# FogBus: A simplified framework for implementing Fog, Edge and Cloud computing applications/research

Shreshth Tuli, Shikhar Tuli, Redowan Mahmud, Rajkumar Buyya

## Abstract

Fog and Edge computing are rapidly transforming IoT and networking ecosystems. They are being used in different ways to improve the communication and simplify day-to-day tasks of public. However no flexible implementation framework/platform is available to realize such an environment. In this paper, we present “FogBus”, a simplified framework for implementing Fog, Edge and Cloud computing applications/research. The framework is built using widely used platforms like Java and web interfaces, to allow the diverse range of embedded devices to be used to realize a fog environment. The system has been tested on a real scenario of Sleep Apnea analysis and deployed on Raspberry Pi’s for computation.

## Introduction

## Model

FogBus is a lightweight and platform independent framework which allows an end-to-end implementation of fog, edge and cloud computing environment. It is designed to run on almost all IoT/embedded devices and support a wide range of machines in terms of their computation capabilities. A driving factor for making this framework independent of operating system and machine architecture is to allow the diversity of embedded devices in the IoT world to be used in the network. The framework integrates all edge devices (using a lightweight balancing framework for small-scale computation) and also cloud/other heavy devices (using the Aneka framework for heavy computation) and provides an end to end solution for integrating computation at different levels to provide better latency (by allocating simple and small-scale work to edge nodes) and high computation power (by allocating heavy computation work to cloud), which in effect to the end user feels like high computation with low latency. This is the fundamental ideology on which fog computing is based.

All devices interact at different levels with different devices. The overall model is based on Master-Slave design where the Master node receives data from sensor for analysis/computation. The Master node then sends this data as a task to different workers: Local worker, Private Cloud or Public Cloud. Therefore, the core components of network include Master Node, Worker Nodes, Sensors and Cloud. All other components including gateways, switches and other interface devices ‘glue’ the core components together.

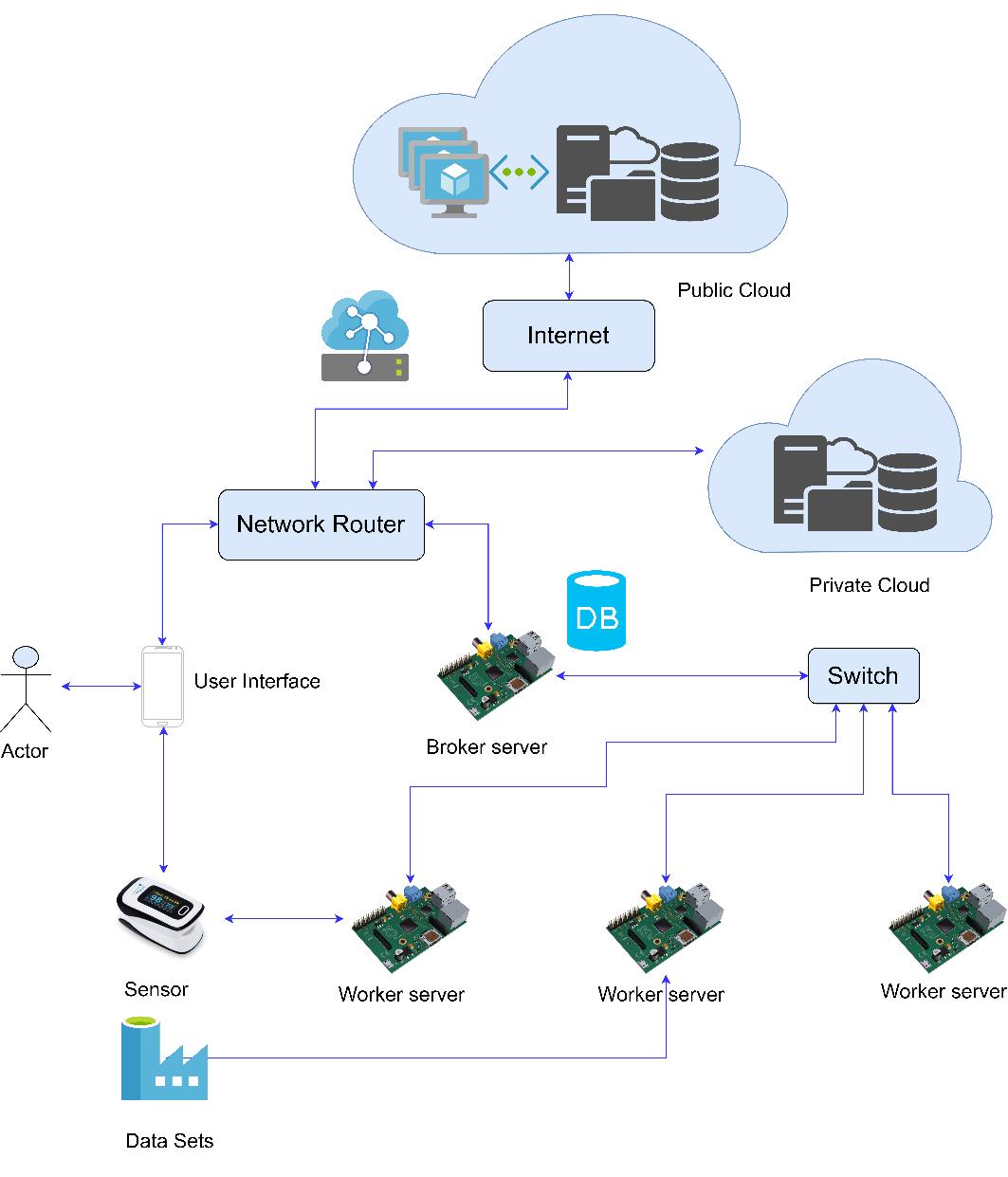
The model is developed to provide a framework the aims of which include:

1. Allow input of data from sensors and output of information/actions to end user/actuators
2. Provide an end-to-end and seamless communication between different fog and cloud resources
3. Provide a platform to easily form an integrated system that can use this interaction to provide services to the end user
4. Implement resource management policies and allow easy extension/implementation of such policies for efficient load distribution
5. Set up a friendly interface between the framework and end user

### Network Level Model

As described before, the core components of the network model are shown in Figure 1. The User connects the local network and interacts with the Master Node using an interface which can be an Android or other smart device. This interface allows communication between sensor/actuator with the Master Node.

