A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing

Minxian Xu, Adel Nadjaran Toosi, Member, IEEE, and Rajkumar Buyya, Fellow, IEEE

Abstract—The attractive features like elasticity, availability and pay-as-you-go pricing model contributed towards rapid adoption of cloud computing. However, the huge energy consumed by cloud data centers makes them to be one of the fastest growing sources of carbon emissions. Approaches for improving the energy efficiency include enhancing the resource utilization to reduce resource wastage and applying the renewable energy as the energy supply. This work aims to reduce the carbon footprint of the data centers by reducing the usage of brown energy and maximizing the usage of renewable energy. Taking advantage of microservices and renewable energy, we propose a self-adaptive approach for the resource management of interactive workloads and batch workloads. To ensure the QoS of workloads, a brownout-based algorithm for interactive workloads and a deferring algorithm for batch workloads are proposed. We have implemented the proposed approach in a prototype system and evaluated with real traces. The results illustrate our approach can reduce the brown energy usage by 21% and improve the renewable energy usage by 10%.

Index Terms—Cloud Data Centers, Renewable Energy Efficiency, QoS, Microservices, Brownout

1 INTRODUCTION

Today’s society and its organizations are becoming ever-increasingly dependent upon information and communication technologies (ICT) with software systems largely hosted on cloud data centers. Clouds offer an exciting benefit to enterprises by removing the need for building own Information Technology (IT) infrastructures and shifting the focus from the IT and infrastructure issues to core business competence. Apart from the infrastructure, elasticity, availability, and pay-as-you-go pricing model are among many other reasons which led to the rise of cloud computing. This massive growth in cloud solutions demanded the establishment of huge number of data centers around the world owned by enterprises and large cloud service providers such as Amazon, Microsoft, and Google to offer their services.

However, data centers hosting cloud services consume a large amount of electricity leading to high operational costs and high carbon footprint on the environment. ICT sector nowadays consumes approximately 7% of the global electricity, and it is predicted that the share will increase up to 13% by 2030 [11]. Among this, the operation of data centers accounts for one of the fastest growing sources of carbon dioxide emissions [2]. In 2013, U.S. data centers solely consumed an estimated 91 billion kWh of electricity (equivalent to the two-year power consumption of all households in New York City) and this is projected to reach 140 billion kWh by 2020 [3].

One of the main sources of energy inefficiencies in data centers is servers which are often utilized between 10 to 50% of their peak load [4]. This issue is amplified by the fact that server machines in data centers do not exhibit complete energy proportionality, that is, servers do not consume electricity in proportion to their load [5]. Even though, cloud providers use techniques such as dynamic consolidation of virtual machines (VMs) [6] to achieve energy savings and avoid underutilized servers, cloud users still waste energy by having many unused or under-utilized cloud resources. RightScale [4] states that the cloud consumers waste between 30-45% of their total cloud consumption [7].

In this regards, microservice architectures and technologies such as containers [8], that are steadily gaining adoption in industrial practice, provide a leap towards more efficient utilization of cloud resources. Containerization allows for higher resource utilization and reduction of cost by running multiple services on the same VM and providing a fine grain controlled on resources. In this paper, we take advantage of container technology to reduce the energy consumption of the system.

Apart from self-contained microservices, renewable energy is another solution gaining momentum to address energy consumption concerns (i.e., the carbon footprint) of cloud computing. In response to the climate change concerns and economic stimulus, many research initiatives have been launched to promote renewable energy use to power cloud data centers in recent years [9][10][11]. Many cloud providers also work on this goal by generating their own renewable energy or drawing power from a nearby renewable power plant. For example, in January 2018, AWS achieved 50% renewable energy usage by investing in clean energy activities including a commercial-scale wind farm in North Carolina [2].

1. https://www.rightscale.com/
Renewable energy systems are shown to be extremely effective in reducing dependence on finite fossil fuels and decreasing environmental impacts. However, powering data centers entirely or partially with renewable energy sources such as solar or wind is challenging as they are non-dispatchable and not always available due to their fluctuating nature. For example, photovoltaic (PV) solar energy is only available during daytime and the amount of power produced depends on the weather and geographical location of the data center. To be able to offer cloud services under such circumstances, cloud resource management systems need to support methods that allocate resources and schedule applications execution by preferring to finish them during the time when renewable energy is available while at the same time need to make sure that user QoS requirement are honored.

In this work, we address the research problem as: by predicting the amount of renewable energy, determining when to use brownout for interactive workloads and when to defer batch workloads, when to consolidate VMs to fewer hosts and scale hosts to maximize the usage of renewable energy while ensuring the QoS of workloads. This research problem has not been addressed by previous work, and the key contributions of the paper are:

- Providing a perspective model for multi-level adaptive resource scheduling to manage workloads and renewable energy;
- Proposing a self-adaptive approach for interactive workloads and batch workloads to ensure their QoS by considering the predicted renewable energy at Denver city based on support vector machine technique;
- Implementing a prototype system derived from the perspective model and the proposed approach on a small-scale testbed;
- Evaluating the performance of the self-adaptive approach in the proposed prototype system.

The rest of the paper is organized as follows: Section 2 discusses the related work for managing energy in the cloud computing environment. Section 3 depicts the system model of our proposed approach, followed by modeling and problem statement in Section 4. The scheduling algorithm with renewable energy is introduced in Section 5. Section 6 provides the detailed information about the implementation of our prototype system, and Section 7 shows the evaluation results of our proposed approach under our prototype system. Finally, conclusions along with the future directions are given in Section 8.

2 RELATED WORK

2.1 DVFS and VM consolidation

A large body of research on the energy efficiency of data centers has been dedicated to the optimization techniques to reduce the energy consumption of servers within a data center using technologies such as dynamic voltage and frequency scaling (DVFS) and VM consolidation [12][13]. Liu et al [14] proposed a heuristic algorithm for big data task scheduling based on thermal-aware and DVFS-enabled techniques to minimize the total energy consumption of data centers. Kim et al. [13] modeled real-time service as VM requests and proposed several DVFS algorithms to reduce energy consumption for the DVFS-enabled cluster. Cheng et al. [21] proposed a heterogeneity-aware task assignment approach to improve the overall energy consumption in a heterogeneous Hadoop cluster without sacrificing job performance. Teng et al. [15] presented a set of heuristic algorithms by taking advantage of DVFS and VM consolidation together for batch-oriented scenarios. Nguyen et al. [16] introduced a virtual machine consolidation algorithm with multiple usage prediction to improve the energy efficiency of cloud data centers. The limitation of DVFS and VM consolidation is that they cannot function well if the whole system is overloaded. Therefore, in this work, we take advantage of brownout-based [22] approach, which can dynamically activate/deactivate optional components in computing systems to handle the interactive workloads.

