ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 1, 408-417 (2020)

www.astesj.com

Special Issue on Multidisciplinary Sciences and Engineering

ASTESJ

ISSN: 2415-6698

Resource Selection Service Based on Neural Network in Fog Environment

Nour Mostafa*

College of Engineering and Technology, American University of the Middle East (AUM), Egaila, Kuwait

ARTICLE INFO

Article history:
Received: 14 January, 2020
Accepted: 18 February, 2020
Online: 25 February, 2020

Keywords:
Cloud
Fog
Fog-to-Cloud
Fog-to-Fog
IoT
e2e delay

ABSTRACT

As an emergent technology in Internet of Things (IoT), the ultimate target of fog computing is to provide a widely distributed computational resources and data repository closer to the network edge providing heterogeneous systems both in terms of software and hardware. The fog system must have the capability to deal with huge number of resources and users at the same time, such increased size sometimes presents the issue of performance degradation. Therefore, the fog should be able to support adaptability, scalability, and extensibility to avoid such degradation by adopting efficiently and effectively an optimal resource selection and allocation model. As many fog users have limited interest in, or knowledge of fog middleware issues, it follows that in a large fog environment the best approach would be to have an automated system for resource selection and allocation. Such an approach eliminates the need for user intervention. This paper proposes a fog resource selection service based on neural network to perform resource selection tasks by coordinating with the metascheduler. Five different selection algorithm were used to evaluate the prediction model for resource selection. In addition to introducing a history update and management algorithm to manage and control the storage of the history log records.

1. Introduction

Fog and IoT systems are handling huge numbers of users, resources and tasks. The fog and IoT environment are widely distributed with great diversity, therefore, it is expected that resource selection will be carried out by run time estimates. With the increase size of the fog and IoT and also with its wider acceptance by the 'ordinary' users, some form of automatic resource selection system is desirable. This resource selection system should have the capability to evaluate available resources on the basis of specific criteria as determined by the user.

Because of its huge volume, the fog requires special services for its efficient utilization. With the expansion in the size of the fog and IoT, users are facing new challenges of resource selection to meet their computational requirements without getting into the fine details of available resources. Although, evolving very rapidly and becoming widely accepted, normal users of the fog and IoT find it very difficult to use middleware technologies. Middleware services provide a means to virtualize and aggregate resources. Fog resources are heterogeneous and very complex and require intelligent solutions for resource discovery and characterization. Therefore, in fog environment, the network and fog designers are facing a serious challenge to grant efficient utilization of such huge and widely distributed resources. Such increased size sometimes presents the issue of performance degradation. Therefore the fog system should be able to support adaptability, scalability, and extensibility to avoid such degradation. In order for the fog system to scale well with the increase size of resources and users, we propose a Fog Resources Selection Service (FResS) for fog systems based on neural network that will meet a user's preferences in an optimal response time. The proposed strategy presents a predictive resources selection model to predict and determine the optimal resource to fulfill the user's task. The work introduced in [1] has been extended in this paper where the IoT framework is integrated within the FResS, in addition to testing and analyzing different selection algorithms. Moreover, a history update and management algorithm has been introduced to manage and control the storage of the history log records. This situation requires that the fog user should be provided with a helper service to facilitate resource selection. Resource selection can be done by taking into consideration either, cost, time, cost and time, performance, or any other constraint. This resource selection service can have its own resource selection algorithms and can also be deployed at varying distribution levels. Currently the majority

*Corresponding Author: Nour Moustafa, nour.moustafa@aum.edu.kw

of metaschedulers are deployed at the cloud level which will be a bottleneck for larger fog and IoT systems. A higher distribution level will alleviate this bottleneck. The future fog and IoT is also intended to be more user focused.

The advanced fog selection for task execution is enabled by the resource selector which uses the FResS to make a rum time prediction to select the best host/resource. Due to the resource-limited fogs, the proposed model avoid incurring overhead when executing the task. In addition to decreasing the overall end-to-end (e2e) latency of the system. The proposed FResS model select resources based on criteria mentioned by the user, which provide QoS levels that meet the users' expectations. Experimental performance evaluation shows an overall improvement in terms of bandwidth consumption and task execution times, in addition to maintaining adequate QoS levels.

The reminder of the paper is organized as follows: Section 2 presents background reading and related work. To provide a deeper understanding of this area, section 3 introduces the fog prediction and selection scheme. Section 4 presents the multi-level load balancing technique. Section 5 presents the evaluation results using workloads and simulations respectively. Finally, the paper concludes with section 6, focusing on the contribution along with future recommendation.

