Fog Computing Serverless Architecture for Real Time Unpredictable Traffic

Navjeet Kaur1 and Ayush Mittal2

1Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India
2System Engineer Specialist, Infosys limited, Chandigarh, India
E-mail: navjeet.kaur@chitkara.edu.in

Abstract. Fog computing is a middle layer in the cloud-IoT model. The additional layer developed to offload the traffic from the cloud by sharing its workload. The offloading of tasks in Fog computing is done by deploying intelligent nodes near the edge of a network to process data locally. The smart nodes are also called Fog nodes, which act as a mini cloud within an area of the network. The Fog network comprises various Fog nodes and a Fog manager which handles their communication and allocates each incoming request to optimal capacity Fog node. The Fog manager at the Fog layer is busier in managing the network than providing the solution to different applications, which should be its primary responsibility. The vast responsibilities list of a Fog manager degrades its efficiency to provide solutions on time. This problem is targeting through this paper, and the proposed work provides a solution through a serverless approach at the Fog layer.

Keywords— Cloud Computing, Fog Architecture, Fog Computing, For Serverless Architecture, Serverless Architecture

1. Introduction
The evolution of information technology is increasing exponentially. The dependency and use of technology is essential these days to accomplish our everyday task in time. The technology progress increases the business productivity of individuals, corporation, and industry. It is also important to update the technology timely as per the increasing demand for Quality of Service (QoS). Fog computing is also one of the latest technology introduced to enhance the QoS provided by the traditional cloud computing by sharing its workload and preprocess it locally within a limited area. The preprocessing of data is usually done by Fog node(s), which behave like a mini cloud, also called as cloudlet, having limited storage and computation capabilities. These Fog nodes are capable of scheduling data locally near the edge of a network. The accomplishment of this scheduling feature of Fog node becomes a challenge in the Fog environment because of the heterogeneous nature of both Fog nodes and tasks to be scheduled. The Fog manager node is handled by a software developer responsible for handling this whole functionality. A software developer who is providing the solution to different problems also have the responsibility of

• Managing and processing incoming requests.
• Managing resources or machines i.e other Fog nodes in its network.
• Load balancing among Fog nodes.
• Continuous management of faults and failure.
An underlying Fog computing layer in the basic cloud model depicted through Figure 1. Previously, application development or solving a problem depends on the platform to platform. The developer who built an application with all the dependencies and libraries works well on their machine. When the application moved to another location for deployment, there is a lot of friction in running the same application with the same efficiency on another machine. This may be due to some missing dependencies or libraries and pose a big issue in smooth application deployment on different location. It has been realised that the development of applications should be platform-independent in order to execute the application anywhere anytime. This problem of platform dependency solved by dockers or containers [1].

The container approach provides this flexibility by increasing the level of abstraction by hiding operating system and machine details from the developer. This approach lets developers to be application-oriented rather than machine-oriented [2]. The container works by creating executable images of the applications by encapsulating applications along with all dependencies. Various authors [3, 4, 5] performs performance analysis of container with respect to older techniques like Virtual Machines (VMs). Moreover, Luo et al. [6] and Yin et al. [7] also proposes a Fog Architecture with Containers where the approach is compared with VM for Fog computing environment also. Further, containers also face issues of management and failures in a large dynamic environment. Container orchestration is needed to manage the life cycle of containers where provisioning and deployment of each one are being managed by a separate engine [8, 9]. The orchestration process improves the container provisioning process, but software developers still need to maintain a lot apart from developing solutions for problems.

Serverless Computing is a new paradigm that primarily focuses on developer burden of managing these technologies and simplified the process by increasing the level of abstraction in the deployment model of the application. A general comparison between container and serverless based deployment is depicted in Figure 2.

Figure 1. Fog Layer in Basic Cloud Model
The serverless paradigm is successfully implemented by Amazon Lambda, Azure functions, Cloud functions, Openwhisk, and many more. A complete review of serverless technologies was discussed by [10]. The paper tries to use the benefits of the serverless approach in Fog computing to process real-time applications before the deadline.

This paper’s main contribution is proposing a serverless architecture of the Cloud-Fog model that provides fast processing of real-time applications. The paper is further divided into different sections, where Section II explained the technology background. Section III describes the problem statement, and proposed model and conclusion with future direction covered in Section IV.

2. Technologies Overview

2.1. Containers and its Orchestration

The container technology such as LXC, OpenVZ, and most popular Docker completely overtakes VM technology such as Xen, VMWare, and KVM in terms of cost, security, and flexibility. The major difference between VMs and containers lies in the abstraction of the operating system. Containers run on a shared Linux kernel, whereas VMs run on a hypervisor, which abstracts computer hardware architecture and connected devices. As a result, virtual machine images consist of a separate operating system on top of which the application runs. Therefore, it is more heavy-weight compared to the container image, which runs on a common operating system. Moreover, container image pack application code with all related dependencies and libraries makes it most convenient to work different docker installed machines at various locations. Figure 3 is the simplified view of container docker. The example shows three different locations A, B and C with docker installed machines where the developer can pack its code with dependencies in a small container managed by a docker engine.

