The Journal of Systems and Software 154 (2019) 125–138

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

New Trends and Ideas

ROUTER: Fog enabled cloud based intelligent resource management approach for smart home IoT devices

Sukhpal Singh Gill[a,b,*], Peter Garraghan[a], Rajkumar Buyya[b]

[a] School of Computing and Communications, Lancaster University, UK
[b] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 10 December 2018
Revised 21 March 2019
Accepted 23 April 2019
Available online 24 April 2019

Keywords:
Fog computing
Cloud computing
Internet of things
Smart home
Resource management
Edge computing

ABSTRACT

There is a growing requirement for Internet of Things (IoT) infrastructure to ensure low response time to provision latency-sensitive real-time applications such as health monitoring, disaster management, and smart homes. Fog computing offers a means to provide such requirements, via a virtualized intermediate layer to provide data, computation, storage, and networking services between Cloud datacenters and end users. A key element within such Fog computing environments is resource management. While there are existing resource manager in Fog computing, they only focus on a subset of parameters important to Fog resource management encompassing system response time, network bandwidth, energy consumption and latency. To date no existing Fog resource manager considers these parameters simultaneously for decision making, which in the context of smart homes will become increasingly key. In this paper, we propose a novel resource management technique (ROUTER) for fog-enabled Cloud computing environments, which leverages Particle Swarm Optimization to optimize simultaneously. The approach is validated within an IoT-based smart home automation scenario, and evaluated within iFogSim toolkit driven by empirical models within a small-scale smart home experiment. Results demonstrate our approach results a reduction of 12% network bandwidth, 10% response time, 14% latency and 12.35% in energy consumption.

1. Introduction

Emerging Big Data and Internet of Things (IoT) applications such as smart cities and healthcare services have risen in societal prominence, demonstrated by an increase of data velocity of 250 MB per minute globally (Chen and Zhang, 2014). Therefore, such applications require substantial data and computational capability to provision service (Perera et al., 2017), possible via deployment within Cloud datacenters. However, such applications when deployed within Cloud datacenters encounter potentially high latency and response times due to large geographical distance and data bandwidth requirements between clients and the datacenter. (Al-Fuqaha et al., 2015). Fog computing has been envisioned as a means to reduce the latency, via extending Cloud datacenters to integrate with the network edge (White et al., 2017; Gill et al., 2018). Thus, IoT environments can leverage fog-assisted Cloud computing to execute latency-sensitive applications.

Resource management – the process of scheduling and allocating resources to applications – is a fundamental concept within distributed systems (Singh and Chana, 2016) in order to adhere to specified Quality of Service (QoS) constraints whilst minimizing overheads pertaining to performance, and energy waste (Singh and Chana, 2016). While there exist a wide plethora of existing schedulers for distributed systems such as MESOS, YARN and BORG in cloud, which have been created to operate within centralized computing infrastructure (Rodriguez and Buyya, 2018). Specifically, these schedulers are not designed to operate within an environment including highly mobile edge devices (Son and Buyya, 2019), latency-sensitive applications, nor wide geographical areas intrinsic to Fog computing environments. Resource management within Fog computing predominantly focuses on managing the compute and storage service between edge devices and the Cloud datacenters to process user tasks with minimum latency and response time (Gill et al., 2018; Singh et al., 2016; Atzori et al., 2010; Deng et al., 2015). Existing IoT and Fog computing resource managers focus on a singular or specific sub-set of metrics including application response time, latency, energy, and network bandwidth (Lee et al., 2016; Yu et al., 2017; Stojkoska and Trivodaliev, 2017; Zhang et al., 2017), Capturing all these parameters within a Fog computing resource management is particularly important within the context of smart homes, which are positioned to process increasingly larger quantities of data from smart devices and appliances connected to

* Corresponding author.
E-mail addresses: s.s.gill1@lancaster.ac.uk (S.S. Gill), p.garraghan@lancaster.ac.uk (P. Garraghan), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.jss.2019.04.058
0164-1212/

IoT systems, whilst simultaneously ensuring high QoS and reduced energy consumption to reduce electricity bills for home dwellers (Garraghan et al., 2018). Due to the complexity of multi-objective optimization of parameters for trade-off decision making in resource management (which has led to existing Fog resource management algorithms implementing FIFO or round-robin based approaches (Lee et al., 2016; Yu et al., 2017)), we believe that exploring nature or bio-inspired algorithms is a promising approach to address this problem for resource management (Singh and Chana, 2016; Kaur et al., 2018).

In this paper, we propose a Fog-enabled Cloud computing resource management framework for smart homes. Our approach, ***R**es**OU**rce managemen**T** t**E**chnique for sma**R**t homes* (***ROUTER***) has been designed to consider and optimize multiple parameters simultaneously including response time, network bandwidth, energy consumption and latency simultaneously via use of a Particle Swarm Optimization algorithm (PSO). Stochastic nature of the particle increases due to this property of PSO and touches rapidly to global minima with a realistic noble solution (Hassan et al., 2005). PSO has become prevalent due to its easiness and its usefulness in extensive range of application with little cost of computation (Chen and Yu, 2005; Hassan et al., 2005). ROUTER has been validated through empirical findings via a case study of IoT based smart home automation which are then integrated into iFogSim for evaluation. The main contributions of this research work are as follows: (i) a detailed requirement and design of an Fog-assisted Cloud architecture to perform effective resource management for various IoT edge devices; (ii) a request handler mechanism for Fog computing jobs, and a multi-objective PSO based resource management technique; (iii) a small-scale empirical study of an IoT smart home environment that leverages a Fog-assisted Cloud computing environment, analyzing the performance of various QoS parameters within different operational contexts.

The rest of the paper is organized as follows. Section 2 presents related work of existing techniques. The proposed technique is presented in Section 3. Section 4 describes the experimental setup and case study. Section 5 describes the results of the evaluation. Section 6 presents conclusions and future work.

2. Related work

Research into IoT applications within Fog computing is growing research field, with various unsolved research challenges (Gill et al., 2018). This section presents the current research on resource management within Fog computing.

Deng et al. (2015) formulated a workload allocation problem to study the tradeoff between energy consumption and delay within a Cloud-Fog computing system. Furthermore, the primary problem is decomposed into three sub-problems to solve independently, and demonstrated that Fog computing is efficient in reducing transmission latency and communication bandwidth, however does not consider system network bandwidth and energy consumption. Do et al. (2015) proposed a proximal algorithm for joint resource allocation in the geo-distributed environment and reducing carbon footprint. Moreover, authors demonstrated that their proposed solution can reduce system carbon footprints whilst offering video streaming as a cloud service. Gu et al. (2015) proposed a cost-efficient resource management technique integrated within a medical Cyber-physical System in which virtual machine placement, task distribution and base station association are investigated. Results demonstrated that the proposed solution performs more effectively in comparison to a greedy algorithm in terms of energy consumption.

Lee et al. (2016) proposed a Gateway-based Fog Computing (GFC) architecture for wireless sensors and actuator networks predominantly consisting of master and slave nodes, managing virtual gateway functions, flows, and resources. Experimental results show that GFC performs more effectively in terms of response time. Yu et al. (2017) proposed a Virtualization based Resource Provisioning (VRP) algorithm for Fog computing and designed an architecture using the concept of parallel and distributed load balancing. Furthermore, the algorithm is evaluated within CloudAnalyst simulator that finds the proposed solution decreases the system energy cost. Stojkoska and Trivodaliev (2017) proposed a conceptual model for smart homes using IoT for fog computing, and suggests that energy consumption can be reduced via integration of geographically distributed renewable energy sources. Zhang et al. (2017) proposed a three-layer hierarchical game framework for resource management in Fog computing to solve the challenges pertaining to fast data processing and minimum response time. This research work reported that Fog devices are more capable to reduce latency as compared to the cloud by experiencing a minor increase in energy consumption. Therefore, the trade-off between latency and power consumption is required to provide more efficient services.

From the literature it is observable that under-provisioning and over-provisioning of resources in existing Fog computing and IoT resource management techniques (Lee et al., 2016; Yu et al., 2017). Fog devices have additional compute and storage power, however it is not feasible for such devices to provide resource capacity equivalent to that of Cloud datacenters, therefore efficient resource management is required to process user requests in a timely manner. To solve this problem, the resource requirement for execution of user tasks should be predicted accurately in advance to utilize resources efficiently. The comparison of existing resource management techniques with the proposed technique (ROUTER) is described in Table 1.

3. ROUTER: Fog-assisted cloud based resource management for IoT and big data analytics

This section presents the proposed resource management technique (ROUTER) for Fog-assisted Cloud resource management for smart homes. The architecture of ROUTER is shown in the Fig. 1.

Based on their functionality, the architecture is composed of three layers, the components of the proposed architecture are discussed below:

Internet of Things (IoT): Edge devices comprising gateways, fog devices, smart home appliances, sensors etc. A user may interact with the Fog computing environment via IoT applications or sensors. The functionality of this layer is enhanced by installing intelligent and applications within end devices.

Fog Computing: Collects data generated by bottom layer (IoT) and establishes communication between edge devices and the Cloud datacenter. The functionality of the intermediate layer is divided into two sublayers: a) *Field Area Network* (end devices interacting with each other via 3 G/4 G/Wi-Fi) and b) *Internet Protocol/Multi-Protocol Label Switching* (used to transfer the data from end devices to centralized cloud system).

Cloud and Big Data: Manages the services which enable the management of resources and processing of big data and IoT tasks. Furthermore, this layer provides QoS to Fog computing applications and the Cloud computing operational management. Applications such as Big Data processing is performed at this layer to handle the large data coming from different IoT applications and process through different stages such as preprocessing, classification and prediction (Gill et al., 2018).

