I. J. Computer Network and Information Security, **2024, 1, 85-96**
Published Online on February 8, 2024 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2024.01.07

Container-to-fog Service Integration using the DIS-LC Algorithm

Aruna. K.*
A.V.C. College of Engineering, Department of Information Technology, Mannampandal, Mayiladuthurai, 609305, India
E-mail: avcce.aruna@gmail.com
ORCID iD: http//orcid.org/0000-0002-0038-3481
*Corresponding author

Pradeep. G.
A.V.C. College of Engineering, Department of Computer Applications, Mannampandal, Mayiladuthurai, 609305, India
E-mail: pradeep.g8@gmail.com
ORCID iD: https://orcid.org/0000-0001-5016-1601

Received: 11 March 2022; Revised: 12 April 2022; Accepted: 17 June 2022; Published: 08 February 2024

Abstract: Containers have newly emerged as a potential way to encapsulate and execute programs. In contrast to virtual machines, each container does not have its own kernel and instead shares the host systems. Containers on the other hand are more lightweight, need fewer data to be sent between network nodes and boot up faster than VM. This makes containers a feasible choice, particularly for hosting and extending the services across the fog computing architecture. The major purpose of this paper is to describe the Distributed Intelligent Scheduling based Lightweight Container algorithm (DIS-LC), which is a revolutionary way for container to fog-services integration and resource optimization. In this proposed algorithm is compared to the least connection algorithm, round-robin algorithm and Ant Colony Optimization-based Light Weight Container (ACO-LWC). Operating cost and traffic cost are used to validate the suggested algorithm. Fog node running costs are divided into two categories: CPU and memory. When compared to current algorithms, quantitative research demonstrates that the proposed DIS-LC scheme gets the greatest performance in terms of all metrics. This demonstrate the algorithm is efficient. Finally, the performance of containerized services and resource management systems is evaluated using the iFogSim simulator.

Index Terms: Container, Docker, Fog Computing, Scalability, Internet of Things, Self-organized Network.

1. Introduction

A container is a generic unit of software that encapsulates code and all of its prerequisites so that the programme may be moved easily and safely from one development platform to another. Containers allow for the separation of services and the light movement of microservices from one fog device to another as needed. Based on the availability of resources and workload functions to address the dynamic micro-service positioning issue for fog devices. Cloud computing is a popular alternative for deploying distributed applications. However, for some IoT applications, the cloud is not an appropriate platform. Fast explosive flows from IoT applications enhance the reaction time for each packet processed by the cloud. To solve this limitation, Cisco proposed fog computing [1] for time-critical IoT applications, which employs network computation to offer users with the best service level possible (Quality of Service). Cloud-based IoT/M2M networks becoming more overwhelmed with large quantities of data traffic as more IoT/M2M (Machine-to-Machine) machines join the distributed network. A Fog system is an extension of cloud computing that creates an additional network, additional locations and computing paradigm in the cloud server. Fog focuses on ease of use, low latency and quick response compared to the cloud. The IoT and fog computing systems are used in the FoT framework. As the fog computation within the local server or tiny server is called FoT server. It manages the user's request. The major components of fog infrastructure are FoT server, FoT devices and FoT gateways. These elements are provided by the FoT for local processing. The fog paradigm is a layered model that focuses on providing ubiquitous access to a common and visible computing resource pool. The core idea behind the model is to protect all of the benefits of the cloud. Fog nodes are located between the edges of communication networks and have the ability to use remote and delay sensitive applications.

iFogSim is the most common simulator based on CloudSim for fog and IoT environments. It is a multi-platform, open source and scalable. Modeling and simulation of infrastructures involving millions of fog nodes, IoT services, sensors, and actuators is possible. iFogSim requires user-defined fog infrastructure, mapping, and IoT service planning techniques, and it allows evaluation with a variety of criteria like latency, network, power consumption and operating expenditures. In the iFogSim environment provides resource management and planning policies. It computes performance metrics and simulates devices, cloud data centers, sensors, network connections, data flows, and flow processing applications. Furthermore, iFogSim has two unique levels of simulated services for power monitoring and resource management: work placement and job planning. It is a high-performance open source tool for fog systems, edge computing and IoT. It is used in conjunction with CloudSim, a popular application for modelling cloud systems and managing resources. The iFogSim simulator was used to evaluate the proposed architecture's scalability. The results of the experiments reveal that the framework is effective at reducing the system's reaction time to satisfy resource restrictions. It can lower the application's average response time. Furthermore, the proposed architecture allows for additional flexibility in terms of microservice isolation and lightweight implementation.

The primary research objective of this paper is:

- It is described how to use a docker container with several services.
- A new Distributed Intelligent Scheduling based Lightweight Container (DIS-LC) algorithm is proposed for container-to-fog-services integration and efficient resource utilizations.
- The proposed algorithm is analyzed using parameters like CPU, memory utilization and network latency of containers with fog nodes.
- Determine the performance of containerized services and resource management solutions with the iFogSim simulator.

The remainder of the paper is structured as follows. Section 2 covers methodology, as well as a related work study and a literature review. Section 3 describes a docker-based infrastructure for multi-service container applications. Section 4 focuses on boosting scalability through the usage of a fog environment. Section 5 focuses on the design and execution of the iFogSim simulator. Section 6 discussed the proposed algorithm. Section 7 addressed the result and discussion. Finally, Section 8 presents the conclusion of the work and discusses the scope of future effort.

2. Related Works

This section discusses the ideas and technologies that are used in our fog-based scalable container service architecture.