2.2 Brownout

Brownout is a self-adaptive approach to manage resources and applications in cloud computing systems. In our previous work [23], we proposed a survey and taxonomy on brownout-based approaches, which summarized the application of brownout in cloud computing systems for different optimization objectives. Xu et al. [17] presented a brownout-based approach to dynamically deactivated application components from the energy perspective. Tomas et al. [22] applied brownout to address the load balancing issues in clouds. Shahrad et al. [24] proposed a practical pricing model for brownout system and aims to increase the utilization of the cloud infrastructure by incentivizing users to dampen their usage fluctuations. Hasan et al. [18] investigated the green energy and user experience trade-off in interactive cloud applications and proposed a controller to provide guarantees of keeping response time within the SLA range in the presence of green energy based on a brownout-enabled architecture.

2.3 Holistic Management with Cooling

Due to the complexity of thermal modeling of data center operation, traditional approaches ignored the impacts of resource management techniques on the cooling power system of data centers. Recently, the holistic management of resources in which both computing and cooling energy are considered in the minimization of the overall consumption of energy has gained considerable attention from the community. Li et al. [19], for example, provided models of capturing thermal features of computer room air conditioning (CRAC) unit of the data center and accordingly propose a VM scheduling algorithm to reduce data center energy consumption while it maintains the SLA violation in an acceptable range. In their work, resource scheduling happens on VM level and the workload type is batch. Al-Qawasmeh et al. [25] presented power and thermal-aware workload allocation in the heterogeneous cloud. They developed optimization techniques to assign the performance states of CPU cores (P-states) at the data center level to optimize the power consumption while ensuring performance constraints. Tang et al. [26] investigated the thermal-aware task scheduling for homogeneous HPC data center,
which aims to minimize peak inlet temperature through task assignment, thus reducing the cooling power. However, the virtualized resources are not considered in their model. Compared with these works, we consider multiple layer resource scheduling and mixed types of workloads.

### 2.4 Renewable Energy

There are many studies in the literature that focused on the optimization of on-site renewable energy use in data centers. Goiri et al. [11] presented a prototype of a green data center powered with solar panels, a battery bank, and a grid-tie which they have built as a research platform. They also describe their method, called GreenSwitch, for dynamically scheduling the interactive and batch workload and selecting the source of energy to use. GreenSwitch aims to minimize the overall cost of electricity while respecting the characteristics of the workload and battery lifetime constraints. Similar to our work, they consider batch and interactive workloads in the data center. Their work focuses on the resource scheduling at the application level, while our work is a multi-layer scheduling approach that considers the application, VMs, and hosts. Liu et al. [20] also focused on shifting workloads and matching renewable energy supply and demand in the data center. They schedule non-critical IT workload and allocates IT resources within a data center according to the availability of renewable power supply and the efficiency of the cooling system. They formulate the problem as a constrained convex optimization and aim to minimize the overall cost within the data center. Different from the optimization of overall costs, we aim to optimize the energy perspective. Another difference is that we use two separate algorithms for interactive workloads and batch workloads, while [20] considers the workloads in an integrated manner. The current paper contributes to the growing body of work in this area.

Table 1 summarizes the comparison among the related work. The most similar works to us are [11] and [20], and we advance them by applying the brownout mechanism and multiple layer scheduling.

### 3 System Model

We propose a system model for adaptive resource scheduling as shown in Figure 1 based on the modified perspective model derived from our previous work [23]. The main difference lies in the type of workloads and the source of energy supply. We consider both interactive workloads and