The Master Node also referred to as the Broker Node collects data from the sensor uses it itself for analysis or sends it to one of the followings:

1. Worker Node in the local network
2. Private Cloud
3. Public Cloud or datacenter

The Master Node also collects results/actions from the devices mentioned above and send it to user interface/actuator. The master or worker nodes can also receive data directly from data sets and perform actions accordingly.

It is in the hands of the user, which service to run and which master to connect to. There can be Master nodes dedicated for specific tasks and in IoT environment multiple services can even run simultaneously requiring connections with multiple masters and/or multiple service running on same master node.

Figure 1: Network Level Model

### Sequence of tasks

A defined sequence of tasks and communication of data helps reduce failure rate and makes it easier to understand the flow of data through the system. Figure 2 below shows the sequence of tasks and data flow in the network.

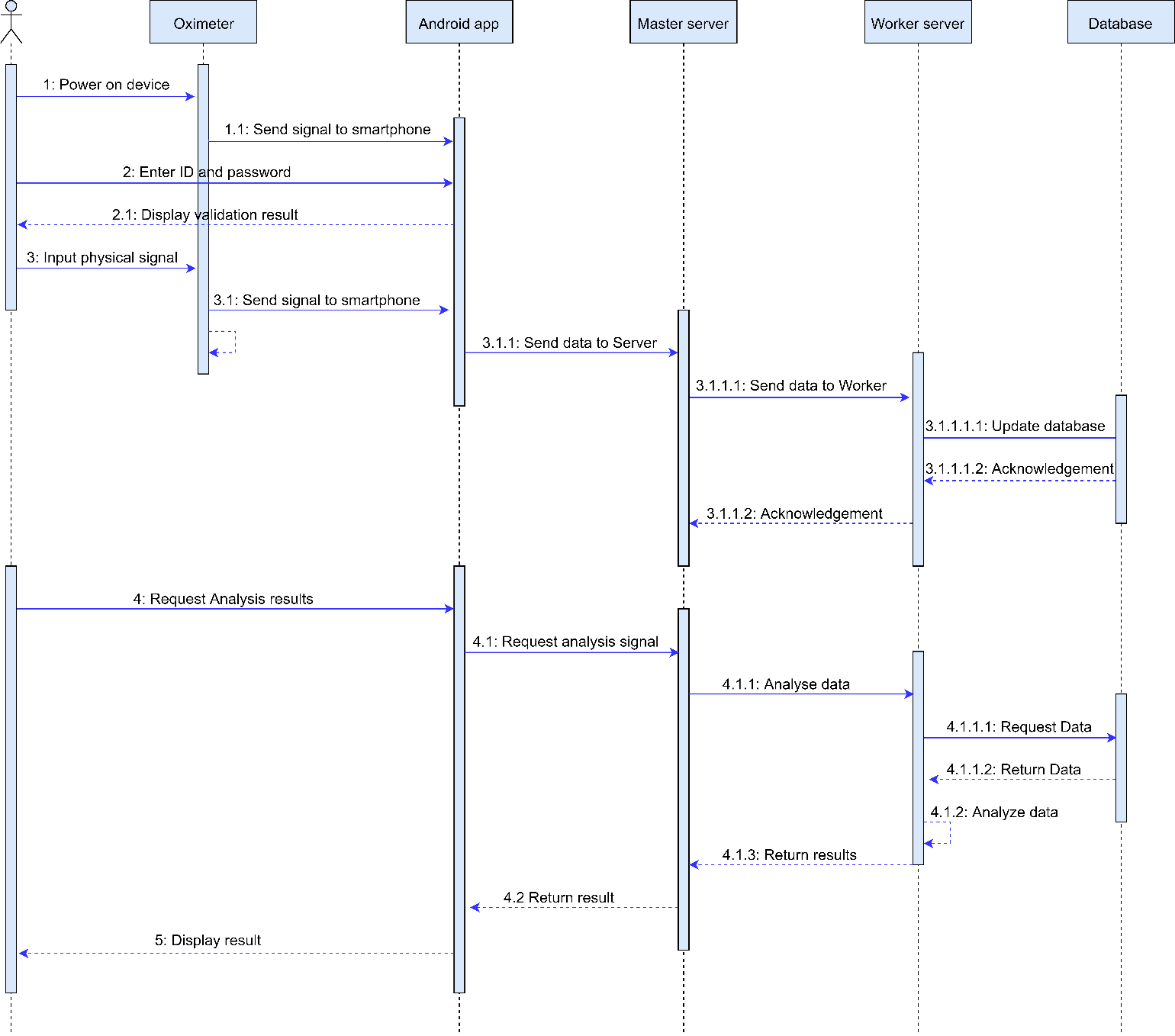


Figure 2: UML Sequence Diagram showing data flow in the network

### Functional Components

1. **Resource Management:**

As devices at different levels of the network, as shown in the ‘Network Level Model’, offer range of computational capabilities and latencies, it is important to distribute work accordingly to ensure optimum QoS. The devices at the ‘edge’ of the network provide very low latency due to close proximity with the user, sensors and actuators, but have tendency to have significantly low computation power and thus higher execution times. The devices far away from users like those on the cloud provide significantly higher latency but due to their much higher computation capabilities provide much lesser execution time. There is thus a trade-off between network propagation latency and computation latency. Depending on different scenarios, different resource management techniques are used.