Some of the recent papers have dealt with the issue of the placement in fog devices by the micro-service. In [2,3], the authors addressed the problem of application placement by putting Virtual Machines (VMs) in fog or edge devices. However, VMs use requires a high resource consumption; the container (light virtualization technology) has been suggested by [4,5] to prevent this overhead by the proposed architecture for speeding up the application deployment. [6,7] suggested placement of containers in the hierarchical fog environment and mapping module algorithm respectively. The goal of fog computing is to bring cloud power (compute, network and storage) to the cloud-to-cloud continuum of things [8,9]. Fog computing, due to its proximity and localization, can deliver lower latency and faster responsiveness than cloud computing. As a result, fog should be viewed as an extension of the cloud. The Fog network's architecture might be hierarchical, indicating data from end-users. Devices can be analyzed at several levels, with each level doing complex data filtering and analytics to determine whether to transfer to the next level for more Fog/Cloud processing. CloudSim is the most popular cloud simulation software for cloud modelling and application development. It's a Java-based discrete event-based simulator that doesn't simulate network components like routers and switches (virtualize). Previous research [10,11] investigated the scalability of an IoT/M2M cloud platform where cloud resources may be scaled up or down dependent on IoT traffic loading. Dealing with IoT traffic only in the cloud, on the other hand, would not entirely solve the scalability issues. IoT traffic must be handled before it enters the cloud to improve the scalability of the IoT/M2M network. This indicates that the Fog network, as a continuum between the cloud and the edge device, must be elastic. i.e. capable of flexibly and dynamically scaling in/out serving instances through Fog nodes to accommodate incoming traffic The one M2M network is a global IoT/M2M network that offers communications with service layer features. Heterogeneous IoT/M2M devices can not only function, but also act as a platform for the convergence of diverse IoT/M2M technologies. Nonetheless, for the single M2M platform, scalability remains an unresolved issue. Our research focuses on migration with the goal of boosting scalability from the cloud to the fog, one M2M to a Fog Computing architecture. Thousands of Fog nodes can be utilized in Fog infrastructures (such as those for cities) to install thousands of IoT service instances and support millions of smart items. It is vital to research and evaluate solutions for such complex systems using simulation methodologies, therefore lowering the cost of evaluation and the time necessary before implementation in real systems.

In [12] SOFT-IoT (Self-Organizing Fog of Things Internet of Things): introduces the concept of fog computing, in which part of the data processing capabilities and service delivery activities are handled locally on "small servers," that is, close to where the data is processed. SOFT-IoT, an alternate option for a centralized cloud computing method, enables algorithms to simplify the concept of computing and focus on key operations performed on virtual machines. SOFT-IoT,

in this sense, enables cloud-level operation; complicated processes can store, analyze and resolve other data. In other words, it happens locally to avoid present infrastructure restrictions and data processing in SOFT-IoT, data processing, and service delivery, minimizing the need for expensive computing resources on remote servers, meaning that the data is geographically far from where it is created. There are now just a few simulators that can replicate a fog environment. Several previous studies used iFogSim to investigate the possibilities of fog in [13], a Java-based programme that is an extension of CloudSim. [14] Improved iFogSim to allow mobility by transferring virtual machines between cloud networks. MyiFogSim, in conjunction with their proposed migration policy, may be used to investigate the policy's impact on application quality of service. The most common CloudSim-based simulator for fog and IoT situations is iFogSim [15]. It is open source, scalable and platform agnostic. It is capable of modelling and simulating infrastructures employing millions of heterogeneous fog nodes, IoT services, sensors and actuators. Defining the Fog infrastructure, mapping and scheduling policies for IoT services and enabling the evaluation of various metrics such as latency, network, energy consumption and operating expenditures [16].

It was used in various studies to model and analyze service allocation options in order to reduce energy consumption, data traffic and meet the application's Quality of Service requirements.

From the preceding discussion, it is obvious that the majority of previous works do not include fog computing for successful container service integration. To boost scalability, we suggest a novel architecture and technique for container-to-fog service integration based on our findings. This paradigm improves the performance of docker systems. As a consequence, the iFogSim simulator was used to simulator fog conditions in order to evaluate the impact of latency, network interference and energy consumption on the container while using fog management methodologies proposed in this paper.

3. Multi-service Containerization

This section focuses on how to oversee and control the container's service while many services are operating on the same computer. Finally, evaluate the container resource use performance.

Docker is an application that manages containers. The key features are the ability to build, ship and operate any environment. For example, it should be able to run a desktop environment, an Android environment, an iOS environment, or a cloud environment. Docker has the capacity to minimize development size by offering a reduced operating system footprint via containers. It is a contemporary technology that automates application deployment within Linux containers. It provides a virtualization layer for operating-system level abstraction and automation. Docker is a revolutionary technology that provides a high-level tool built on top of the Linux container API of Linux Container [8] and adds new functionalities. The major docker components are:

- Docker Engine: it is used to build Docker containers and Docker mages.
- Docker Hub: this is the repository that houses numerous Docker images.
- Docker Compose: this is used to create programmes that employ many docker containers. Because of their low weight, docker containers are incredibly scalable.

Using docker-compose, many programs may be run inside a single container. It is a platform for creating and managing multi-docker software. Each container in this case runs the program independently but can communicate if desired. Docker compose files that are simple to create in YAML, which stands for Yet another Markup Language which is an XML-based language. Docker composes ensures that all resources (containers) can be accessed by users with a single command. There are three steps to using docker compose i) define the application environment with a Dockerfile ii) define the services by using docker compose YAML file iii) use the command prompt to perform the docker compose up command. Figure 1 depicts the docker compose building components.

