Resource Provisioning Framework for IoT Applications in Fog Computing Environment

Rakshith G*, Rahul M V†, Sanjay G S‡, Natesha B V§, Ram Mohana Reddy G¶
Department of Information Technology
National Institute of Technology Karnataka
Surathkal, Mangaluru - 575025, India
Email: {*15it134.rakshith.g, †15it229.rahul, ‡15it138.sanjay}@nitk.edu.in, {§nateshbv18, ¶profgrmreddy }@gmail.com

***Abstract*—The increasing utility of ubiquitous computing and dramatic shifts in the domain of Internet of Things (IoT) have generated the need to devise methods to enable the efficient storage and retrieval of data. Fog computing is the de facto paradigm most suitable to make efficient use of the edge devices and thus shifting the impetus from a centralized cloud environment to a decentralized computing paradigm. By utilizing fog resources near to the edge of the network, we can reduce the latency and the overheads involved in the processing of the data by deploying the required services on them. In this paper, we present resource provisioning framework which provisions the resources and also manages the registered services in a dynamic topology of the fog architecture. The results demonstrate that using fog computing for deploying services reduces the total service time.**

***Index Terms*—Fog computing, latency, IoT, resource provisioning**

I. Introduction

The use of IoT devices in various fields has increased manifold in the recent times. This is mainly due to advances in technology and the prevalence of IoT [1]. However, using the cloud computing techniques to handle resources in a centralized manner needs to be changed. This is to allow low and predictable communication latency and location awareness in large-scale distributed systems for the IoT applications.

However, cloud computing lacks the above stated characteristics [2]. By using a centralized cloud for all computations, we are neglecting the storage and computational resources available on the IoT devices whose capabilities are steadily increasing. The non-utilization of these resources leads to an increased processing cost and substantial delay. The novel distributed paradigms which enable the utilization of these resources are edge and fog computing [3], [4], [5].

Fog computing [3], [4] aims to reduce the involvement of the centralized cloud by processing the data produced by IoT sensors and other devices locally. This leads to faster task execution and reduced communication latency. This mechanism of decentralized processing can be addressed by a fog computing framework. But the main challenge in the fog computing environment is to decide the placement of the services on the geographically distributed and resource constrained fog computing nodes. Also the fog resource provisioning framework has to monitor and analyze IoT data and provision of virtualized resources [6] to these devices to enable computation.

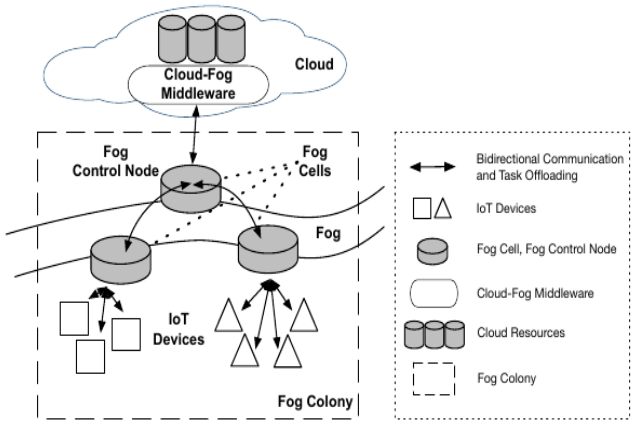

Fig. 1: Overview of an edge-fog architecture [7]

Fig. 1 gives the overview of the edge-fog architecture. The fog landscape is composed of fog nodes to which various IoT devices are connected. There may be one or more fog control nodes which coordinate and control the other fog nodes in the topology. The data produced by the IoT devices are processed locally by the fog nodes and if required it is sent to the cloud for further processing.

We envision a framework that is capable of deploying services on the fog nodes by taking into consideration the resource utilization of each of the fog cells and appropriate provisioning resources for the IoT services. The proposed framework is also aware of the dynamic nature of the topology and hence, it is made to work out of the box in any network configuration of the fog cells.

The rest of this paper is organized as follows. Section II discusses the related work, Section III describes the proposed methodology, Section IV discusses the experimental results with analysis and finally, the paper is concluded with future directions in Section V.