There is scope of considerable reduction of latency caused due to propagation of data if low-computation workload is allocated to edge devices. Even though their execution time would be higher but the critical factor here would be the time caused by data transfer over the network. On the other hand, significant reduction in time can be achieved if high-computation workload is sent to more powerful machines like cloud because in such tasks edge devices might lead to deadlocks and cause the whole system to collapse.

The Resource manager of the model should decide the following:

1. Allocate task to a local worker node or not: As the latency offered by local machines is much lower in terms of the network propagation of data this is a crucial decision to be taken by the master node. Sending to cloud would require considerable overhead and might not be suitable for real-time and mission critical tasks like healthcare, robotics, etc.
2. Allocate task to itself or not: Some small tasks can be performed by the master itself. As the master deals with the load balancing and other tasks like maintaining integrity of data, it might be that the master node is itself busy. Even then some small tasks may arise on rare occasions which require immediate results. At such stages, the master may decide to perform it itself to eliminate data propagation over the network and other authentication and handshaking protocols.

It is a design decision in the implementation strategy how these decisions are taken. Some parameters that are important for taking such decisions include:

1. Quality of results: Each application has its own quality thresholds. As the required quality of results increases, more complex algorithms may be required. Sometimes the algorithm remains same but more number of iterations are required to reach the required confidence threshold. This has a direct impact on the computation difficulty and how well and fast a machine is able to perform repetitive tasks.
2. Deadline: The time in which the results are required plays a crucial role in determining the load balancing scheme. As hinted earlier, devices at different levels have different execution times and data propagation times. Based on data flow rate and application runtime it is important to balance tasks among different levels of devices.
3. Frequency of data sharing: Closely related to point 2, some tasks might require small amount of data to be analyzed but with high frequency for example -----. In such cases execution time needs to be very low and network bandwidth very high. The device themselves should allow high I/O data flow.
4. Computation Complexity of Application: The order of complexity of application determines the number of task request to be sent to cloud. For very high complexity tasks it might me suitable to only use cloud resources to prevent bottlenecking of the software because of low computational capabilities of edge nodes.
5. Cost of Computation: Different levels have different currency for computation. Devices over the cloud are metered and are charged based on time of usage. Embedded devices in IoT applications are charged based on the initial physical component cost and then the cost of energy expenditure for functioning of these devices. By normalizing these different currencies using conversion factors, one can determine the cumulative cost for running the framework per unit time/task/user. Inherent to the model requirements, but depending on the scenario, this cost needs to be minimized.

Different load balancing algorithms can be implemented that specialize in improving some of these parameters based on user needs. A weighted sum of quantifiable factor of each of these parameters and the weights themselves would determine which balancing technique should be used for a specific situation.

**2. Data Input-Output**

There exist several ways of interacting and sharing data from sensors and among fog nodes. It is crucial that this data sharing mechanism is failsafe in terms of data theft and fraudulent manipulation. The streams of data might be or not be dedicated. Another important part is that the protocol followed for communication among fog devices should be supported by all nodes, complex and rarely used encryption standards may restrict diversity of devices that can be used.

The sharing of data with sensors and actuators needs to be as close to real time as possible. For mission critical application proper data transfer frequencies are important. For the case of static transfer frequencies, very high frequencies can lead to computation lag from the worker nodes, very low frequency can cause reduction in the Quality-of-service (QoS).

The current model uses HTTP protocol for communication between different nodes in the local network and the Aneka’s protocols for task distribution to cloud and other devices. All data is currently transferred in plain text using the HTTP REST APIs specifically GET and POST.

**3. Computation over data**

Another important criteria for model development is that it is generic in terms of the service it provides. In the IoT world there are several applications in which Fog, Edge and Cloud Computing can be used, and there are several others that haven’t even been thought of. It is important that the frame work allows services to be used that can be changed, modified and/or extended easily.

The computation/analysis requirement of the service also affects the Resource management policy and data transfer rate requirements. Based on different services and end-user requirements, performance metrics can be designed and the framework can be optimized accordingly.

The model uses Java runtime environment for data analysis due to it’s ability to be run in diverse machines.

**4. Storage of Data**

Many applications require data to be stored for future use. This data being shared among multiple nodes is prone to attacks. It is thus important to implement strategies/techniques that keep data resilient and prevent it from tampering. Not only keeping saved data secure is important, but the methods used for data extraction and sharing among nodes is also a concern.

**5. Data Interface**

Interaction with the user plays a key role in obtaining the user requirements and also acts as an intermediary between the network and sensors/actuators. It is crucial for this interface to be accessible from a wide range of devices and be friendly for interaction. Some requirements of this interface include:

* Set the architecture of the network in terms of worker nodes
* Connect to Sensors and actuators for real time data sharing
* Allow modification and extension of different components for application specific optimization
* Ability to set/modify the resource management policy
* Display/Modify/Erase data collected
* Enable or Disable Master as a worker
* Enable or Disable Aneka tasks to be spun

The current model uses PHP based web scripts as an interface with the user and also backend interface among master and worker nodes.