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Comparison for related work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>Technique</td>
</tr>
<tr>
<td>Beloglazov et al. [6]</td>
<td>√</td>
</tr>
<tr>
<td>Dholakia et al. [7]</td>
<td>√</td>
</tr>
<tr>
<td>Liu et al. [8]</td>
<td>√</td>
</tr>
<tr>
<td>Nguyen et al. [9]</td>
<td>√</td>
</tr>
<tr>
<td>Sheikh et al. [10]</td>
<td>√</td>
</tr>
<tr>
<td>Hasan et al. [11]</td>
<td>√</td>
</tr>
<tr>
<td>Li et al. [12]</td>
<td>√</td>
</tr>
<tr>
<td>Beloglazov et al. [13]</td>
<td>√</td>
</tr>
<tr>
<td>Toor et al. [14]</td>
<td>√</td>
</tr>
<tr>
<td>Our Approach</td>
<td>√</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>Server (host) $i$</td>
</tr>
<tr>
<td>$P_{si}$</td>
<td>Power consumption of server $i$</td>
</tr>
<tr>
<td>$P_{is}$</td>
<td>Power when $i$ is idle</td>
</tr>
<tr>
<td>$P_{pro}$</td>
<td>Power when $i$ is fully loaded</td>
</tr>
<tr>
<td>$n_{vm}$</td>
<td>The number of VMs deployed on host $i$</td>
</tr>
<tr>
<td>$U_{t}^{s}$</td>
<td>The utilization of the $j$th VM on host $i$</td>
</tr>
<tr>
<td>$t$</td>
<td>The time interval</td>
</tr>
<tr>
<td>$T$</td>
<td>The scheduling period</td>
</tr>
<tr>
<td>$T_{sv}$</td>
<td>The brownout dimmer value at time interval $t$</td>
</tr>
<tr>
<td>$n_{ms}$</td>
<td>The number of microservices on VM $j$</td>
</tr>
<tr>
<td>$U_{j}$</td>
<td>The utilization of microservice $ms$</td>
</tr>
<tr>
<td>$CoP$</td>
<td>The function to calculate the cooling efficiency of cold air</td>
</tr>
<tr>
<td>$f_{sup}$</td>
<td>Cooling air supply temperature</td>
</tr>
<tr>
<td>$P_{cb}$</td>
<td>Cooling power for host $i$</td>
</tr>
<tr>
<td>$P_{j}$</td>
<td>Total power of host $i$</td>
</tr>
<tr>
<td>$P_{a}$</td>
<td>Total power of data center at time interval $t$</td>
</tr>
<tr>
<td>$d(t)$</td>
<td>Total workloads at time interval $t$</td>
</tr>
<tr>
<td>$a_{m}(t)$</td>
<td>The interactive workloads at time interval $t$ with size $m$</td>
</tr>
<tr>
<td>$b_{n}(t)$</td>
<td>The batch workloads at time interval $t$ with size $n$</td>
</tr>
<tr>
<td>$D$</td>
<td>The maximum resource capability in system</td>
</tr>
<tr>
<td>$S_{0}$</td>
<td>The start time of batch workload $b_{n}(t)$</td>
</tr>
<tr>
<td>$E_{n}$</td>
<td>The execution time of batch workload $b_{n}(t)$</td>
</tr>
<tr>
<td>$U_{n}$</td>
<td>The deadline of batch workload $b_{n}(t)$</td>
</tr>
<tr>
<td>$n_{sla}$</td>
<td>SLA violation ratio in the system</td>
</tr>
<tr>
<td>$n_{uni}$</td>
<td>The number of failed requests</td>
</tr>
<tr>
<td>$n_{req}$</td>
<td>All the number of requests coming into system</td>
</tr>
<tr>
<td>$K_{1}$</td>
<td>Available renewable energy at time interval $t$</td>
</tr>
<tr>
<td>$d(t)^{+}$</td>
<td>Power resulted from workloads on server</td>
</tr>
<tr>
<td>$c(d(t))$</td>
<td>Power resulted from cooling part reduction</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Allowed SLA violation ratio</td>
</tr>
<tr>
<td>$arg_{wa}(t)$</td>
<td>Average response time interactive workloads at time interval $t$</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Allowed average response time</td>
</tr>
<tr>
<td>$U_{h}$</td>
<td>Host utilization of host $i$ at time interval $t$</td>
</tr>
<tr>
<td>$T_{U_{h}}$</td>
<td>Threshold to determine overloaded hosts</td>
</tr>
<tr>
<td>$T_{U_{here}}$</td>
<td>Threshold to determine underutilized hosts</td>
</tr>
<tr>
<td>$U_{0}$</td>
<td>The start time of available renewable energy</td>
</tr>
<tr>
<td>$n_{av}$</td>
<td>The number of available renewable energy</td>
</tr>
<tr>
<td>$E_{out}$</td>
<td>The end time of available renewable energy</td>
</tr>
<tr>
<td>$n_{ov}$</td>
<td>The number of overloaded hosts at time interval $t$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>The percentage of utilization from batch workloads</td>
</tr>
<tr>
<td>$N_{h}$</td>
<td>The set of deactivated microservices on host $i$</td>
</tr>
<tr>
<td>$U_{(S_{m})}$</td>
<td>The utilization sum of deactivated microservices</td>
</tr>
<tr>
<td>$d(t)$</td>
<td>Predicted total workloads at time interval $t$</td>
</tr>
<tr>
<td>$a_{m}(t)$</td>
<td>Predicted interactive workloads at time interval $t$ with size $m$</td>
</tr>
<tr>
<td>$b_{n}(t)$</td>
<td>Predicted batch workloads at time interval $t$ with size $n$</td>
</tr>
<tr>
<td>$T_{d}$</td>
<td>Deferred time of batch workload</td>
</tr>
<tr>
<td>$td$</td>
<td>Start time of batch workload after deferred</td>
</tr>
<tr>
<td>$S_{d}$</td>
<td>Start time of deferred</td>
</tr>
<tr>
<td>$P_{d}$</td>
<td>Predicted renewable energy at $td$</td>
</tr>
<tr>
<td>$P_{d}$</td>
<td>Predicted power consumption at $td$</td>
</tr>
<tr>
<td>$n_{act}$</td>
<td>The number of active hosts</td>
</tr>
<tr>
<td>$n_{req}$</td>
<td>Number of requests when host is overloaded</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>The difference between required host and active hosts</td>
</tr>
</tbody>
</table>
batch workloads in the application layer and consider green and brown energy together in the energy supply layer.

In the users layers, users submit their service requests to the system. The users can define QoS constraints for the submitted requests, such as budget and deadline. The submitted requests are forwarded to the application layer. From the service providers’ viewpoint, these workloads generated by users are processed by the applications hosted in the cloud infrastructure.

We consider two types of applications: the interactive application (such as web application), and batch application. The interactive application should be executed as soon as possible to ensure the QoS. We consider the interactive application to support brownout, thus the microservices of the interactive applications can be identified as optional or mandatory. The optional ones can be deactivated to save resource usage, if deemed necessary. For the batch application, the workloads can be deferred for execution if their deadline is ensured.

Applications provided by service providers to offer services for users are managed by the application hosting engines, such as Docker [27] or Apache Tomcat. Applications can be deployed on either virtualized platform (virtual resources) or cloud infrastructure (physical resources). The host application engine can be container-based management platforms, e.g. Docker Swarm [28], Kubernetes [29] or Mesos [30], which provide management of container-based applications. The virtualized platform manages the virtualized resources, for example, the Virtual Machines managed by VMware [31]. As for the resource allocation and provisioning in cloud infrastructure, they can be managed by infrastructure management platform, e.g. OpenStack [32].

The bottom layer is the energy supply layer, which provides the mixed of energy sources for powering the system. The brown energy comes from a coal-based thermal power plant, which has high carbon footprint. The green energy comes from the renewable energy, such as solar power.

To support the resource provision, monitor and allocation in the system, a controller is required based on MAPE-K architecture model and fits into the feedback loop of the MAPE-K process, which has modules including Monitor, Analyze, Plan and Execute to achieve the adaptation process in cloud computing system. Sensors and Actuators are used to establish interactions with the system. Sensors gather the information from different levels in the system, including application hosting engine, virtualized platform, cloud infrastructure, and energy usage. The Sensors can be the devices attached to hardware, e.g. power meter. The collected information is provided to the Monitor module.

The Analyze module analyzes the received information from the Monitor module, and the Plan module makes decisions for applying scheduling policies, in which the scheduling policies are implemented. According to the decisions, the Execute module schedules resources via actuators on the application hosting engine and the virtualized platform to enable/disable optional microservices in interactive applications or defer the workloads of batch applications to be supplied by renewable energy. These operations can be fulfilled via the APIs provided by the application hosting engine or the virtualized platform.

The Knowledge pool in MAPE-K model is applied to store the predefined objectives (energy efficiency or SLA constraints) and trade-offs (e.g. trade-offs between energy and SLA). The rules in Knowledge pool, such as SLA rules, can be updated according to resource scheduling algorithms. The Knowledge pool also contains models like predicting the supplied amount of renewable energy, which can be used by scheduling algorithms.