II. Related Work

Hong et al. [8] presented an abstract overview of a high-level programming model consists of APIs and a simple resource provisioning methodology based on the utilization limits of fog resources. This is established on a Platform

TABLE I: Summary of Key Existing Works

Author	Methodology	Remarks
Hong et al. [8]	High level programming model	Support for dynamic scalability and they did not consider the placement of services on fog based resources.
Vaquero et al. [9]	Network Function Virtualization and Software-Defined Networking	Interface between edge and fog components more flexible. But they did not consider the resource provisioning and service placement in fog architecture.
Yi et al. [10]	Three tier architecture	Proof of concept platform on which services are deployed.
Sarkar and Misra [11]	Three tier architecture	Substantial improvement in energy consumption and service latency over cloud.

as a Service approach, in which, the model is hosted on a cloud service, and developers only need to take care of the implementation, testing, and deployment of solutions on this platform. This is responsible for the dynamic scaling of the executed applications during the run time.

The computing instances are generated on-demand to enable the dynamic scaling of the environment as per the specified scaling policies. Parameters such as CPU utilization, and network bandwidth are continuously monitored to trigger the scaling jobs when necessary. To enable dynamic transparent scaling, they also made use of a spatio-temporal object store which also enables the sharing of the application-wide data.

Vequero et al. [9] presented an approach by using Network Function Virtualization (NFV) along with Software-Defined Networking (SDN) at the edge of the network. Using this the dynamic nature and the flexibility of the network along with the user services are made better. The above approaches make the interface between the edge and the fog components of the network more flexible. Recent research challenges in this area are also listed which include the resource management, monetization and discovery of applications.

Yi et al. [10] proposed a three tier architecture with user, fog and cloud. The middle fog layer is composed of six constituents to manage task requests, deployment of services and management of the network.

Some application scenarios are mentioned and an experimental proof of concept is explained. The proof of concept platform is made up of two subsystems, both of which are connected to a cloud service. Separate Open Stack configurations are deployed on both of these subsystems. The parameters like bandwidth, latency, execution time of a facial recognition application are considered for evaluation of the test bed. These results throw light on the betterment of the fog computing paradigm over a centralized cloud scenario.

S. Sarkar and S. Mishra [11] proposed a theoretical fog computing model with a three tier architecture. In the lowest tier, IoT devices are clustered. In tier 2, Edge and fog-cloud gateways are present and the tier 3 consists of the cloud. All the possible states of the system are defined. Performance metrics for the model include energy consumption and service latency. This model has a substantial improvement over the cloud computing paradigm.

Finally we used some selected criteria to evaluate the outcome of the literature review. These criteria include some general concepts as well as some specific topics. The criteria considered are as follows.

1) The Internet of Things (IoT) [1]
2) An Architecture for Fog Computing (FC)
3) Dynamic Topology (DT) [12]
4) Programming Model (PM)
5) A Resource Provisioning Approach (RP) [6]

The internet of things played an important role in developing the fog computing framework. All the frameworks including the fog computing follow a programming model. One of the neglected things in such frameworks is the dynamic topology. This is chosen because of the easy mobility of the IoT devices and the need to track them. Finally, the ultimate goal of a framework is to provision the resources and deploy services over the provisioned resources. Hence we consider resource provisioning as a major factor.

More specifically, dynamic topology is not considered which can change over time and provisioning of resources to maximize the resource utilization and minimize the latency of the network if any. Table I summarizes some of the key existing works.

III. Proposed System

A. Architecture

We propose a resource provisioning framework in the fog computing environment for deploying the IoT application services as shown in Fig. 2. The complete framework is composed of IoT devices, fog cells and fog control nodes. A group of fog control nodes along with arbitrary number of fog cells constitute a fog colony. The user request for task execution is the task request. The service images are the undeployed binary files that utilize the IoT devices to provide some service to the user and these are stored in a shared database. The services are deployed and executed on fog cells or fog control nodes by application of a resource provisioning algorithm. Finally, an application constitutes a set of services which are to be deployed for successful execution of the above said application.

A fog cell has its own compute and storage capabilities. It is responsible for running the IoT services. The IoT sensors are directly interfaced with the fog cell. They aggregate the data from the IoT sensors and if required propagate this data to the upper level fog devices. The responsibilities of a fog cell also include execution of the incoming task requests. For processing of the received task requests, emphasis is laid on in-node processing. If there is need of further processing, the