2. Related Work

The emergence of cloud, fog and IoT approaches as a comparatively new approaches to distributed computing, which has become popular in the last few years. These approaches addresses the problem of organizing large scale computer societies, called Virtual Machine, which are able to use and share large sets of distributed resources (e.g. computational resources and data repositories) among them. With the emergence of several cloud and fog service providers, it is expected that system performance and run time estimates will be used for resource selection, which make organization to move from one service provider to another to meet their needs. Predictions are a very important step towards automatic resource management [1][2]. As the capabilities of IoT devices are limited, more attention is required when allocating tasks on these devices to assign the workload optimally on available resources.

In [3] authors aims to enable parallel execution by splitting the allocated services on the available resources. To achieve low delay on service allocation the proposed model distributed/matched the desired services among the available Fog-to-Cloud (F2C) resources. The author in [4] proposed a resource allocation strategy for resource selection by adopting the Price Timed Petri Nets (PTPNs) which allow the user to select the resources autonomously based on the price cost and time cost to complete a task. In addition to sorting the resources into groups based on their credibility, then the users will be assigned to different groups because the credibility that they valuate the resources is different. The authors in [5] proposed an online algorithm for task allocation in mobile IoT networks by considering two scenarios to address the scenario of tasks information is unknown and incomplete local task information, respectively. The task allocation decision is done based on the current network status, energy consumption in order to satisfy the mobile device battery capacity constraint and to minimize the energy consumption of the current location.

In [6], the authors proposed an online task scheduling scheme called FairTS by adopting and combining the deep reinforcement learning (DRL) and dominant resource fairness (DRF) techniques. The proposed scheme views the agent's policy, take the current state using neural network to learn from experience, and outputs the action selection probability vector. The agent observe the resource availability, pick a task, and decide its fog resource allocation based on a resource availability that meet the task completion deadline to achieve resource fairness among tasks.

The author in [7] proposed offloading scheme, where the task data submitted by the user is uploaded to its nearby fog node, then the fog node will take a decision for offloading the task to its neighbor fog node or remote cloud to satisfy the delay deadline received from the end user. In [8] the authors used the machine learning as a prediction model to predict cloud processing resource consumption. The proposed model uses an artificial intelligence algorithm for future request prediction by dividing the incoming request into multiple consumption classes depending on their consumption amount, then the request is distributed among processing unites. For an incoming request, the adopted algorithm is used to predict the resource amount consumed to complete the task, a placement algorithm is used to allocate it to the best processing unit.

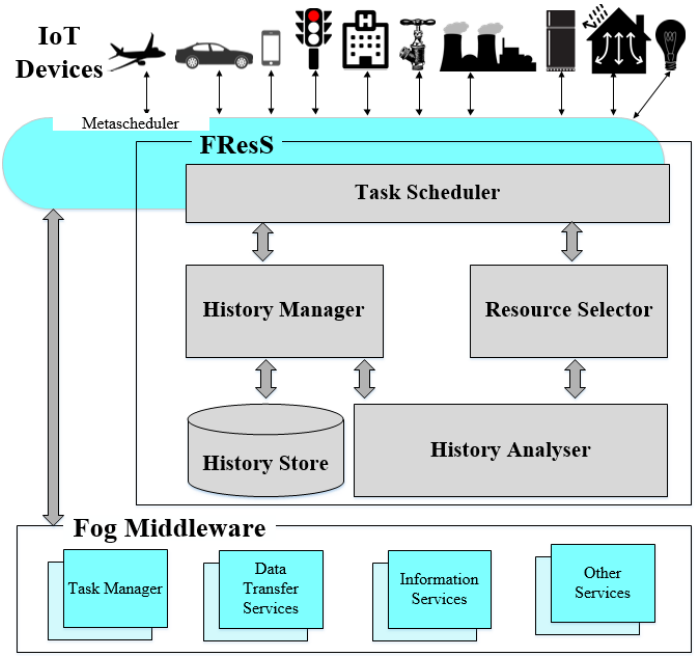
Although there exists research investigating the resources selection and allocation issues in fog computing, however it is still premature. As mentioned earlier, this project resulted in the development of two novel components: a run time prediction model and a distributed architecture of a Fog Resource Selection Service (FResS), which provides a degree of autonomy in determining the optimal resources to execute a particular task.

3. Fog Prediction and Selection Scheme

In our paper [1] a novel approach has been introduced in fog computing, namely, a Fog to Fog (F2F) resource selection algorithm called FResS, that provide automatic resource selection system. This resource selection system have the capability to evaluate available resource on the basis of specific criteria as determined by the user. The idea was to develop a new technique which require no user input. This work resulted in a run time prediction model based on historical execution logs and a fog resource selection service. The main modules of the FResS framework are: Task Scheduler, Resource Selector, and History Analyzer. These modules perform different resource management functions, the most obvious of which is resource prediction and selection. All of these modules are discussed separately in the coming sections.