The docker engine creates an image of the application, which can run on a different system with the
same performance. The main reason behind this is dockers systems share a common operating system which is missing in hypervision based systems which need to manage different operating system [11]. Although containers resolve the platform dependency issue, there is another issue of developer burden of managing the containers with an increase in demand for service. This problem is managed by a container management tool, also called container orchestration [2]. The management tool completely manage the provisioning, fault, and failure of containers. Still, the application developer needs to do a lot of management rather than just building solutions to problems. There are many other open issues of containers discussed by Paraiso et al. [12] which leads to the serverless approach to emerging as the ultimate need of high-demand issues in cloud-Fog computing.

2.2. Serverless Computing
Cloud Serverless Architecture make use of two services which are

- Microservices: This service structure an application into a set of independently deployable, loosely coupled, and highly manageable services. Although the identification of microservices is a tedious task, it has many advantages like failure or fault isolation concerning the whole application and flexibility of choosing the right technology for specific service deployment [13].

- Function as a service: This service lets the developer build a small unit of code called functions executed when some event occurs [14]. A function is a part of a particular micro-service of an application where other dependent functions are running.

Figure 4 illustrates application execution through microservices and functions where an application is divided into several microservices and each microservice holds some functions that might hold some dependencies on each other. Serverless computing also known as function as a service (faas), lets developers build applications without the worry of back-end management. Although it does not mean that servers are not there. The approach is highly efficient in terms of unpredictable traffic and cost. Also, it auto-scale things as per increasing or decreasing demand for the requested service.

3. Problem Description and Proposed Solution
3.1. Problem Description
In Fog computing, there are various networks of Fog nodes where a Fog manager supervises each system. A Fog manager’s responsibilities handled by a software developer at the back end in terms of processing request(s), writing codes for applications, maintaining servers, computing powers, optimal task-node pairing, and recovery in case of failures. This huge list of responsibilities on the developer imposed severe degradation of their efficiency in terms of their primary responsibility, i.e. focusing on business problems and developing solutions. However, the serverless approach is implemented in cloud computing presently. We plan to implement serverless near the edge of the network through Fog computing.
3.2. Proposed Solution
The problem is solved by introducing the serverless approach in the Fog layer as well in the cloud layer to handle the unpredictable traffic in real-time applications. The serverless approach helps software developers build applications comprised of functions called micro-services that run a particular micro-service when triggered. In a serverless approach, the developers need not maintain servers to run their applications. It automatically scales itself as per the incoming traffic, and most importantly, it is highly cost-efficient as it charged only when the function is triggered or running. Moreover, the serverless approach also provides high availability and fault-tolerant. In this approach, complete underlying infrastructure managed by different cloud providers and software developers as a Fog manager needs to manage the incoming requests by providing appropriate solutions. The proposed architecture is given in Figure 5 divided into four different layers which are

- Presentation Layer: This layer deals with the actual applications used by the end-user through smart devices like any web application, smart wearable gadgets, or any end product used by the customer to place service requests.

- Application Layer: Software developers of the application(s) deals with this layer. The layer represents the actual system where the application is running. The developer process each new request based on its requirements and develop functions as per the new different request. If the function already exists for a particular placed request, the developer will just trigger the given input values.

- Serverless Fog Layer: The layer consists of various Fog nodes network with a Fog manager per network. Each Fog manager is responsible for managing its Fog network. Moreover, a Fog manager monitors Fog nodes’ health in terms of their workload at regular intervals. Fog nodes are the ones where the data is cached and processed. The layer is responsible for processing incoming requests locally by triggering functions from its local repository.

- Serverless Cloud Layer: The use of this layer might only be needed to save all results for future analysis and for dealing with an application that is too intensive to be handled by the local Fog network.

3.3. Detailed view
The serverless function is that compute service that lets you run code without provisioning or managing servers. It executes your code only when needed and scales automatically. The most significant advantage of adopting a serverless function is its cost-effectiveness in unpredictable traffic as the provider charge only when the system is running [15]. The detailed view of the Fog serverless architecture given in Figure 6. End users place the requests at the presentation layer. Then, the application layer where the developer maintains requests will trigger the specific function or create a solution for the requested service at the serverless interface. The serverless service trigger the lambda function known as Fog function for each service request, which further initiates a mini container. A mini container can manage a limited amount of service requests, and if the load increases, the Fog function initiates a new container service to handle the incoming requests. The Fog function can add or remove containers as per the demand where the Fog manager further pair it with an efficient Fog node act as a Virtual machine to execute the service. A load manager node manages the Fog node management, their workloads, synchronization with Fog manager node, and input/output. Further, monitoring service keeps track of time for which the function is executing on machines for cost calculation at the end. The cloud layer is used for the final data repository for future analysis and statistics.