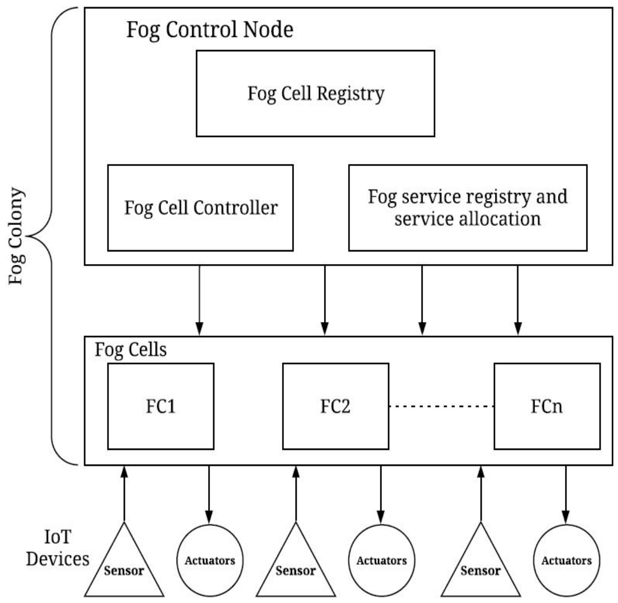

Fig. 2: Proposed Framework

intermediate results can then be forwarded to the parent fog control nodes which may have better compute capabilities.

A fog control node is a fog cell with additional responsibilities. The proposed framework with the list of modules is shown in the Fig 2. The fog control node in the top most level exposes an interface through which the user can submit task requests. The fog control nodes in the lower levels are tasked with monitoring the resource utilization and forwarding the user request to its child fog cells to perform the computations if none of the child fog cells are capable of performing the task (to maintain QoS). It is also responsible for maintaining the topology of the framework by regularly monitoring heart-beats received from the child fog cells. Each fog control node maintains and periodically updates a list of the resource utilizations of its children fog cells. This way, the fog control node at every level is aware of the capabilities of its child fog nodes and hence at the top most level, the primary fog control node knows exactly where to deploy the incoming requests. However, this is also controlled by the resource provisioning algorithm used which is illustrated in further sections.

B. Algorithms and Techniques

The edge-fog computing framework is responsible for provisioning of the resources in the children fog cells such that it ensures maximum utilization of the resources. The framework is also made generic so as to plug any of the resource provisioning approaches as seen fit by the user.

Since the task requests for the IoT device are dynamic, the framework should dynamically allocate resources to the fog nodes and also monitor and assess the allocated resources to the fog nodes. This is done to ensure that none of the fog nodes are overloaded with service requests, the others being relatively free. Thus the workload is balanced between the fog nodes. The parameters we consider to optimally provision the resources are the following.

- Number of containers run on each of the fog cells.
- The total free memory of the fog cells.
- The total CPU utilization of the fog cells.

We consider the cost function which is defined as the weighted sum of the above three parameters.

$$C = \sum_{i=1}^{n} \sum_{j=1}^{3} w_{ij} \times component_{ij} \times f_j(t)$$

where, i ranges from 1 to the number of nodes, and w_i is a weight between 0 and 1, $component_i$ is an array containing the values of the above mentioned parameters for each node, and $f_j(t)$ represents that these values vary with time.

We make use of first fit algorithm (Algorithm 1) and the branch and bound based exact algorithm (Algorithm 2) for service placement on fog nodes. One ensures timely deployment of the service requests and the other ensures the overall optimality of the allocation. Since we want to a most optimal deployment to the fog cells, the deployment therefore represents an optimization problem. And therefore, the use of branch and bound based exact algorithm is justified. Two cost functions based on varying the weights are considered respectively for each of the two algorithms. While, there is no optimization of the cost function in the first fit algorithm, whereas the branch and bound algorithm deals with the optimization of the cost function. The first fit algorithm alone does not guarantee an optimal placement of the fog cells to the incoming requests. This is because the first fit algorithm does not look beyond the greedy nature of selecting the first node that satisfies the requirements. Hence, we considered the branch and bound based exact algorithm.

The branch and bound algorithm, although proceeds in a way similar to brute force approach, proceeds carefully as to not explore the entire search space. Hence, it is a much optimized algorithm. The cost function we consider is empirically tested based on the weights assigned to the parameters under consideration is cost per unit resource usage in the fog node.

The only problem with branch and bound algorithm is that it cannot be used for single use task requests like the first fit algorithm. In the first fit algorithm, it really does not matter if the size of the request set is 1. However, if the request set size is set to 1, the whole point of branch and bound algorithm is defeated and it behaves just like the first fit algorithm.

Hence, we make use of both the algorithms in a combined manner. Initially, the task requests are allocated to the fog nodes in a first fit approach. However, the branch and bound algorithm is run periodically to make sure that the whole assignment is the most optimal. Hence, our proposed combined approach is robust and optimal at the same time.

