017].

[20] S. Nair, S. Porwal, T. Dimitrakos, A. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, A. Khan, Towards secure cloud bursting, brokerage and aggregation, in: IEEE 8th European Conference on Web Services (ECOWS), IEEE, 2010, pp. 189–196.

[21] L. Zhang, F. Fowley, C. Pahl, A template description framework for services as a utility for cloud brokerage, in: 4th International Conference on Cloud Computing and Service Science, SCITEPRESS – Science and Technology Publications, 2014.

[22] K. Gouda, T. Radhika, M. Akshatha, Priority based resource allocation model for cloud computing, International Journal of Scientific Engineering and Technology (IJSET) 2 (1) (2013) 215–219.

[23] K. Dinesh, G. Poornima, K. Kiruthika, Efficient resources allocation for different jobs in cloud, International Journal of Computer Applications 56 (10) (2012) 30–35.

[24] V. Chang, Y.-H. Kuo, M. Ramachandrana, Cloud computing adoption framework: A security framework for business clouds, Future Generation Compter Systems 57 (2016) 24–41.

[25] V. Chang, M. Ramachandran, Towards achieving data security with the cloud computing adoption framework, IEEE Transactions on Services Computing 9 (1) (2015) 138–151.

[26] F. Li, M. Vogler, M. Claeßens, S. Dustdar, Efficient and scalable iot service delivery on cloud, in: IEEE Sixth International Conference on Cloud Computing (CLOUD), 2013, pp. 740–747.

[27] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, A. Rindos, Sdiot: a software defined based internet of things framework, Journal of Ambient Intelligence and Humanized Computing 6 (4) (2015) 453–461.

[28] D. Yazar, A. Dunkels, Efficient application integration in ip-based sensor networks, in: In Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, BuildSys '09, 2009, pp. 43–48.

[29] S. Nastic, S. Sehic, D.-H. Le, H.-L. Truong, S. Dustdar, Provisioning software-defined iot cloud systems, in: International conference on future internet of things and cloud (FiCloud), 2014, pp. 288–295.

[30] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, D. Savio, Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services, IEEE Transactions on Services Computing 3 (3) (2010) 223–235.

[31] M. Yuriyama, T. Kushida, Sensor-cloud infrastructure—physical sensor management with virtualized sensors on cloud computing, in: Proceedings of the 13th International Conference on Network-Based Information Systems (NBiS '10), Takayama, Japan, 2010, pp. 1–8.

[32] M. Kovatsch, M. Lanter, S. Duquennoy, Actinium: A restful runtime container for scriptable internet of things applications., in: Internet of Things, 2012, pp. 135–142.

[33] J. Soldatos, M. Serrano, M. Hauswirth, Convergence of utility computing with the internet-of-things, in: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), IEEE, 2012, pp. 874 – 879.

[34] K. Dar, A. Taherkordi, R. V. R. Rouvoy, F. Eliassen, Adaptable service composition for very-large-scale internet of things systems, in: Proceedings of the Workshop on Posters and Demos Track PDT'11, ACM, 2011, pp. 1–11.

[35] E. Avilés-López, J. García-Macías, Tinysoa: a service-oriented architecture for wireless sensor networks, Service Oriented Computing and Applications, Springer 3 (2) (2009) 99–108.

[36] O. Lassila, S. Dixit, Interleaving discovery and composition for simple workflows, in: Proc. Semantic Web Services, AAAI Spring Symp. Series, 2004, pp. 22–26.

[37] J. Rao, P. Kungas, M. Matskin, Composition of semantic web services using linear logic theorem proving, Information Systems 31 (4/5) (2006) 340–360.

[38] R. Zhang, I. Arpinar, B. Aleman-Meza, Automatic composition of semantic web services, in: Proceedings of International Connference of Web Services, 2003, pp. 38–41.

[39] J. Luo, J. Zhou, Z. Wu, An adaptive algorithm for qosaware service composition in grid environments, Service Oriented Computing and Applications 3 (3) (2009) 217–226.

[40] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, Automating daml-s web services composition using shop2, in: Proceedings International Semantic Web Conference, 2003, pp. 195–210.

[41] S. R. Ponnekanti, A. Fox., Sword: A developer toolkit for web service composition, in: In Proceedings of The Eleventh World Wide Conference, 2012.

[42] A. Martinez, M. Patino-Matinez, R. Jimenez-Peris, Zenflow: a visual web service composition tool for bpel4ws, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2005, pp. 181–188.

[43] B.Pi, G. Zou, C. Zhong, J. Zhang, H. Yu, A. Matsuo, Flow editor: Semantic web service composition tool, in: IEEE Ninth International conference on Service Computing (SCC), 2012, pp. 666–667.

[44] B. Yamini, D. Selvi, Cloud virtualization: A potential way to reduce global warming, in: Recent Advances in Space Technology Services and Climate Change (RSTSCC), 2010, pp. 55–57.

[45] L. F, M. N., Seeking quality of web service composition in a semantic dimension, IEEE Transactions on Knowledge and Data Engineering (TKDE) 23 (6) (2010) 942–959.