In the following sections, we will introduce our proposed approach and the prototype system that is derived
from this perspective model.

4 PROBLEM MODELING

In this section, we will discuss the modeling and the optimization problem. Table 2 shows the symbols and their meanings utilized throughout this paper.

4.1 Power Consumption

4.1.1 Server Power Consumption

The server power model is derived from [33], which is based on the average CPU utilization. As we consider multilayer scheduling, the utilization of hosts, VM and microservices are modeled:

\[ P_i^s = \begin{cases} 
    P_{i,\text{idle}} + \theta_i \sum_j w_{i,j} \times P_{i,\text{dynamic}} & , w_i > 0 \\
    0 & , w_i = 0
\end{cases} \quad (1) \]

where \( P_i^s \) is the power consumption of the host \( i \) in the data center, which is composed of two parts: the idle power \( P_{i,\text{idle}} \) and dynamic power \( P_{i,\text{dynamic}} \). The dynamic power part is related to the VM utilization on the host. If there is no VM running on the host, it means the host can be switched into the low power mode and consume low energy. The \( w_i \) represents the number of VMs deployed on host \( i \). \( U_{i,j}^{vm} \) represents the utilization of \( j \)th VM on host \( i \). The \( \theta_i \) is the dimmer value of brownout at time interval \( t \).

The utilization of VM is the sum of microservices utilization running on the VM, which is modeled as:

\[ U_{i,j}^{vm} = \sum_k U_{i,j}^{ms,k} \quad (2) \]

where the \( ms \) is the id of microservice and \( A_j \) is the number of microservices. Since CPU computation is main power consumption component of servers, in our server model, we mainly focus on the power draw by the CPU utilization.

4.1.2 Cooling Power Consumption

For cooling power consumption, we use the model from HP lab data center [33] as follows:

\[ CoP(T_{sup}) = 0.00687T_{sup}^2 + 0.0008T_{sup} + 0.458 \quad (3) \]

The \( CoP \) in Equation (3) is a function to estimate the cooling efficiency of cold air supply temperature \( T_{sup} \) provided by cooling equipment, which is related to the target temperature that room is aimed to be maintained.

We consider the data center thermal control is managed by Computer Room Air Condition (CRAC) system. The system contains multiple CRAC units, which transfers cold air to the hosts to reduce hotspots. Based on server power consumption and cooling efficiency, we can calculate the power consumed \( P_i^c \) by cooling equipment for host \( i \) as:

\[ P_i^c = \frac{P_i}{CoP(T_{sup})} \quad (4) \]

The total power draw by the server part and the cooling part can be represented as:

\[ P_i = P_i^s + P_i^c \quad (5) \]

The total power of data center with \( n \) servers:

\[ P_t = \sum_{i=1}^{n} P_i \quad (6) \]

4.2 Workloads Model

In this work, we consider two types of workloads: (1) interactive workloads and (2) batch workloads. The interactive workloads are response time sensitive, thus these workloads should be executed immediately with the response time specified in the SLA, while the batch workloads can be deferred for execution as long as the deadline is satisfied. Based on the different characteristics of these workloads, the interactive workload at time \( t \) is denoted as \( a_m(t) \), and the batch workload is presented as \( b_n(t) \), with start time \( S_n \), execution time \( E_n \), and deadline \( D_n \). Therefore, the total workload at \( t \) is:

\[ d(t) = \sum_{m} a_m(t) + \sum_{n} b_n(t) \quad (7) \]

The value of \( d(t) \) should be \( 0 \leq d(t) \leq D \), in which \( D \) is the maximum resource capability of the system.

We use \( sla \) to denote the SLA violation that the system allows for interactive workloads, which represents the ratio of the requests fail to execute within predefined SLA, which is modeled as:

\[ sla = \frac{\text{num}_f}{\text{num}_a} \quad (8) \]

The \( \text{num}_a \) is the number of all the requests coming into the system, and \( \text{num}_f \) is the number of failed requests. To ensure the QoS, we have the constraint on all requests as \( sla \leq \alpha \). For the interactive requests the average response time \( avg_a(t) \) should be below the threshold defined by service provider \( avg_a(t) \leq \beta \).

4.3 Optimization Objectives

We assume the scheduling period as \( T \), and time interval we schedule resources is denoted as \( t \). We assume the available renewable energy at time \( t \) is \( R_t \). As the server power and cooling power is related to workloads, we use \( d(t)^+ \) to denote the power consumption resulted from the workload execution on servers, and \( c(d(t)) \) represents the cooling power resulted from workload, thus \( P_t = d(t)^+ + c(d(t)) \), our optimization objective is modeled as:

\[ \min \sum_t \left( \max(P_t - R_t, 0) \right) \]

\[ \text{s.t.} \ 0 \leq d(t) \leq D, \ \forall t \]

\[ 0 \leq \theta_t \leq 1, \ \forall t \]

\[ sla \leq \alpha, \forall t \]

\[ avg_a(t) \leq \beta, \forall t \]

where aims to minimize the usage of brown energy by maximizing the usage of renewable energy. Meanwhile, the constraints including maximum capacity \( D \) in the system, dimmer value \( \theta_t \), SLA \( sla \) and average response time \( avg_a(t) \) should be satisfied.

5 SCHEDULING WITH RENEWABLE ENERGY

5.1 Green-aware Scheduling Algorithm

To schedule the interactive and batch workloads in an energy efficient manner by considering renewable energy, we propose a Green-aware scheduling algorithm, which is shown in Algorithm 1. During the observation period \( T \),
Algorithm 1: Green-aware scheduling algorithm

Input: host utilization $U^t_i$, utilization thresholds $TU_{up}, TU_{low}$
Output: brownout energy usage $\sum \mu_i(t)$

$n^0_i = \sum_k (\max(P_i - R_k, 0))$

for $t \leftarrow 0$ to $T$
do
\hspace{1em} if $n^0_i > 0$ then
\hspace{2em} $S_i \leftarrow$ brownout algorithm for interactive workloads
\hspace{2em} $T^D \leftarrow$ deferring algorithm for batch workloads
\hspace{2em} $\min \leftarrow \sum_k (\max(P_i - R_k, 0))$
\hspace{1em} else if $U_{avg} < TU_{low}$ then
\hspace{2em} VM consolidation algorithm
\hspace{1em} host scaling algorithm
\hspace{1em} $\min \leftarrow n_a$
\hspace{1em} else
\hspace{2em} process interactive workloads in normal mode
\hspace{2em} process batch workloads in normal mode
\hspace{1em} end
\endfor