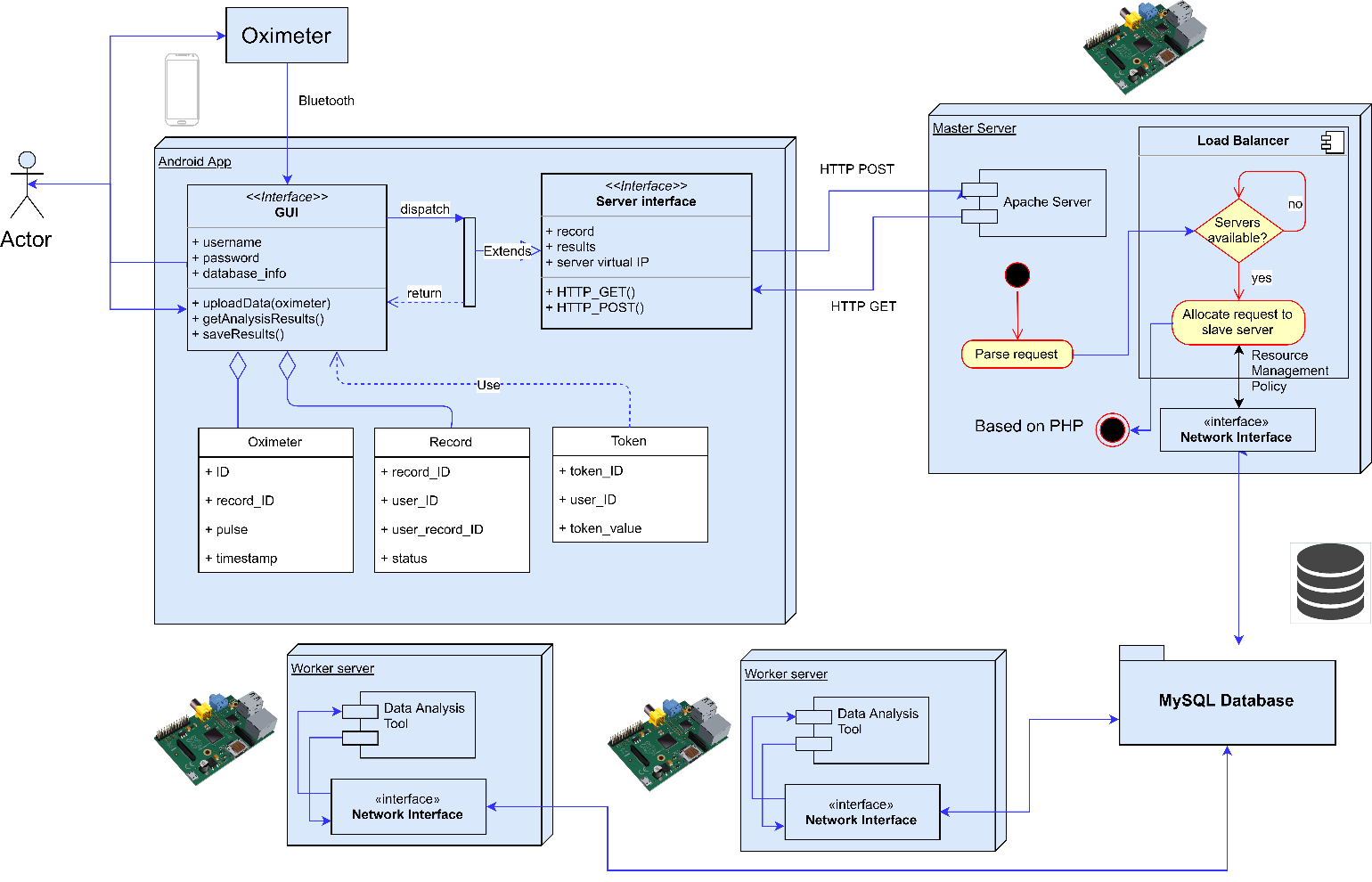


Figure 3: Complete model with different functional components

## Implementation

The software has been built using platforms that are supported by a wide range of devices including those used for IoT application. The main software is divided into four different components:

1. Interface Web Scripts (Based on PHP)
2. Aneka analysis code (Based on C# and .NET)
3. Analysis Jar file (Based on Java)
4. User Interface (Android App, or web-based interface)

The web scripts require an Apache server to be set up in all nodes and MySQL server in master. Each of these have different roles and are distributed among different nodes. Assuming that the local network is secure, all public cloud interaction is based on Blockchain to prevent hackers from accessing and manipulating data.

### Master Node

The master node is the first point of contact for the end user. It has the following roles:

1. Take user login information and check with the Database

2. Maintain Database of registered users

3. Maintain configurations including Worker nodes’ IP addresses

4. Take data for analysis

5. Distribute data with other parameters among worker nodes and Aneka (or itself)

The different scripts which together perform these roles are:

1. **Index.php**

This is the first screen that is displayed when the user opens the software’s web interface. An HTML form is allowing users to input login information including username and password is displayed first. This form takes in the login information and sends it to Master node using the POST method. The script then checks the input information with the MySQL Database. Using the *mysqli\_connect*command, the database name and table name are verified. After the form, the database and table connection success are shown on the web page.

The login information is checked using the SQL query to find entry with the given details. If the number of returned rows is greater than 0 then the login is successful and page navigates to “home.php”. If the login details are not found in the table then a prompt is displayed to allow re-entering the login details.

1. **Home.php**

This acts as a menu or navigation page for the user and allows user to go to either the “manager.php” page or the “session.php” page.