The analysis of Algorithm 1 leads us to the following. Sorting step takes $O(n \log n)$ time and searching for the first

Algorithm 1 First Fit Algorithm

```
Input: Set fognodes, Dict requirements
Output: Node allocated_node
Cost Function Used: -0.4*memory/100 + 0.45*containers/100 + 0.1*cpu_percent/100
    // init fields
utils ← {}
for node ∈ fognodes do
    utils[node] ← get_util_node(node)
end
sorted_utils ← sorted(utils) based on cost function
for node ∈ sorted_utils do
    available_resources ← node[nodename]
    if available_resources[memory] ≥ requirements[memory] and available_resources[cpu_percent] $\leq 100 - requirements[cpu]$ and available_resources[containers] < 5 then
        return allocated_node ← node[nodename]
    end
end
```

fit takes $O(n)$ time resulting in the total time complexity of the algorithm of the order of

$$O(n \log n) + O(n) = O(n \log n)$$

where, n is the number of fog nodes.

The steps involved in the branch and bound approach include enumeration of the search space of the candidates. The algorithm visits each branch which represent the subsets of the search space. Each branch is then checked for upper and lower estimated bounds on the optimal solution. The branch is discarded when it cannot produce a better result when compared to the already found best solution by far by the algorithm.

Efficient estimation of bounds is the key ingredient to this algorithm. If the above said bounds are unavailable, the algorithm proceeds like a brute force search. The objective of this approach is to find a value as the input to the function that either maximizes or minimizes the output of the cost function.

A branch and bound algorithm operates according to the following two principles (steps): First step is to recursively branch the search space into smaller spaces, then optimizing the value of the function on each of these smaller sub-spaces. Only branching alone will not produce optimal results as it will amount to a brute force search. In order to completely optimize the objective function, the algorithm maintains the bounds of each of the branches. Once the bounds are found, the second step is to prune the inefficient branches, or rather, the branches that do not contribute to the minimization of the cost function below the current best from the search space and the details are described in Procedure 1.

The analysis of Algorithm 2 leads us to the following. We note that Procedure 1 is internally called by Algorithm 2. Procedure 1 takes $O(n \times j)$ time where n is the number of fog nodes and j is the number of job requests. And, Procedure 1 is run for each job request for every fog node. Hence the overall time complexity of Algorithm 2 is

$$O((nj)^2)$$

This is a huge improvement from the brute force approach of $O(j^n)$.

Algorithm 2 Branch and Bound-based Exact Algorithm

```
Input: Set fognodes, Set requests
Output: List allocations, Dict costmatrix
Cost Function Used: $\frac{2}{5}$*containers + 0.04*(total_cpu_usage) + 0.01*(total_memory_in_use)
    // init fields
costmatrix ← {}
for node ∈ fognodes do
    utils[node] ← get_util_node(node)
    costmatrix[node] ← []
end
for node ∈ fognodes do
    for request ∈ requests do
        node_utils ← utils[node]
        reqs ← request.requirements
        costmatrix[node].add($\frac{2}{5}$*node_utils[containers] + 0.04*(node_utils[cpu_percent]+reqs[cpu]) + 0.01*(node_utils[memory_percentage]+reqs[memory]))
    end
end
assigned_indices ← []
allocated_nodes ← []
allocations ← []
heap ← []
cost_incurred ← 0
for node ∈ fognodes do
    for i ∈ (0,len(costmatrix[node])) do
        if i ∉ assigned_indices then
            min_cost ← Call Procedure 1
            cost_level ← costmatrix[node][i] + min_cost
            heap.push(cost_level,i)
        end
    end
    assignment = heap.pop()
    allocated_nodes.add(node)
    assigned_indices.add(assignment[1])
    allocations.add((node,assignment[1]))
    cost_incurred ← cost_incurred + costmatrix[node][assignment[1]]
    heap ← []
end
return allocations, costmatrix
```

Fig. 3 shows the flow and deployment of a task request which is submitted to the fog control node after registering the service with the fog control node. Note that we refer to the fog control node at the topmost level as the fog master since it controls the deployment in the current fog colony.