Algorithm 2: Brownout Algorithm for Interactive Workloads

Input: time interval $t$, the number of overloaded hosts $n^0_i$
Output: deactivated microservice $S_i$

for host $i$ in the host list do
\hspace{1em} if $t < \mu^t_i$ or $t > \mu^t_i$ then
\hspace{2em} $\theta_i = \sqrt{\frac{t}{\mu^t_i}}$
\hspace{2em} $U^t_i = \theta_i \times \mu^t_i$
\hspace{2em} if $U^t_i > TU_{up}$ then
\hspace{3em} finding deactivated microservices $S_i$ on host $i$
\hspace{3em} $\min \leftarrow \mu^t_i - U(S_i)$
\hspace{3em} deactivate the microservices
\hspace{2em} end
\hspace{1em} else if $R_t < P_i$ then
\hspace{2em} $\theta_i = \sqrt{\frac{t}{\mu^t_i}}$
\hspace{2em} $U^t_i = \theta_i \times \mu^t_i$
\hspace{2em} finding deactivated microservices $S_i$ on host $i$
\hspace{2em} $\min \leftarrow \mu^t_i - U(S_i)$
\hspace{2em} deactivate the microservices
\hspace{1em} end
\endfor

Algorithm 3: Deferring Algorithm for Batch Workloads

Input: batch workload $b_i(t)$ with start time $S_i$, execution time $E_i$, and deadline $D_i$
Output: deferred time $T^D$

for $t = S_i$ in $b_i(t)$ do
\hspace{1em} if $t > t^*$ then
\hspace{2em} execute $b_i(t)$
\hspace{1em} else
\hspace{2em} defer $T^D$ time for execution
\hspace{2em} $td = t + T^D$,
\hspace{2em} $\forall t_d \leq D_j - E_j$, $td > t^*_d$
\hspace{2em} $d(t_d) = \sum n_m a_m(t_d) + \sum n_b b_n(t_d)$
\hspace{2em} $R_{j,d} > P_{j,d}$
\hspace{2em} $S_j = td$
\hspace{2em} update $P_{j,d}'$
\hspace{2em} end
\hspace{1em} else if $t^*_d \leq td \leq t^*_{j,d}$ then
\hspace{2em} if $R_i > P_i$ then
\hspace{3em} execute $b_i(t)$
\hspace{3em} else
\hspace{4em} defer $T^D$ time for execution
\hspace{4em} $td = t + T^D$, $\forall t_d \leq D_j - E_j$
\hspace{4em} $d(t_d) = \sum n_m a_m(t_d) + \sum n_b b_n(t_d)$
\hspace{4em} $R_{j,d} > P_{j,d}$
\hspace{4em} $S_j = td$
\hspace{4em} update $P_{j,d}'$
\hspace{3em} end
\hspace{2em} execute $b_i(t)$
\hspace{1em} end
\endfor

at each time interval $t$, the algorithm will firstly identify the number of overloaded hosts (line 2). If the overloading situation exists, the algorithm will manage the interactive workloads and batch workloads with different algorithms: brownout algorithm for interactive workloads (Algorithm 2) and deferring algorithm for batch workloads (Algorithm 3) to minimize brown energy usage (lines 4-6). If the system is not overloaded and the average utilization is below the underutilized threshold (line 7), the algorithm will apply VM consolidation algorithm to pack VMs on fewer servers, thus the idle servers will be switched into the low power mode to save energy (lines 8-10). If the system is running at the normal status, then the workloads will be executed in the normal fashion.

5.2 Brownout Algorithm for Interactive Workloads

The pseudocode of the brownout algorithm for interactive workloads is shown in Algorithm 2. The algorithm schedules resources differently according to whether the renewable is available or not. 1) During the time when renewable energy is not available (line 2), the brownout is triggered, and the dimmer value is generated. The dimmer value $\theta_i$ is computed based on the severity of overloads in the system (line 3). With the dimmer value, the expected utilization reduction $U^t_i$ on host $i$ is computed (line 4). Then the algorithm selects a set of microservices $S_i$ to deactivate, thus the utilization is reduced. The difference between the expected utilization reduction $U^t_i$ and the sum of utilization of selected microservices $U(S_i)$ is minimized (lines 6-8). To minimize the difference, the microservice selection process is based on the LUCF algorithm [17], which sorts the microservices according to their utilization in a list, and finds the sublist which has the utilization that is closest to $U^t_i$. 2) When the renewable energy is available but less than the total required energy, the brownout is also triggered (line 11). The dimmer value is calculated based on renewable energy and required energy as noted in line 12. Then the rest steps are the same as in the first part of Algorithm 2, which finds the microservices and deactivate them. 3) When sufficient renewable energy is available, brownout will not be triggered.

5.3 Deferring Algorithm for Batch Workloads

Algorithm 3 shows pseudocode for processing the batch workloads. The batch workloads are executed when their start time $S_j$ is coming (line 1). The workloads are processed based on the time period that the workloads are in. For the workloads which have the start time before the renewable energy start time $t^*$, the objective is to defer their execution to the time when the renewable energy is available while ensuring their deadlines (lines 2-12). 1) If the deadline is
before \( t_d \), it means the workload cannot be deferred to be processed by renewable energy, so the workload can be executed at \( t \) (lines 3-4). If the workload can be deferred, the algorithm defers its time with \( T^D_j \), then the algorithm updates the workloads at time \( t_d \), which equals to \( t + T^D_j \). The deferred time \( T^D_j \) should satisfy the constraint, e.g. not failing the deadline, the renewable energy is enough at \( t_d \), and should not be deferred to after \( t_e \). If the constraints are satisfied, the algorithm updates the predicted power consumption at \( t_d \). When the start time of the workload is during the time when renewable energy is available and sufficient, the workload is executed; otherwise, the workload will be deferred. Similar to the first part of Algorithm 3, the deferred time \( t_d \) also needs to satisfy the constraints in Equation (9) but removing the constraint that workload is executed before \( t_e \). 3) When the time is after \( t_e \), it means the renewable energy is not available any more, therefore, the workloads are executed as soon as possible to comply with the deadlines.