Cloud computing contains a wide variety of services that can enhance application operation to minimize latency of executing tasks on Fog devices whilst decreasing Cloud economic costs. There exist different types of services, which operate in tandem comprise:

Table 1
Comparison of Existing Techniques with Proposed Technique (ROUTER).

Authors	Applicable Network	Fog Nodes	Nodal collaboration	Focus	Performance Parameters (QoS)			
					Response Time	Energy	Latency	Network Bandwidth
Deng et al., 2015	Mobile Network	Servers	Master slave	Application management	✖	✔	✔	✖
Do et al., 2015	Vehicular Network	Servers	Peer to Peer	Application management	✖	✔	✖	✖
Gu et al., 2015	Mobile Network	Base Stations	Peer to Peer	Network Management	✖	✔	✖	✖
Lee et al., 2016	IoT	Network Devices	Peer to Peer	Resource Management	✔	✖	✖	✖
Yu et al., 2017	IoT	Network Devices	Peer to Peer	Resource Management	✖	✔	✖	✖
Stojkoska and Trivodaliev, 2017	Mobile Network	Base Stations	Cluster	Application management	✖	✔	✖	✖
Zhang et al., 2017	Vehicular Network	Servers	Master slave	Network Management	✔	✖	✖	✖
ROUTER (Proposed)	**IoT**	**Network Devices and Servers**	**Peer to Peer**	**Application, Network and Resource Management**	✔	✔	✔	✔

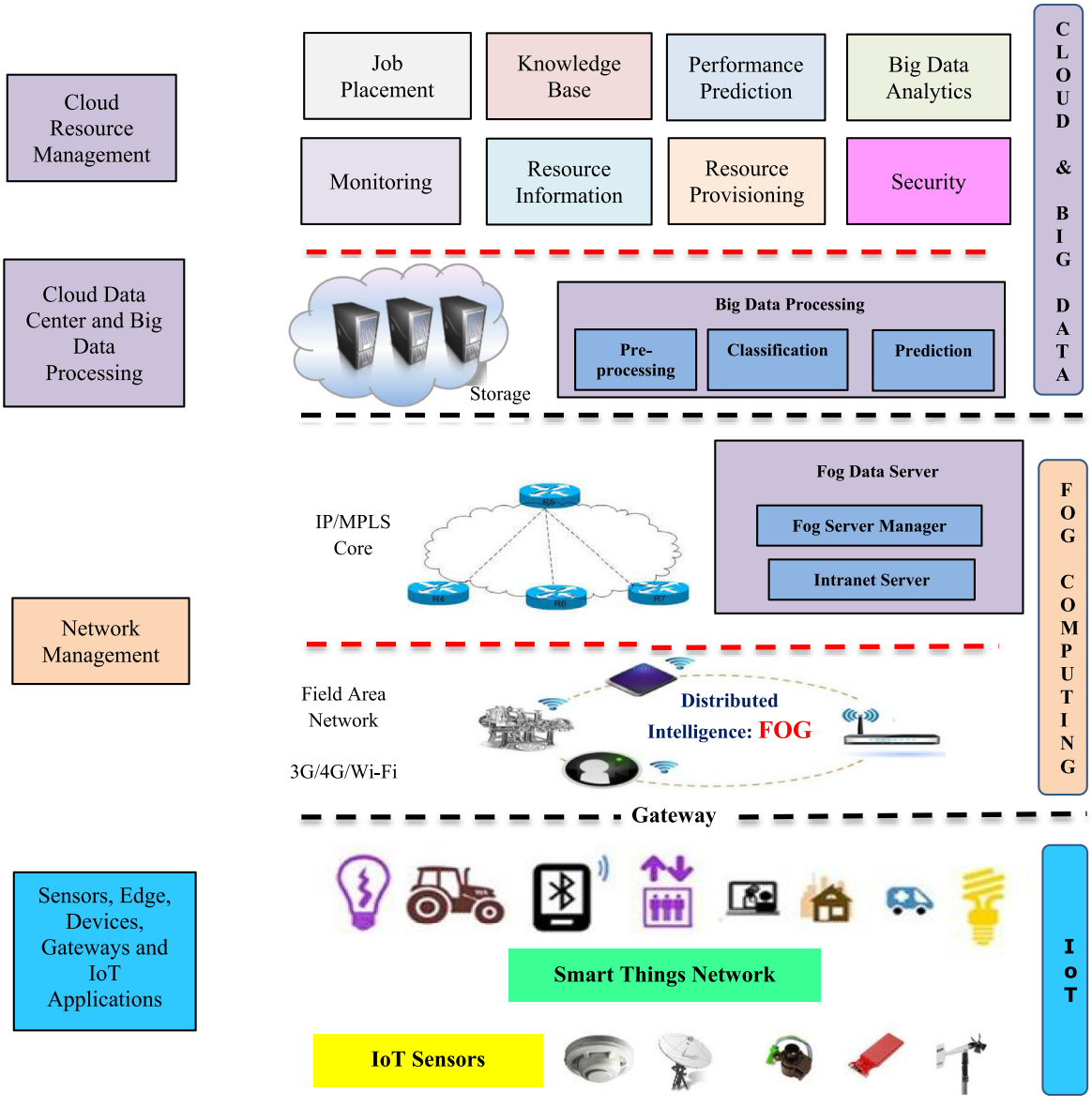


Fig. 1. ROUTER Architecture.

- *Monitoring:* Monitoring of service/application status and performance.
- *Knowledge Base:* Stores historical information pertaining to resource and application demand to improve decision-making processes in future IoT-based applications.
- *Job Placement:* Processes information provided by *Monitoring* services that contain available Cloud resource status at a particular period of time. This information is leveraged to discover the best machines to schedule jobs (tasks) for execution. This is further interconnected with *Resource Provisioning* to find allocation requirements of new resources for existing tasks.
- *Big Data Analytics:* Collects data from different IoT devices to perform different data processing operations spanning data pre-processing, classification, and prediction (Gill et al., 2018). This module assists in determining threshold values for performance parameters for resource scheduling decision making.
- *Resource Information:* Obtains information from *Monitoring* and *Knowledge Base* to profile applications and resources.
- *Security*: Provides authorization and authentication to applications and services to manage user credentials.
- *Resource Provisioning:* Control and provides resource allocation to various network, Fog and Cloud resources. Due to changing the number of applications and requirements of applications with the entire system, resources are allocating dynamically in response to QoS and operational constraints.
- *Performance Prediction*: Performance of free cloud resources is visualized by utilizing the information of *Knowledge Base* service and this information is further forwarded to the *Resource Provisioning* service to determine application resource requirements.

3.1. Request handler mechanism

Fig. 2 shows the interaction of Fog Data Server (FDS) with IoT devices and Cloud Data Server (CDS) in terms of the design model. *IoT layer* contains end devices such as gateways and sensors to retrieve information from the end user. It then forwards the user information to FDS for further processing. The fog layer contains multiple *FDSs*. The FDS comprises one Fog Server Manager (FSM), which manages all FDS resources required for job execution. Further, the request can be forwarded to cloud layer for execution in case of unavailability of resources at the FDS level. The cloud layer has a number of *CDS*. Fig. 3 describes the interaction of cloud layer, fog layer and IoT layer to handle a typical job request.

There are two types of job processing requests. First, at the FDS (denoted by i_f) and another at the CDS (denoted by i_c), which is requested by FDS in the case of unavailability of resources at the Fog layer. Initially, the IoT layer submits a job request (i_f) to the closest FDS (say FDS_1) intended to accelerate job execution. The FSM checks whether the resource demand of that particular request is satisfied or not at FDS_1. If the FDS_1 satisfies the resource demand of request (i_f) then the FSM starts its execution and tracks its execution status.

If the FDS_1 partially satisfies the demand of the job request (i_f) then the FSM has to wait for Minimum Constraint Time (M_{st}), otherwise the job request is forwarded to *the CDS*. If all the resources are occupied at *the* FDS_1 but is in its initial release state, then the job request (i_f) must wait for Minimum Constraint Time (M_{st}) to release the resources and then commence execution. If all the resources are busy executing other FDS_1 yet some requests are failing during execution, then the FSM will discover another FDS_2 to offload requests. If all the resources are unavailable in all of the FDS within the Fog cluster, then job request (i_f) are propagated to the CDS over appropriate communication network and now this request is denoted as (i_c) and user will receive a message *"Wait for processing"* and then must wait for maximum allocated time (M_{time}) to release the resources at CDS. FSM then sends the job request (i_c) to closest CDS for further processing. The CDS provides resources for execution of job requests with minimum response time and latency, and then sends an acknowledgement to the FDS. The latency and response time values are predefined via analysis and modelling of historical system data and both the parameter have some fixed value for a certain interval (we have considered one-hour duration for intervals). Based on the performance of resources (execution time and energy consumption), the value of latency and response time is redefined at every interval. The next section describes the working of Fog server manager for scheduling of resources.

3.2. Fog server manager

This section describes the Fog server manager for scheduling resources to execute job requests.

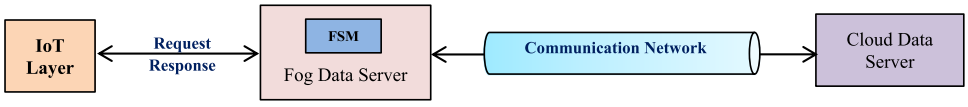


Fig. 2. Functional Components.

