IoT - Fog Model Building on Agro and Fish Farming Applications

[1]Dr. K. Rangaswamy, [2]Dr. S. Murali Mohan, [3]B. Rupa Devi, [4]Lakkala Jayasree, [5]B. V Chandra Sekhar

[1]Associate Professor, Department of CSE (DS), Rajeev Gandhi Memorial college of Engineering and Technology, Nandyal, Andhra Pradesh.
Email: rangaswamy19@gmail.com
[2]Professor, Department of Electronics & Communication Engineering, Mother Theresa Institute of Engineering and Technology, Palamaner, A.P.
Email: muralimohanieee@gmail.com
[3]Associate Professor Department of Computer Science & Engineering, Annamacharya Institute of Technology and Sciences, Tirupati.
Email: rupadevi.aitt@annamacharyagroup.org
[4]Assistant Professor, Department of Computer Science and Engineering, Sri Padmavati Mahila Visva Vidyalayam, Tirupati.
Email: jayasreemohan15@gmail.com
[5] Assistant Professor, Department of CSE, Rajeev Gandhi Memorial college of Engineering and Technology, Nandyal, Andhra Pradesh
Email: bvcalfa@gmail.com

Abstract

The design of a fog application is complicated because it includes the application architecture and the deployment diagram from software packages to connectivity computer systems and the runtime infrastructure. This entire IoT of elements must be taken into consideration. There are many different design options available for each of these facets, and they all impact the overall quality of service and cost of the application that is produced. This paper suggests a strategy for assessing haze IoT application forms early in development. Our strategy uses modeling and simulation to provide forecasts for the quality of service and cost, allowing developers to choose a great application design through an interactive process.

Keywords: Application Design, IoT, Fog Computing, Simulation

1. Introduction

The Internet of Things (IoT) is seeing widespread adoption of connected devices, dramatically increasing the data available to developers. The Internet of Things (IoT) applications available today can employ this knowledge to make possible more sophisticated use cases[1]. The conventional approach to building an IoT app entails collecting data at the network's periphery, transmitting it to the cloud computing system for processing, and returning the results to the network's edge. This can be used, for example, to turn on a light when motion is detected in a smart home situation[2], [25]. Various services, such as AWS IoT and the Azure IoT Hub, utilize this method because it can be implemented easily.

On the other hand, some drawbacks include sluggishness, unnecessary data transfers, and the risk of private information falling into the wrong hands[3]. AWS Greengrass reduces

the need for network bandwidth by offloading some processing to the edge node, allowing the node to continue functioning even when network partitions are present. However, this approach is limited by the processing power of the network edge, which is often insufficient to handle the execution of compute-intensive tasks. Utilizing the computing power offered by more powerful machines, such as cloudlets, that are part of the core network is an apparent solution to this issue that can be implemented. The term "fog" is commonly used to refer to this execution environment, which is made up of network edge[4], machines located inside the core network, as well as the cloud[5], [24]. Usually, programmers must think about the software architecture, the runtime infrastructure, and the Application Software Deployment Infrastructure Machine Mapping when designing an application for the fog[6].

There are many design possibilities for each of the three aspects, and each can be merged with possibilities from the other two. Because of this, an application could be designed in many different ways. When deciding on a specific design, one should thoroughly analyze how the change will affect the quality of service (QoS) and the cost. It is meant to carry out such an evaluation. However, it is likely to be complex because the additional benefits of effectively utilizing fog can substantially improve IoT systems[7]. This paper aims to support such evaluations so developers can select a design option more easily. As a result, the following are the contributions that we make:

We present an approach that can be utilized during the design phase to evaluate applications based on fog computing. Implementing our methodology, which we call Fog Explorer, is demonstrated as a proof of concept. We present a scenario illustrating how our methodology can be applied[8]. When developing a fog-based Internet of Things application design, many options exist for deployment mappings, application architecture, and runtime facilities[9]. This opens up a wide variety of potential layouts. To help developers assess their designs, we have adopted an iterative simulation and analysis process. We utilize simulations to provide quality of service and cost estimates for engagingly adjusting deployment lookup tables. The simplified infrastructure and application models form the basis for the simulations. This information helps designers weigh the pros and cons of potential design decisions and find a happy medium between the quality of service and cost[5].