5.4 Host scaling

Algorithm 4: Hosts scaling algorithm

\[
\text{Input} : \text{number of hosts} n \text{ in data center, number of active hosts} n_a, \text{number of requests when host is overloaded} \text{numthr}, \text{predicted number of requests num(t) at time} t. \\
\text{Output} : \text{number of active hosts} n_a \\
1. n_a \leftarrow \lceil \text{num}(t) + \text{numthr} \rceil \\
2. n' \leftarrow n_a - n \\
3. \text{if} \ n' > 0 \text{ then} \\
4. \quad \text{Add n' hosts} \\
5. \quad \text{while} \ P_t \leq P_{th} \text{ do} \\
6. \quad \quad \text{Add another host} \\
7. \quad \text{update} P_t \\
8. \end \\
9. \text{else if} \ n' < 0 \text{ then} \\
10. \quad \text{Remove |n'| hosts} \\
11. \text{else} \\
12. \quad \text{no host scaling} \\
13. \end \\
14. \text{return} n_a
\]

We use a modified host scaling algorithm from [35] by considering renewable energy as shown in Algorithm 4. With profiling data, we configure the thresholds of requests that leads to overloads, in which the average response time violates the predefined constraints. The algorithm calculates the difference \( n \) between the number of required servers and actual servers. 1) When more servers are needed, then it adds \( n \) servers into the system (lines 3-4). If the renewable energy is still enough, then it tries to scale more servers into the system to improve the QoS (lines 5-8). 2) If servers are already enough, then remove \( |n'| \) servers from system to reduce energy. 3) If \( n \) is 0, then it means no host scaling is required.

5.5 Renewable energy prediction

In this work, we focus on the solar energy as it is one of the most common sources of renewable energy. We use Support Vector Machine (SVM) to predict the solar irradiation or PV power output for the availability of renewable energy. SVM is a machine learning approach and has been applied to data analysis successfully and has showed that it can achieve a better learning capacity and smaller prediction error value than many other methods for various datasets. In the studies related to solar irradiation prediction, SVM has been used to forecast and train solar radiation model [36] [37].

Since you do not have the access to the hourly solar irradiance at Melbourne City, in this paper, we use the historical data from NREL Solar Radiation Research Laboratory. The solar panels of the Laboratory are located at Denver, Colorado, US (Latitude 39.742° North, Longitude 105.18° West), which has a similar weather to Melbourne instead. We use the hourly-based solar irradiance data from September 1 2018 to November 1 2018. SVM prediction approach has two phases: the training phase and testing phase. 80% data is used for the training phase, and 20% data is used for the testing phase. Once the process is finished, the test data and prediction results are compared to calculate the error rate. We use SVM R toolbox for our purpose.

The obtained results are shown in Figure 2. It shows that the SVM-based approach can achieve the values close to actual ones. In the testing phase, the coefficient of determination \( (R^2) \) is 0.763 and correlation coefficient \( (r) \) is 0.873. The selected parameters for svm are regularization parameter \( C = 4 \) and Kernel bandwidth \( \epsilon = 0 \). In the experiments section, we applied this trained model to predict solar irradiance. The solar irradiance can be easily calculated with the conversion efficiency of solar panels, e.g. 20%, we assume all solar irradiance will be available as power energy.

6 Prototype System Implementation

We configure our testbed [38] to develop a prototype system to evaluate the proposed approach. Figure 3 shows the implemented architecture of our prototype system. Cloud resource management platform and microservices management platform have been developed and widely used for years, thus, in this work, we design and implement our prototype based on these mature platforms.

Cloud IaaS resource management platform, OpenStack, is responsible for managing cloud resources, including CPU, memory, and bandwidth. The monitored data of resources is collected by status collector and can be used for resource provisioning and optimization. Microservice management

platform, Docker Swarm, is responsible for managing service images, monitoring service resource utilization and managing the service life cycles. Other Docker APIs can also be used to run operations on services.

Based on the two management platforms for cloud resources and services, SA (Self-Adaptive) controller is designed to manage and monitor both of them to achieve the multiple level resource scheduling. When requests are submitted to the system, like interactive workloads or batch workloads, the resource allocator in SA controller manages cloud resource management platform and service management platform simultaneously to accept and process requests by providing the requested amount of resources. Apart from allocating resources to requests, the resource allocator can also optimize resource utilization. For instance, brownout can be triggered to deactivate optional microservices to reduce resource utilization. The service provider can also configure the available resource scheduling policies for the energy efficiency purpose.

To provision and optimize the resources by means of resource allocator, the resource monitor needs to collect the resource usage at different levels, including services utilization, VMs utilization, and hosts utilization. To minimize the overheads of frequently monitored data collection, the collection time intervals should be well configured by the service provider. For instance, the brownout mechanism can be checked every five minutes as the brownout costs are not high, while the VM migration and host scaling operations can be executed with longer time intervals, e.g. one hour.

In the following subsections, we introduce the implementation of our prototype system in details.

6.1 Implementation

To implement our prototype system, we take advantage of the OpenStack cloud resource management platform and Docker Swarm service management platform. The system is implemented with Java, OpenStack, Docker Swarm, An-sible, Eaton Power Distribution Units (ePDU) API. Our prototype system uses these open source tools to provide a self-adaptive approach to optimize, manage and provision resources for different types of workloads.

OpenStack platform is used to manage hosts and VMs. The hosts in OpenStack are called compute nodes and are running with Nova Compute Node component to connect the hypervisor and OpenStack controller. VMs are managed by Nova API to create, migrate and remove VM instances. The Neutron OVS Agent and OpenVSwitch are providing services related to the network.

Docker Swarm Platform manages the service provided by service providers. The images of services are stored in the service repository component, which can fetch the images from remote to local. The services are managed by the service manager via Docker APIs, including creation and deletion. The status of services are monitored by a service monitor where we monitor service utilization and liveness.

Our prototype system is based on these services to manage the resources and services to handle the requests. Below, we introduce the details of the components in our prototype.

Resource Allocator: It interacts with OpenStack controller via OpenStack APIs and Docker Swarm Controller via Docker APIs. It manages the physical resources on compute nodes, and these physical resources can be used for creating and deploying VMs on the nodes. Resource Manager knows the amount of resource that is used or remaining on each compute node, like the number of cores, memory, and storage. When creating a VM instance, it can also specify the instance capacity (CPU, memory, operation system and etc.) as well as other information related to VMs, such as location, images of VMs and IP address. The virtual network in a compute node is also managed by Resource Manager that uses the Neutron component, which is deployed on each compute node.