Step 1. The *FDS* checks local resource availability and has the ability to manage system resources for request execution.
Step 2. The *IoT layer* forwards the user request to *FDS*, which enqueues request.
Step 3. The *FDS* handles the user request with following conditions:

i) **if** required resources are available at the *FDS level*, **then** the FSM processes the request on the FDS and submits an acknowledgement to the *IoT layer* regarding execution status.
ii) **if** a portion of required resources are available at the *FDS level*, **then** the job request needs to wait for a particular time period (Threshold value for each request i_f), **otherwise** Goto (v).
iii) **if** the *FDS* is already executing other requests, which are close to completion, **then** the new job request needs to wait for a particular time period (Threshold value for each request i_f), **otherwise** Goto (v).
iv) **if** one of the resources executing a particular request fails at the *FDS* level, **then**, the request management behaviour will follow the behaviour as in (ii) condition.
v) **if** no resources are available in the *FDS*, **then** the job request is forwarded to *the CDS*.

Step 4. *CDS* executes the job request and sends an acknowledgement to *FSM*.

Fig. 3. Request Handler Mechanism.

3.2.1. Objective function

The main objective of the fitness function is to optimize the performance parameters energy consumption ($E_{Consumption}$), network bandwidth ($N_{Bandwidth}$), latency ($Latency$) and the response time (R_{Time}) to facilitate requests originating at the IoT layer. This fitness function (Eq. (1)) effectively compromises the following performance parameters

$$Fitness\,value = \alpha\, E_{Consumption} + \beta\, N_{Bandwidth} + \gamma\, Latency + \mu\, R_{Time} \quad (1)$$

where $0 \leq \alpha < 1$, $0 \leq \beta < 1$, $0 \leq \mu < 1$ *and* $0 \leq \gamma < 1$ denotes weights to prioritize components of the fitness function. The *Network Bandwidth* is defined as the number of bits transferred/received in one second. The *Latency* is defined as the delay before the transfer of job request for processing. The *Response Time* is defined as the length of time taken for a system to react to a job request first time. The *Energy Consumption* is the sum of energy consumed by the processors, the switching equipment, the storage devices, the network devices and other components such as fans or conversion losses (Al-Fuqaha et al., 2015).

3.2.2. Particle swarm optimization based resource scheduling algorithm

Particle Swarm Optimization (PSO) is motivated by the social activities of species such as group of birds seeking food sources (Chen and Yu, 2005) and works based on a global search method. The PSO algorithm denotes the number of particles as a population, which are first initialized randomly. The PSO improves the fitness value (as calculated using Eq. (1)) of a particle in every generation. In the PSO algorithm, the particle's position is denoted as: a) global optimal state ($Global_{OptimalState}$): best particle among group based on fitness value of all the particles b) and local optimal state ($Local_{OptimalState}$): it is best fitness value of a particular particle. Further, [Eq. (1)] is used to update particle's velocity and position in every generation. Every particle regulates its position based on the value of $Global_{OptimalState}$ and $Local_{OptimalState}$ in every generation. The PSO can be used to solve resource scheduling problems due to (i) usefulness and easiness with less computation cost and (ii) achieving global minima relatively quickly (Chen and Yu, 2005). Deteailed terminology of PSO used in this research work is presented in Table 2.

There is a partial solution in genome for every particle, which is considered as a resource identifier. The main motivation for the PSO-based scheduling is to identify the best resource identifier, which creates the best solution for the particular optimization problem such as resource scheduling. The selection process of non-PSO based resource identifier stops after a pre-defined number of iterations. We set a fixed number of iterations to keep the computation time low. In the PSO-based method, a new solution would be rejected if its fitness value is less than the current solution. Algorithm 1 presents the pseudo code of PSO based resource scheduling algorithm.

The working of PSO based resource scheduling algorithm can be described as follows:

1. Initializes the random feasible solution based on the request list and resource list.
2. Select the best heuristic from low-level heuristics.
3. Every request represents the resource identifier with initial solution, which accesses the value of fitness function.
4. Randomly initializes the request's position and request's velocity.
5. At each request position, select a low-level heuristic and calculates fitness value ($Local_{OptimalState}$).
6. If at particle position the Fitness ($Local_{OptimalState}$) is greater than Fitness ($Global_{OptimalState}$) then $Global_{OptimalState}$ takes the value of $Local_{OptimalState}$.
7. Identify the fitness value at best global position of the request.
8. [Eq. (1)] is used to update the value of particle position and velocity after selection of request from population. Furthermore, it computes the fitness value of the new position and compares with its previous position.
9. If the value is better than the local best value then it assigns the request's present position to the global best value.
10. Fitness at $Local_{OptimalState}$ and $Global_{OptimalState}$ is compared. If the fitness at $Local_{OptimalState}$ is greater than at $Global_{OptimalState}$ then it assigns the value of $Local_{OptimalState}$ to $Global_{OptimalState}$.

Apply to the resource scheduling problem after selection of a low-level heuristic. The scheduling of resources is continued until all the jobs are scheduled.

4. Performance evaluation

To demonstrate the feasibility of the proposed approach, we have developed the framework and scenario into a Fog computing based environment using CloudSim (Calheiros et al., 2011) and iFogSim (Gupta et al., 2017). In this research work, event simulation functionalities of CloudSim have been used to implementing functionalities of iFogSim architecture. CloudSim entities such as datacenters and communication amongst datacenters through message sending operations are included. Therefore, the core CloudSim layer is responsible for handling events between fog computing components in iFogSim (Gupta et al., 2017). iFogSim implementation is established by simulated services and entities. The proposed technique has been validated via deployment of a smart home automation experiment case study. The application model of IoT-based smart home automation is built into iFogSim in order to validate the proposed technique through real-time application (in other words, data from the experiment is directly fed into the

Table 2
PSO Terminology.

PSO Terminology	Description
Particle	Denoted as an independent instance in a search space and its position is affected by the value of $Local_{OptimalState}$ and $Global_{OptimalState}$. Further, the performance of a particle is measured by its fitness value. A request is considered as a particle for this research work.
Population Size	It is a set of number of job requests, which are coming from IoT/edge devices.
Initial Random Velocity	The movement of every particle is dependent on 1) preliminary random velocity and 2) two randomly weighted effects: a) the affinity to reach neighborhood's best earlier position and b) the affinity to reach best earlier position of a particle. Resources are mapped to requests based on these two affinities. Request will be processed on that resource which has higher value of fitness.
Particle Velocity	The probability distribution for the particle determines the value of particle velocity.
Particle Position	Present state of the particle (request), which can be completion state, execution state, ready state, waiting state or submission state.
Global Best Position ($Global_{OptimalState}$)	Best position of particle (job request) attains among the total group of particles (job request list).
Local Best Position ($Local_{OptimalState}$)	Best position of particle (job request) as particle attains

Algorithm 1 PSO Based Resource Scheduling Algorithm.

1. **Input Value**: No. of job requests and No. of resources
2. **Outcome**: Resource scheduling for an execution of Job Requests
3. **Begin**
4. Initialize variables: Resource list, Job Request List, Randomly Allocating Input Value
5. $Population_{Size}$ = Size of Population
6. $Initial_{RandomVelocity}$ = Initial Random Velocity
7. $Particle_{Velocity}$ = Velocity of Particle
8. $Particle_{Position}$ = Position of Particle
9. R_P = Random Position
10. $Initial_{PopulationSize}$ = Initial Population Size
11. $Global_{OptimalState}$ = Global Optimal State
12. $Local_{OptimalState}$ = Local Optimal State
13. M_{IC} = Maximum Iteration Count
14. Counter = 1
15. **while** (counter $\geq$ 0)
16. counter ++
17. **if** (counter $\geq$ $Population_{Size}$)
18. **break**
19. $Particle_{Velocity}$ $\leftarrow$ $Initial_{RandomVelocity}$
20. $Particle_{Position}$ $\leftarrow$ R_P ($Population_{Size}$)
21. $Local_{OptimalState}$ $\leftarrow$ $Particle_{Position}$
22. $\forall$ $Particle_{Position}$ $\in$ $Initial_{PopulationSize}$, Compute Fitness Function [Eq. (1)]
23. **if** *Fitness Value* ($Global_{OptimalState}$) $\geq$ *Fitness Value* ($Local_{OptimalState}$) **then**
24. $Global_{OptimalState}$ $\leftarrow$ $Local_{OptimalState}$
25. Counter = 1
26. **while** (counter < M_{IC}) **do**
27. counter **++**
28. **for** $Particle_{Position}$ $\in$ $Initial_{PopulationSize}$ **do**
29. $Particle_{Velocity}$ $\leftarrow$ Update_Particle_Velocity ($Particle_{Velocity}$, $Global_{OptimalState}$, $Local_{OptimalState}$)
30. $Particle_{Position}$ $\leftarrow$ Update_Particle_Position ($Particle_{Position}$, $Particle_{Velocity}$)
31. **if** *Fitness Value* ($Particle_{Position}$) $\leq$ *Fitness Value* ($Local_{OptimalState}$)
32. **then**
33. $Local_{OptimalState}$ $\leftarrow$ $Particle_{Position}$
34. $Global_{OptimalState}$ $\leftarrow$ $Local_{OptimalState}$ **if** *Fitness Value* ($Local_{OptimalState}$) $\leq$ *Fitness Value* ($Global_{OptimalState}$) **else** $Global_{OptimalState}$
35. **return** ($Global_{OptimalState}$)
36. **while** *queue is not empty* **do**
37. $\forall$ *resource* $\in$ *resource list* **do**
38. Job request = dequeue from unprocessed job request queue
39. schedule job request (based on fitness value [Eq. (1)])
40. **if** all the job requests not executed **then Goto** 15
41. **Finish**

Fig. 4. Front View of Smart Home.