A detailed architectural diagram of FResS is shown in Figure 1 highlighting the fact that FResS works along with the metascheduler. The FResS architecture was developed by keeping in mind its distribution level, which can be either IoT level, or fog level. The recommended distribution level is the IoT level, if deployed at the IoT level, FResS stores historical data related to the single IoT, it makes the management of execution logs straightforward because the overall volume of data will be very small. It will also provide fast and accurate predictions, again

because of smaller and related data. If deployed at the fog level it can become a single point of failure and a performance bottleneck. Implementation details will be very much similar for the different levels of distribution.



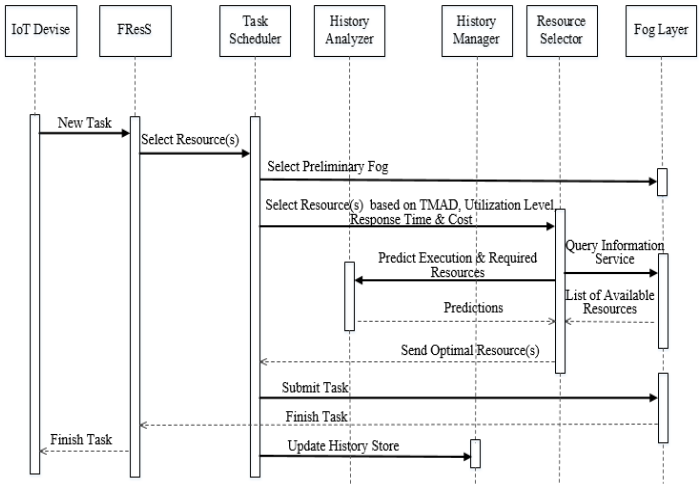
architecture

3.1. Task Scheduler Module

This is the main component interacting with the fog middleware through a metascheduler. Its implementation depends on the underlying fog middleware. Depending on its distribution level, whether it is IoT level or fog level, the Task Scheduler Module can have different implementation details. At a minimum it provides interfaces between the metascheduler and the FResS components, but it can also act as a metascheduler in itself. For evaluation purposes a simplified Task Scheduler Module was implemented by removing all external constraints.

The Task Scheduler sits between an IoT devise/user and the fog middleware. It provides interfaces to the IoT, the fog middleware, and other modules of the metascheduler. The sequence diagram shown in Figure 2 explains the sequence of events during a typical task submission process. The Task Scheduler Module and the metascheduler module coordinate with the lower layers of the fog to execute different tasks. Lower layers of the fog provide basic services i.e. data transfer, task submission, task management, and information services. Once a task is submitted by the user to the Task Scheduler, it will be connected to a preliminary fog based on users' location, and then the Task Scheduler Module forwards it to the Resource Selector Module.

The configuration plays a key role in selecting resources for each task and predicting its execution time. If the task selector returns more than one resource for a particular task, then an optimal resource will be selected by the Task Scheduler to fulfill users' request based on utilization level, which allow the users' preference to be considered while assigning the task to resource(s) which is more suitable for large scale domain such as fog computing. The progress of the tasks and resources will be constantly monitored by the proposed model, which take decisions based on that. The History Manager will receive the execution logs from the Task Scheduler after every execution, and subsequently predictions will be made for future tasks using these logs. The sequence diagram in Figure 2 shows the interaction among the different modules of FResS during a typical task submission process.



FResS model

3.2. Resource Selector Module

The Resource Selector Module analyses the workload of the incoming task to provide automatic resource selection by using the run time predictions. The Resource Selector depending on the configuration, along with their run time predictions, will forward a list of selected resources to the Task Scheduler. The Task Scheduler Module returns the resource list to the metascheduler to complete task submission. The resource selection on the fog is a multi-step process. The first step tests the eligibility of the discovered resources against the essential QoS criteria, e.g. privacy, memory, cost, availability, and delay. The list of the selected resources should be capable of executing the current task if there is no deadline constraint. Step one is performed by the metascheduler on its own.

In step two, the ability of resources to meet deadlines will be evaluated by evaluating run time predictions. For optimum task allocation, these predictions are then utilized by the metascheduler. To make optimum resource selections the metascheduler also considers additional constraints of user preferences, data locality, co-allocation, cost, workflow constraints, and advance reservation. These additional constraints are also evaluated by the metascheduler after receiving a ranked list of resources along with predictions from FResS. Predictions help the metascheduler to make a decision on the required duration of advance reservation. The co-allocation decision is facilitated with this predicted knowledge of expected run times.

The Resource Selector Module receives its input from the Task Scheduler Module along with the currently available resource list. It forwards incoming tasks to the History Analyzer Module to generate run time predictions for the available resources. The History Analyzer Module then uses algorithm 6, discussed in the next section, to generate run time predictions. For the evaluation

of FResS an experimental strategy was devised in which five algorithms 1, 2, 3, 4, 5 were used by the Resource Selector Module for resource selection. These algorithms are named as history, static, random, history plus static, and history plus random.