Resource Monitor: It is used to monitor the running status of the whole system from different levels, including hosts, VMs and services. We use OpenStack Ceilometer and Gnocchi components to measure the data at the host and VM level. Ceilometer is responsible for monitoring the utilization of resources of VMs and then sends the collected data to Gnocchi to aggregate the data for all the hosts. We use Docker APIs to collect the resource utilization of services deployed on VMs. Apart from monitoring the resource utilization, we also use ePDU APIs to monitor the power consumption of hosts. With these monitored data, other components, like Power Estimator and Policy Manager can use these data to make decisions, which will be introduced later.

Application Scheduler: We design our main controls in the Application Scheduler component. When requests are submitted by users, the Application Scheduler decides which requests in the batch workloads should be deferred, which microservice should be temporarily deactivated by brownout mechanism, which VM should be migrated to
another host and which host should be switched to the low power mode. With the retrieved data from the Resource Monitor component, these decisions are made with the policies in the Policy Manager. After the decisions are made, Resource Provisioner exploits Resource Manager to allocate the resources to VMs, services, and requests.

**Power Consumption Estimator:** To achieve our objective of managing energy and support our scheduling policies, we have a power consumption estimator to predict the power consumption at a specific time period. For example, for the batch workloads, we proposed a deferring algorithm, thus we need to estimate the power consumption at the deferred time period to calibrate our algorithm. We use the workloads model shown in Equation (7) to estimate the workloads and then convert it to the total energy consumption based on the model in [20].

**Policy Manager:** It contains the implemented scheduling policies in our prototype, including Algorithms 1 to 4. The Policy Manager component uses the retrieved data from Resource Monitor, and makes decisions based on system status. For example, a VM is migrated from an underutilized host to other hosts, thus the idle host can be switched to the low power mode to save power consumption; when the renewable energy is not sufficient and the system is overloaded, to ensure the QoS of service, brownout can be triggered to relieve the overloaded situation. The customized workloads processing policy, VM migration policy and host scaling policy can also be implemented for the policy manager.

**ePDU API:** Eaton Power Distribution Units (ePDU) is an effective power management and monitoring device. It has outlets that electric devices can be connected to. It also provides the features to read the power consumption of hosts as well as turn on/off the outlets remotely. We implemented Python scripts based on ePDU APIs to read the power data at per second rate to support part of the functions in Resource Monitor. Our scripts can also operate the hosts remotely by turning on/off the power supply to hosts to support the decision in Policy Manager. For example, a host needs to be scaled out if the whole system is underutilized; or hosts should be scaled in to support more requests.

**Renewable Energy Predictor:** For supporting our renewable energy experiments, we implement a renewable energy predictor that predicts the renewable energy at Denver city based on the historical data. Our current renewable energy predictor is based on the support vector machine. As introduced in Section 5.5, our SVM models show that it can achieve a high accuracy. The data based on this component can also be incorporated into the scheduling policy.

## 7 Performance Evaluation

### 7.1 Environmental Settings

**Hardware:** We utilize a micro data center of Melbourne CLOUDS lab as testbed. Our data center is consist of 9 heterogeneous servers. Table 3 shows the capacity specification of the servers and their energy consumption information. To monitor the power consumption of individual machines, we use two ePDUs and all the servers are connected to them. Apart from the power monitor, the ePDUs also enable us to switch on/off power outlets connected with individual server remotely through the network. Figure 4 shows the servers and ePDUs of our testbed. The total maximum power of the IT equipment in our system is 1.27 kWh for 8 hosts (one IBM X3500 M4 machine is regarded as the OpenStack control node and is not considered).

We assume our system is equipped with 1.63 kWh PV panel, which has 30% more power than the hosts, as the cooling part normally consumes about 20% to 30% of server energy if the target temperature is 25 degree [9]. We consider to control the data center temperature as 25 degree, according to Equation (3), \( T_{sup} = 25 \), then we get \( CoP(T_{sup}) = 0.211 \). In the following experiments, we use this value to compute the power from the cooling equipment, e.g. if the hosts consume 10 kWh, then the cooling part is 2.11 kWh.

**Software:** The operating systems of the servers are CentOS Linux Distribution. We use OpenStack [32] to support our cloud platform and manage the VMs. One of our most powerful machines is selected as our controller node, and other nodes are acting in the same role. In VM instances, we deploy Docker [39] containers to provide services in the form of microservices and use Docker Swarm to manage the containers cluster. Some other required software, like Java, Ansible are also installed in the VMs.

### 7.2 Workload

To make the experiments as realistic as possible, we use real traces from Wikipedia and Facebook. For the interactive workloads, we use the real trace from Wikipedia requests on

![Fig. 4. Testbed](image-url)
2007 October 17 to replay the workload of Wikipedia users. The trace includes data on requests time stamp and their accessed pages. We filter the requests based on per second rate and generate the requests rate. The original request rate is around 1,500-3,000 per second.

For the batch workloads, we use the traces collected in October 2009 by Facebook for applications that are executed under Hadoop environment. Referring to [11], we configure the map phase of each job takes 25-13000 seconds, and the reduce phase takes 15-2600 seconds. The deadline for processing jobs is generated based on uniform distribution with \( \mu = 6 \) hours and \( \sigma = 1 \) hour in \( N(\mu, \sigma^2) \). We also assume the workloads consume the maximum of cluster utilization as 27% as same as in [11].

7.3 Application

We use the Weave Shop web application that implemented with containers as the application in our scenario. The Weave Shop is a shopping system for selling socks online and has multiple microservices, including user microservice to handle user login, user database microservice for user information storage, payment microservice to process transactions, front-end microservice to show the user interface, catalog microservice for managing item for sale and etc. As these microservices are implemented independently, they can be deployed and controlled without impacting other microservices. The application is deployed by a configuration file, and part of the microservices are configured as optional, e.g., recommendation engine.

7.4 Results

To evaluate the benefits of our proposed approach for renewable energy usage, we perform the comparison of our proposed approach (SA) and a baseline algorithm (HS), which applies VM consolidation [5] and host scaling [35] that dynamically adds/removes hosts in system, while the green-aware scheduling algorithm in our proposed approach is not applied.

Figure 5 shows the baseline energy consumption of interactive workloads, batch workloads and cooling during the observed time period (one day). The blue line shows the actual renewable power production. The system is consuming brown energy at night time from 0:00 to 5:00 and 18:00 to 24:00. The solar energy is available at day time during hours 6:00 to 17:00. Even with taking advantage of VM consolidation and host scaling, the solar energy consumption of the system is not fully utilized. For example, at hour 11:00, the total energy consumption of system is about 1400 Wh, while the available solar energy is more than 1500 Wh.

