Fog-assisted Cloud based Resource Management for IoT and Big Data Analytics: A Case Study with Smart Home Application

Sukhpal Singh Gill and Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia
sukhpal.gill@unimelb.edu.au, rbuyya@unimelb.edu.au

Abstract

The Fog computing paradigm offers a virtualized intermediate layer to provide data, computation, storage, and networking services between cloud data centers and end users. The megatrend of Internet of Things (IoT) based real-time applications such as health monitoring, disaster management and traffic management requires lesser response time and latency to process a huge amount of data (Big Data). Therefore, fog computing is a solution to improve performance, in which cloud is extended to the edge of the network. In this paper, we proposed a novel resource management technique using fog-assisted cloud computing environment, which manages IoT devices and analyses big data while optimizes performance parameters such as response time, network bandwidth, energy consumption, security and latency. The performance of proposed technique is evaluated using iFogSim toolkit and proposed technique has been verified with the help of a case study of IoT based smart home automation.

Keywords: Fog Computing, Cloud Computing, Big Data, Quality of Service, Internet of Things, Smart Home, Edge Computing, Smart City

1. Introduction

The emerging big data and Internet of Things (IoT) applications such as smart cities, healthcare services etc. are increasing, which needs fast data processing to improve the performance of computing systems [1]. However, these applications are facing large delay and increased response time because computing systems need to transfer data to the cloud and then from cloud to the application, which affect its performance [2]. Fog computing is a solution to reduce the latency, in which cloud is extended to the edge of the network. IoT environment uses fog-assisted cloud computing for processing of data to make smarter decisions in a permitted time period. The data collected from different IoT devices have a large variety and volume (also known as Big Data), which also needs fog servers with high processing power. As a result of regular capturing and collection of datasets, they grow with the velocity of 250 MB/minute or more [3]. The continuous exchange of data in IoT environments is required for efficient decision making and real-time analytics for smart cities [1]. Data is stored and processed on cloud servers after collection and aggregation of data from smart devices of IoT networks.

Due to the changing requirements of IoT applications, cloud servers can be configured using pay-per-use mode to reduce the cost to develop an IoT application. Further, on-demand highly scalable cloud platforms are required to process the volume of data with large magnitude [4]. Cloud data processing cannot meet the requirements of an IoT application when low latency is required because sources of data are distributed across different sites. In addition, due to a large amount of data processing at the cloud, computing system does not process at the required speed which leads to communication failures. Therefore, fog computing is an alternative paradigm which provides the function of networking, compute and storage service between end devices and cloud data centers to process user tasks with minimum latency and response time [5]. Further, Fog computing offers support for developing Internet of Everything (IoE) applications which require predictable/real-time latency (networks of actuators and sensors in industry and transportation). In this new paradigm, application components (dedicated fog devices or routers and smart gateways) between cloud and sensors are running on both edge devices and cloud [6]. To fulfill the IoT application requirements like wide geographical distribution and low latency, fog computing supports distributed data analytics, cloud integration, interface heterogeneity, communication protocols and computing resources. End devices are sending raw data continually to the cloud which makes the cloud a bottleneck of the system [7]. To solve
this problem, fog computing is an effective solution to process received data nearer to the data source itself which improves the scalability to process more number of request without putting burden of processing on the cloud.

Fog computing is proficient in filtering and processing the substantial amount of arriving data on edge devices, creating the data processing architecture distributed and thus scalable. Resource management is the main part of the architecture and comprises components that consistently manage resources in such a way that application-level Quality of Service (QoS) constraints are fulfilled and resource wastage is reduced [16]. To this end, scheduler component plays a key role in keeping track of the state of available resources (information provided by the Monitoring service) to find the best candidates for hosting an application component. A novel resource management technique is required, which can use fog-assisted cloud computing environment to manage IoT devices and consider the important QoS parameters.

The motivation of this research work is to design a Resource Management technique for Fog Computing (RMFP) for fast processing of user tasks, which can improve the user satisfaction. Further, proposed technique improves the performance of IoT based applications and optimizes the QoS parameters such as response time, network bandwidth, energy consumption, security and latency. Finally, proposed technique has been verified with the help of a case study of IoT based smart home automation. 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 results of the evaluation. Section 5 presents conclusions and future work.