2. Modeling and Simulation

demonstrates the iterative simulation and analysis developer workflow: first, build a high-level model of the infrastructure (1a) and the application (1b). Identifying the various machines and how they are connected, while the application model outlines the application modules and the data streams that flow between them. Note that there is no need to implement anything to carry out the simulation process. This is because application modules comprise very high-level data on the development tools of an implementation that will later need to be implemented. After creating the first infrastructure and software model, designers can create a deployment model by positioning software applications on infrastructure machines (2) based on those models. Every placement update influences QoS and cost, so a new simulation run should be triggered by it (3)[10]. Studying these effects allows for deriving recommendations regarding optimizing placements and the infrastructure and application model (4). Because of this information, developers can iteratively improve their designs and

compare the merits of various design solutions.

Model of the Infrastructure

When designing a system, programmers usually only have a hazy idea of the backend systems used. Of the system they are designing. As a result, we cannot assume that we have complete information and benchmarking results on the currently available performance. Therefore, the model must support an abstract, high-level infrastructure description. This can be achieved by concentrating on six properties, three of which pertain to the machines and three to the connections between them.

The performance Indicator property of each machine evaluates the system's efficiency about a standard machine. For example, if the performance indicator for a machine is 4.0, it indicates that the machine in question is approximately four times "faster" than the machine serving as the reference. In addition to the performance measure, machines have characteristics that characterize the amount of memory (referred to as available memory) and the price of that memory (referred to as memory Price). If it becomes necessary, more features can be added to our model. For example, these supplementary properties can describe storage capabilities and costs, while quality-of-service properties like availability. On the other hand, we consciously decided to limit our approach to the design phase to a straightforward model.

The latency property estimates the time it takes for two machines to communicate with one another for each connection. In addition, bandwidth and bandwidth Price describe the available bandwidth and its associated cost. A machine can have strong links to another machine. This allows for independently modeling outgoing and incoming data, as well as considering connections that are either slow but inexpensive or expensive.

Building Application Model

Events frequently drive applications for the Internet of Things: "things" produce streams of data, the digital data are evaluated to identify certain occurrences, and events cause actions to be taken by other "things." Each feature may be carried out in a different location when the fog is present[11]. For this reason, the architecture of an application should be built on self-contained modules that can be executed independently of one another and share data and information through the interconnection of data streams, as is also done[12].

About these kinds of We identified three distinct categories of application modules: sources, services, and sinks. Data is generated by sources and processed by services, sending the results to sinks, which then receive the data. Model of the application consisting of four separate application modules, including one origin, one service, as well as two sinks. Because our method is intended to make model definitions possible at a very early stage of the development process, we have again chosen to employ a condensed collection of module properties. In our model, sources such as Internet of Things Sensors generate a steady stream of data at a predetermined rate (output Rate), which would be processed sequentially by a set of services over a predetermined period. A time frame is associated with this processing (reference Processing Time). With this abstraction in place, designers need only worry about the rate at which data is generated for every unit of time rather than the specifics of how often information is transmitted or how much data is being generated at any given time. Sent 40 kilobytes of data to the Aggregator provider from the temperature sensors. Five seconds are spent by this service grouping the same data before it is sent to the first of two designated storage locations (Storage 1 and

Storage 2). The service's output rate is calculated in real-time by dividing the total data by a dynamically determined factor that comes in with the output Ratio that corresponds to it. This output Ratio defines how well the service alters the volume of the data stream coming in[13].

It is important to remember that services never "freeze" while processing data but continuously receive and process data in a series of iterations. We have simplified this issue by requiring devs only to estimate how long the production process will take on a reference device, as benchmarking real software on facilities is the only way to determine the actual operation time, which is not feasible during the design phase. The time spent calculating the correct processing time is thus reduced[14]. Given the reference point of the machine the module will be used on and the comparison processing times of the subsystem, we can estimate the actual computation. Using the reference machine (with its performance measured at 1.0) as an example, we can infer that it will take the Aggregator 5s to process the data. Each service, as well as the reference, has a corresponding mode that, when chosen, specifies how the application module's output rate is divided up among the various outgoing streaming data. Depending on the mode setting, either the total amount of data is sent to all subsequent devices at the determined rate (mode = "all") or each successive device receives the entire determined production rate (mode = "individual") generates is proportionally distributed across all streaming information that is sent out (mode = "total"), depending on which option is selected. If the Aggregator service, for instance, had four outgoing streams of data rather than two, then the amount of data contained in each data stream would be 5 kilobytes per second because the 20 kilobytes per second that were calculated as the output rate would need to be divvied up by four. The amount of memory a module will consume during the runtime is specified by the required memory property, which is present in all modules. The only properties developers must specify for data streams are the modules they connect. The quantity of If you know the reach its full potential, output ratios, and modes for the module, you can figure out the required bandwidth, which we'll call the required bandwidth.