Figure 6 demonstrates the energy consumption of our proposed approach by using Algorithms 1 to 4. The blue line still shows the actual renewable power production, but the decision making is happening based on our SVM prediction model. We can observe that the power consumption of the batch workloads during 0:00 to 8:00 has been reduced, which results from the deferring operations: batch workloads are deferred to the time when solar energy is available, e.g., hour 6:00. Some batch workloads are still executed during hours from 0:00 to 8:00 due to the deadline constraints, which cannot be deferred to the time when renewable energy is available. Thus, we can find that the brown energy usage during 0:00 to 8:00 has been reduced compared in Figure 6. For example, at hour 1:00, the total power is reduced from 1221 to 815 Watts.

During the time when solar energy is available, our proposed approach has improved the usage of renewable energy, in which the energy consumption follows the line the of predicted renewable energy. For instance, at hour 11:00, the usage of solar energy is increased from 1387 Wh to 1544 Wh compared with Figure 5.

We also note that the power consumption during the time when brown energy is the only source of power supply, the energy is also reduced, which exploits the brownout mechanism to reduce the energy consumption. For instance, the power at hour 18:00 is decreased from 1391 Watts to 1195 Watts.

Combing the results in Figure 5 and 6, we conclude that our proposed approach can improve the usage of renewable energy and reduce the usage of brown energy.

The average response time and CDF of response time for interactive workloads in our proposed approach are demonstrated in Figure 7. The overall measured response

---

5. https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository
The response time is 403 ms. The results also show that 95% of requests are responded in 900 ms, and 99% of requests are responded within 1 second. It shows that our proposed approach reduces brown energy usage while ensuring the QoS. The reason is that the brownout approach can relieve the over-loaded situation, thus ensuring the response time.

To illustrate the reason of power reduction by our proposed approach, Figure 8 compares the active hosts during the observed time period between the baseline and our proposed approach, as switching the idle hosts into the low power mode is the most effective way to save power. The results demonstrate that our proposed approach uses fewer hosts during the time period when renewable energy is not sufficient, e.g., hours from 0:00 to 8:00 and 18:00 to 24:00, while when the renewable energy is available, more hosts are scaled in to utilize more renewable energy, such as the time from 10:00 to 15:00. In this way, the power of all the active hosts is reduced and the usage of renewable energy is improved.

Figure 9 demonstrates the comparison of brown and renewable energy usage. During the night time (0:00 to 5:00 and 18:00 to 24:00), both approaches only use brown energy. Benefiting from proposed algorithms, our approach reduces the brown energy usage by 28% from 13.9 kWh to 10.8 kWh. During the daytime (6:00 to 17:00), both renewable energy and brown energy are used in two approaches. Our proposed approach consumes 5% more total energy in the day time, while it uses 10% more renewable energy than the baseline from 9.9 kWh to 10.9 kWh. In total power usage comparison, the brown energy usage is reduced 21%, and the renewable energy usage is improved 10%.

In summary, experiments show that our proposed approach can improve the renewable energy usage for both interactive and batch workloads by applying brownout mechanism and deferring the execution of batch workloads. Our proposed approach can switch more machines into low power mode when renewable energy is not sufficient while the QoS of workloads are also ensured.

8 Conclusions and Future Work

Our self-adaptive approach for managing applications and harnessing renewable energy brings up many opportunities to optimize the energy efficiency problem in cloud computing environment. In this paper, we proposed a multiple layer perspective model for interactive and batch workloads by considering renewable energy. We also introduced a self-adaptive and renewable energy-aware approach deriving from the perspective model. The proposed approach improves the usage of renewable energy and reduces the usage of brown energy while ensuring the QoS requirement of workloads. We proposed a solar radiation prediction method based on SVM to predict the solar power at Denver City. The prediction method is integrated into our proposed approach. We apply brownout mechanism to dynamically deactivate/activate optional components in the system for interactive workloads and use a deferring algorithm to defer the execution of batch workloads to the time when renewable energy is available. VM consolidation and host scaling are also applied to reduce the number of active hosts.

We developed a prototype system to evaluate the performance of our proposed approach. In the prototype system, the physical resources are managed by OpenStack and the services are managed by Docker Swarm. We take advantage of the APIs from these platforms to monitor, manage, and provision the resources to services. The effectiveness of our proposed approach is showed through the experimental evaluation, including the workloads from real traces. The results show that our proposed approach is able to improve the usage of renewable energy while satisfying the constraints of workloads.
In current experiments, we assumed the maximum capacity of solar panel availability and applied the cooling model to calculate the cooling efficiency to get the cooling power based on power from hosts. The renewable energy and cooling part can be improved for accurate measurements by installing the solar panel and power meters for cooling equipment. We plan to extend our prototype system for multiple clouds in the different time zones to support workload shifts in data centers and minimize the carbon footprint in a global view.

ACKNOWLEDGMENTS

This work is supported by China Scholarship Council and ARC Discovery Project.

REFERENCES


Minxian Xu received the BSc degree in 2012 and the MSc degree in 2015, both in software engineering from University of Electronic Science and Technology of China. He is working towards the PhD degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. His research interests include resource scheduling and optimization in cloud computing. He has co-authored several peer-reviewed papers on CSUR, T-SUSC, T-ASE, CCPE, ICSOC and ICC.

Adel Nadjaran Toosi has joined Faculty of Information Technology at Monash University as a lecturer in May 2018. Before he joined Monash University, he worked as a Research Fellow in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory in the School of Computing and Information Systems (CIS) at the University of Melbourne for more than three years. He received his Ph.D. degree in Computer Science and Software Engineering from the University of Melbourne in 2015. His thesis was one of the two theses nominated for the Chancellor’s Prize for Excellence in the Ph.D. Thesis and John Melvin Memorial Scholarship for the Best Ph.D. Thesis in Engineering. Adel has made significant contributions to the areas of resource management and software systems for cloud computing. His research interests include Distributed Systems, Cloud Computing, Software-Defined Networking (SDN), Green Computing, and Soft Computing. He is currently working on resource management for Software-Defined Networking (SDN)-enabled cloud computing environments. For more details, please visit his homepage: http://adelnadjarantoosi.info.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He served as a Future Fellow of the Australian Research Council during 2012-2016. He has authored over 625 publications and seven text books including “Mastering Cloud Computing” published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=114, g-index=245, 66,900+ citations). Dr. Buyya is recognized as a “Web of Science Highly Cited Researcher” in 2016 and 2017 by Thomson Reuters, a Fellow of IEEE, and Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to Cloud computing. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com