2. Related Work

Research on management of IoT based applications using fog computing environment is growing exponentially, but different research issues are still pending to address [18] [20]. This section presents the current research on resource management for fog computing. Deng et al. [8] formulated the workload allocation problem mathematically to study the tradeoff between energy consumption and delay in a cloud-fog computing system. Further, the primal problem is decomposed into three sub-problems to solve independently and demonstrated that fog computing is efficient in reducing transmission latency and communication bandwidth, but network bandwidth and energy consumption are not addressed. Cuong et al. [9] proposed a proximal algorithm for joint resource allocation in the geo-distributed environment and reducing carbon footprints. Further, demonstrated that proposed solution reduces the carbon footprints while offering video streaming as a cloud service. This approach focuses only on energy consumption. Lin et al. [10] proposed a cost-efficient resource management technique which is integrated with a medical cyber-physical system in which virtual machine placement, task distribution and base station association towards cost-efficient system are investigated and shown that proposed solution performs better than the greedy algorithm in terms of energy consumption.

Wangbong et al. [11] proposed a Gateway-based Fog Computing (GFC) architecture for wireless sensors and actuator networks which mainly consists of master and slave nodes, and manages virtual gateway functions, flows, and resources. Experimental results show that GFC performs better in terms of response time. Yu et al. [12] proposed a Virtualization based Resource Provisioning (VRP) algorithm for fog computing and designed an architecture using the concept of parallel and distributed load balancing. Further, the algorithm is tested on Cloud-Analyst simulator that finds proposed solution performs better in terms of energy cost. Stojkoska et al. [13] proposed a conceptual model for smart home using IoT for fog computing and suggested that energy consumption can be reduced by integrating all the renewable energy sources, which are distributed geographically. Zhang et al. [14] proposed a three-layer hierarchical game framework for resource management in fog computing to solve the challenges such as fast data processing, minimum response time, etc. This research work reported that fog devices are more capable to reduce latency as compared to the cloud by experiencing a little larger energy consumption. Therefore, the trade-off between latency and power consumption is required to provide more efficient services.

There is a problem of under-provisioning and over-provisioning of resources in existing resource management techniques [11] [12]. Effective resource management in the virtual environment is required, which can improve resource utilization and user satisfaction. Fog devices have additional compute and storage power, but it is not possible for these devices to provide the resource capacity of cloud, therefore efficient resource management is required to process the user requests in a timely manner. To solve this problem, 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 (RMFP) is described in Table 1.
<table>
<thead>
<tr>
<th>Authors</th>
<th>Applicable Network</th>
<th>Fog Nodes</th>
<th>Nodal collaboration</th>
<th>Focus</th>
<th>Provisioning Metrics (QoS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Response Time</td>
</tr>
<tr>
<td>Deng et al. [8]</td>
<td>Mobile Network</td>
<td>Servers</td>
<td>Master slave</td>
<td>Application management</td>
<td>✓</td>
</tr>
<tr>
<td>Cuong et al. [9]</td>
<td>Vehicular Network</td>
<td>Servers</td>
<td>Peer to Peer</td>
<td>Application management</td>
<td>×</td>
</tr>
<tr>
<td>Lin et al. [10]</td>
<td>Mobile Network</td>
<td>Base stations</td>
<td>Peer to Peer</td>
<td>Network Management</td>
<td>×</td>
</tr>
<tr>
<td>Yu et al. [12]</td>
<td>IoT</td>
<td>Network Devices</td>
<td>Peer to Peer</td>
<td>Resource Management</td>
<td>×</td>
</tr>
<tr>
<td>Zhang et al. [14]</td>
<td>Vehicular Network</td>
<td>Servers</td>
<td>Master slave</td>
<td>Network Management</td>
<td>✓</td>
</tr>
<tr>
<td>RMFP (Proposed)</td>
<td>IoT</td>
<td>Network Devices and Servers</td>
<td>Peer to Peer</td>
<td>Application, Network and Resource Management</td>
<td>✓</td>
</tr>
</tbody>
</table>

The existing works have considered only one or two of the provisioning metrics from response time, latency, energy, security and network bandwidth but all the five QoS parameters have not been considered simultaneously to the best knowledge of the authors. Moreover, most of the existing work focuses on application and network management except GCF [11] and VRP [12]. IoT environment is only considered in [11] and [12]. IoT based resource management is done in GCF [11] and VRP [12], but optimizes only one provisioning parameter. The proposed technique (RMFP) manages cloud resources for the processing of requests of IoT applications with minimum response time, latency, energy, security and network bandwidth as QoS parameters. Finally, RMFP has been verified with the help of a case study of IoT based smart home automation.