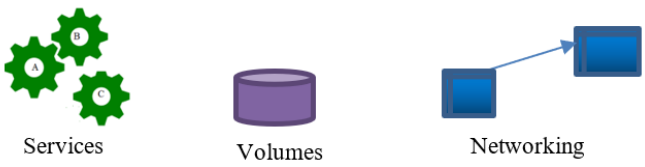

Fig.1. The building blocks of docker compose

Although it can use one container network to access distinct services housed in different containers, it can obtain the same benefits by running numerous services in the same container. These services must be accessed through the container host or network, such as the Apache / Engines HTTP Server with FTP Server or several microservices operating on the same container with different rules. The file structure of the multiple services on the container is shown in Figure 2.

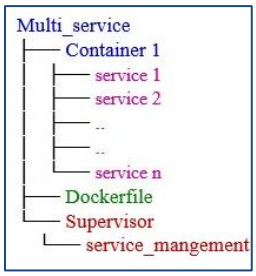

Fig.2. Multiple process in one container

Figure 3 depicts a docker compose.yml file. To run two or more lightweight services inside the same container by using the "Supervisor".

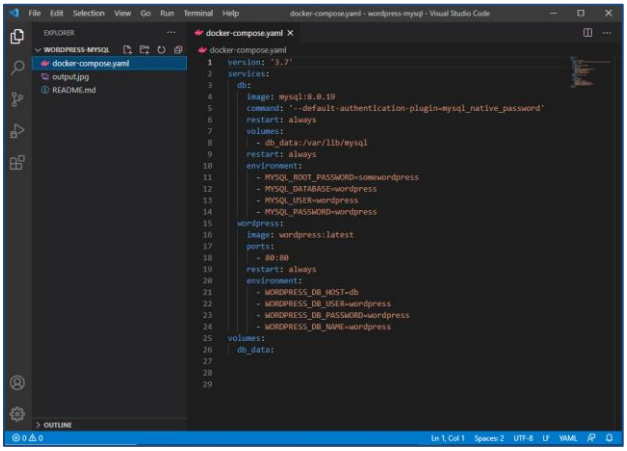

Fig.3. Docker compose.yml file

For example, the same container runs MySQL and word press simultaneously as shown in Figure 4. It extracts the image files from the docker hub registry.

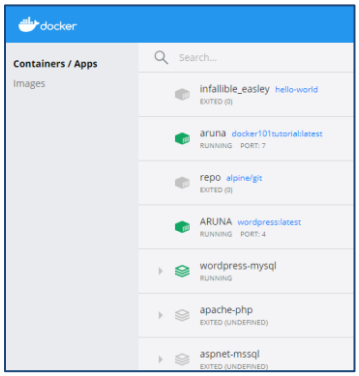

Fig.4. Docker Container with multiple services

As seen in Figure 5, It displays the container's resource use. Container monitoring parameters include CPU, memory, block I/O and network I/O. All services were managed via the supervisor process.

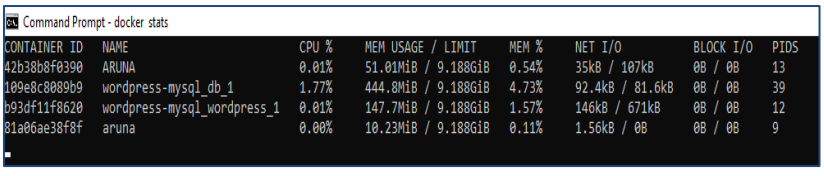

Fig.5. Container resource utilization

Supervisor allows us to better track the processes: The code is extremely straightforward and easy; to install the supervisor and establish a directory for child images to store configuration, use the instructions listed below.

```
RUN apt-get -y install supervisor && \
mkdir -p /var/log/supervisor && \
mkdir -p /etc/supervisor/conf.d.

# supervisor base configuration
ADD supervisor.conf /etc/supervisor.conf

# default command
CMD ["supervisord", "-c", "/etc/supervisor.conf"]
```

By default, all pictures expand from a base image that simply contains supervisor. This basic image includes a configuration file /etc./supervisor.conf, which causes the supervisor to operate as a foreground process, keeping the containers and operating mode active. It contains all configuration files in the directory /etc./supervisor/conf.d/, as well as the start of various programmes. All of the children's containers link to the supervisor setup by adding their special.sv.conf service to the appropriate directory. The container then begins, and all operations begin. The monitor is in charge of monitoring and maintaining the installed processes.

4. Proposed Model for Container-to-Fog-services Integration

Integration of container and fog computing gives up a slew of new opportunities. Fog computing nodes conduct complicated tasks on behalf of IoT devices, and the topological proximity of fog computing to IoT provides for a number of advantages (e.g., low latency). However, because certain IoT devices are mobile, the fog benefits may be negated. When a device moves, the communication path to the appropriate fog service may become longer, impacting fog advantages (due to fog proximity) and overall performance. To overcome this issue, the fog service might be transferred to the fog computing environments and kept close to the clusters and IoT device(s) that are being served. The Figure 6 illustrates the integration of a container with fog to increase service scalability.

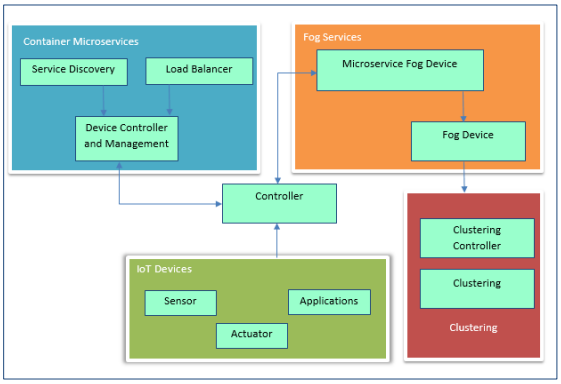

Fig.6. Proposed architecture of container-to-Fog-services integration

Fog services can be built as containers to increase overall performance and speed. Containers to increase fog node resource consumption and decrease service delays.