1. **Manager.php**

The “manager.php”, handles all the worker IPs for the “session.php”. It maintains the configuration file “config.txt”, in which every line contains one IP address of a worker. “config.txt” also holds whether Master and Aneka could be given the task of analysis or not. The “Remove all Workers” allow the configuration file to be reset to default state where there are no registered worker IP addresses and both Master and Aneka are enabled for task processing.

Whenever a new IP address is set it is appended to the configuration file. The first line contains the Master and Aneka enable configuration.

Another task performed by the manager page is to synchronize the Jar file used for analysis. The “Sync Jar File” button sends a request to the manager page of each worker node to synchronize the executable jar file from master. The worker node then downloads the master’s copy of the executable and overwrites it’s own with it.

1. **Session.php**

The “session.php” script performs most of the software functionalities. The initial part of the code welcomes user by using the username tag of the GET request received from the home page. The user can manually input the data for send data via the GET request to the master node. After inputting the data, if the “Analyze” button is clicked, then the content of “config.txt” is parsed to obtain list of worker IP addresses.

Variables *$toMaster* and *$toAneka* store if the task is to be given to the Master or Aneka or not. It is initialized to false if first line in “config.txt” is “DisableMaster” or “DisableAneka” and true otherwise.

The next few lines form the code for the “Failover” and “Load Balancing” schemes of the software. The variable *$ips* is an array of the IP addresses and *$loads* is their corresponding loads. The PHP method file\_get\_contents() allows us to get a string form of the webpage passed as the argument. An error operator lets it return FALSE in case of errors. If any HTTP request for load returns FALSE then variable *$my\_var* is set to 100, and error message is displayed on the screen. Setting load to 100 has the effect that this node would never get the task and hence removed from load balancing IPs. The next time the worker IPs are accessed for load, this is checked again and if the particular worker node is available, it is taken into the load balancer.

Using a for loop, the *$loads* array is populated by accessing the “load.php” of the workers corresponding to the IP address. If the load of any worker is < 80%, then the variables *$toMaster* and *$toAneka* are set to false. In effect, when master is enabled as worker, this sets *$toMaster* to true only when all worker nodes have more that 80% load. Same is the case for Aneka container. If the task is not given to the master or Aneka, i.e. *$toMaster* and *$toAneka* are false, then the IP of the worker with minimum load gets the task. The task is sent using the GET method to the “worker.php” script of the worker with that IP address. Otherwise, randomly either the master or Aneka gets the task. If the task is sent to master, then it is sent using GET method the “worker.php” script in the local machine. If the task is sent to Aneka, then it is sent using the GET method to the “workerAneka.php”.

The load balancing strategy used is shown in Figure 4.

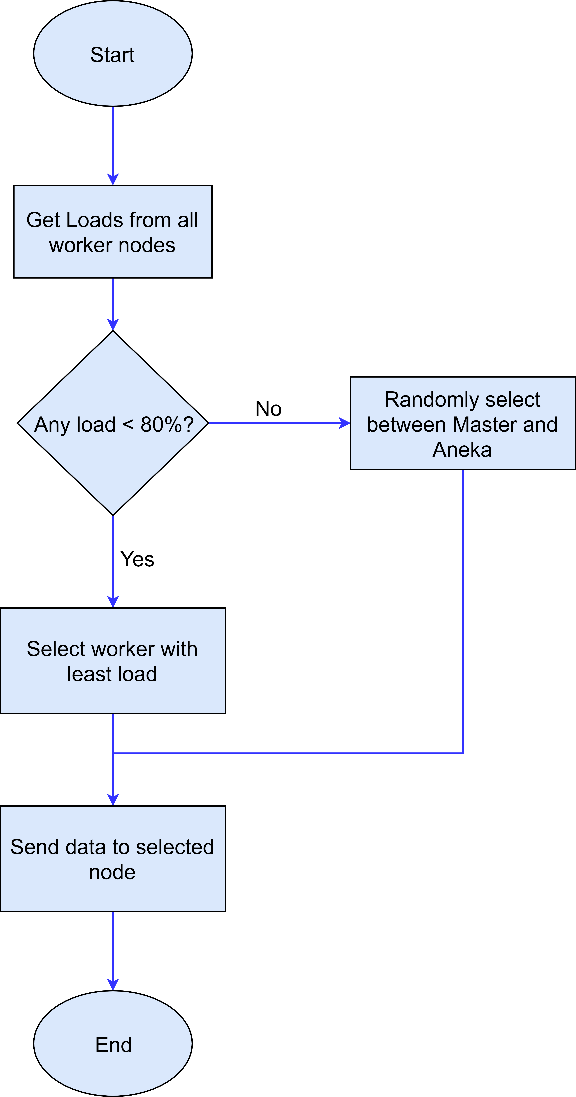


Figure 4: Simple Load balancing scheme implemented in code

The received results from the worker node are displayed on the webpage and a cumulative graph shows all the data that has been recorded since the user was registered.

### Aneka

1. **WorkerAneka.php**

This script acts as an interface between the Aneka software and the master session script. It receives data to be analyzed from the session script and saves it to the data file (data.txt). When analysis is complete, the contents of the result file (result.txt) are displayed on the webpage.

1. **Aneka code**

The Aneka code parses the data file every 500ms and checks if pending analysis exists. If yes then it forms Aneka tasks/threads and launches it to other Aneka containers which might be on the cloud or in local network.

Blockchain has been implemented in this to ensure data integrity and secure the data and results from hackers. Whenever a new thread or task is spun, the proof of work is calculated by the master node and is sent to other nodes for verification. If verification passes then the block of data is added to the chain, otherwise the data is discarded. Each time new task is sent, the chain is validated by checking the correctness of the last block’s hash value and matching the previous has value of the last block with the hash value of second last block.

management has also been implemented so that data from unknown sources can not be added to the blockchain, thus ensuring no outside and unwanted manipulation can be done. Each Aneka node is given a public and private key. The signatures are formed using private key and verified using public keys.