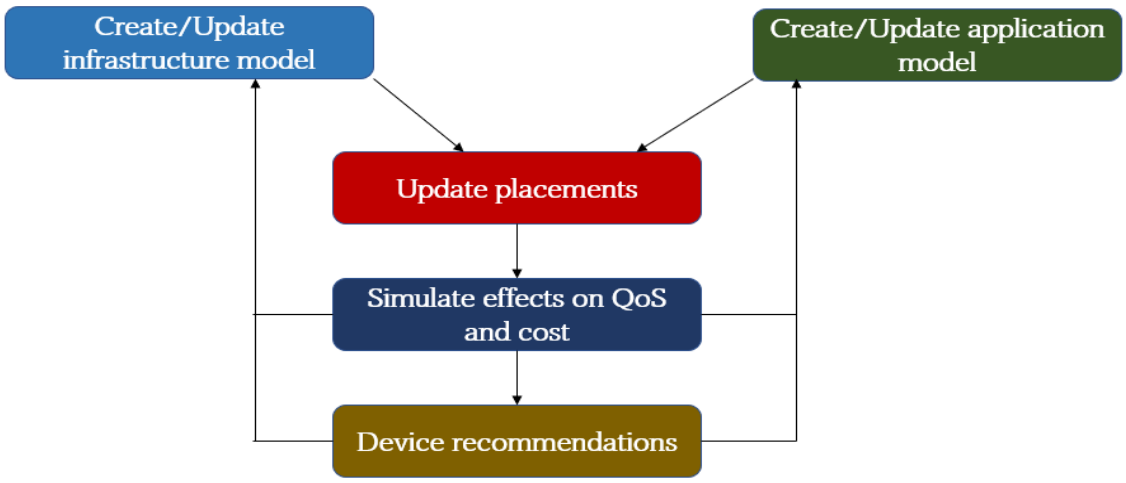

Fig 1. Modeling Design and Outline

Model -1

Developers ought to be able to perform an interactive evaluation of the effects of the module placement alternative on both qualities of service and cost based on the application and infrastructure model. We perform simulations of effects on a total of four metrics using the present model's properties: processing cost, process time, transmission cost, and also transmission time. The time metrics define the time it takes to process a single data item on a given machine or send it over a given connection instead of the cost metrics, which describe the average price imposed per second in a particular configuration. For determining these metrics, we suggest using an implementation of a tool. After each module placement, such an engaging simulation tool must carry out several calculations, which we will describe in more detail below.

1. The routing of data streams: The tool must ascertain which connections are utilized. The discrete components' streaming data must be utilized to achieve this goal. For example, let's pretend we have a model of facilities comprising three machines, A, B, and C. There is a connection between machines A and B, and from there to machines C and D. The AB connection and the BC connection will transmit data between the detector and the aggregation site when the detector is located on node A. The aggregation site is located on node C. In more complex scenarios with multiple potential connection routes, the tool can utilize the shortest path algorithm to ascertain the links offering the lowest bandwidth usage price.

2. Utilization of Resources: The tool needs to calculate how the placement of something affects the amount of bandwidth and memory used by machines and connections. The new amount of memory used is the same as the previous amount, especially with the amount necessary for the placed module. Similarly, with the data stream's needs factored in, the current rate of used available bandwidth for each link is the sum of the connection's actual rate and the amount needed by the stream. All the data feeds included in the module are subject to this rule. When a machine has a module removed, the free bandwidth and memory usage both drop by the values. Putting the Aggregator service on machine C, for instance, could lead to a memory leak if machine C requires 500 MB of memory bandwidth utilization of 1500 megabytes, and connections AB and BC each utilize a bandwidth of 40 kilobits per second.

3. Under provisioning The tool needs to determine the under-provisioning ratio, also known for every connection and machine, under the Provision Ratio. The ratio defined here is the one that resources to available resources. If more resources are available than utilized, the ratio equals 1. In any other case, it is calculated by dividing the resources used (such as bandwidth or memory) by the total available resources. For instance, installing the Under-provisioning ratio of 10 would occur if a machine provided an aggregator service with only 100 MB of memory.

4. Different cost metrics for different situations Costs associated with transmitting and running individual data streams and application modules must be calculated by the tool module separately. Because of this, it needs to consider the practice of under-provisioning its resources.