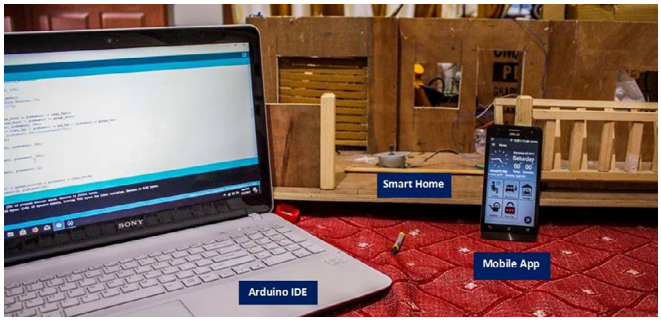


Fig. 5. Interaction of Smart Home Components with Mobile App using Arduino IDE.

simulator to provide edge-device operational behavior for the resource manager).

4.1. Case Study: IoT based smart home automation

In order to demonstrate an example smart home case study, we interconnected multiple IoT devices wirelessly controllable by using a smartphone. The scenario we have created consists of a home consisting of three rooms (Garage, Lobby, and Bedroom), that are capable of manipulating various devices and appliances within each room to which consist of AC, fan, bulb and doors. Fig. 4 depicts the front view of smart home, whilst Fig. 5 describes an interaction of smart home components with mobile app using Arduino IDE.

Figs. 6 and 7 depicts the interaction of devices in the smart home application, and integration of different components, respectively. The smart home contains an Arduino board and different home appliances such as AC, fan, bulb and doors. The components are interacting with each other via the following sequence:

- *Android to ESP8266:* Initially, an Android device generates a signal to fetch required information from the smart home. This signal is transferred to the ESP8266 module wirelessly using the server created by the ESP over the local hotspot. This connection uses a *connection id* between ESP and Android device,

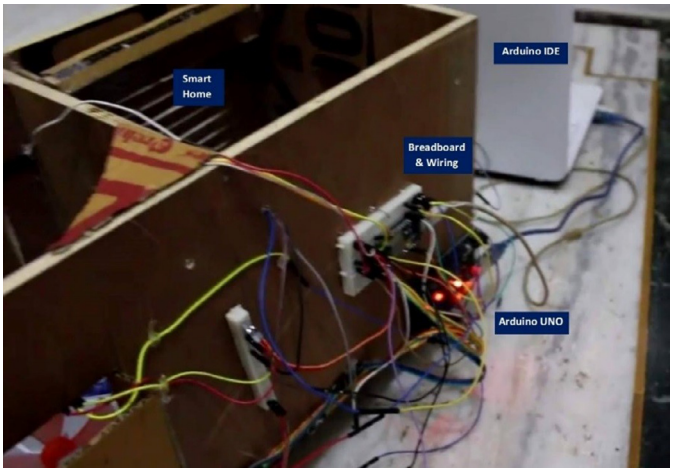


Fig. 6. Interaction of Arduino IDE and Arduino UNO.

where ESP sends the HTTP packet to initiate the connection. This data is then further processed at the ESP8266 module.

- *Intranet Server:* The Arduino based hardware is designed to provide an interface between the android application and appliances. This is used to retrieve incoming data from sensors and converts into digital and send it to android application over Internet using Intranet server. This is also used to generate the signal for a specific appliance selected by the user.
- *ESP8266 to Arduino:* ESP receives the signal/data from the server created at the specific static IP address. The Arduino then matches the header with the prescribed header format and then further breaks down the signal and uses the resultant data to enable or disable the desired pins.
- *Controlling Device States:* The Arduino directs the pins received in the signal to turn ON/OFF home appliances as per user requirements. The device status is then updated within the Android application.
- *Intrusion/Breach Detections:* When the security feature in the Smart Home App is turned ON, the Passive Infra-Red (PIR) sensor (Sahoo and Pati, 2017) will be turned ON to detect the heat signals and motion inside the room. If any movement is detected, it will activate a buzzer and an SMS of the detected intrusion is sent to the owner's phone. Similarly, when the door is opened, the signal breaks and the owner is alerted with a message of breach from the door.
- *Live Video Feed:* The device actives an IP camera connected to the Wi-Fi hotspot to create a live view in the application. Therefore, when the server is started to project the video, its IP address is be used inside Smart Home App to create the image.

Fig. 8 shows the interface of the smart home. The user can control basic operations such as device selection, turn on/off home appliancchange light colors, fan speed, acquire sensor details, add/view event, and watch live feed camera. The home screen shows the live view of various rooms as shown in Fig. 8(g), and sensor information such as temperature sensor, humidity sensor, number of devices connected to smart home and consumption of electricity. A user can further create a new event if required by using the "Add Task" shown in Fig. 8(e).

The use case diagram of smart home automation shown in Fig. 9 describes the interaction of different actors user, app database and sensors. Fig. 10 shows the class diagram of smart home automation to describe the interaction of different classes with their different functions. Alert class describes the important aspects of real-time applications such as latency, response time and deadline. User will be alerted if response time is more than threshold value. Further, alert can be generated if deadline of a particular request is missing. Moreover, user can be intimated when latency is more than its threshold value.

4.2. Implementation of proposed technique in iFogSim

Fig. 11 describes the component mapping for smart home automation within a simulation environment using the. iFogSim toolkit. Different sensors are used to control different activities such as voltage, light, motor speed (motion), room temperature and security of smart home. PIR sensor detects the movement of objects even beyond the boundaries of the smart home and detects heat signature from the light. IP camera is used as an *edge device*. ATmega328P based Arduino board is connected to every appliance of the smart home. Smart Home App is communicating with Fog device using the HTTP communication protocols (ESP8266 module).

The following classes within iFogSim are modified to implement IoT based smart home application within the greater Fog environment:

FogDevice: Describes the hardware features of Fog devices and their relations with sensors and other Fog devices. We have extended *PowerDatacenter* class of CloudSim (Calheiros et al., 2011) to allow the main attributes of the FogDevice class to access downlink and uplink bandwidths (specifying the communication capacity of Fog devices), storage size, processor and memory. Functions of this class specify the scheduling of resources among application modules executing on it and their deployment and release after execution. Moreover, we have developed a *Listener* module, which receives the data from different sensors as shown in Fig. 11.

Sensor: In the iFogSim toolkit, IoT sensors are represented by instances of the Sensor class. Features of a sensor, extending from its connectivity to output aspects, are represented by attributes of this class. The class holds a reference attribute to the gateway Fog device to which the sensors are attached. We used

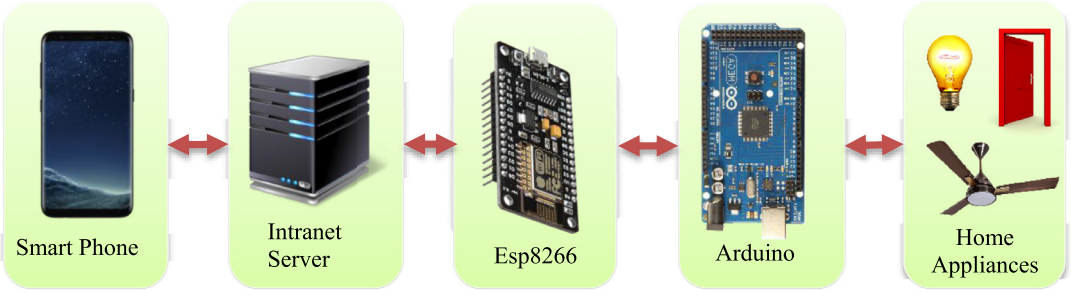
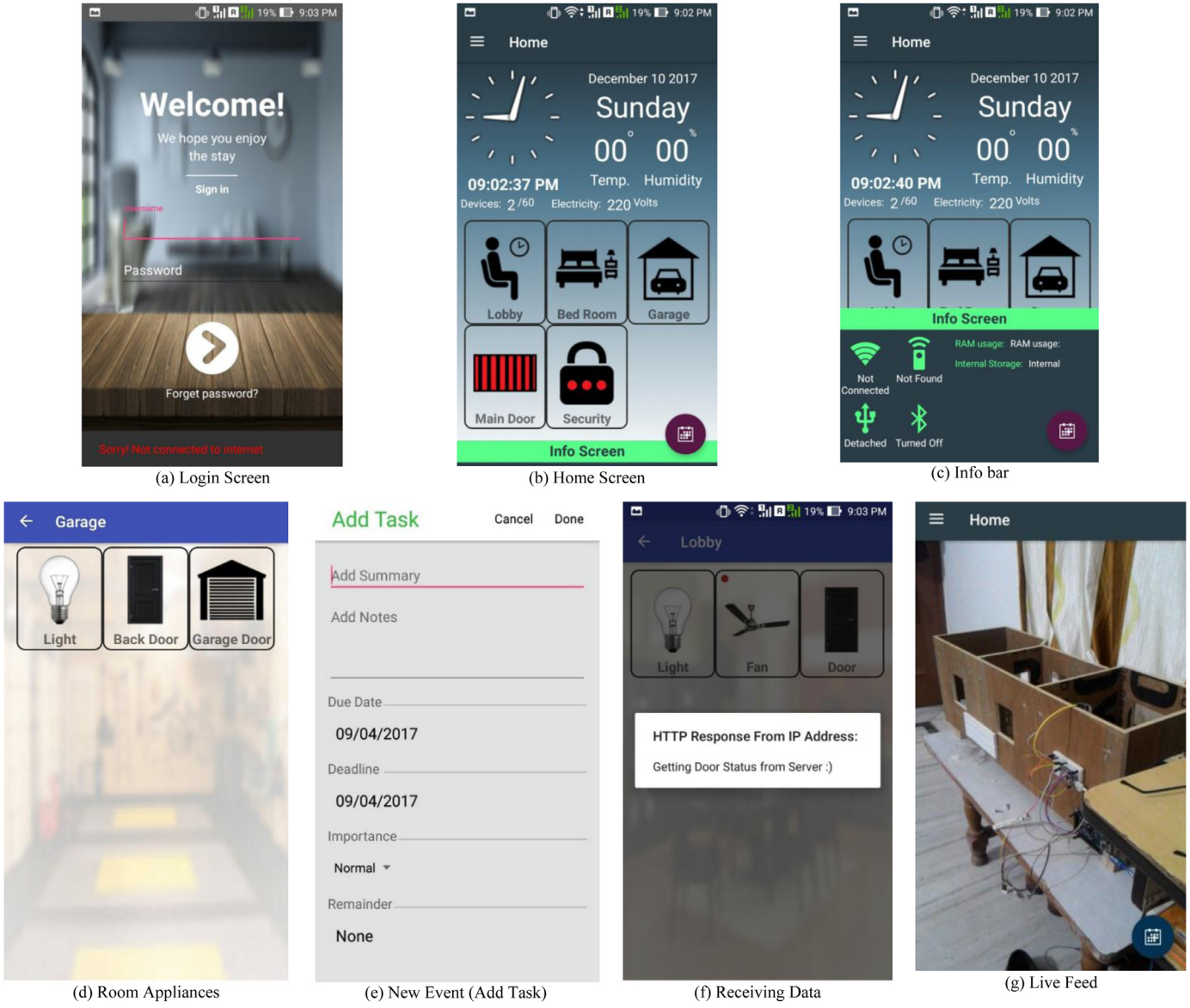