4.1. Expand Scalability with Fog

This section focuses on the scalability of containerized applications by using fog. In addition, the scaling process of the proposed fog architecture is also discussed.

Scalability is defined as a computer device, network, or program's capacity to handle growing quantities of work in terms of computational power and storage resources [17]. Scalability refers to the machine's ability to enhance efficiency when extra resources are introduced and scaled in two directions, namely upward (vertically) or outward (horizontally) [18]. Scaling up adds services to a single system node by placing servers, processors, or storage on a single device whereas scaling down adds additional system nodes, such as adding a new computer to a distributed computing programme. Table 1 defining factor that each method has its own set of advantages when scaling horizontally vs. vertically.

Thousands of Fog nodes can be employed in the Fog infrastructure to deploy thousands of IoT resources and service millions of people. Before implementing such complicated systems, solutions should be examined and assessed using modelling software to reduce costs and time assessment on the real systems. The Fog is utilized not only to load the crowded cloud, but also for low-latency applications. FoT is an IoT and Fog computation, data collecting or data transfer paradigm from tiny server computers. The FoT server handles user requests inside local servers. The FoT registration, FoT equipment and the FoT service access gateway are all included. The container where the request can be performed may support these. Figure 7 shows the machine architecture. The fog paradigm connects to the FoT container at the local level.

Table 1. Horizontal Vs. vertical scaling

Vertical scaling	Horizontal scaling
It requires a hardware upgrade for the server.	It involves connecting additional processing units/physical devices to the server. It has traditionally been used for high-level computation as well as application.
Increased I/O operations, CPU/RAM power, and disc capacity are all considerations to consider while scaling vertically.	Increased no. of nodes in the cluster and reducing the responsibilities.
Example: Virtual machine Hypervisor	Example: Google (gmail, facebook etc.,)

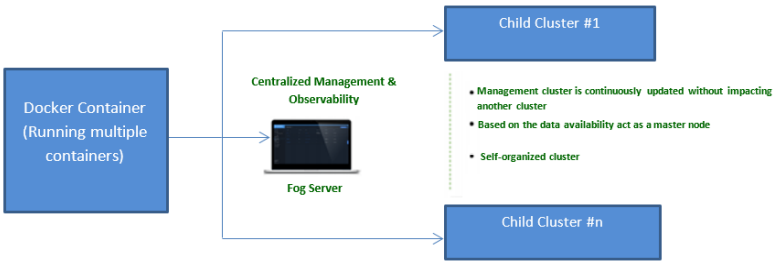

Fig.7. Fog server system architecture

The fog server interfaces to the container app. The fog server used to manage user requests and the child's cluster centrally and to observe them. The fog server monitors and updates the orchestration continuously. The maximum threshold of resource utilization is 100%. The output thresholds are in this case set at 85% of CPU and RAM usage. The scaling server can be scaled by one, if the CPU and RAM fog nodes use 85 percent. Working as the primary node that organizes the community group based on resource availability. The FoT server must determine the need for nearby containers as well as the concentration of application data. Fog-resistant systems are compatible with the gateway, which necessitates the activation and compilation of docker-based packages.

4.2. Development of a Dynamic Network

The dynamic network framework is designed for usage in cloud environments. In other words, dynamic network data processing and service distribution take place locally near the location of IoT data collection in order to address the need for heavy computing resources in distant servers, i.e. geographically distant from data generation, with the goal of overcoming existing technological limitations. Fog computing implementation has led in the creation of hierarchically organized multi-layer fog computing models. These structures typically define the organizational setup for all computer nodes that are involved. However, due to network features, computational node structure may increase efficiency, i.e. latency or bandwidth, between nodes by integrating specialized connections that do not follow the hierarchical method. The autonomous fog node is offered as an alternative to the hierarchical structure. These nodes are structured in a flat paradigm that takes advantage of network performance. This strategy uses less bandwidth by utilizing basic exchanges rather than direct messaging. The cloud extends the fog of things to the network's edge. Synchronization of these resources necessitates attention to a variety of communication modalities [19]. As Figure 8 shows, A Fog network of self-organizing computer nodes forms three self-network groups.

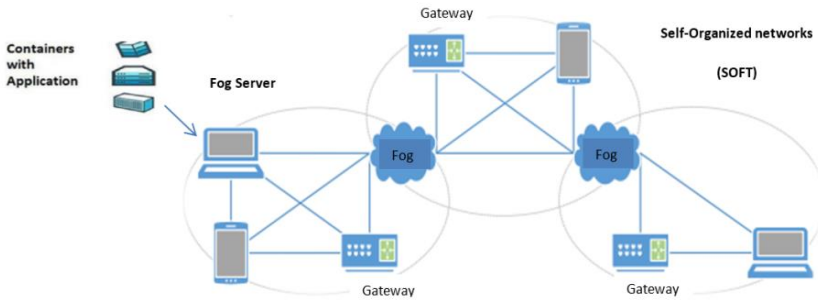

Fig.8. Generic fog deployment scenario

The network of all involved computer nodes is referred to as a fog network. Fog devices that connect sensors to the network are generally called gateways. The Self-Organizing Fog of Things (SOFT) is a computer node that uses logic in order to connect new computer nodes to the fog network and share information. To allow multiple fog nodes which act as collective applications, self-network groups can be executed. Two self-organized networks of the same community are known as neighbors. Self-organized networks may belong to the same or different groups. There are small numbers of nodes in each group. Network of the same group shares each other’s details through group graphs.