### Worker

The Worker node, displays CPU load, analyzes data and displays results.

1. **Load.php**

The “load.php” uses the PHP method: sys\_getloadavg() to get the system CPU load and displays it on the webpage.

1. **Manager.php**

The worker’s manager page allows user to set the Master IP address which is used for synchronizing the executable Jar file. It also displays the current set IP address of the Master which is saved in the configuration file (config.txt) and the analysis file name that is being used for computation on the data.

1. **Worker.php**

The “worker.php” script receives data to be analyzed using the GET method. It saves the data in “data.txt” and sets the first line to “Analysis Done = false” and executes the executable Jar application. The Java based analysis application parses this data, saves results in “result.txt” and changes the first line of “data.txt” to “Analysis Done = true”. This script waits till the first line changes to the latter and then parses the result file. The parsed output is displayed on the web page.

1. **Analyzer.jar**

This is the analysis software that performs computation on the data received and outputs results in the result file (result.txt).

The complete working of the software with details indicating how different scripts and applications interact; is shown in Figure 5.

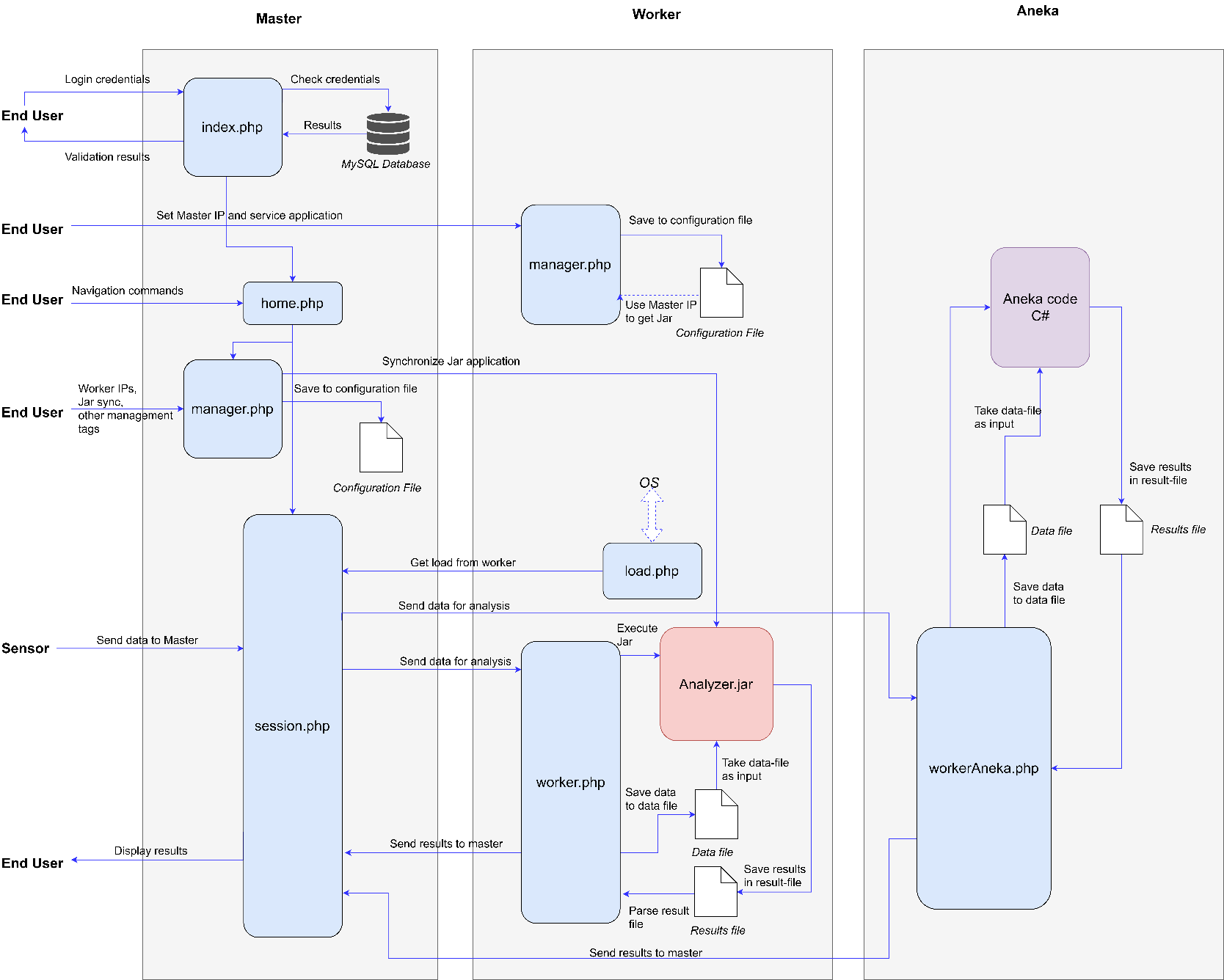


Figure 5: Complete working explanation of the implemented software

## Application Case Study: Sleep Apnea analysis

The FogBus framework has been deployed and tested with a real-life application: Sleep Apnea Analysis. Sleep Apnea is a disease in which air stops flowing into the lungs for 10 seconds or longer while in sleep. This reduces the oxygen level and sensing that the patient has stopped breathing, the brain triggers to wake up just enough to breathe. In some cases, this can happen over 30 times in an hour. Many people have sleep apnea, (also known as sleep apnoea) but may not even know it. In fact, sleep apnea affects more than three in 10 men and nearly two in 10 women, so it's more common than you might think.

