pFogSim: A Simulator for Evaluating Dynamic and Layered Fog Computing Environments

|  |  |  |  |
| --- | --- | --- | --- |
|  | \*A Report on Progress | |  |
|  |  |
| Clayton Johnson  [Department of Computer Science and Software Engineering](https://www.eng.auburn.edu/comp/index.html)  Auburn University  Auburn, USA  cpjohnson@mavs.coloradomesa.  edu | Jacob Hall  [Department of Computer Science and Software Engineering](https://www.eng.auburn.edu/comp/index.html)  Auburn University  Auburn, USA  jih0007@tigermail.auburn.edu | Jay McKinley  [Department of Computer Science and Software Engineering](https://www.eng.auburn.edu/comp/index.html)  Auburn University  Auburn, USA jbmckin@clemson.edu | Shehenaz Shaik  [Department of Computer Science and Software Engineering](https://www.eng.auburn.edu/comp/index.html)  Auburn University  Auburn, USA  szs0117@tigermail.auburn.edu |

Sanjeev Baskiyar

[Department of Computer Science](https://www.eng.auburn.edu/comp/index.html)

[and Software Engineering](https://www.eng.auburn.edu/comp/index.html)

Auburn University

Auburn, USA

baskisa@auburn.edu

***Abstract*—A new approach is proposed for designing and testing fog computing configurations. pFogSim allows for an n-layered fog design instead of limiting users to a static one or two-layered design. Further, pFogSim accomplishes this with its Puddle based architecture for grouping and pooling distributed resources ultimately allowing realistic, real-world networks to be tested. pFogSim fills the void in which simulations for large, practical networks are lacking.**

***Keywords—Fog Computing, Fog Networks, EdgeCloudSim, pFogSim, CloudSim, Cloud Computing, Dynamic Networking,***

***Edge Computing, Distributed Computing***

1. **Introduction**

pFogSim allows the simulation of n-layered fog networks for a smart city environment. While multiple simulators exist in the marketplace, it should be noted that most of those assume a centralized approach to orchestration. Where a centralized approach refers to a system in which the orchestrator knows the current dynamic state of the entire network and all nodes within it. While a centralized approach may be useful on a small scale, on the scale of a SmartCity, where there may be thousands of nodes within a network[4], having the dynamic state of the entire network available to a single system is not realistic. That is not to say that other simulators only work with centralized approaches, rather there is not a useful implementation of a decentralized approach. What is brought to the table by pFogSim is a simulation of a decentralized orhcestration using the puddle based Hierarchical Autonomous Fog Architecture (HAFA) proposed in [4].

Current proposals also offer a static number of layers of fog devices if any. pFogSim is capable of simulating these environments as well as deeper n-layered designs. Built on top of the EdgeCloudSim[1] simulator, pFogSim adds simulated routing, task migration, and the Puddle design. The distributed Puddle design natively included in pFogSim allows applications to be run close to their originating devices, reducing latency and network traffic to higher-level nodes such as cloud servers. In this paper we will discuss in detail the hierarchical Puddle structure, the service deployment associated with said structure, the dynamic task migration algorithm, the routing system implemented, the architecture of the simulator, as well as the benefits of this simulator and the Puddle structure.

# Defined terms

Before proceeding it is useful to define some terms used further in the paper.

## Dynamic Network/Node Information

In this paper dynamic network information or dynamic state of the network refers to the attributes of the network or a node on the network that fluctuate throughout the context of the simulation. Examples of dynamic state info include: number of tasks currently running on a machine, current CPU or RAM utilization of a machine, or current amount of network traffic at a particular location. These attributes, among others, are considered dynamic because they will change without direct intervention, tasks will start and complete, resources will be used and freed throughout the simulation.

# Static Network/Node Information

Static network information refers to the attributes with do not change within the context of the simulation. Attributes such as the location of datacenters or buildings, and the hardwired links between them, are considered static because they will not change without direct intervention.

# Centralized Approach/Orchestrator

A centralized approach refers to a system where which the orchestrator knows the current dynamic and static state of the entire network and all nodes within it, updated in real time.

# Localized Approach/Orchestrator

A localized approach refers to a system where the orchestrator knows the current dynamic and static state of only a local subset of the network.

# Non-Centralized Approach/Orchestrator

A non-centralized approach refers to a system where the orchestrator has *only* static information about the network.

1. **Related Work**

# A. iFogSim

iFogSim is a simulator designed for a single layer distribution of fog devices [2]. iFogSim was originally to be the basis for our simulator. Its main benefit is its ability to simulate multipart applications running on separate machines. However, iFogSim does not support mobility of devices or advanced routing which is critical to the goals of this study.