Deployment of the application at the FoT gateway is provided by the container using the SOFT technique. Inputs can be extracted from actuators and sensors, which must be transferred to the host via FoT devices [20]. Focusing on protocols such as MQTT (Message Queue Telemetry Transport), SOFT supports input authentication modules for application development. As a result, in order to execute, this sort of application needs evaluate and interpret data from the surrounding (or remote) region. In the absence of a high computing resource, the distant server requires SOFT to handle data generation and local service failures through the fog. The container's primary job is to create and implement applications for the essential services. Self-organized content is a type of IoT service that includes profile tracking, deployment, recovery, and administration. Local fog servers are linked to containerized terminal devices. The fog server responds to end-device requests and distributes a resource [21-23]. Using a self-organized fog of things to externalize scalability in order to reorganize container orchestration.

5. Implementation

The scalability of fog is examined in this part using the iFogSim simulator, which provides for the flexibility of setting the fog device specs. Because the fog network infrastructure is low-cost, simulation-based studies are the most commonly employed ways for verifying the efficacy of proposed mechanisms.

iFogSim is a free simulation and tested fog workload scenario for the deployment in testbed without any costs and complexities. It is an open-source, multiplatform and very scalable. This is a high-performance toolset for fog computing, edge computing and IoT. It allows for the modelling and simulation of a fog computing environment in order to evaluate resource management latency and delay. An iFogSim [24] which is also based on CloudSim is one of the fog simulators most commonly used. In order to simulate actual structures and adopt the sensing, processing and actuation model, the iFogSim can be used for distinguishing the components from these three groups. The primary structural characteristics are i) fog devices having the ability to set CPU, RAM, MIPS, uplink and downlink bandwidth and idle power levels (including cloud resources, fog resources, smart devices) (ii) geographic position and gateway communication actuators (iii) sensors that create data in the form of information representing tuples estimate the time required for pause and inventory management. It incorporates more adaptable ways of resource management based on the topic of research. It works with CloudSim, which is a popular tool for cloud modelling and resource management. Figure 9 shows how the importing is done in iFogSim GUI topology.

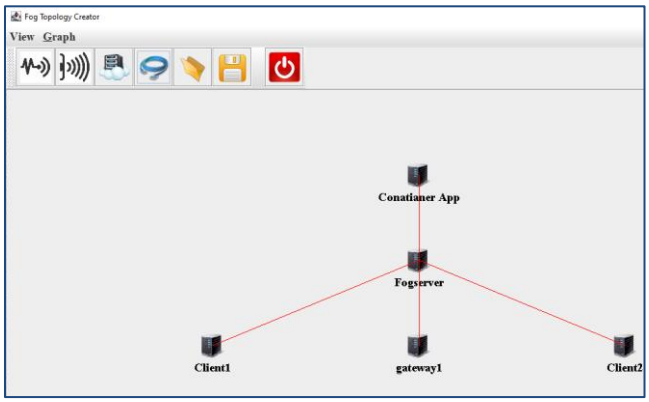

Fig.9. iFogSim GUI for building network topology

```
sys:1261:651
==================
=============
COORD MAP{}
sys:1261:651
==================
================
=============
COORD MAP{}
Adding edge between Fogserver & Client1
Adding edge between Fogserver & gateway1
Adding edge between Fogserver & Client2
Adding edge between Conatianer App & Fogserver
############################
{FogDevice [mips=2000 ram=3000 upBw=1000 downBw=1100]=[], FogDevice [mips=1000 ram=3000 upBw=800 downBw=1000]=[], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1200]=[], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1000]=[Edge [dest=FogDevice [mips=2000 ram=3000 upBw=1000 downBw=1100]], Edge [dest=FogDevice [mips=1000 ram=3000 upBw=800 downBw=1000]], Edge [dest=FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1200]]], FogDevice [mips=500 ram=45000 upBw=1000 downBw=1200]=[Edge [dest=FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1000]]]}
############################
```

```
sys:1261:651
==========
==========
==========
COORD MAP{FogDevice [mips=2000 ram=3000 upBw=1000 downBw=1100]=Coordinates [abscissa=315,
ordinate=390], FogDevice [mips=1000 ram=3000 upBw=800 downBw=1000]=Coordinates [abscissa=630,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1200]=Coordinates [abscissa=945,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1000]=Coordinates [abscissa=630,
ordinate=260], FogDevice [mips=500 ram=45000 upBw=1000 downBw=1200]=Coordinates [abscissa=630,
ordinate=130]}
```

```
Start Node: Client1
Start Node: gateway1
Start Node: Client2
Start Node: Fogserver
Target Node: Client1
Target Node: gateway1
Target Node: Client2
Start Node: Conatianer App
Target Node: Fogserver
sys:1261:651
==========
==========
==========
COORD MAP{FogDevice [mips=2000 ram=3000 upBw=1000 downBw=1100]=Coordinates [abscissa=315,
ordinate=390], FogDevice [mips=1000 ram=3000 upBw=800 downBw=1000]=Coordinates [abscissa=630,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1200]=Coordinates [abscissa=945,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1000]=Coordinates [abscissa=630,
ordinate=260], FogDevice [mips=500 ram=45000 upBw=1000 downBw=1200]=Coordinates [abscissa=630,
ordinate=130]}
Start Node: Client1
Start Node: gateway1
Start Node: Client2
Start Node: Fogserver
Target Node: Client1
Target Node: gateway1
Target Node: Client2
Start Node: Conatianer App
Target Node : Fogserver
Sys: 1261:651
==========
==========
COORD MAP{FogDevice [mips=2000 ram=3000 upBw=1000 downBw=1100]=Coordinates [abscissa=315,
ordinate=390], FogDevice [mips=1000 ram=3000 upBw=800 downBw=1000]=Coordinates [abscissa=630,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1200]=Coordinates [abscissa=945,
ordinate=390], FogDevice [mips=1200 ram=3000 upBw=1000 downBw=1000]=Coordinates [abscissa=630,
ordinate=260], FogDevice [mips=500 ram=45000 upBw=1000 downBw=1200]=Coordinates [abscissa=630,
ordinate=130]}
Start Node: Client1
Start Node: gateway1
Start Node: Client2
Start Node: Fogserver
Target Node: Client1
Target Node: gateway1
Target Node: Client2
Start Node: Conatianer App
Target Node: Fogserver
sys: 1261:651
```