Procedure 1 Computation of Minimum Cost

```
Input: Dict costmatrix, Node node, int req_index, int
       cost_incurred, List assigned_indices, allocated_nodes
Output: int min_cost
// init fields
cost ← cost_incurred
local_assigned_indices ← assigned_indices
local_assigned_indices.add(req_index)
for $n$ in $costmatrix$ do
    if $n == node$ then
        continue
    end
    if $n \in$ allocated_nodes then
        continue
    end
    min_val ← MAX_INT
    min_index ← -1
    for $i \in$ (0,costmatrix[n]) do
        if $i \notin$ local_assigned_indices and costmatrix[n][i] $<$
          min_val then
            min_index ← i
            min_val ← costmatrix[n][i]
        end
    end
    local_assigned_indices.add(min_index)
    cost ← cost + min_val
end
return cost
```

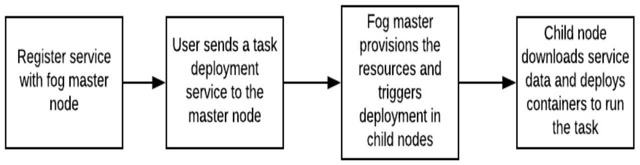
Register service with fog master node
User sends a task deployment service to the master node
Fog master provisions the resources and triggers deployment in child nodes
Child node downloads service data and deploys containers to run the task

Fig. 3: Workflow of deployment of task requests

Each of the fog control nodes and fog cells employ different kinds of databases. Each fog cell uses a local redis instance. The fog control node runs two instances, one is the local redis instance, and the other is the shared redis instance. The local instances in both the cases are used to store the configuration parameters, IP addresses of the parent nodes and device statistics such as the resource utilization of themselves. The shared redis instance is used to store the service images of the deployable services. Whenever the fog cells are allocated a new task, they can download the service images from this shared redis instance.

The different technologies used in the fog nodes as described below:

Using celery, we deployed a complete monitoring system that is responsible to get the heartbeats from the corresponding child nodes to check whether they are alive or not. It also monitors the resource utilization in each of the children nodes and avoid any resource provisioning if they are critically low on resources.

REST is the main factor in driving the communication in micro services architecture due to the heterogeneous and light software component foot print. The minimalistic feature of the Flask python framework gives an edge to it in order to use it in our project for communication purposes.

Celery is used as a broker to transport the service data and to monitor the resource usage and to get the hearbeats of the children nodes. Each type of node (fog control node and fog cell) run one worker each of Celery. The worker in the fog control node is responsible for getting the heartbeat and monitoring the resource utilization's in the children nodes. It is also responsible for triggering the children to download the service data from the master node. The celery worker in the fog cells are responsible for execution of the services deployed.

IV. Experimental Results and analysis

A. Experimental Set up

We make use of the low cost devices such as Raspberry Pi's to run this framework on them. We ran the experiments on a cluster of three Raspberry Pi 3 Model. All of them run on Hypriot which is a lightweight operating system for IoT that enables built-in docker support. The framework, ie., the fog cells and the fog control nodes themselves are containerized. Other services also can be containerized and executed on this platform using Docker hooks.

Each incoming task request is first mapped onto a service which is then deployed on the IoT devices using Docker containers [13]. The nodes which are eligible for being deployed upon are fog control and fog cell nodes.

B. Results and Discussion

To evaluate the performance of the proposed fog computing framework, we consider the following parameters.

- Fog deployment time: The time elapsed from the receipt of the request, including the provisioning time and finally till the services are deployed on the provisioned resources.
- Provisioning time: The time taken for the provisioning algorithm to run.

Fig. 4 presents the variation of total execution time to the deployment time. Each deploy request is considered as an event. As evident from Fig. 4, the deployment time compared to the overall execution time is of the order of one-tenth of the execution time. The initial spike is because of the image creation for the first time. Fig. 5 represents the provisioning times when the topology is made up of one fog control node and one or two fog nodes. As evident, when there is only one available node for deployment, the provisioning algorithm needs to just check for the available resources in the available node and assign the jobs. Hence the time taken here is much lesser than the case with two nodes where the provisioning algorithm needs to first decide where to deploy and then check for the available resources in the chosen node.