Fig. 7. Interaction of Different Components.



(a) Login Screen (b) Home Screen (c) Info bar

(d) Room Appliances (e) New Event (Add Task) (f) Receiving Data (g) Live Feed

Fig. 8. Different Operations of Smart Home App.

reference attributes of Sensor class to simulate the behavior of different sensors, which are gathering different types of information at IoT layer as shown in Fig. 10.

Actuator: Defines a method to perform an action on arrival of a tuple from an application module to perform different operations of smart home automation as described in Table 3. When user preforms any operation, this class override the defined method to execute corresponding operation. The latency of different devices is defined using attributes of this class as shown in Table 4.

Communication Network: The physical topology (tree topology) of the smart home automation is modeled in iFogSim via FogDevice, Sensor and Actuator classes as described in Fig. 11.

Controller: The Controller object launches the AppModules on their assigned Fog devices following the placement information provided by Module Mapping object and periodically manages the resources of Fog devices as shown in Fig. 11. When the simulation is terminated, the Controller object gather results of cost, network usage and energy consumption during the simulation period from the Fog devices.

Tuple: Central unit of communication amongst Fog entities. The sensors in iFogSim generate tuples that can be referred as tasks in Cloud computing. The creation of tuples (tasks) is event driven and the interval between generating two tuples is set following deterministic distribution while creating the sensors. The instances of Tuple class in iFogSim (Gupta et al., 2017) are represented as tuples, which are inherited from the Cloudlet class of CloudSim (Calheiros et al., 2011). Categorization of tuples is done with its type and destination and source application modules and it is described in Table 3. The length of data encapsulated in the tuple and processing requirements (defined as Million Instructions (MI)) are specified by the attributes of the class.

- **Application:** The smart home application is modeled as a directed acyclic graph (DAG), the vertices of the graph representing modules that perform processing on incoming data and edges denoting data dependencies between modules as shown in Fig. 11. These entities are realized using the following classes.
- **AppModule:** Instances of *AppModule* class represent processing elements of fog applications and realize the vertices of DAG. *AppModule* is implemented by extending the class *PowerVm* in CloudSim. For each incoming tuple, an *AppModule* instance processes it and generates output tuples that are sent

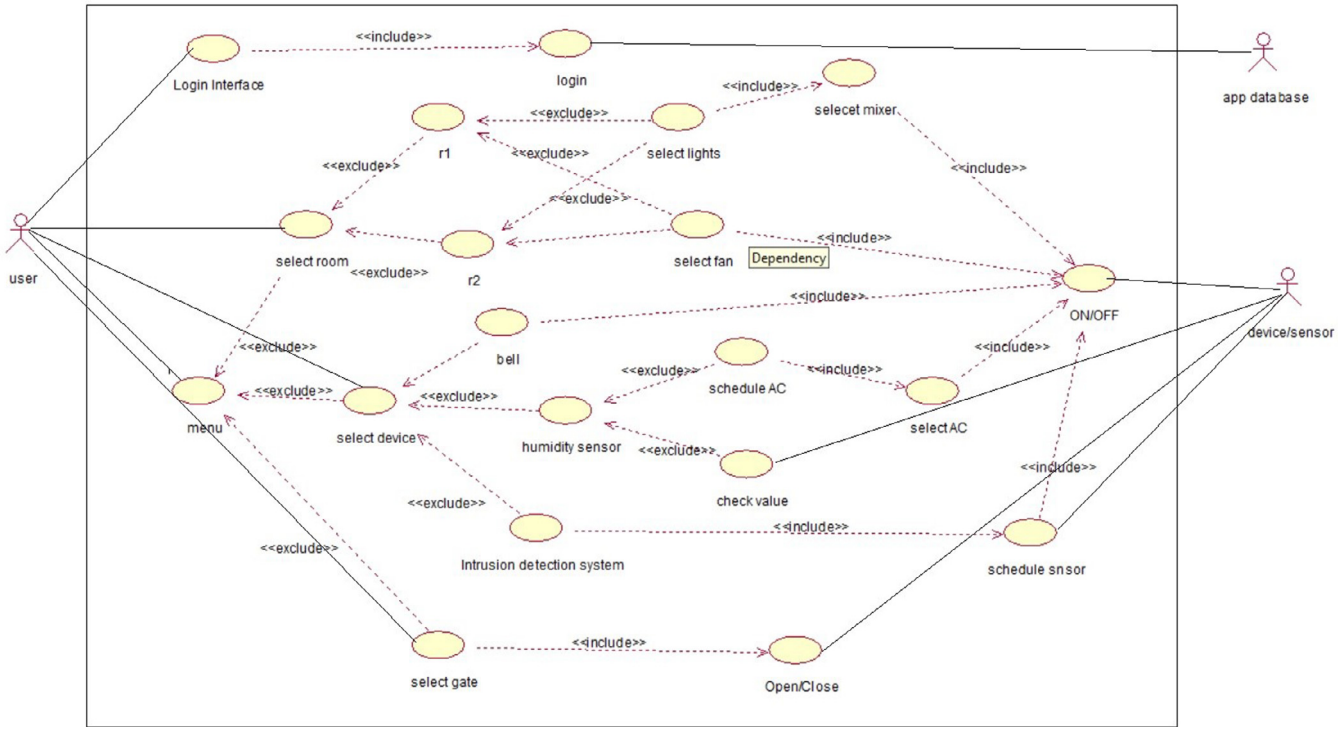


Fig. 9. Use Case Diagram of Smart Home Automation.

to next modules in the DAG. The application modules of SHA are Admin, Owner, System, Appliances, Events, Database and Sensors/IP Camera as shown in Fig. 12 and the description of above-mentioned application modules is given in *Section 4.2.1*.

- **AppEdge:** An AppEdge instance denotes the data dependency between a pair of application modules and represents a directed edge. Each edge is characterized by the type of tuple it carries, which is captured by the tupleType attribute of AppEdge class along with the processing requirements and length of data encapsulated in these tuples. The edges between the application modules in the smart home application are described in Table 3.
- **AppLoop:** AppLoop is an additional class, used for specifying the process-control loops of interest to the user. In iFogSim, the developer can specify the control loops to measure the end-to-end latency. An AppLoop instance is fundamentally a list of modules starting from the origin of the loop to the module where the loop terminates. There are two loops "monitor() and update()" in SHA as shown in Fig. 12.
  - **Monitoring Service:** Fog server manager is used to monitor the resource utilization statistics during scheduling of resources.
  - **Resource Management Service**: We have used *edge-ward placement strategy* for the deployment of application modules close to the edge of the network and customized resource scheduling policy by overriding the method updateAllocatedMips inside the class FogDevice (as discussed in *Section 3*). Proposed resource scheduling policy schedules the fog devices for execution of different application modules to perform various operations of smart home application. The pseudo code for resource scheduling policy is given in Fig. 4.

The detailed description to model and simulate Fog computing environment in iFogSim for different applications can be found in (Gupta et al., 2017).

4.2.1. Application Model: smart home automation

Fig. 12 shows the application model of the Smart Home Automation (SHA), which describes the sequence of operations of an application and their type of tuples.

The application modules are modeled in iFogSim using the AppModule class. As depicted in Fig. 12, there are data dependencies between modules, and these dependences are modeled using AppEdge class in iFogSim. The control loop of interest for SHA application is modeled in iFogSim using AppLoop class. The application receives signals by different sensors and an actuator DISPLAY displays the current status of smart home to the user through preconfigured mobile device. SHA application consists of different major modules as shown in Fig. 12. The functions of these modules are as follows:

1 **Admin**: An administrator can add/remove or configure new smart devices to the Smart Home environment. The other functions of an administrator are: 1) to create, configure or delete user settings via the administration user interface and 2) to reset all settings to defaults or a saved configuration.
2 **Owner**: The Owner of SHA enabled mobile device can select appliances, turn/on off devices, select attributes and receive SMS of an intrusion detection.
3 **System**: The system module automatically choose device if user is connected to home network and notifies the current status of home to user.
4 **Appliances**: The user can control the basic functionalities of their home appliances. For instances, turn on/off, changing the color of lights, speed of fans, etc.
5 **Events**: SHA application provides the functionality of reminding the current occurring events to the user. The user has to add an event in SHA application with the option of reminding or not. If not, application will not remind for event, but the user can have look of event going to occur.
6 **Database**: The SHA application communicates with a database module to send, receive and store sensor information. This module provides encrypted back-end database.