To analyze the current resource utilization of each device using the iFogSim emulator. The *executeTuple()* command is used in the fog system to process logic when the device alters its resource utilization. These statistics can also be bundled as a tuple and sent to the resource management layer to be used to build use-aware resource utilization strategies.

6. Proposed Distributed Intelligent Scheduling Algorithm

The proposed Distributed Intelligent Scheduling algorithm is used to gradually divide the scheduling problem into smaller independent sub problems of increasing granularity, which can be executed in parallel. While the search of a feasible plan may compromise global completeness and optimality. The distributed scheduling technique significantly reduces the complexity of a large-scale multiple-resource scheduling problem.

Distributed Intelligent Scheduling based Lightweight Container (DIS-LC) algorithm:

The purpose of fog nodes is to minimize two cost functions namely operating cost and traffic cost. The operating cost of fog nodes is made up of two metrics: CPU and memory. The CPU cost (in MIPS) measures the amount of CPU consumed by a certain fog node when doing IoT operations, as well as the idle CPU consumption cost when the node is not in use. The memory cost (RAM) measures the amount of memory needed by the fog node to support container system

services, as well as the cost of idle memory while the node is not in use.

Formally, let define the operating cost $oper_{cost}(f)$ $of f \in F_n$.

$$oper_{cost}(f) = CPU_{cf} + RAM_{cf} \tag{1}$$

Where f refers to the single fog node and F_n refers to the set of active fog nodes. The $oper_{cost}(f)$ is the operational cost of a certain fog node, CPU_{cf} is the CPU cost on a certain fog node f and RAM_{cf} is the RAM cost on a certain fog node f.

In a container context, the devices must send the collected data to the fog nodes at certain bandwidth rates. Depending on the underlying demand, the active physical connections might have varying availability levels at various times. Furthermore, fog nodes must connect with one another from time to time in order to share relevant information. As a result, the traffic cost of the fog nodes is estimated as the bandwidth cost associated with each link plus the delay between the existing hops.

Formally, let the traffic cost $Traf_{cost}(f)$ of a fog node $f \in F_n$ communicating with an container system $c \in C_n$ as follows,.

$$Traf_{cost}(f) = \sum_{c, f \in C_n, F_n} BW_{cf} .opt_{cf} + \sum_{f, f' \in F_n} BW_{ff'} + NetworkLatency_{cf} \tag{2}$$

Where $Traf_{cost}(f)$ is the traffic cost on certain fog node, $Opt_{cost}(f)$ is the operational cost on a certain fog node, BW_{cf} is the bandwidth capacity on the link of fog node f, $NetworkLatency_{cf}$ is the network latency between fog node f *and* c refers to container, C_n is multiple container system, f is the fog node and F_n is the set of fog nodes.

Thus each fog node $f \in F_n$ has to minimize the following objective function

$$Overall_{cost} = Opr_{cost}(f) + Traf_{cost}(f) \tag{3}$$

Where $Overall_{cost}$ is the overall cost on certain fog nodes.

To accomplish the overall performance of the container, the following limitations must be considered

- The total amount of resources allocated to any fog node should not exceed the node's available resource capacity.

$$\sum_{c \in C_n} = A_c^r Opt_{cf} <= A_f^r, \forall_{f \in F_n,} \forall_{r \in r_n,} \forall_{c \in C_n} \tag{4}$$

Where A_c^r is the amount of resource of type r on a container system and A_f^r is the amount of resource of type r on a fog node.

- The entire amount of traffic arriving at any fog node should be less than the amount of traffic that the fog node can handle.

$$Traf_{cost}(f) < \varphi_{f,} \forall_{f \in F_n} \tag{5}$$

Where φ_f the maximum amount of traffic is that fog node f can afford.

The Distributed Intelligent Scheduling based Lightweight Container (DIS-LC) algorithm is executed by each fog node accepts as input a queue recording the container service to the fog node executing the resources to serve the tasks generated by the container system that issued the message and that the container device is in the fog node's preference list as shown in Figure 10.

Algorithm 1: Distributed Intelligent Scheduling based Lightweight Container (DIS-LC)

Input : Set C of container system and set *F* of fog nodes.
Output : Mapping of container system to a given fog node *f*.
Repeat
  While fog edge device (fog $_{edge}$) is not empty **do**
    If $A_f^r > A_c^r$ and c ∈ fog $_{edge}$ **then**
      Send accept reply of *f*
      Adjust the resource of f
    else
      Send reject reply to *c* (*container service)*
      Reject all *container service*
      Remove service from the container
    end if
  end while

Fig.10. DIS-LC algorithm

If all requirements are met by the container, the fog node sends an accept message to the underlying container system indicating to perform its functions. The fog node updates its available resources by deducting the resources required to serve the container service. If the number of resources available on the fog node is insufficient, a reject message is delivered to the fog device indicating that the fog cannot service its responsibilities at this time and the fog rejects any container systems whose rank in the preference list is lower than that of the rejected.