[46] H. Elshaaf, D. Botvich, Trustworthiness inference of multi-tenant component services in service compositions, Journal of Convergence 4 (1) (2013) 31–37.

[47] M. Alrifai, T. Risse, W. Nejdl, A hybrid approach for efficient web service composition with end-to-end qos constraints, ACM Transactions on the Web (TWEB) 6 (2) (2012) 111–130.

[48] E. Constante, F. Paci, N. Zannone, Privacy-aware web service composition and ranking, in: 2013 IEEE 20th International Conference on Web Services (ICWS),, 2013, pp. 131–138.

[49] H. Jun, Y. Meng, X. Gong, Y. Liu, Q. Duan, A novel deployment scheme for green internet of things., IEEE Internet of Things Journal 1 (2) (2014) 196–205.

[50] T. Mukesh, A framework for power saving in iot networks, in: In 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, pp. 369–375.

[51] A. Zeeshan, W. Yoon, A survey on energy conserving mechanisms for the internet of things: Wireless networking aspects, Sensors 15 (10) (2015) 24818–24847.

[52] J. Liang, J. Chen, H. Cheng, Y. Tseng, An energy-efficient sleep scheduling with qos consideration in 3gpp lte-advanced networks for internet of things, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 3 (1) (2013) 13–22.

[53] P. Bartalos, M. B. Blake, Engineering energy-aware web services toward dynamically-green computing, in: Proceedings of the Service-Oriented Computing-ICSOC 2011 Workshops, Springer, 2012, pp. 87–96.

[54] E. Park, H. Shin, Reconfigurable service composition and categorization for power-aware mobile computing, Parallel and Distributed Systems 19 (11) (2008) 1553–1564.

[55] Z. Luo, J. Zhou, Z. Wu, An adaptive algorithm for qosaware service composition in grid environments, Service Oriented Computing and Applications 3 (3) (2009) 217–226.

[56] P. Rodrguez-Mier, M. Mucientes, J. C. Vidal, M. Lama, An optimal and complete algorithm for automatic web service composition, International Journal of Web Service Research 9 (2) (2012) 1–20.

[57] P. Rodriguez-Mier, M. Mucientes, M. Lama, A dynamic qosaware semantic web service composition algorithm, in: in International Conference on Service-Oriented Computing, 2012.

[58] X. Wang, J. Wang, Z. Zheng, Y. Xu, M. Yang, Service composition in service-oriented wireless sensor networks with persistent queries, in: 2009 6th IEEE Consumer Communications and Networking Conference, IEEE, 2009, pp. 1–5.

[59] T. Baker, B. Al-Dawsari, H. Tawfik, Y. Ngoko, Greedi: An energy efficient routing algorithm for big data on cloud, Ad Hok Networks 35 (2015) 83–96.

[60] T. Baker, Y. Ngoko, R. Tolosana-Calasanz, O. F. Rana, M. Randles, Energy efficient cloud computing environment via meta-director framework, in: 6th IEEE international Conference on Developments in E-Systems Engineering (DeSE 2013), IEEE, 2013, pp. 130–136.

[61] Y. Yan, B. Xu, Z. Gu, S. Luo, A qos-driven approach for semantic service composition, in: IEEE International Conference on Commerce and Enterprise Computing, 2009, pp. 523–526.

[62] M. Aiello, E. E. Khoury, A. Lazovik, P. Ratelband, Optimal qos aware web service composition, in: IEEE International Conference on Commerce and Enterprise Computing, 2009, pp. 491–494.

[63] W. Jiang, S. Hu, Z. Huang, Z. Liu, Q. D. Handler, Two-phase graph search algorithm for qos-aware automatic service composition, in: International Conference on Service-Oriented Computing and Applications, 2010.

[64] P. J. Antonio, S. S. P. Fernandez, A. Ruiz-Cortés, Qos-aware web services composition using grasp with path relinking, Expert Systems with Applications 41 (9) (2014) 4211–4223.

[65] G. Zou, Y. Chen, Y. Xiang, R. Huang, Y. Xu, Ai planning and combinatorial optimization for web service composition in cloud computing, in: Annual International Conference on Cloud Computing and Virtualization (CCV 2010), 2010, p. 28.

[66] H. Kurdi, A. Al-Anazi, C. Campbell, A. A. Faries, A combinatorial optimization algorithm for multiple cloud service composition, Computers and Electrical Engineering 42 (C) (2015) 107–113.

[67] M. Klusch, , A. Gerber, Fast composition planning of owl-s services and application, in: In Proceedings of the 4th Eruopean Conference on Web Services, 2006, pp. 181–190.

[68] R. Karunamurthy, F. Khendek, R. H. Glitho, A novel architecture for web service composition, Journal of Network and Computer Applications 35 (2011) 787–802.

[69] H. Nacer, D. Aissani, Semantic web services: Standards, applications, challenges and solution, Journal of Network and Computer Applications 44 (2014) 134–151.