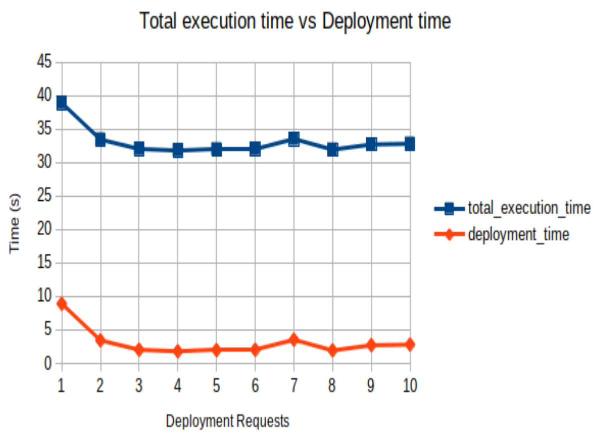

Fig. 4: Execution and deployment times of requests

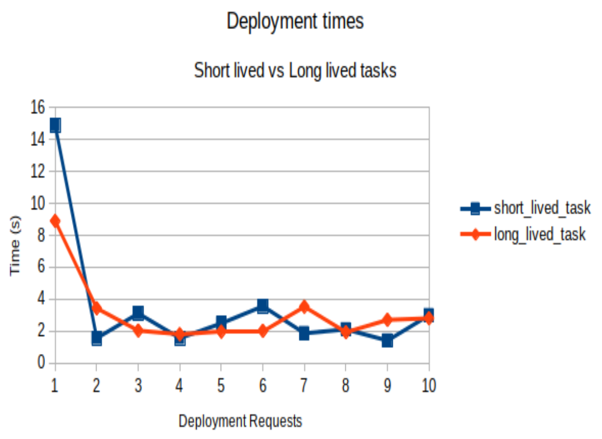

Fig. 6: Deployment times for short and long running tasks

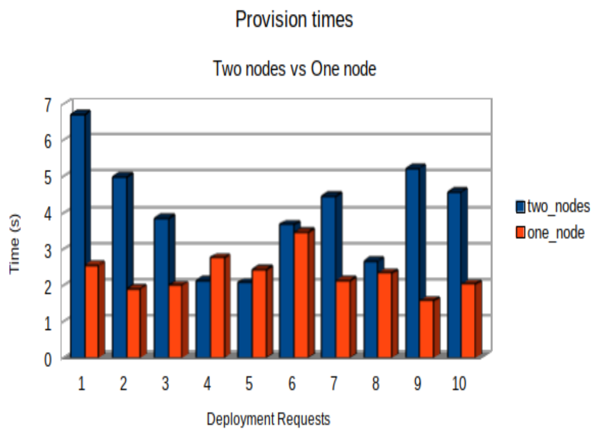

Fig. 5: Provisioning times

Short lived tasks are the tasks that finish within 10 seconds while long running tasks are the tasks that take at least 30 seconds to complete. The results in Fig. 6 also suggest that the deployment time does not depend on the execution of the tasks as it only depends on the container initialization and start-up times.

V. Conclusion

The Internet of Things is a popular and a promising trend in terms of scientific and economic value. Fog computing is the latest trend in the field of Internet of Things. During the implementation phase, two important algorithms, namely, first-fit and branch & bound based algorithms were developed and evaluated. Lastly, the framework was evaluated on the parameters described above and meaningful and insightful results were obtained. The future work includes scaling this framework to more than three nodes in total and evaluating the performance of the provisioning algorithm. More and more services can be deployed and tested based on other parameters like network bandwidth and power consumption.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, “Cloud computing resource scheduling and a survey of its evolutionary approaches,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[3] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: A platform for internet of things and analytics,” in *Big data and internet of things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[5] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, “Fog computing: Principles, architectures, and applications,” in *Internet of Things*. Elsevier, 2016, pp. 61–75.

[6] E. Kalyvianaki, “Resource provisioning for virtualized server applications,” University of Cambridge, Computer Laboratory, Tech. Rep., 2009.

[7] O. Skarlat, K. Bachmann, and S. Schulte, “Fogframe: Iot service deployment and execution in the fog,” *KuVS-Fachgespräch Fog Computing 2018*, p. 5, 2018.

[8] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, “Mobile fog: A programming model for large-scale applications on the internet of things,” in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ser. MCC ’13. New York, NY, USA: ACM, 2013, pp. 15–20. [Online]. Available: http://doi.acm.org/10.1145/2491266.2491270

[9] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2677046.2677052

[10] S. Yi, Z. Hao, Z. Qin, and Q. Li, “Fog computing: Platform and applications,” in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, Nov 2015, pp. 73–78.

[11] S. Sarkar and S. Misra, “Theoretical modelling of fog computing: a green computing paradigm to support iot applications,” *IET Networks*, vol. 5, no. 2, pp. 23–29, 2016.

[12] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, “Incremental deployment and migration of geo-distributed situation awareness applications in the fog,” in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 258–269.

[13] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors,” in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 275–287.