3. Fog-assisted Cloud based Resource management for IoT and Big Data analytics

This section presents the proposed resource management technique for IoT and Big Data analytics using the Fog-assisted cloud. The architecture of proposed technique is shown in the Figure 1, in which it is divided into three main layers: 1) Cloud and Big Data, 2) Fog Computing and 3) IoT. Based on functionality, the architecture has four layers, in which big data and cloud are separate layers. The components of the proposed architecture are discussed below:

a) Internet of Things (IoT): This layer (bottom layer) contains end devices like gateways, fog devices, sensors etc. A user can interact with fog paradigm through IoT applications using IoT sensors. The functionality of this layer is enhanced by installing intelligent and innovative applications on end devices.

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

c) Cloud and Big Data: Top layer manages the services which enable the management of resources and processing of big data and IoT tasks. Further, this layer offers the quality of service to applications of fog computing and manage cloud data centers at the infrastructure level. Big Data processing is done at this layer to handle the large data coming from different IoT applications. Many middleware-like services are implemented in this layer to optimize the use of fog and cloud resources for IoT applications. The main aim of these services is to maintain the performance of applications by keeping latency at acceptable levels by executing tasks on fog devices and to decrease the cost of using cloud simultaneously. The different types of services, which are working in a collective manner are described below:
Monitoring: This service monitors the status and performance of the other services and applications and other services can access this information if required.

Knowledge Base: This service stores the past information about demands of resource and application to improve decision-making process in future IoT based user applications.

Job Placement: This service processes the information provided by Monitoring service which contains the information about the state of available cloud at a particular period of time. Further, this information is used to find the best components to contain jobs (tasks) and flows coming for execution. This is further interconnected with the Resource Provisioning service to find the requirements for the allocation of new resources for execution of the current number of flows and tasks.

Raw Data Management: Views about other services and data sources are directly accessed by this layer. Both complex (MapReduce) and simple querying (NOSQL REST APIs or SQL) are used to obtain views from the data for other services.

Resource Information: This service obtains information from Monitoring service and Knowledge Base, which is used to make application and resource profiles.
• **Security**: This service provides the information to applications and services about cryptography, authorization and authentication is required to manage user credentials and requirements on latency.

• **Resource Provisioning**: For hosting of application, this layer attains network, fog and cloud resources. Due to changing the number of hosted applications and requirements of applications with time, resources are allocating dynamically. Other services like Monitoring, Performance Prediction and Profiling provides information for provisioning of resources. For example, execution of low latency-oriented tasks on fog devices when free resources are accessible.

• **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 find the resource requirement to process the large number of pending tasks and flow.

### 3.1 Design Model for Resource Provisioning and Scheduling

Cloud-fog environment is used to design a model for scheduling of resources, which has three layers such as cloud layer (top-most), fog layer (intermediate) and IoT layer or client layer (bottom-most). Figure 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, fog devices and sensors to get information from the end user and forwards the user information to FDS for further processing. FDS comprises one Fog Server Manager (FSM) and the number of Virtual Machines (VMs) to process user requests using the server virtualization at FDS. FSM manages all the virtual resources in FDS for execution of user requests.

![Figure 2: Functional Components](image)

Further, the user request can be forwarded to cloud layer for execution in case of unavailability of resources at FDS. The following steps describe the interaction of cloud layer, fog layer and client layer.

| **Step 1**. All the data centers are organized in cloud and fog layer. Every cloud layer has the number of CDS and fog layer has the number of FDS. |
| **Step 2**. FSM of every FDS checks resource availability and has the accountability to manage VMs. |
| **Step 3**. Firstly, IoT layer forwards the user request to FDS, and then FDS loads the request to its FSM. |
| **Step 4**. FSM processes the user request with following conditions: |

  i) **If** all the demanded resources are available to FDS, **then** it processes the user request and user submits an acknowledgement to FSM regarding execution status.

  ii) **If** only some demanded resources are available to FDS, **then** a user request is divided into a number of subtasks as per resource availability.

  iii) **If** FDS is already executing other requests but initial release (final) state, **then** user request needs to wait; **then** process it after current execution.

  iv) **If** all the resources are executing user requests at one FDS but some are failing during execution, **then** it will again process the user request as in (ii) condition.