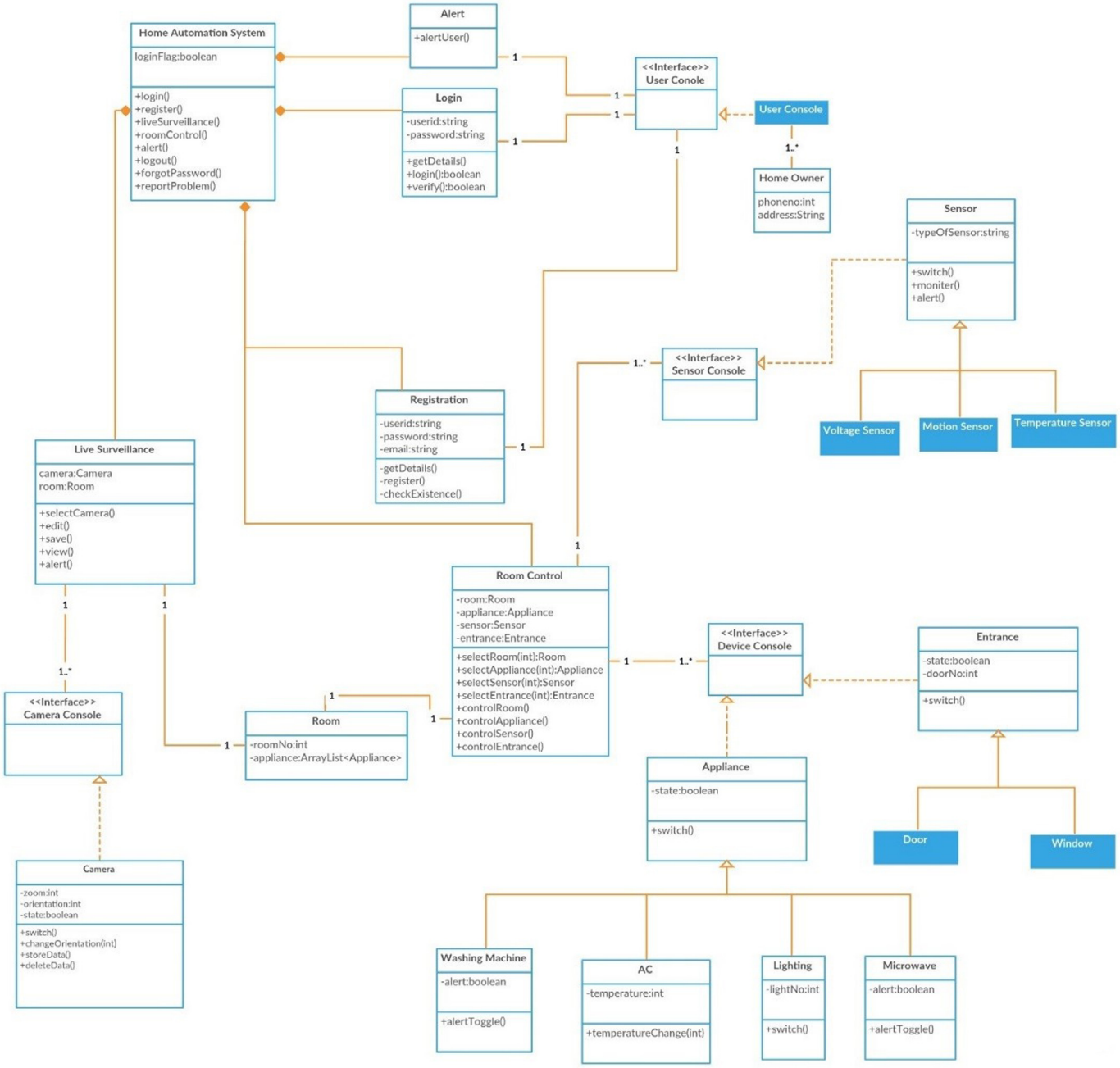


Fig. 10. Class Diagram of Smart Home Automation.

7 **Sensors/IP Camera**: SHA application monitors the data coming from the sensors. For instances, check home temperature and humidity using temperature and humidity sensor, check current power consumption by the house using kWh measuring sensor, etc. SHA application monitors the outside activities of home using live feed camera and intruder detection system. Intruder detection system contains PIR sensors all around the house to detect any proximity to the house and alert the owner of that house.

The properties of tuples (modeled using Tuple class) carried by edges between the modules in the smart home application are described in Table 3.

The latency of different devices from source to destination is described in Table 4.

The configuration (CPU GHz, RAM size and Power) of different fog devices is described in Table 5.

5. Evaluation Results

The experiments have been performed with different QoS parameters, such as response time, latency, energy consumption and network bandwidth.

5.1. Benchmark techniques

To evaluate the performance of the resource management technique ROUTER, we selected compared it against two similar techniques from the literature: Gateway-based Fog Computing (GFC)

Table 3
The description of Intermodule Edges in the Smart Home Application.

Operation Name	Tuple Type	Description	CPU Length (MIPS)	Network Length (Bytes)
Register New Mobile Phone/Device	Add User	Add new user to Smart Home Application	2000	48
Get Status of Event	Return Status	Returns the status of every event after its occurrence	2200	60
Update Information of User	Update User	Update the user details	2800	63
Unregister Mobile Phone/Device	Delete User	Delete user form SHA database	2000	50
Sign Up	Login	User performs login to application in order to get access to device	3500	57
Verification of Registered Device	Verify	Verify the details of user for authentication	2200	45
Choose Home Appliance	Select Appliance	Select the appliance, which can be AC, microwave, fan, light, washing machine etc.	2000	52
Get Status	Check Status	Check the status of the security of home	2200	54
Show Status	Display Status	Display the checked status on mobile display	3100	50
Fog Device Selection	Choose Device	Enable authorized user to choose a communicating device	2200	50
Choose Variables	Select Attributes	Select attributes for *Set Value* Function	3500	55
Assign Value to Variables	Set Value	Enable user to adjust values according to the appliances and device capacities using open adjustment panel.	3000	50
Change Appliance Details	Update	Update the appliance information	2000	50
Turn ON-OFF Electric Appliance	Turn On/Off	Enable the user to turn on/off the chosen appliance	2200	66
Display Task	View Event	Enable the user to add the selected task	3100	65
Create Task	Add Event	Enable the user to add the new tasks and also reminds you about their occurrence.	2700	66
Delete Task	Remove Event	Enable the user to remove the particular task	2300	65
Get Information about Home	Notify User	Notify the current status of home to user	3600	50
Get Sensor Information	Return Signals	Enable the user to learn info about the device from the sensors	3450	55
Watch Live Feed Camera	Sensing/ Monitor	Enable the user to watch live view of outside his house through IP camera.	3500	55

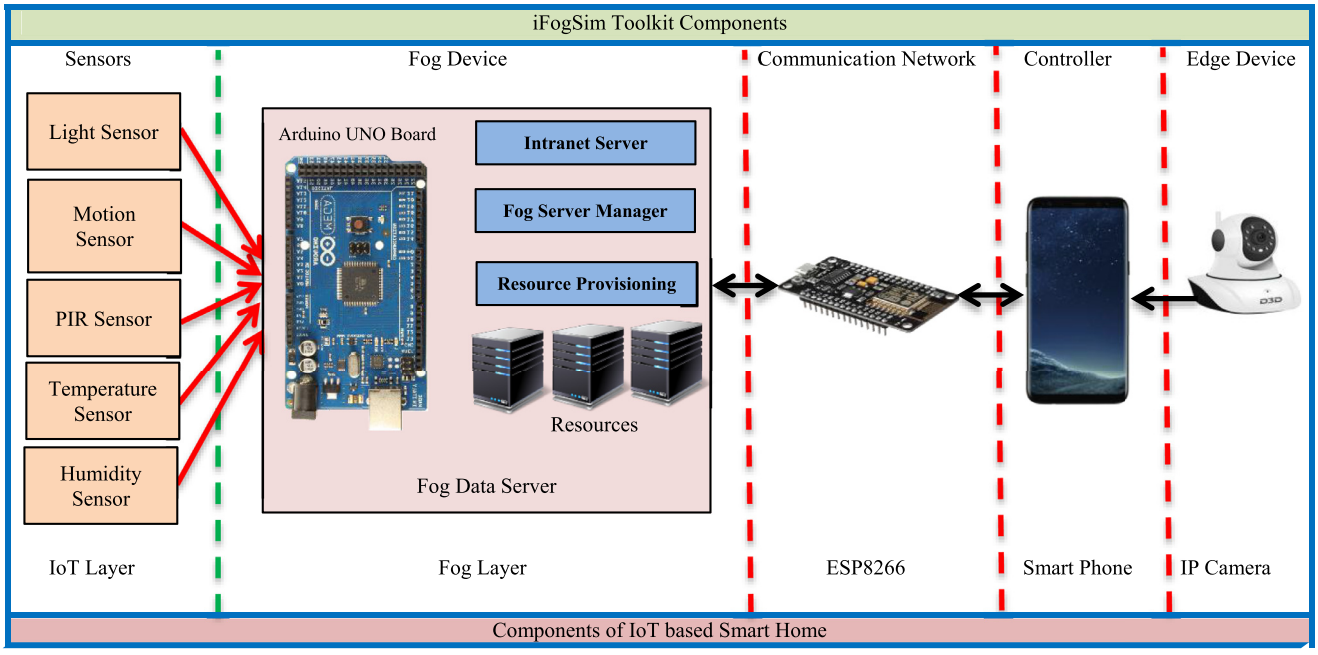


Fig. 11. The Mapping of the Components of Smart Home Automation with iFogSim Toolkit.