# B. EdgeCloudSim

EdgeCloudSim is what the pFogSim simulator is extending. EdgeCloudSim was chosen because it has support for mobile edge devices and is a more useful networking model than iFogSim for this specific case. Another benefit is that EdgeCloudSim makes use of the XML file format for inputting simulation specifications instead of the JSON format used by iFogSim. Similar to iFogSim, however,

EdgeCloudSim assumes a single layer of fog devices which is the goal through the extension with pFogSim.

1. **Simulator Architecture**

pFogSim is an extension of the EdgeCloudSim simulator. The base provided by EdgeCloudSim is used and expanded upon to account for more realistic networking and mobility. Testing the HAFA Puddle Architecture is a significant goal of this simulator as well. Figure 1 below shows the relationships between the modules used by pFogSim.

# A. Orchestration

Orchestration is the core functional part of pFogSim. The Orchestration module contains the five basic orchestrators packaged with pFogSim as well as the Puddle class.

The Puddle class is used to represent the Puddle head in an actual system and contains all the information about the Puddle and its machines. The Radix class is a supporting class used to sort nodes based on distance to a particular location. The Clustering module is used by the orchestrator to organize nodes into their logical layers and Puddles. The clustering algorithms are used in both the FogHierCluster and FogCluster files.

The PuddleOrchestrator class extends the EdgeOrchestrator base provided in EdgeCloudSim and uses the Puddle and Radix classes to provide service deployment according the HAFA Puddle architecture optimized on physical distance.

The CentralizedOrchestrator offers the ideal placement of tasks on the network optimized on physical distance from task to host.

The LocalOnlyOrchestrator shows the response when all tasks are sent to the only the closest bottom level node. The task is rejected if the machine is unable to handle the task.

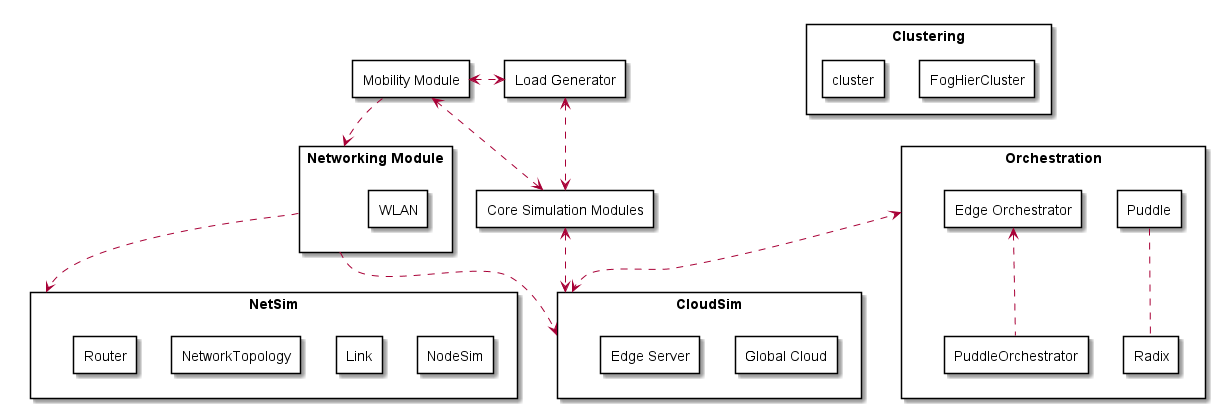
The EdgeOnlyOrchestrator show the response when tasks are only placed on the lowest level of fog nodes, preferring closer nodes.

The CloudOnlyOchrestrator sends all tasks to the cloud.

# B. NetSim

NetSim is the largest module code-wise. The NetSim module contains the entire support for simulating a full network and routing environment, including the NodeSim, Link, NetworkTopology, and Router classes.

The NodeSim and Link classes represent the physical location and network attributes of the machines on the network separate from their resources. These classes are used by the NetworkTopology to form a map of the entire network, representing only the location and static connections. The NetworkTopology is considered to be static for all intents and purposes. That is, it does not contain dynamic information about the network, but rather, represents the physical characteristics of the network that are not likely to change within the scope of the simulation.

 The Router class uses the map represented by the NetworkTopology to find the least latency path from one node to another. While the router does contain the network map, it does not require the up to date dynamic state of the network and thus does not truly qualify as a centralized system.