5. Individual measurements of time The tool needs to determine the amount of time necessary for transmission and processing for every data stream and module. If the respective source of information is not adequately provided, each time metric will be infinite. The system will experience internal pressure that will never be managed. After all, the input will always exceed the

capabilities of either transmission or processing. If this is not the case, the processing time corresponds to the estimated processing time of the machine (which considers the machine's performance indicator), and the transmission time for each connection is equal to its latency.

6. Total quantitative measurements: The tool needs to determine a real and cost metric every time. This allows designers to get an overall sense of the characteristics of a design by looking at a single value. For instance, the sum of the modules' processing costs equals the total processing cost.

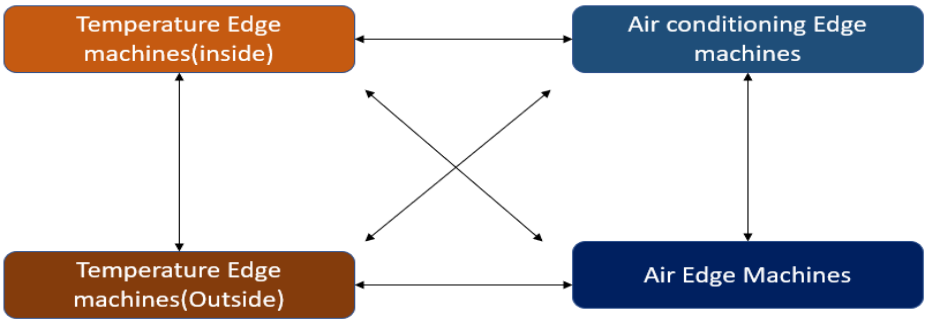

Fig 2. Modeling Design on Temperature and Air Edge Devices Communication

Recommendations: The tool needs to provide devs with suggestions on optimizing placements and the current application model or infrastructure. Data such as under-provisioned computers and connections can be the basis for these recommendations. In addition, the recommendations need to assist in identifying invalid module placements in which a missing infrastructure linkage disrupts the data streams. In addition, the recommendations should highlight which machine and connection resources might be reduced without impacting the quality of service or the cost. We decided to go with this set of measures because they provide a comprehensive overview of the effects of a particular design on QoS and cost. In addition, our tool can calculate these performance measures before the application is implemented and before the developer has complete information about the runtime infrastructure. We need modeling data about application components and the planned run-time infrastructure to do this. If the more extensive quality of service and cost metrics are required, or if additional recommendations must be derived, it is also possible to expand the properties of the application and infrastructure model. However, doing so will increase the complexity of the analysis.

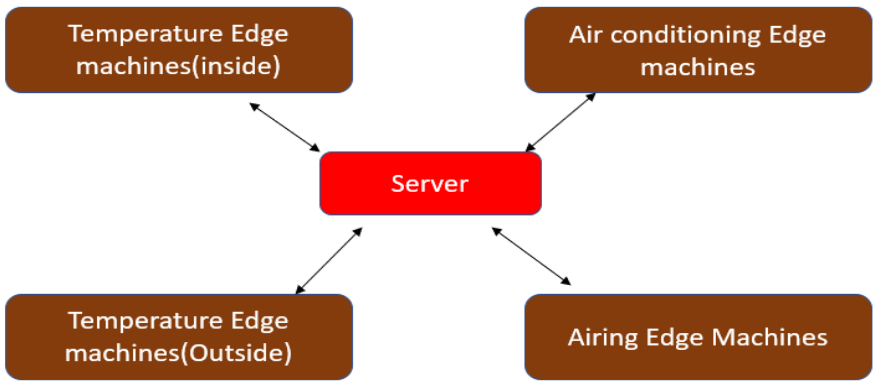

Fig 3. Server Loading – Fog and IoT with Edge Devices – Communication

Scenario

In this section, we put the modeling and simulation procedure to use by applying it to a real-world example. Because of space constraints, we have chosen to illustrate our point with an example from edge computing. On the other hand, the procedure is not significantly altered for more complex scenarios wherein machines have been geo-distributed across the rim, core network, and cloud locations. In the scenario that we have planned, a company wants to reduce the cost of energy by regulating the temperature within one of their houses based on the readings from various sensors. Because there are multiple possible styles for an application like this that involves smart buildings, the company must consider the effects on quality of service and cost before deciding which design must be implemented[15]. Numerous factors, such as the strength of the wind, the amount of direct sunlight that enters the building, the difference in temperature between the interior and exterior of the building, and the presence of air pollution, are important to the climate that exists inside of a building and should therefore be monitored. In addition, there are numerous methods for controlling the climate, such as opening and closing the windows, shades, and shutters, turning the temperature up or down or using air conditioning or heating. In addition, predictions of the weather and analyses of past data make it possible to make more informed choices [13].