  v) **If** all the resources are unavailable in FDS within its fog cluster, **then** the user request is forwarded to CDS.

| **Step 5**. **If** the user has not expected the result of their request within maximum allocated time, **then** user has to wait for processing. |
| **Step 6**. For further processing user request is transmitted to CDS. |
| **Step 7**. CDS provides the resources to the user directly for execution of user request with minimum response time and latency and sends an acknowledgement to respective FSM. |

Figure 3 shows the pseudo code for resource management (provisioning and scheduling) to process the user request in an efficient manner. We have implemented the algorithm in IoT layer and Fog layer using iFogSim toolkit [17] to execute the user requests coming through IoT based applications. There are two types of processing of user requests, one at FDS (denoted by $i_f$) and another at CDS (denoted by $i_c$), which is requested by FDS in case of unavailability.
of resources. Initially, IoT layer submits user request ($i_f$) to the closest FDS (say $FDS_1$) for fast processing and FSM checks whether the resource demand is satisfied or not at $FDS_1$. If $FDS_1$ satisfies the demand of user request ($i_f$) then FSM starts its execution and tracks execution status. If $FDS_1$ satisfies the partial demand of user request ($i_f$) then FSM divides the Total task $T$ is divided into a number of subtasks ($T=t_1 \ast t_2 \ast t_3 \ast \ldots \ldots \ast t_n$) as per resource availability and starts execution at different VMs. For example: if there are 4 resources available, then Task $T$ is divided into 4 tasks ($T=\frac{T}{4}$).

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request $i_f$:</td>
<td>Request from IoT layer to FDS.</td>
</tr>
<tr>
<td>Request $i_c$:</td>
<td>Request from FDS to Cloud Data Server (CDS).</td>
</tr>
<tr>
<td>$CD$:</td>
<td>CDS process the request.</td>
</tr>
<tr>
<td>$FDS$:</td>
<td>FDS process the request.</td>
</tr>
<tr>
<td>FSM:</td>
<td>Each FDS comprises FSM</td>
</tr>
<tr>
<td>$VM$:</td>
<td>VMs at FDS.</td>
</tr>
<tr>
<td>$M_{time}$:</td>
<td>Minimum constraint time to release the resources.</td>
</tr>
<tr>
<td>$T_{time}$:</td>
<td>Threshold value for each request $i_f$.</td>
</tr>
<tr>
<td>$T$:</td>
<td>Total task</td>
</tr>
<tr>
<td>$t_1$:</td>
<td>Number of resources available in $FDS_1$</td>
</tr>
<tr>
<td>$t_2$:</td>
<td>Number of resources available in $FDS_2$</td>
</tr>
<tr>
<td>$t_1, t_2, t_3, \ldots, t_n$:</td>
<td>Sub tasks of total task</td>
</tr>
</tbody>
</table>

1. for each request $i_f$, 2. Every user request $i_f$ is submit to closest location $FDS$ as user’s location 3. Every $FDS$ processes the user request 4. FSM start processing of the service request 5. if all demanding resources (VMs) are available to first $FDS$ then 6. $FDS$ processes $i_f$ and submits an acknowledgement to FSM regarding execution status 7. endif 8. if only some demanding resources (VMs) are available to $FDS$ then 9. Total task $T$ is divided into number of subtasks as per resource availability 10. $T=t_1 \ast t_2 \ast t_3 \ast \ldots \ldots \ast t_n$ 11. endif 12. if $FDS$s already executing other requests but at initial release state then 13. User have to wait for $M_{time}$ and then submits its request to $FDS$ 14. endif 15. if all the resources (VMs) are busy with other execution at one $FDS$ but some are failing during execution 16. then $GOTO$ step 4. 17. if all the resources (VMs) are unavailable in $FDS$ within its fog cluster then 18. Request $i_f$ is propagated to $CDS$ over appropriate communication network 19. endif 20. if $T_{time} \leq M_{time}$ then 21. User will receive a message “$Wait$ for processing” 22. else $GOTO$ step 4 23. endif 24. for every request $i_c$ 25. Every request $i_c$ is referring to closest location $CDS$ as $FDS$ location 26. Every $CDS$ processes the service request 27. $CDS$ loads the result for user 28. $CDS$ sends an acknowledgement to respective FSM 29. end for 30. end for

Figure 3: Pseudo Code for Resource Provisioning and Scheduling in Fog Computing

If all the resources are busy at $FDS_1$ but it is at initial release state, then the user request ($i_f$) must wait for Minimum Constraint Time ($M_{time}$) to release the resources then start execution. If all the resources are busy with other execution at $FDS_1$ but some requests are failing during execution, then FSM finds another $FDS_2$. If all the resources are unavailable in all the FDS within its fog cluster, then user request ($i_f$) is propagated to 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 sends the user request ($i_c$) to closest CDS for further processing. CDS provides the resources for execution of user request with minimum response time and latency and sends an acknowledgement to respective FSM. The value of latency and response time is predefined and both the parameter has 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 after every interval.