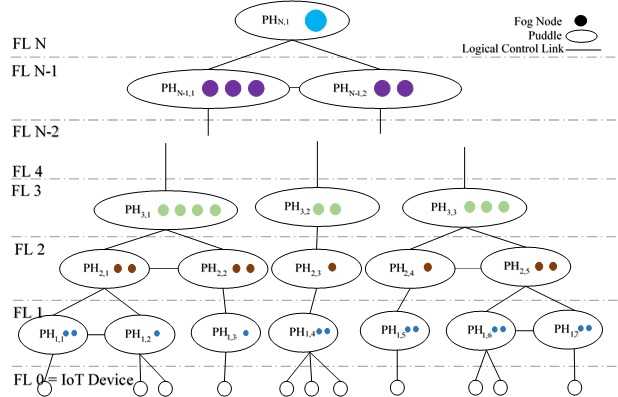
**Figure 1**

1. **Puddle Orchestrator**

# A. Hierarchical Structure

The Puddle structure divides the fog devices (nodes) into a logical hierarchy with the cloud being the highest level and the edge devices such as sensors and actuators being the lowest level. Devices on the same level are assumed to be homogeneous (or very nearly so). We are assuming that any network provider would desire to have consistent and predictable network devices as it decreases training costs and replacement costs. Within a level, the devices are further divided into “Puddles” of machines that are physically close to one another. This is done by our FogHierCluster algorithm. To begin the process, FogHierCluster requires a list of all nodes within the network. Then it separates the nodes out by the level by which they reside on the network. For each level, FogHierCluster hands off the list of nodes within said level to FogCluster to allow the cluster data to be built appropriately. The clusters are finally made with a hierarchical clustering algorithm. The layers, each already clustered, are then lain atop one another and organized vertically. They are passed later to Puddles and the Network model.

Out of each Puddle, one of the machines is designated as the Puddle head, within the code this is represented by the Puddle class. The Puddle head maintains a running total capacity of resources in the Puddle as well as the highest single instance capacity for the Puddle, as well as the list of machines in that Puddle. The Puddle head also holds information for the links to parent and child Puddles relative to this Puddle.



**Figure 2**

Puddles are organized into parent-child relationships between levels. All Puddles (except the cloud itself), are associated with a parent Puddle that is one level higher and physically nearby. Each Puddle (except those on the bottom layer), is therefore also associated with a list of one or more child Puddles that are one level below and physically nearby. The Puddle structure is strictly logical, used for service deployment, and does not restrict network traffic.

# B. Puddle Service Deployment

The service deployment algorithm for PuddleOrchestrator is a localized algorithm designed to leverage the Puddle architecture. The first step is to identify the minimum fog layer on which the task or application can be run. Once the minimum layer is found, the nearest Puddle of that layer (called the local Puddle) is contacted and the Puddle head is queried if the task can be run on that Puddle. If the Puddle does not have the available resources, a nearby Puddle of the same level is identified as an alternate.

The nodes of the local Puddle (or alternate) and the all the nodes on its parentage line are collected into a list. That is, it collects a list of the nodes belonging to the local Puddles of the minimum layer and up. For example, in a 7 layer environment if a task needed the resources of at least level 4 then it would collect a list of the nodes from the local level 4, 5, 6, and 7 Puddles. Even if an alternate Puddle was selected for the lowest level, it would still follow the lineage of the local Puddle instead of the alternate to get the closest Puddles of the higher layers. This algorithm is distributed because it only requires knowledge of the local Puddle head, which communicates with its ancestor Puddle heads to collect the local list of candidate nodes.

Once a list of candidate nodes has been collected, they are sorted by distance to the task. Since it is sorting based solely on distance it is possible to use radix sort as the basis for this step in the algorithm. Further, since it is not concerned with the entire network, but only a small portion of it, a rather inconsequential time complexity may be assumed. If integer *n* is let to be the total number of nodes on the network, and the integer *p << n* be the maximum number of nodes in one Puddle, and the constant *s* be the number of fog layers in the environment, then the number of nodes through which we must sort follows as *s\*p*. Using radix sort, the time complexity is *O(s\*p)*, but since *s* is a constant for the system, the time complexity of sorting these nodes to be *O(p)*.

From the sorted list of nodes it chooses the closest machine which can handle the task regardless of that machine’s layer. The reason for this is to optimize both network latency and compute utilization. If choices are restricted to only the lowest level a closer node might be ignored simply because it is of a higher level. By selecting the very closest node that can run the computation, it is guaranteed that it has the lowest network latency available and lowering the chance of starving a node of energy.