For example, when a cold front is coming, it makes more sense to put it on hold for a few more minutes before bringing the temperature down with the help of an air-con if the temperature threshold values have only been slightly exceeded. This is especially true if the temperature thresholds have been exceeded for an extended period. We will only concentrate on the scenario's heating component for our assessment. In addition, we only track the temperature on the inside and exterior of the building, and we only use two methods to control the climate: airing and air conditioning. In other words, the building must be cooled down as much as possible by daylighting at specific moments when the temperature outside is lower than the temperature inside the building. Air conditioning, which uses much energy, should only be used if there's no other option. The general type of situation set up. Several temperature sensors are dispersed throughout the building, and interior and exterior measurements are taken every second. An actuator is installed in each building's glass, allowing the window to be opened and closed as desired.

Additionally, several air conditioners are positioned throughout the building, each controlled by a separate actuator. The primary focus of our scenario evaluation is the selection of appropriate runtime facilities for the Internet of Things application that will control the actuators. The decentralized and the centralized runtime infrastructures will be evaluated as potential options. In a decentralized setup, edge devices communicate with one another directly to share information. In contrast, edge devices only interact with a (central) server in a centralized setup, which performs most processing[16]. Every possible choice for the runtime infrastructure has its benefits and drawbacks. Decentralized systems, for instance, are inherently incapable of having a single failure point and, thus, are typically able to deal well with network partitioning. Data access and bug fixing are made simpler using a centralized configuration. Because none of these choices is optimal in every circumstance, an analysis of each circumstance is required. As the evaluation needs to occur early in the design phase, the proposed approach aims to make this evaluation possible and helps decision-makers make choices despite their limited

data on application architecture and runtime infrastructure. In addition, our methodology is still adaptable enough to assess the results of a wide range of modifications, such as increasing or decreasing the total network edge number or recalculating the resources' costs.

Models for Both Infrastructure and Applications

The first thing we do in our process is build an initial continuous integration for each runtime environment. We combined all edge machines attached to the same category of "things" into a single machine to simplify the modeling process. Even though this is only feasible when all computers and their interconnection have identical hardware resources, it is still possible for us to build a separate machine for each end device if the resources are changed. Our centralized infrastructure models. At this point, the performs analysis on the measurements and sends commands to the edge devices. The edge devices, in turn, only forward these instructions to the actuators without performing any additional processing. Although the devices at the network's edge do not require substantial storage and processing capabilities[17], the server in the center does. The bandwidth utilization is relatively high compared to a decentralized system since all raw sensor data must be transferred to the centralized server[18]. Our strategy calls for creating an application model in addition to the infrastructure model.

Regarding the scenario, we came up with the one shown in figure 4. It is suitable for use with either of the aforementioned runtime infrastructures. Sensor sources constantly send information to event dispatcher (ED) services, which check to see if they need to send out a new event. Air Conditioning Manager and Airing Manager are both responsible for receiving events and performing analysis on them. By the findings of the analysis, they issue directives to the two electro-hydraulic drains. It is important to remember that every actuator "thing" has a corresponding sensor because the managers need to peruse the current state, such as whether or not an air conditioner is operating.

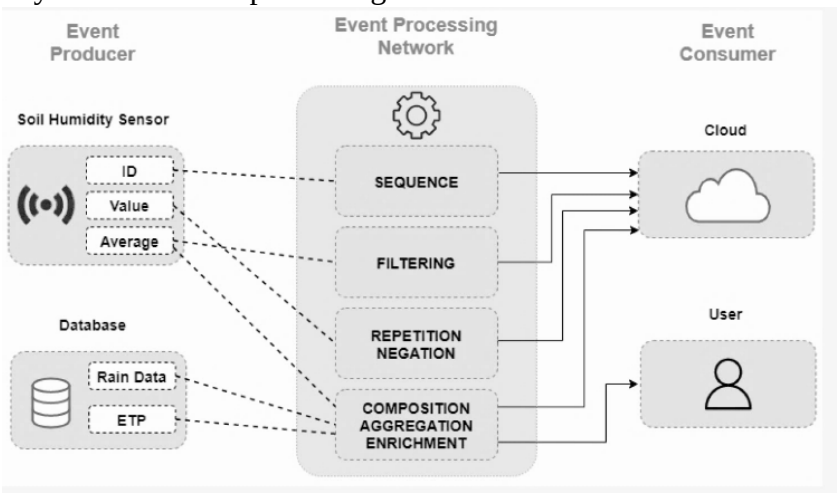

Fig 4. Application Building with IoT- Fog Edge Devices at Field – Agro Monitoring Modelling