4. Performance Evaluation

The proposed technique has been tested in fog computing based simulated environment using CloudSim [15] and iFogSim [17]. In this research work, basic event simulation functionalities of CloudSim [15] have been used for implementing functionalities of iFogSim architecture. CloudSim entities like datacenters and communication among datacenters through message sending operations are considered. Therefore, the core CloudSim layer is responsible for handling events between fog computing components in iFogSim [17]. iFogSim implementation is established by simulated services and entities. The proposed technique has been verified with the help of the case study of smart home automation.

4.1 Case Study: IoT based Smart Home Automation

To validate the proposed technique, the case study of IoT based smart home automation is demonstrated in this section. In recent years, there has been a huge boom in the world of intelligent devices (IoT) for home automation. All such systems are designed with a single agenda to ease the human and appliance interaction. Although we can easily use these components individually, but every component has its own configuration, which adds additional overhead to the processing mechanism and makes it complex. This case study presents the solution for connecting multiple devices (IoT devices) through an application which can be accessed wirelessly using a smartphone and provides security at the home automatically. Figure 4 describes the integration of different components such as smartphone, Intranet server, ESP8266 (ESpruino Pico), Arduino board and different home appliances such as AC, fan, bulb and doors.

![Interaction of Different Components](image)

**Figure 4: Interaction of Different Components**

The components are interacting with each other in 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, where ESP sends the HTTP packet to initiate the connection. This data is then further processed at the ESP8266 module.

- **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**: Arduino then directs the pins received in the signal to turn ON/OFF home appliances as per the requirements of a user. The status of the device is then updated in the Android application.

- **Intrusion/Breach Detections**: When the security feature in the Smart Home App is turned ON, the Passive Infra-Red (PIR) sensor [19] will be turned ON to detect the heat signals and motion inside the room. If any movement is detected, it will turn on the 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:** In this case, the device to be used as the IP camera has to be the first device to be connected to the Wi-Fi hotspot for getting the live view in the App. Therefore, when the server is started in the device to project the video, its IP address will be used inside Smart Home App to get the live view.

Figure 5 shows user interaction with smart home App and user can control the basic operations such as choose device, turn on/off home appliances, changing the color of lights, the speed of fans, change attributes, get sensor info, add/view event and watch live feed camera. The user should have valid authentication details to access this App. The home screen shows the live view of different places and information bar, which contains the information of sensors 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 “Add Task”.

(a) Login Screen  
(b) Home Screen  
(c) Info bar  
(d) Room Appliances  
(e) New Event  
(f) Receiving Data

Figure 5: Different Operations of Smart Home App
Figure 6 shows the use case diagram of smart home automation, which describes the interaction of different actors such as user, App database and sensors.

Figure 7 shows the class diagram of smart home automation to describe the interaction of different classes with their different functions.

4.2 Implementation of Proposed Technique in iFogSim

Figure 8 describes the mapping of the components of smart home automation with simulation environment i.e. iFogSim toolkit [17]. Different sensors are used to control different activities such as light, motor speed, 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 of iFogSim are customized in this research work to implement IoT based smart home application using fog environment:

- **FogDevice**: This class describes the hardware features of Fog devices and their relations with sensors and other Fog devices. Further, we have extended PowerDatacenter class of CloudSim [15] to make main attributes of the FogDevice class, which can 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. We have developed a Listener module, which receives the data from different sensors as shown in Figure 8.
- **Sensor:** In iFogSim toolkit, IoT sensors are represented by instances of the Sensor class. Features of a sensor, extending from its connectivity to output aspects, which 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 references attributes of Sensor class to simulate the behavior of different sensors, which are gathering different types of information at IoT layer as shown in Figure 7.

![Class Diagram of Smart Home Automation](image)

Figure 7: Class Diagram of Smart Home Automation
- **Actuator**: This class 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 2. When user performs any operation, this class overrides the defined method to execute corresponding operation. The latency of different devices is defined using attributes of this class as shown in Table 3.