# Dynamic Service Migration

This simulator supports dynamic service migration of tasks across the network. As mobile devices move, they often leave the Puddle where their task is running. This poses a problem for latency-sensitive applications. In order to maintain low latency for all tasks, if a task moves a specified distance away from where its task is running, then the service will be migrated to a more local machine Migration of tasks is accomplished by reapplying the service deployment algorithm and finding the appropriate machine for the task to run on. Once the new machine is identified the task can be moved with its data and bound to the new machine to continue running. Data transfer time is accounted for during this transfer to allow for more realistic simulation.

1. **Centralized Orchestrator**

# **Structure**

The centralized orchestrator shows the ideal placement for tasks on the network. The optimal placement is calculated based on the distance from the task to the host.

# Centralized Service Deployment

The service deployment for the centralized approach is a modified version of the puddle deployment algorithm. Instead of using the local puddle information, the central approach uses information from the entire network and deploys the task to the closest node that can handle the task. The orchestrator takes the list of all nodes on the network and sorts them based on distance (using the same radix sort algorithm detailed in the puddle orchestrator). The orchestrator will then iterate through the list until it finds a node that can handle the task, then deploy the task to that node.

1. **Local Only Orchestrator**

# **Structure**

The local only orchestrator places the task on the closest bottom layer fog device.

# Local Only Service Deployment

The local only orchestrator searches the lowest level of fog devices and attempts to deploy the task to that device. If the task cannot be run on that device then the task is rejected and cannot be run.

1. **Edge Only Orchestrator**

# **Structure**

The edge only orchestrator deploys all tasks to the lowest layer of fog devices.

# Centralized Service Deployment

The edge only orchestrator sorts all of the bottom level fog devices by distance and iterates through them until a device is found that can run the given task.

1. **Cloud Only Orchestrator**

The cloud only orchestrator places all tasks on the cloud.

1. **Rounting and Network Model**

# A. Network Representation

The network is represented by nodes and the static links between them. The nodes are interpreted from the same XML file that EdgeCloudSim[2] uses to read its hosts. Only a few additional node attributes were needed to completely satisfy the simulator’s needs. The links are read from their own separate XML file. Each node in the network representation corresponds to node in the Puddle architecture, and contains the node’s location and the static links associated with that node. A node also stores other useful data including whether the associated device is a Wi-Fi access point, whether the device is moving and if so according to what vector, and the WLAN ID associated with the device.

A link contains the locations of the link’s endpoints (called the left and right endpoints), both of which must be valid nodes, and the left and right latencies for the link. The so called left and right latencies refer to the static delay associated with travelling the link from the right node to the left node (left latency), and from the left node to the right node (right latency). These left/right values are assigned by the distance from the endpoints of the link. The default value used is that one kilometer will add 0.03 millisecond to the transfer. [5] All these links are assumed to be bidirectional.

The links and nodes together form a network topology. The network topology structure holds all of the nodes in the network and ensures that there are no isolated nodes (i.e. a node with no links to anything else) and no dangling links (a link where there is no node at one or both of its endpoints). The network topology also contains the list of Puddles that the clustering algorithm returns.

# B. Routing

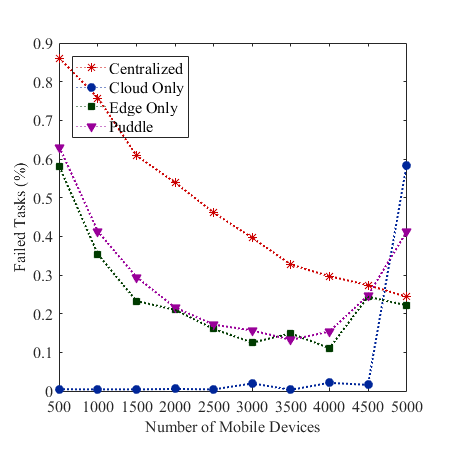
The simulator assumes a central router. This assumption is to abstract routing as a baseline for the ideal routing for the network. We make this assumption on the basis that the router does not need to maintain the full status of the entire environment. Rather, the router only requires the static map of the network and the links that connect it. This is purely for the simulator and any interaction between a node and the router is to simulate the activity of a realistic one. This router does not assume every node has the image of the full network. It cannot be emphasized enough how this simulator is made for a decentralized system in which it is unrealistic to know the state of the network at every time. The router does not violate this since it only ever gives information the nodes would naturally have in any given circumstance.