Model -2

After creating both models, developers are now able to begin the process of placing sources, services, and sinks on machines. The sources and sinks are located on the edge devices of both runtime infrastructures. While in the centralized configuration, all the

services are located on the server, and the decentralized configuration places the ED and Manager services on the network's edge. With the help of these deployment mappings, the creators of the smart building will be able to start investigating how the quality of service and cost metrics are affected when the properties of the infrastructure and application models are altered[19]. Ultimately, the question of which runtime facilities are " better " is determined by the myriad aspects of a particular situation. These considerations include the number of actuators and sensors, the frequency and volume of different sensors, the tasks and analyses carried out by EDs and Manager assistance, and the transmission and processing resources cost. Developers can run the required computer models to receive advice, improve their design, as well as compare different innovative solutions if they follow our approach and do so to obtain these goals [20], [21]. We prepared a demo10 for this situation using evaluating effects on quality of service and cost for the centralized and decentralized runtime infrastructure but also trying out various model properties. The evaluation presented in this study is done using a case study. The outcomes of the demonstration utilizing the default settings. In this scenario, the centralized infrastructure has a higher cost but good QoS metric values; consequently, the centralized runtime infrastructure must be selected if a quality level of this nature is required. In conclusion, the information made available by Fog Explorer enables developers to make educated decisions regarding issues such as selecting particular runtime facilities without the necessity for early prototype implementations[22].

Real-Time Agriculture Application Modelling and Results

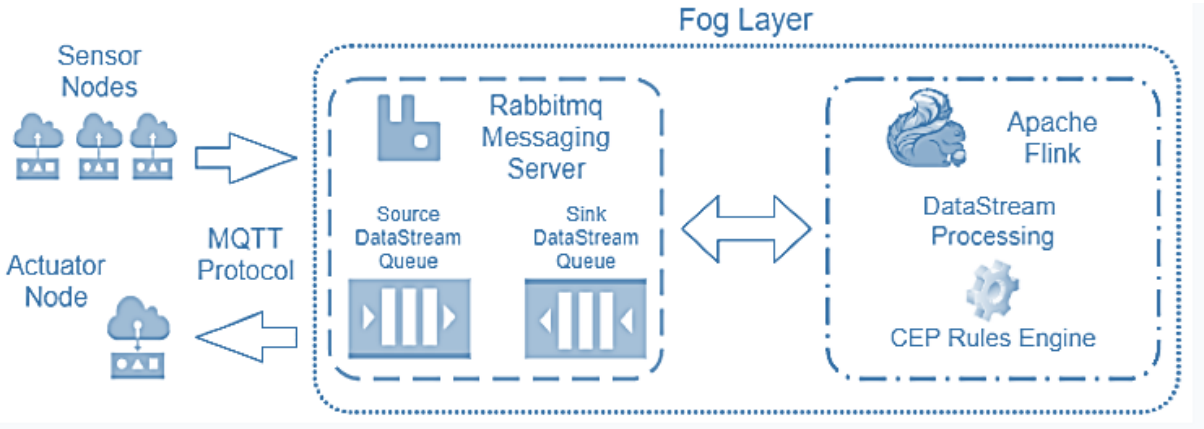

Fig 5. Node-wise Sensors Integration with IoT – Fog and Edge devices

The figure displays the hardware and software used in Experiment 2 to facilitate communication between nodes equipped with sensors and actuators and the Fog layer. The field equipment we used included three soil moisture sensors and an actuator for the zone's irrigation system. Python was our tool of choice. Model both parts, including soil moisture progression based on irrigation level (on or off)[23]. A Rabbitmq messaging server with two queues (data input and output) and a

module dedicated to the data stream and complex event processing (CEP) were implemented using the FlinkCEP library in Apache Flink, making up the Fog tier. The MQTT protocol enabled interaction between the Fog layer and the edge devices. Because RabbitMQ includes a built-in plugin for receiving MQTT messages, it was selected as the preferred messaging service.