- **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 Figure 2.

- **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 Figure 8. When the simulation is terminated, the Controller object gathers results of cost, network usage and energy consumption during the simulation period from the Fog devices.

- **Tuple**: It makes the central unit of communication among entities in Fog. The sensors in iFogSim generates 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 [21]. The instances of Tuple class in iFogSim [17] are represented as tuples, which are inherited from the Cloudlet class of CloudSim [15]. Categorization of tuples is done with its type and destination and source application modules and it is described in Table 2. 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 Figure 8. 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 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 Figure 8 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 2.

- **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 Figure 8.

- **Monitoring Service**: Fog server manager is used to monitor the resource utilization statistics.

- **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 figure 3.

The detailed description to model and simulate Fog computing environment in ifogSim for different applications can be found in [21].

### 4.2.1 Application Model: Smart Home Automation

Figure 8 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 AppModule class. As depicted in Figure 8, there are data dependencies between modules, and these dependences are modeled using AppEdge class in ifogSim. Finally, the control loop of interest for SHA application is modeled in ifogSim using AppLoop class. The application is fed 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 Figure 8 and 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.
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 2.

Figure 8: Application Model of the Smart Home Automation

Table 2: The description of Intermodule Edges in the Smart Home Application

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Tuple Type</th>
<th>Description</th>
<th>CPU Length (MIPS)</th>
<th>Network Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register New Mobile Phone/Device</td>
<td>Add User</td>
<td>To add new user to Smart Home Application</td>
<td>2000</td>
<td>48</td>
</tr>
<tr>
<td>Get Status of Event</td>
<td>Return Status</td>
<td>It returns the status of every event after its occurrence</td>
<td>2200</td>
<td>60</td>
</tr>
<tr>
<td>Update Information of User</td>
<td>Update User</td>
<td>To update the user details</td>
<td>2800</td>
<td>63</td>
</tr>
<tr>
<td>Unregister Mobile Phone/Device</td>
<td>Delete User</td>
<td>To delete user form SHA database</td>
<td>2000</td>
<td>50</td>
</tr>
<tr>
<td>Sign Up</td>
<td>Login</td>
<td>User perform login to application in order to get access to device</td>
<td>3500</td>
<td>57</td>
</tr>
<tr>
<td>Verification of Registered Device</td>
<td>Verify</td>
<td>To verify the details of user for authentication</td>
<td>2200</td>
<td>45</td>
</tr>
<tr>
<td>Choose Home Appliance</td>
<td>Select Appliance</td>
<td>To select the appliance, which can be AC, microwave, fan, light, washing machine etc.</td>
<td>2000</td>
<td>52</td>
</tr>
<tr>
<td>Get Status</td>
<td>Check Status</td>
<td>To check the status of the security of home</td>
<td>2200</td>
<td>54</td>
</tr>
<tr>
<td>Show Status</td>
<td>Display Status</td>
<td>To display the checked status on mobile display</td>
<td>3100</td>
<td>50</td>
</tr>
<tr>
<td>Fog Device Selection</td>
<td>Choose Device</td>
<td>To enable the authorized user to choose a communicating</td>
<td>2200</td>
<td>50</td>
</tr>
</tbody>
</table>
The latency of different devices from source to destination is described in Table 3.

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Latency (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Camera</td>
<td>Smartphone</td>
<td>6</td>
</tr>
<tr>
<td>Smartphone</td>
<td>Wi-Fi Gateway</td>
<td>2</td>
</tr>
<tr>
<td>Wi-Fi Gateway</td>
<td>ISP Gateway</td>
<td>4</td>
</tr>
<tr>
<td>ISP Gateway</td>
<td>Cloud Data Server (CDS)</td>
<td>100</td>
</tr>
</tbody>
</table>

The configuration (CPU GHz, RAM size and Power) of different fog devices is described in Table 4.