The serverless approach is of great benefit to the developer, who can now focus on developing a solution to different problems as an independent function. When the developer creates a Fog function, need to specify configuration information, such as the amount of memory and maximum execution time that you want to allow for Fog function. When a Fog function invoked, serverless service launches an execution context based on the configuration settings provided. The execution context is a temporary runtime environment that initializes any external dependencies of your function code, such as database connections or HTTP endpoints. It takes time to set up an execution context and do the necessary “bootstrapping,” which adds some latency each time the Fog function invoked. This latency is only when
a function invoked for the first time or after it has been updated because the Fog function tries to reuse the execution context for subsequent invocations of the function.
Figure 7 explained the developer view where the developer developed independent functions for a particular service request. Note that in the diagram, the term micro-services are independent functions of an application where each function is a service. For example, there are different independent services requests, like service A, service B, service C and so on. For each independent service, a developer will code its solutions in various ways. It can code on its machine, pack it and send it to the serverless service or directly build function on the Fog interface application. Further, serverless service will abstract the rest of the functionality from the developer and create a lambda function, a single executable function, also known as Fog function[16]. There are various other services provided by the serverless framework like

- Container Service along with orchestration: This service will initiate a container or docker service per Fog function. A container can manage a limited amount of service requests. If the load increases, the Fog function will initiate a new container service to handle the incoming request, which was previously handled by the developer. The orchestration service also is managed by the serverless approach, where the containers’ faults and failures are handled.

- Log Service: This service is used to manage the Fog functions and create log or history maintaining the versions.

- Monitoring Service: This service is used to track function execution time to calculate the cost to be paid by the customer of the serverless approach. The serverless approach costs only for the time period when the function is being executed on the Fog node.

![Serverless Architecture from Developer View](image)

**Figure 7. Serverless Architecture from Developer View**

Serverless Fog function automatically executes the function created by the developer on high-availability computing infrastructure and manages the entire administration of computing resources, including server and OS maintenance, capacity provisioning, and auto-scaling, code, and security patch deployment, and code monitoring and logging. The developer just needs to provide a solution.

3.4. *Pros and Cons of the Proposed Solution*

**Pros:**

- The Serverless model has the biggest advantage to a Fog manager in sharing its responsibilities in terms of server maintenance, storage, scaling resources in case of unpredictable traffic, and ensuring high availability.

- The proposed approach is also cost-efficient as the clients need to pay for only the time when a particular request is in process and not for the idle time.

**Cons:**

- The approach completely depends upon a cloud provider to provide support to the software developer.

- The Fog manager might feel less control over the system.
4. Conclusion and Future Directions

Fog computing is in its early stage of development. An intelligent node is deployed near the edge of the network to process the requests locally instead of processing from the cloud which are far away from the actual location. Moreover, the deployment of intelligent Fog nodes near the edge of the network faces a lot of difficulty due to the devices’ limited resource capacity. The Fog nodes network usually managed by a Fog manager node or software developer who builds real-time solutions for problems. The present system is quite a management dependent rather than application dependent. The proposed method targets this issue of manager node who is overburdened with the responsibilities of management systems and application development. The serverless approach brings this efficiency and lets the developer build solutions irrespective of any backend management. The cloud is already implementing these functionalities, and our purpose is to apply it on Fog computing with the overall motive of a better system. The future of serverless computing is bright, but there are many open issues of the serverless approach which required improvement like

- Serverless approach is not a good fit for all kind of applications.
- Limited programming languages choices till now.
- Cost get increases when traffic is predictable.

Our future work will focus on these issues and improve the model on these parameters.

References

[1] Merkel D 2014 Linux journal 2014 2
[2] Burns B, Grant B, Oppenheim D, Brewer E and Wilkes J 2016 Queue 14 70 – 93
[3] Felter W, Ferreira A, Rajamony R and Rubio J 2015 IEEE international symposium on performance analysis of systems and software (ISPASS) 171 – 172
[4] Joy A M 2015 International Conference on Advances in Computer Engineering and Applications 342 – 346
[5] Barik R K, Lenka R K, Rao K R and Ghose D 2016 International Conference on Computing, Communication and Automation (ICCCA) 1204 – 1210
[6] Luo J, Yin L, Hu J, Wang C, Liu X, Fan X and Luo H 2019 Future Generation Computer Systems 97 50 – 60
[7] Yin L, Luo J and Luo H 2018 IEEE Transactions on Industrial Informatics 14 4712 – 4721
[8] Tosatto A, Ruiu P and Attanasio A 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems 70 – 75
[9] Hightower K, Burns B and Beda J 2017 Kubernetes: up and running: dive into the future of infrastructure (“ O’ Reilly Media, Inc.” )
[10] Lynn T, Rosati P, Lejune A and Emeakaroha V 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) 162 – 169
[11] Bernstein D 2014 IEEE Cloud Computing 1 81 – 84
[12] Paraiso F, Challita S, Al-Dhuraibi Y and Merle P 2016 IEEE 9th International Conference on cloud Computing (CLOUD) 718 – 725
[13] Newman S 2015 Building microservices: designing fine-grained systems (“ O’ Reilly Media, Inc.” )
[14] Castro P, Ishakian V, Muthusamy V and Slominski A 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) 2658 – 2659
[15] Pérez A, Molii G, Caballer M and Calatrava A 2018 Future Generation Computer Systems 83 50 – 59
[16] Cheng B, Fuerst J, Solmaz G and Sanada T 2019 IEEE International Conference on Services Computing (SCC) 28 – 35