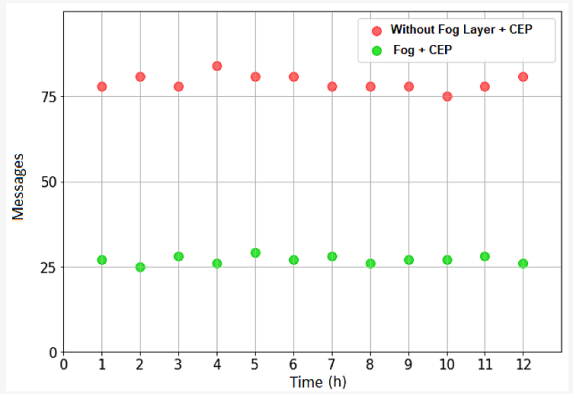

Fig 6. Messages – Responses from the Device

The outcomes of the experiment we ran to evaluate the system's performance in the following metrics: number of false positives, number of false negatives, accuracy, precision, recall, F1-Score, and data traffic. Three examples will be provided below.

	Soil Moisture	Temperature
Accuracy	0.992	0.993
Precision	0.996	0.997
Recall	0.995	0.995
F1-Score	0.995	0.996

Table 1: Performance Metrics

This experimental results scenario with 100 sensors on fog confirms the strategy's performance in a network with a more sizable number of nodes. As a result, 1800 soil moisture measurements and 1800 temperature measurements were taken to ensure the proposal could be scaled. The proposed method was used to collect soil moisture in the field, resulting shown in Figure 7. Using CEP and fusion techniques, field-collected temperature data is analyzed and displayed as a matrix of positive and negative results.

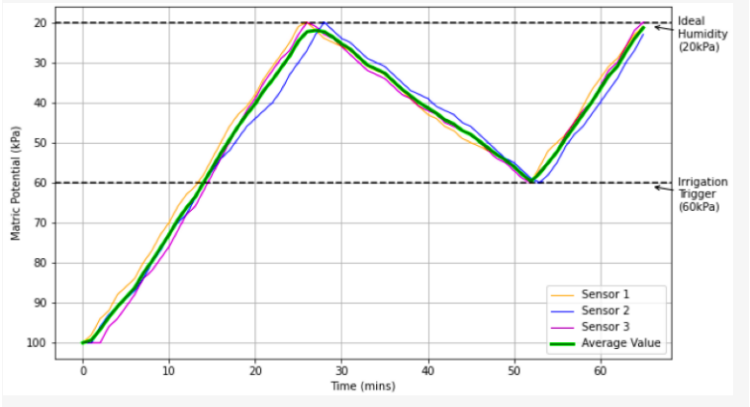

Fig 7. Evaluation of the performance

3. Conclusion

In this article, we proposed a method for assessing fog-based Internet of Things (IoT that can be employed right from the beginning of the design phase. When developers use our methodology, they can compare various design options using quality of service metrics and cost considerations. This strategy is built on a modeling and simulation process in which models of runtime infrastructure and software architecture are created, and simulations of the impacts of application module positions on infrastructure computers are carried out. Because there is a possibility that only a small amount of information will be accessible for the modeling, the proposed simulation procedure only needs high-level model meanings. It can, as a result, be used early in the development process. We proffered Fog Explorer, our prototypical execution, as a proof of concept. We also demonstrated how our methodology could be utilized by providing an example scenario.

References

[1] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. Bin Zikria, "Role of IoT technology in agriculture: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 319, 2020.

[2] S. D. Bhogaraju, K. V. R. Kumar, P. Anjaiah, J. H. Shaik, and others, "Advanced Predictive Analytics for Control of Industrial Automation Process," in *Innovations in the Industrial Internet of Things (IIoT) and Smart Factory*, IGI Global, 2021, pp. 33–49.

[3] G. Seeja, O. Reddy, K. V. R. Kumar, S. Mounika, and others, "Internet of Things and Robotic Applications in the Industrial Automation Process," in *Innovations in the Industrial Internet of Things (IIoT) and Smart Factory*, IGI Global, 2021, pp. 50–64.

[4] Y. Kalyani and R. Collier, "A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture," *Sensors*, vol. 21, no. 17, p. 5922, 2021.

[5] T.-C. Hsu, H. Yang, Y.-C. Chung, and C.-H. Hsu, "A Creative IoT agriculture platform for cloud fog computing," *Sustain. Comput. Informatics Syst.*, vol. 28, p. 100285, 2020.