technique (Lee et al., 2016) and Virtualization based Resource Provisioning (VRP) technique (Yu et al., 2017) discussed in *Section 2*. We further detail precise functionality and differences with our approach below:

- GFC (Lee et al., 2016) is a gateway-based fog computing architecture for wireless sensors and actuator networks which consists of master and slave nodes, and manages virtual gateway functions, flows, and resources. In GFC, gateway and master node are connected by Ethernet interface, and master node controls the virtual path among slave nodes. Further, slave node performs the resource management for scheduling of resources to process job requests. GFC uses First Come First Serve (FCFS) based resource scheduling algorithm to schedule the resources to optimize response time. The GFC is implemented using CloudSim toolkit by extending new class, which contains the implementation of three fog nodes. Authors have done without using ifogsim by adding new class, which extends the resource scheduling class of CloudSim. They focused only on

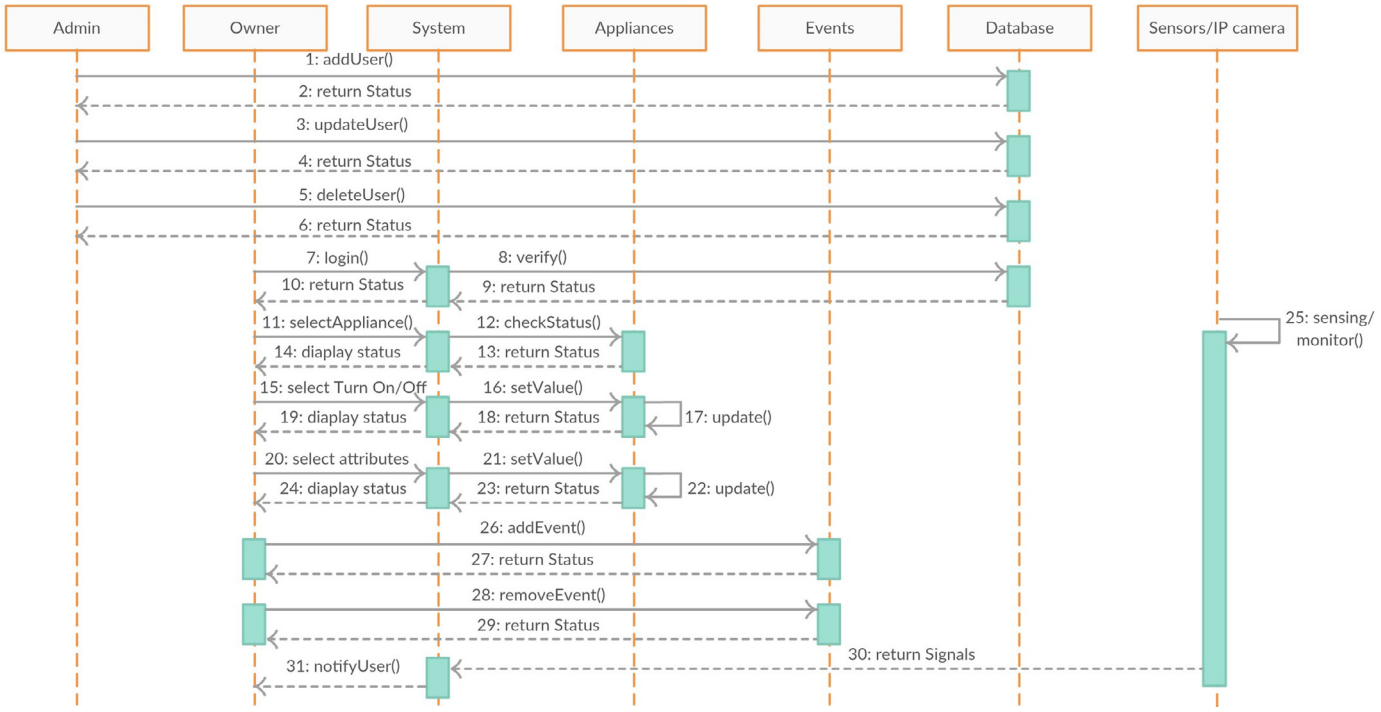


Fig. 12. Application Model of the Smart Home Automation.

Table 4
Latency of different Devices.

Source	Destination	Latency (secs)
IP Camera	Smartphone	6
Smartphone	Wi-Fi Gateway	2
Wi-Fi Gateway	ISP Gateway	4
ISP Gateway	Cloud Data Server (CDS)	100

Table 5
Configuration of different Fog Devices.

Device Type	CPU GHz	RAM (GB)	Power (W)
VM	3.0	4	107.339
Wi-Fi Gateway	3.0	4	107.339
Smartphone	1.6	1	87.53
ISP Gateway	3.0	4	107.339

single performance parameter (response time) with limited fog nodes and problem the starvation can occur in case of larger job request, which further leads delay the execution of pending deadline-oriented jobs.

- VRP algorithm (Yu et al., 2017) uses the concept of parallel and distributed load balancing to develop virtualization based resource scheduling algorithm. VPR uses round robin based scheduling algorithm to process the job requests, which gives fixed time quantum to every job request, which can behave same as FCFS if time quantum is too large. If time quantum is too short much of time is spent in process switching and hence latency and response time increases. Further, the algorithm is tested on Cloud-Analyst simulator that finds proposed solution performs better in terms of energy cost of only processor.

ROUTER operates by using PSO based resource scheduling technique, which uses multi-objective fitness function to optimize the four different QoS parameters simultaneously. ROUTER forwards the job request to CDS if the FDS is not able to process within threshold time. Furthermore, ROUTER is validated via integration with a lab-controlled smart home automation case study described in Section 4, which is further integrated into an application model built within the iFogSim application layer. Both VRP and GFC use dummy jobs to evaluate their performance while ROUTER uses real-time traffic generated from smart home application. In order to evaluate the performance of ROUTER, GFC and VRP effectively, we used the identical simulation environment described in Section 4.

5.2. Analysis results

Network Bandwidth: Fig. 13(a) shows the average network bandwidth of 1789.6 B/s, 2714.45 B/s and 2830.25 B/s for all resource managers ROUTER, GFC, and VRP. It is observable that both GFC and VRP have a similar network bandwidth of 2770 B/s, ROUTER on average uses 1790 B/s, which is 12.36% and 14.43% less than GFC and VRP, respectively. This is because, ROUTER processes data of IoT devices effectively while fulfilling the QoS requirements at runtime. Another reason of better performance is that PSO achieves global minima quickly, which distributes load effectively during scheduling of resources.

Latency: We analyzed the latency of each resource management technique (i.e. the delay before transfer of user requests for job processing). With increasing the number of operations, the value of latency increases as shown in Fig. 13(b). It is observable that ROUTER has a lower latency in contrast to both GFC and VRP (as operations increase). The average value of latency in ROUTER technique is 10.14% and 14.44% less than GFC and VRP respectively. The reason is because ROUTER executes job requests at Fog Data Server (FDS) instead of sending job requests to Cloud Data Server (CDS) which would result in a larger communication delay.

Response Time: Fig. 13(c) shoes the time taken for a system to react to a user request. With increasing the number of operations, response time increases. The average value of response time in ROUTER technique is 14.03% and 15.65% less than GFC and VRP respectively. The reason for reduced response time is due to the request handling mechanism provisioning resources for job requests