7. Result and Discussion

It is evaluated and the research objectives have been achieved with ability of container services using the parameters such as response time (ms), resource consumption (CPU, memory, I/O) and node load. The proposed algorithm DIS-LC is compared with three existing series namely least connection, Round Robin and ACO-LWC. The result shows that the proposed approach has utilizes the CPU at higher usage rate than the existing series as indicated in Figure11. As a result, this strategy is simple to apply in multiple container services for maximum efficiency.

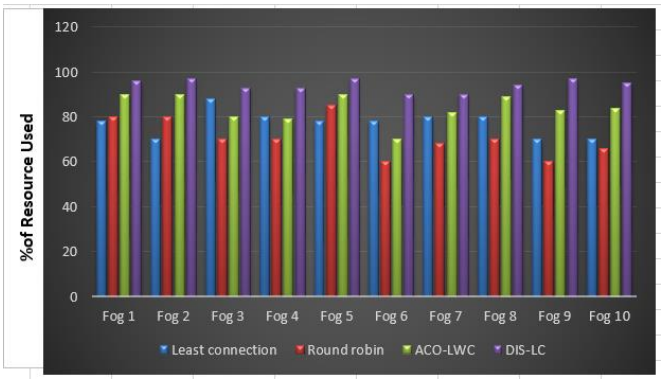

Fig.11. CPU Utilization

The CPU usage (measured in milli seconds) of all four methods is shown in Table 2. The CPU consumption for least connection, round robin, ACO-LWC method and DIS-LW is 0.43ms, 0.27ms, 0.10ms and 0.5ms respectively. This demonstrates that the proposed DIS-LC approach consumes relatively little CPU time and improving CPU efficiency.

Table 2. Comparison of CPU usage

Algorithm	CPU usage (ms)
Least connection	0.43
Round robin	0.27
ACO-LWC	0.10
DIS-LC	0.5

Figure12 shows memory usage in a fog node system with ten fog nodes. The first conclusion drawn from these graphs states that an increase in the number of fog nodes for multi-container jobs results in lower fog node resources.

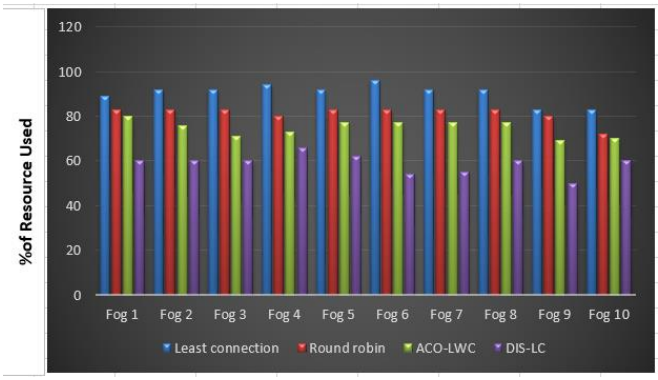

Fig.12. Memory utilization

It is expected that having more fog nodes reduces the system load. The second conclude remarks of the proposed work proves that 100% resource usage on all fog nodes when fog nodes are increased dynamically. The suggested technique allows these fog nodes achieves higher resource usage. The usage of memory for the least connection, round

robin, ACO-LWC and DIS-LC algorithms are calculated 2.13 GB, 1.53 GB, 144.3 MB and 108.5 MB respectively. As a result, the proposed DIS-LC algorithm consumes relatively less memory and improves the system's RAM speed.

8. Conclusion and Future Work

In this research, it is suggested a new DIS-LC algorithm for efficient resource utilization of container-to-fog-services integration dynamically regulates the system and include a self-network controller. The performance of the container services and its resource management are measured using iFogSim simulator tool. This fog architecture increases the scalability in a flat mode and leverages the properties of the network and its efficiency. In addition, Self-organization enables interoperability of local ecosystems in the fog. It is able to dynamically scale in/out the one M2M middle node instances according to the different traffic. The scalability and performance of container with fog is improved by using the DIS-LC algorithm. As a result, the suggested DIS-LC algorithm uses fewer resources than other algorithms. In fog conditions, it also increases the system performance of multiple container service. In future work, it is suggested to enhance the performance by scaling the server in/out across at various fog hierarchy stages.

Acknowledgment

The authors thank the institution and stakeholders for their support and the reviewers for their insights into the paper, because these comments have led to improved work.

References

[1] A. Pires, J.Simao, and L.Veiga, “Distributed and Decentralized Orchestration of Containers on Edge Clouds”, Journal of Grid Computing”, Vol.19, No.3,2021,doi:10.1007/s10723-021-09575-x.

[2] C.L.Tseng and J.LFuchun, “Extending Scalability of IoT/M2M Platforms with Fog Computing”, in *Proc. of the IEEE 4th World Forum on Internet of Things (WF-IoT)*, Singapore, pp.825-830, 2018.

[3] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang *et al*., “Virtual machine placement and workload assignment for mobile edge computing,” in *Proc. of the IEEE 6th International Conference on Cloud Networking (CloudNet*), Prague, pp. 1–6, 2017.

[4] A. Ahmed and G. Pierre, “Docker Container Deployment in Fog Computing Infrastructures,” *IEEE International Conference on Edge Computing (EDGE)*, USA, pp. 1-8, 2018, doi: 10.1109/EDGE.2018.00008.