[6] E. Guardo, A. Di Stefano, A. La Corte, M. Sapienza, and M. Scatà, "A fog computing-based iot framework for precision agriculture," *J. Internet Technol.*, vol. 19, no. 5, pp. 1401–1411, 2018.

[7] S. Jaiganesh, K. Gunaseelan, and V. Ellappan, "IOT agriculture to improve food and farming technology," in *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*, 2017, pp. 260–266.

[8] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming," *Ieee Access*, vol. 7, pp. 156237–156271, 2019.

[9] X. Zhang, Z. Cao, and W. Dong, "Overview of edge computing in the agricultural internet of things: key technologies, applications, challenges," *Ieee Access*, vol. 8, pp. 141748–141761, 2020.

[10] M. R. M. Kassim, "IoT applications in smart agriculture: Issues and challenges," in *2020 IEEE conference on open systems (ICOS)*, 2020, pp. 19–24.

[11] K. V. R. Kumar, K. D. Kumar, R. K. Poluru, S. M. Basha, and M. P. K. Reddy, "Internet of things and fog computing applications in intelligent transportation systems," in *Architecture and Security Issues in Fog Computing Applications*, IGI Global, 2020, pp. 131–150.

[12] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. H. D. N. Hindia, "An overview of the Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3758–3773, 2018.

[13] O. Friha, M. A. Ferrag, L. Shu, and M. Nafa, "A robust security framework based on blockchain and SDN for fog computing enabled agricultural internet of things," in *2020 International Conference on Internet of Things and Intelligent Applications (ITIA)*, 2020, pp. 1–5.

[14] F. M. R. Junior, R. A. C. Bianchi, R. C. Prati, K. Kolehmainen, J.-P. Soininen, and C. A. Kamienski, "Data reduction based on machine learning algorithms for fog computing in IoT smart agriculture," *Biosyst. Eng.*, 2022.

[15] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM*

Comput. Surv., vol. 50, no. 3, pp. 1–43, 2017.

[16] C. S. M. Babou, B. O. Sane, I. Diane, and I. Niang, "Home edge computing architecture for smart and sustainable agriculture and breeding," in *Proceedings of the 2nd International Conference on Networking, Information Systems \& Security*, 2019, pp. 1–7.

[17] F. Sharofidinov, M. S. A. Muthanna, V. D. Pham, A. Khakimov, A. Muthanna, and K. Samouylov, "Agriculture management based on lora edge computing system," in *International Conference on Distributed Computer and Communication Networks*, 2020, pp. 113–125.

[18] M. Uddin, M. Ayaz, A. Mansour, el-H. M. Aggoune, Z. Sharif, and I. Razzak, "Cloud-connected flying edge computing for smart agriculture," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 6, pp. 3405–3415, 2021.

[19] S. D. Bhogaraju and V. R. K. Korupalli, "Design of Smart Roads-A Vision on Indian Smart Infrastructure Development," in *2020 International Conference on COMmunication Systems & Networks (COMSNETS)*, 2020, pp. 773–778.

[20] D. J. Reddy and M. R. Kumar, "Crop yield prediction using machine learning algorithm," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2021, pp. 1466–1470.

[21] K. Ramana, R. Aluvala, M. Rudra Kumar, G. Nagaraja, A. Vijaya Krishna, and P. Nagendra, "Leaf Disease Classification in Smart Agriculture using Deep Neural Network Architecture and IoT," *J. Circuits, Syst. Comput.*, 2022.

[22] M. Kumar, K. Dubey, and R. Pandey, "Evolution of emerging computing paradigm cloud to fog: applications, limitations and research challenges," in *2021 11th International Conference on Cloud Computing, Data Science \& Engineering (Confluence)*, 2021, pp. 257–261.

[23] M. N. Akhtar, A. J. Shaikh, A. Khan, H. Awais, E. A. Bakar, and A. R. Othman, "Smart sensing with edge computing in precision agriculture for soil assessment and heavy metal monitoring: A review," *Agriculture*, vol. 11, no. 6, p. 475, 2021.

[24] Reddy, K. Uday Kumar, S. Shabbiha, and M. Rudra Kumar. "Design of high-security smart health care monitoring system using IoT." *Int. J* 8 (2020).

[25] Ramana, Kadiyala, et al. "Leaf Disease Classification in Smart Agriculture using Deep Neural Network Architecture and IoT." *Journal of Circuits, Systems, and Computers* (2022).