Sleep Apnea analysis is very difficult and cumbersome. An overnight or laboratory-based sleep study requires you to stay overnight at hospital in a sleep unit or laboratory usually consisting of a number of private, quiet, single rooms.  You will sleep with sensors hooked up to various parts of your body.  Physicians usually recommend this test for more complicated or difficult to diagnose cases.

The doctors use a term Apnea Hypopnea Index (AHI) and the Electrocardiogram (ECG) data to diagnose if a person is suffering from Sleep Apnea or not. Using open source algorithms, we deployed an analysis application that could analyze data from a Bluetooth enabled Pulse Oximeter and give diagnosis results based on the data.

This was divided into two parts: Data analysis application and the Android Interface with Bluetooth Oximeters.

Physical Infrastructure:

1. Master Node: Dell XPS 13
2. Worker Node: 2x Raspberry Pis
3. User Interface: Android Device
4. Sensor: Pulse Oximeter

All nodes were connected using a local hotspot.

### Data Analyzer

References:

<https://github.com/subrahmanyamanigadde/sleepapnea>

<https://github.com/monarch-initiative/sleep-apnea-clustering>

The sleep apnea analysis app was developed using a combination of the above two open source repositories.

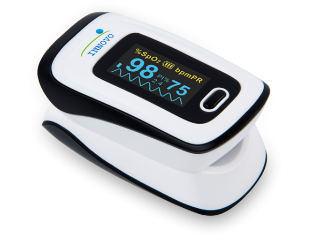
The analysis application takes the data file as input and stores results in the result file. When new data is found in the data file, it parses it and splits the string with “,” as a delimiter, and converts all strings to integers. Two data sets are parsed in this way: heart rate and blood oxygen level. For oxygen level, in a broad sense, the analysis is done in the following way:

* There is a *dip* Boolean variable, which stores whether there is a *dip* in oxygen level, a count which stores the number of times *dip* changes to true and a min which stores minimum oxygen level
* Whenever the oxygen level goes below 88, *dip* turns to true and stays true till oxygen level comes above 88. This is verified with an increase in the heart rate corresponding to or with an offset with the timestamps along a dip in oxygen level.
* This count becomes the AHI (Apnea - Hypopnea Index), used to determine the disease severity
* Based on standard thresholds, the disease probability of the sleep apnea is determined

For the heart rate data, the analysis is done using the following method:

* The Minimum and Maximum heart rates are determined in the input data
* The average heart rate and average rise or fall of the heart rates is determined
* Those close to the dips in oxygen level are filtered
* Based on some thresholds the ECG diagnosis is determined

Using both the diagnosis results, disease severity of the patient can be evaluated.

Once done, the result is written to the result file, and the first line of data fie is changed to “Analysis Done = true” indicating the worker script to take results from the result file.

### Android Interface

References:

MIT App Inventor: <http://appinventor.mit.edu/explore/front.html>

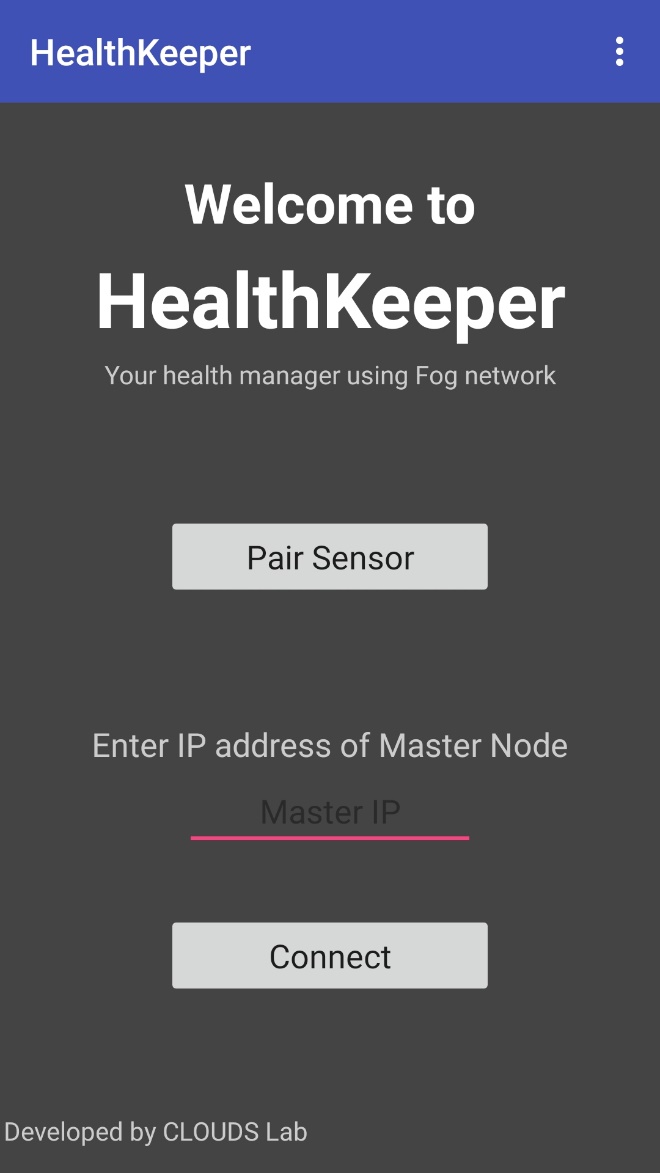
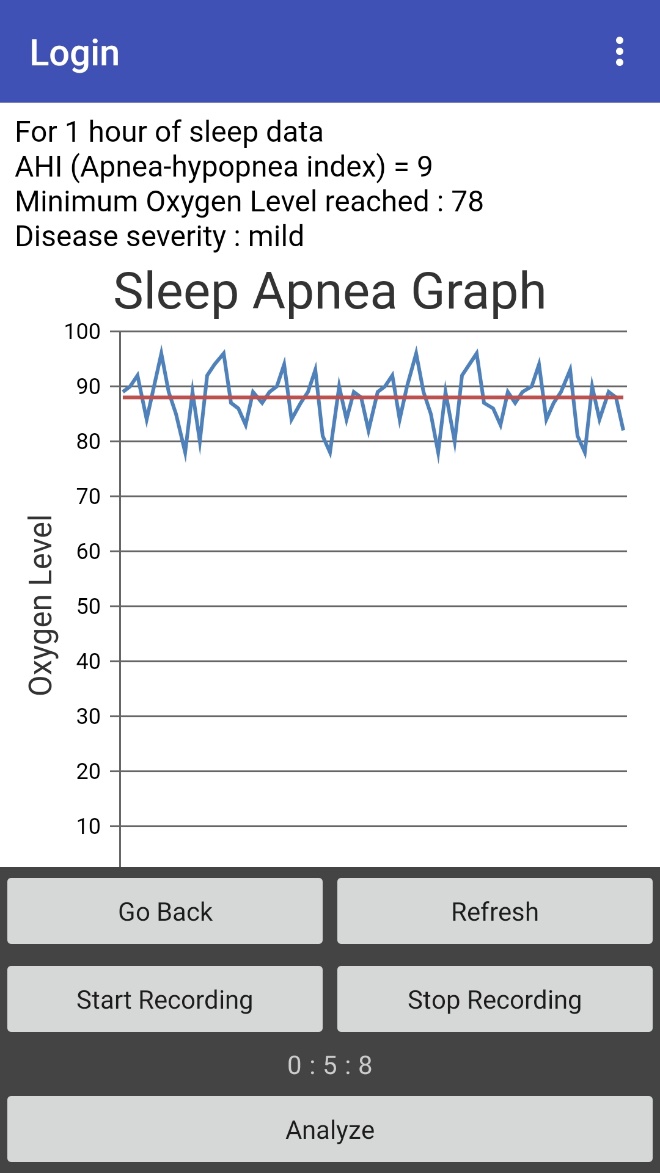
App Inventor JavaBridge: <http://www.appinventor.org/jBridgeIntro>