[5] D.V.Leon, L.Miori, J.Sanin, N.E.Ioini, S.Helmer *et al*., “A Lightweight Container Middleware for Edge Cloud Architectures,” *Fog Edge Computing: Principles and Paradigms*, pp. 145–170, 2019.

[6] E. Yigitoglu, M. Mohamed, L. Liu and H. Ludwig, “Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing,” in *Proc. of the IEEE 6th International Conference on AI &Mobile Services (AIMS)*, USA, pp. 38–45, 2017.

[7] M.Taneja and A.Davy, “Resource Aware Placement of IoT Application Modules in Fog-cloud Computing Paradigm,” in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Portugal, pp. 1222–1228, 2017.

[8] OpenFog Consortium, “OpenFog Reference Architecture for Fog Computing,” 2017.

[9] P.Hu, S.Dhelim, H.Ning and T.Qiu, “Survey on Fog Computing: Architecture, key Technologies, Applications and Open Issues”, *Journal of Network and Computer Applications*, Vol. 98, pp.27-42, 2017.

[10] Maryam Sheikh Sofla, Mostafa Haghi Kashani, Ebrahim Mahdipour and Reza Faghih Mirzaee, “Towards effective offloading mechanisms in fog computing”, *Multimedia Tools and Applications*, Vol.81, pp.1997–2042, 2022.

[11] B.David and J.L.Fuchun, “OpenStack-based Highly Scalable IoT/M2M Platforms”, in *Proc. of the IEEE International Conference on Internet of Things (iThings)*, United Kingdom, pp.711-718, 2017.

[12] H.Gupta, V.D.Amir, K.Soumya and R. Buyya, “iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments”, *Software: Practice and Experience*, Vol.7, no.9, pp.1-22, 2017.

[13] M.M.Lopes, M.A.M.Capretz and L.F. Bittencourt, “MyiFogSim: A Simulator for Virtual Machine Migration in Fog Computing”, in *Proc. of the 10th International Conference on Utility and Cloud Computing*, USA, pp. 47–52, 2017.

[14] M. Redowan Mahmud and R. Buyya, “Modelling and Simulation of Fog and Edge Computing Environments using iFogSim Toolkit”, *Fog and Edge Computing*, 2018, DOI:10.1002/9781119525080.CH17.

[15] R. Mahmud, S. Pallewatta, M.Goudarzi and R.Buyya, “iFogSim2: An Extended iFogSim Simulator for Mobility, Clustering and Microservice Management”, *Edge and Fog Computing Environments*, 2021.

[16] https://linuxcontainers.org/lxc/getting-started, 6.04.2022.

[17] Amina Mseddi, Wael Jaafar, Halima Elbiaze and Wessam Ajib, “Joint Container Placement and Task Provisioning in Dynamic Fog Computing”, *IEEE Internet of Things Journal*, 2019, doi 10.1109/JIOT.2019.2935056.

[18] V.Hurbungs, V.Bassoo and T.P. Fowdur, “Fog and edge computing: concepts, tools and focus areas”, *International Journal of Information Technology*, Vol. 13, No.2, pp.511–522, 2021, doi: 10.1007/s41870-020-00588-5.

[19] Ahmed M. Alwakeel, “An Overview of Fog Computing and Edge Computing Security and Privacy Issues”, *Sensors*, Vol. 21, No. 24, pp. 8226, 2021, doi:10.3390/s21248226.

[20] Abdelaali Chaoub, Aarne Mammela, Pedro Martinez-Julia and Ranganai Chaparadza, “Self-Organizing Networks in the 6G Era:State-of-the-Art, Opportunities, Challenges and Future Trends”, 2021,doi.org/10.48550/arXiv.2112.09769.

[21] Zainab Javed, Waqas Mahmood, "A Survey Based Study on Fog Computing Awareness", International Journal of Information Technology and Computer Science, Vol.13, No.2, pp.49-62, 2021.

[22] Mohamed A. Elsharkawey, Hosam E. Refaat, " MLRTS: Multi-Level Real-Time Scheduling Algorithm for Load Balancing in

Fog Computing Environment", International Journal of Modern Education and Computer Science, Vol.10, No.2, pp. 1-15, 2018.
[23] Vishal Kumar, Asif Ali Laghari, Shahid Karim, Muhammad Shakir, Ali Anwar Brohi,"Comparison of Fog Computing & Cloud Computing", International Journal of Mathematical Sciences and Computing, Vol.5, No.1, pp.31-41, 2019.
[24] H. Gupta, A. V. Dastjerdi, S. K. Ghosh and R. Buyya, "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments," *Software: Practice and Experience*, Vol. 47, no. 9, pp. 1275–1296, 2017.

Authors' Profiles

Aruna. K. received B.Tech. degree in Information Technology from Anna University, Chennai, in 2005. Received M.E degree in CSE from Priyar Maniammai University, Vallam, in 2011. She is pursuing her Ph.D. in the field of Networking at Anna University, Chennai. She is currently working as an Assistant Professor at A.V.C. College of Engineering, Mayiladuthurai.

Dr. Pradeep. G. received his Ph.D. degree from Anna University Chennai in 2016. He is currently working as a professor at A.V.C. College of Engineering, Mayiladuthurai. His research interest includes Web Service, Information Retrieval and Service Oriented Architecture.

How to cite this paper: Aruna. K., Pradeep. G., "Container-to-fog Service Integration using the DIS-LC Algorithm", International Journal of Computer Network and Information Security(IJCNIS), Vol.16, No.1, pp.85-96, 2024. DOI:10.5815/ijcnis.2024.01.07