The router takes source and destination nodes and the network topology. It reads the network topology as a weighted directed graph with nodes being vertices and links and delays representing edges and their weights. The router then runs Dijkstra’s algorithm on the graph from the given source node. Afterwards, the router finds the destination node and builds a linked list of the nodes that form the shortest path from the source to destination.

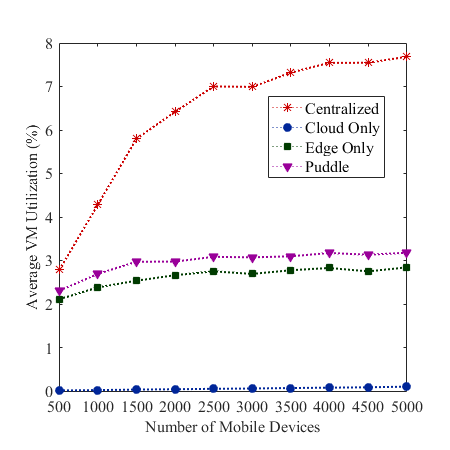
The router passes this linked list to the network model to calculate total latency. The NetworkModel is an abstract class in EdgeCloudSim responsible for calculating proper latencies in the network. The network model calculates the local delay due to congestion at each node and adds all the congestion and static delays to find the total delay from source to destination. The default network model packaged with pFogsim is the ESBModel representing an Equal Share Bandwidth model for calculating the local delays due to congestion. The ESBModel will take the total available bandwidth for a given router and dived it equally among the resources connected to the device.

1. **Testing**

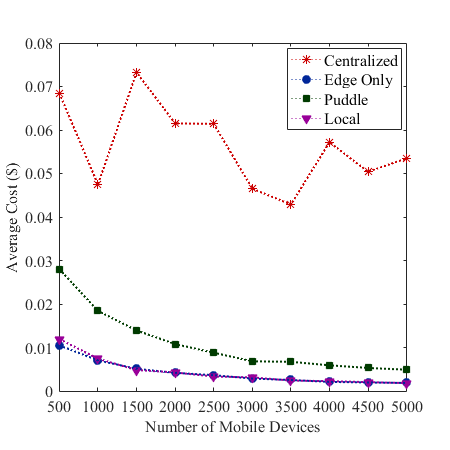
We can track average network delay, percentage of failed tasks, average cost, VM utilization, distance from task to host, and hops from task to host, among other metrics. A sample run was executed with a data set consisting of approximately 1100 nodes representing a sample of public building locations from the city of Chicago. There are some sample graphs below showing example outputs of the simulation. These plots are generated by MatLab scripts from the log files produced by the simulator.

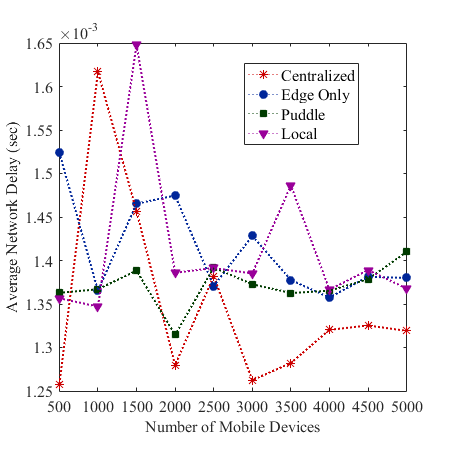


**Figure 3**

****

**Figure 4**

****Figure 5****



****Figure 6****

1. **Limitations**

# Co-Located Resources

The simulator requires that all resources have unique locations. If there are multiple resources in a single location, they should be entered as single object in the code.

**REFERENCES**

1. C. Sonmez, A. Ozgovde and C. Ersoy, "EdgeCloudSim: An environment for performance evaluation of Edge Computing systems," 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, 2017, pp. 39-44.
2. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. Softw Pract Exper. 2017;47: 1275–1296.
3. Scoca, Vincenzo & Aral, Atakan & Brandic, Ivona & De Nicola, Rocco & Uriarte, Rafael. (2018). Scheduling Latency-Sensitive Applications in Edge Computing. 158-168. 10.5220/0006706201580168.
4. Shehenaz Shaik and Sanjeev Baskiyar. 2018. Hierarchical and Autonomous Fog Architecture. In ICPP 2018: 47th International Conference on Parallel Processing Companion Proceedings, August 13-16, 2018, Eugene, OR, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/ 10.1145/3229710.3229740
5. Yousefpour, Ashkan, Genya Ishigaki, Riti Gour, and Jason P. Jue. "On reducing iot service delay via fog offloading." IEEE Internet of Things Journal (2018).