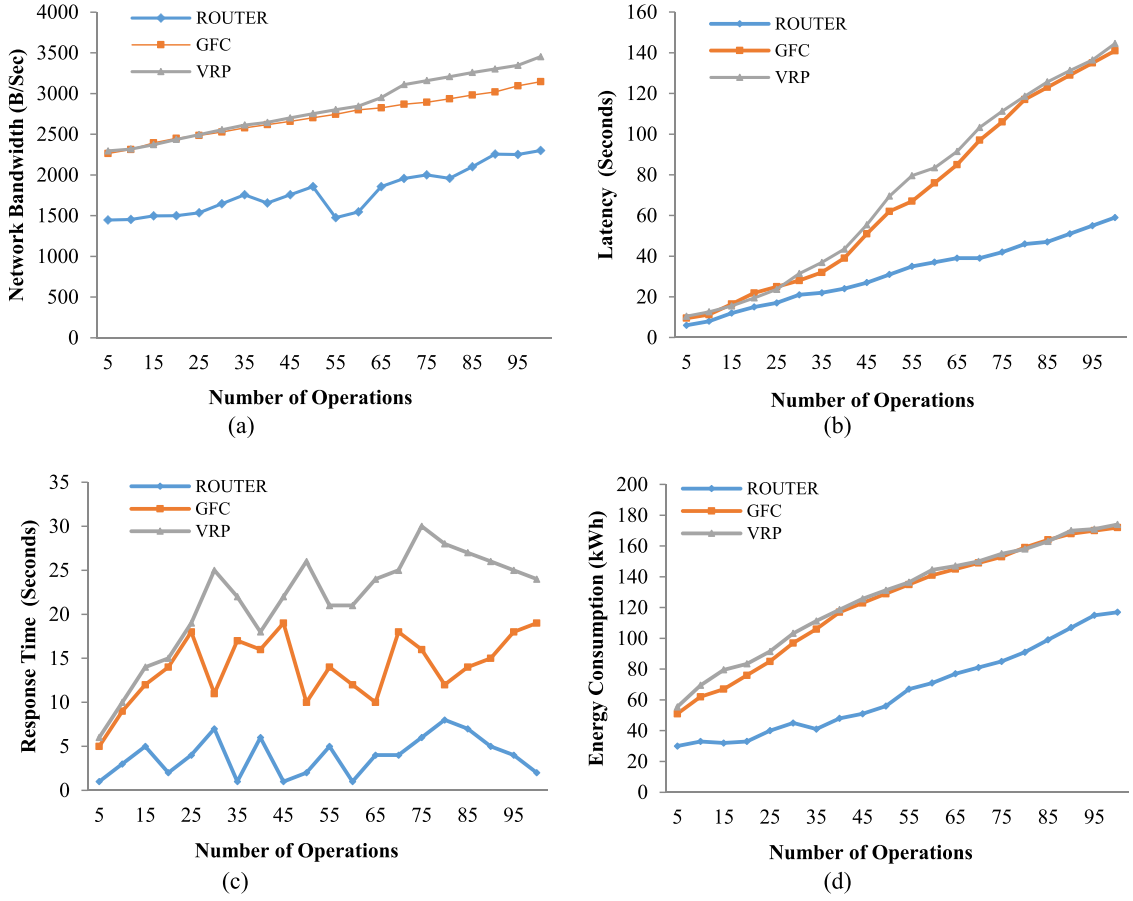


Fig. 13. Evaluation results for resource managers ROUTER, GFC, and VRP: (a) Network Bandwidth, (b) Latency, (c) Response Time, (d) Energy Consumption.

before actual scheduling of resources. Furthermore, ROUTER tracks the state of all resources at each point of time, enables it to take an optimal decision than GFC and VRP.

Energy Consumption: It is the sum of energy consumed by the processor, switching equipment, storage device, network device and other components such as fans, conversion losses (Al-Fuqaha et al., 2015). With increasing the number of operations, the value of energy consumption increases as shown in Fig. 13(d). The average value of energy consumption in ROUTER technique is 12.35% and 13.45% less than GFC and VRP respectively. An effective scheduling of resources using PSO reduces significant amount of network traffic, which leads to reducing the number of idle resources (processor, switching equipment, storage device, network device) that reduces the wastage of energy.

6. Conclusions and future work

In this research paper, QoS-aware resource management technique (ROUTER) is proposed using fog-assisted cloud computing environment, which manages IoT devices efficiently. Furthermore, we designed a case study of IoT based smart home automation to validate the proposed technique. The performance of the proposed technique has been evaluated in Fog computing environment using iFogSim toolkit. Experimental results demonstrate that the proposed technique reduces the network bandwidth by 12.36%, response time by 10.14%, latency by 14.03% and energy consumption by 12.35% and it detects intrusions to provide security.

In future, the proposed technique can be enhanced to work with some other parameters such as scalability, cost, reliability and availability. In fog computing system, trade-off between delay and power consumption is an open research area. Further, the proposed technique will be verified in a real fog environment for the practical realization. In future, ROUTER architecture can be generalized to other fog computing applications such as agriculture, healthcare, weather forecasting, traffic management and smart city.

Acknowledgments

This research work is supported by the Engineering and Physical Sciences Research Council (EPSRC) - (EP/P031617/1), Melbourne-Chindia Cloud Computing (MC3) Research Network and Australian Research Council (DP160102414). We thank Redowan Mahmud, Shashikant Ilager, Sara Kardani, Shreshth Tuli, and Damian Borowiec for their useful suggestions.

References

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M., 2015. Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutor. 17 (4), 2347–2376.
Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: a survey. Comput. Netw. 54 (15), 2787–2805.
Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exper. 41 (1), 23–50.
Chen, C.L.P., Zhang, C.-Y., 2014. Data-intensive applications, challenges, techniques and technologies: a survey on Big Data. Inform. Sci. 275, 314–347.
Chen, G., Yu, J., 2005. Particle swarm optimization algorithm. Inf. Control-Shenyang 34 (3), 318.
Deng, R., Lu, R., Lai, C., Luan, T.H., 2015. Towards power consumption-delay trade-off by workload allocation in cloud-fog computing. In: 2015 IEEE International Conference on Communications (ICC). IEEE, pp. 3909–3914.

Do, C.T., Tran, N.H., Pham, C., Alam, M.G.R., Son, J.H., Hong, C.S., 2015. A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing. In: 2015 International Conference on Information Networking (ICOIN). IEEE, pp. 324–329.
Garraghan, P., Yang, R., Wen, Z., Romanovsky, A., Xu, J., Buyya, R., Ranjan, R., 2018. Emergent Failures: rethinking Cloud Reliability at Scale. IEEE Cloud Comput. 5 (5), 12–21.
Gill, S.S., Arya, R.C., Wander, G.S., Buyya, R., 2018b. Fog-based Smart Healthcare as a Big Data and Cloud Service for Heart Patients using IoT. In: International Conference on Intelligent Data Communication Technologies and Internet of Things. Springer, Cham, pp. 1376–1383.
Gill, S.S., Chana, I., Singh, M., Buyya, R., 2018a. CHOPPER: an Intelligent QoS-aware Autonomic Resource Management Approach for Cloud Computing. Clust. Comput. 21 (2), 1203–1241.
Gu, L., Zeng, D., Guo, S., Barnawi, A., Xiang, Y., 2015. Cost-Efficient Resource Management in Fog Computing Supported Medical CPS. In: IEEE Transactions on Emerging Topics in Computing, pp. 1–12.
Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R., 2017. iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Softw. Pract. Exper. 47 (9), 1275–1296.
Hassan, R., Cohanim, B., Weck, O.D., Venter, G., 2005. A comparison of particle swarm optimization and the genetic algorithm. In: 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference, p. 1897.
Kaur, A., Singh, V.P., Gill, S.S., 2018, August. The Future of Cloud Computing: opportunities, Challenges and Research Trends. In: 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on. IEEE, pp. 213–219.
Lee, W., Nam, K., Roh, H.-G., Kim, S.-H., 2016. A gateway based fog computing architecture for wireless sensors and actuator networks. In: 2016 18th International Conference on Advanced Communication Technology (ICACT). IEEE, pp. 210–213.
Perera, C., Qin, Y., Estrella, J.C., Reiff-Marganiec, S., Vasilakos, A.V., 2017. Fog computing for sustainable smart cities: a survey. ACM Comput. Surv. (CSUR) 50 (3), 32.
Rodriguez, M.A., Buyya, R., 2018. Container-based cluster orchestration systems: a taxonomy and future directions. Softw. Pract. Exper..
Sahoo, K.C., Pati, U.C., 2017. IoT based intrusion detection system using PIR sensor. In: Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2017 2nd IEEE International Conference on. IEEE, pp. 1641–1645.
Singh, S., Chana, I., 2016. A survey on resource scheduling in cloud computing: issues and challenges. J. Grid Comput. 14 (2), 217–264.
Singh, S., Chana, I., Singh, M., Buyya, R., 2016. SOCCER: self-optimization of energy–efficient cloud resources. Clust. Comput. 19 (4), 1787–1800.
Son, J., Buyya, R., 2019. Latency-aware Virtualized Network Function provisioning for distributed edge clouds. J. Syst. Softw. 152, 24–31.
Stojkoska, B.R., Trivodaliev, K., 2017. Enabling internet of things for smart homes through fog computing. In: 2017 25th Telecommunication Forum (TELFOR), Belgrade, pp. 1–4.
White, G., Nallur, V., Clarke, S., 2017. Quality of service approaches in IoT: a systematic mapping. J. Syst. Softw. 132, 186–203.
Yu, L., Jiang, T., Zou, Y., 2017. Fog-assisted Operational Cost Reduction For Cloud Data Centers. IEEE, pp. 13578–13586. Access 5.
Zhang, H., Zhang, Y., Gu, Y., Niyato, D., Han, Z., 2017. A hierarchical game framework for resource management in fog computing. IEEE Commun. Mag. 55 (8), 52–57.

Sukhpal Singh Gill is currently working as a Research Associate at School of Computing and Communications, Lancaster University, UK. Dr. Gill was a Postdoctoral Research Fellow at Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia. He has published more than 45 papers in highly cited journals and conferences as a leading author with H-index 16. His research interests include Software Engineering, Cloud Computing, Internet of Things, Big Data and Fog Computing. For further information on Dr. Gill, please visit: www.ssgill.in

Peter Garraghan is a Lecturer in the School of Computing & Communications, Lancaster University. He has industrial experience building large-scale systems and his research interests include distributed systems, Cloud datacenters, dependability, and energy-efficient computing.

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 also edited several books including "Cloud Computing: Principles and Paradigms" (Wiley Press, USA, Feb 2011). He is one of the highly cited authors in computer science and software engineering worldwide (h-index=124+, g-index=281, 80,000+ citations). Microsoft Academic Search Index ranked Dr. Buyya as #1 author in the world (2005–2016) for both field rating and citations evaluations in the area of Distributed and Parallel Computing. "A Scientometric Analysis of Cloud Computing Literature" by German scientists ranked Dr. Buyya as the World's Top-Cited (#1) Author and the World's Most-Productive (#1) Author in Cloud Computing. Recently, Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" in both 2016 and 2017 by Thomson Reuters, a Fellow of IEEE, and Scopus Researcher of the Year 2017 with Excellence in InnovativeResearch 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 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.