The android app “HealthKeeper.apk” allows the android device to act as an intermediary between the Oximeter sensor and the Master server. Rather than sending data manually through the GET HTML form, this app records and sends data automatically. The app has been developed on an open source platform: “MIT App Inventor” which can be seen at this link. The source file for “HealthKeeper.apk” can be found here. The application is divided into two screens namely: Welcome screen and Session screen. As shown in the End User tutorial, the Welcome screen asks user to pair the Bluetooth Oximeter with the Android device and enter the Master server’s IP address

The masterip is the global variable which stores the IP address entered by the user. The list global variable stores a list of masterip and Bluetooth device address. The PairSensor is a list picker to select from the available bluetooth devices. After picking, the BluetoothClient connects to the selected address and displays it on the screen. When the “Connect” button is clicked, the list objects are appended and sent to the Session screen.

The Session screen is the main screen that handles all interaction with the master server. The blocks below show the variable initializations. The data variable stores the data received from sensor in list form, loop variable is a boolean for timer, and url variable stores the web address of the master node’s index page.

The “Refresh” button connects the “WebViewer” to the url. When the StartRecording button is clicked, the timer is reset, the BluetoothClient connects to the address and starts collecting data from the sensor till the StopRecording button is clicked. When the StopRecording button is clicked, the data is submitted to the master node via the GET method. When Analyze button is clicked, the data is submitted for analysis and results displayed on the WebViewer. The analysis function is shown in Figure 6, which Connects the WebViewer to the session.php with GET request: analyze value set, which ensure that data is submitted for analysis.

Session screen of the app

Welcome screen of the app

## Images of Sleep Apnea System

## Conclusions

## Related Work

## Further Scope of developments

This software forms a base for setting up Fog Computing Environment which is not OS or architecture specific. Though the framework is complete by itself and fully functions to perform real-time, sensor-based Sleep Apnea analysis, but has a large scope of improvement and further developments:

Load Balancing Scheme

As discussed earlier, the current load balancing scheme is naive and can be greatly improved. Currently, the load balancing scheme focuses on task distribution based on CPU load, whichever device has the least load, the task is given to it. There can be a cumulative ranking parameter which takes into account many other factors as per requirement. These factors can be and are not limited to: Network Bandwidth, Memory Load, etc. Different weights can be given to these parameters are per the scenario.

### Data Integrity

Health Analysis data is important for patients as their treatment and lives depend on it, thus it is important to save this data from fraudulent manipulation or sabotage from hackers. Thus, to maintain data integrity many techniques like Blockchain can be implemented to ensure that data is secure.

### Data Privacy

Some applications require data to be secure as well as kept private. The current system is prone to attacks and unwanted display of data to others. Privacy policies like encrypting the data can be used to ensure that the data cannot be seen by others.

### Data Authenticity

The current system allows any device to connect to master and share or view data. This can be used by hackers to forge DDoS or similar attacks. As Fog platforms also contains low range devices with limited threshold management, such attacks even at a low scale can destroy such devices. Thus, a signature-based validation technique can be used to ensure user and data authenticity.