<table>
<thead>
<tr>
<th>Device Type</th>
<th>CPU GHz</th>
<th>RAM (GB)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>3.0</td>
<td>4</td>
<td>107.339</td>
</tr>
<tr>
<td>Wi-Fi Gateway</td>
<td>3.0</td>
<td>4</td>
<td>107.339</td>
</tr>
<tr>
<td>Smartphone</td>
<td>1.6</td>
<td>1</td>
<td>87.53</td>
</tr>
<tr>
<td>ISP Gateway</td>
<td>3.0</td>
<td>4</td>
<td>107.339</td>
</tr>
</tbody>
</table>

4.3 Experimental Results

The experiments have been performed with different QoS parameters, such as response time, latency, energy consumption, network bandwidth and intrusion detections. To test the performance of proposed technique (Resource Management technique for Fog Computing (RMFP)), we selected two similar techniques from literature: Gateway-based Fog Computing (GFC) technique [11] and Virtualization based Resource Provisioning (VRP) technique [12] as discussed in Section 2. Note: The detailed description of metrics is given in previous research works [4] [5].

Test Case 1 - Network Bandwidth: It is defined as the number of bits transferred/received in one second. We have analyzed the network bandwidth consumed for RMFP, GFC and VRP resource management techniques with the different number of operations as shown in Figure 10. With increasing the number of operations, the value of network bandwidth increases. The average value of network bandwidth in RMFP technique is 9.36% and 12.22% less than GFC and VRP respectively.
**Test Case 2 - Latency:** It is defined as the delay before the transfer of user request for processing. We have analyzed the value of latency for RMFP, GFC and VRP resource management techniques with the different number of operations as shown in Figure 11. With increasing the number of operations, the value of latency increases. The average value of latency in RMFP technique is 7.42% and 8.12% less than GFC and VRP respectively. The reason is because RMFP executes user requests at Fog Data Server (FDS) instead of sending requests to Cloud Data Server (CDS), which further improves the performance of proposed technique in an efficient manner.

**Test Case 3 - Response Time:** It is defined as the length of time taken for a system to react to a user request. We have analyzed the value of response time for RMFP, GFC and VRP resource management techniques with the different number of operations as shown in Figure 12. With increasing the number of operations, response time increases. The average value of response time in RMFP technique is 13.73% and 15.45% less than GFC and VRP respectively.
**Test Case 4 - 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 loss [2]. We have analyzed the value of energy consumption for RMFP, GFC and VRP resource management techniques with the different number of operations as shown in Figure 13. With increasing the number of operations, the value of energy consumption increases. The average value of energy consumption in RMFP technique is 10.15% and 10.93% less than GFC and VRP respectively.

![Figure 13: Effect of Change in Number of Operations on Energy Consumption](image)

**Test Case 5 - Intrusion Detection**: The proposed technique (RMFP) uses Passive Infra-Red (PIR) sensor [19] to detect the motion (intrusions) inside the room. If any movement is detected, it will turn on the 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. Figure 14 shows the occurrence and detection of number of intrusions.

![Figure 14: Occurrence and Detection of Number of Intrusions](image)

**5. Conclusions and Future Work**

In this research paper, QoS-aware resource management technique is proposed using fog-assisted cloud computing environment, which manages IoT devices efficiently. Further, 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 12.22%, response time by 15.45%, latency by 8.12% and energy consumption by 10.93% 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.
Acknowledgement

One of the authors, Dr. Sukhpal Singh Gill [Postdoctoral Research Fellow], gratefully acknowledges the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia, for awarding him the Fellowship to carry out this research work. We would like to thank all the anonymous reviewers for their valuable comments and suggestions for improving the article. We thank Redowan Mahmud, Shashikant Ilager and Sara Kardani for their useful suggestions.

References


Author’s Bibliography

Sukhpal Singh Gill is a Postdoctoral Research Fellow at Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia. Dr. Gill joined Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (TIET), Patiala, India, in 2016 as a Faculty. Dr. Gill obtained the Degree of Master of Engineering in Software Engineering from TIET, as well as a Doctoral Degree specialization in “Autonomic Cloud Computing” from TIET. Dr. Gill received the Gold Medal in Master of Engineering in Software Engineering. He is a DST Inspire Fellow [2013-2016] and worked as a SRF-Professional on DST Project, Government of India. He has done certifications in Cloud Computing Fundamentals, including Introduction to Cloud Computing and Aneka Platform (US Patented) by
ManjraSoft Pty Ltd, Australia and Certification of Rational Software Architect (RSA) by IBM India. His research interests include Software Engineering, Cloud Computing, Internet of Things, Big Data and Fog Computing. He has more than 40 research publications in reputed journals and conferences. For further information on Dr. Gill, please visit: www.ssgill.in

Dr. 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=118, g-index=255, 72,200+ 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 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 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