Data: workload history
Input: available resource list
Input: task submission description file
Result: selected resource
initialization;
getPredictions(new task submission description file, available resources);
selected resource is one with the minimum run time prediction;
return selected resource;

Algorithm 1: Selecting resource using history

Data: workload history
Input: available resource list
Input: task submission description file
Result: selected resource
initialization;
getPredictions(new task submission description file, available resources);
selected resource is one with the highest mips rating;
return selected resource;

Algorithm 2: Selecting resource using static performance capability

Data: workload history
Input: available resource list
Input: task submission description file
Result: selected resource
initialization;
getPredictions(new task submission description file, available resources);
select a resource randomly from the available resource list;
return selected resource;

Algorithm 3: Random resource selection

Data: workload history
Input: available resource list
Input: task submission description file
Result: selected resource
initialization;
getPredictions(new task submission description file, available resources);
select 10 resources with the least run time predictions;
from these resources pick resource with highest mips rating;
return selected resource;

Algorithm 4: Selecting resource using history and mips rating

Data: workload history
Input: available resource list
Input: task submission description file
Result: selected resource
initialization;
getPredictions(new task submission description file, available resources);
select 10 resources with the least run time predictions;
from these resources pick any resource randomly;
return selected resource;

Algorithm 5: Selecting resource using history and random selection

Selecting the fastest resource depending on its static number crunching capacity is a straightforward solution to the resource selection problem. It is approximately comparable to the Condor matchmaking technique [9]. It is mentioned above as static resource selection, which suffers from the problem that it does not have a feedback mechanism to inform the system about any anomaly. The history based resource selection on the other hand has a feedback mechanism to rectify any anomaly resulting from changed resource capability.

The random resource selection results in an even distribution of tasks among candidate resources without considering the difference in their capabilities. On heterogeneous fog system, it is not a desired outcome where high power resources are required to get more tasks. The random resource selection also does not provide any estimates of expected execution time, leaving the user in a 'wait and see' situation. The historical resource selection is combined with last two algorithms, 4 and 5 to overcome the drawback of static and random resource selection. Another shortcoming of the historical resource selection is the uneven load distribution similar to the static resource selection. However, the presence of a feedback loop shown in Figure 4 reduces the severity of this problem. The performance of all these algorithms will be evaluated in detail in the simulation result section. Once similar tasks are found, run time predictions are generated for all resources in the list. The finished list is returned to the Resource Selector Module.

3.3. History Analyzer Module

The incoming task along with the list of available resources is forwarded to the History Analyzer Module to generate run time predictions by examining through the history execution logs to find similar tasks using algorithm 6, this process is conducted using Artificial Neural Networks (ANNs) [10]. Once similar tasks are found, run time predictions are generated for all resources in the list. The complete list is then returned to the Resource Selection Module. ANNs are suitable for training over hundreds or even thousands of passes of data sets[11]. The increased size of the fog and IoT systems brings new challenges of scalability and extensibility and the need for automated processing becomes clear therefore, accurate results can be achieved. The design of ANNs generally comprises three layers, input, middle (or "hidden"), and output layers [10]. The input layer is the starting point where the data enters the system. The input data are passed to the hidden layer for processing which is the intermediate processing unit, and then passes the new signal onto the output layer. The learning phase plays a key role in neural network for processing information

using the interconnection weights. The strength of the input data is measured by weights, therefore, the process of finding the best set of weights for the neural network is referred to as training or learning. In order to get accurate results, the weights are modified at the input layer by passing input values, and then the network's predictions will be measured to check how the predictions' values are close to the training sets. In addition, the accuracy of the prediction results is compared to the actual system results as shown in the simulation result section 5, Figure 3 shows a single unit network.

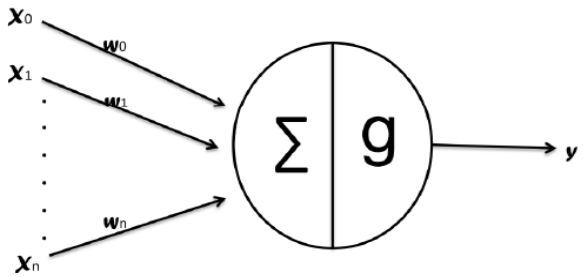


Figure 3: An Artificial Neuron Unit [12]

An amount proportional will be changed to the difference between the desired output and the actual output to achieve weight adjustment [13]. The general mathematic definition [14] is as shown in (1) describes the relationship between the input and output mapping:

$$y(x) = g(\sum_{i=0}^{n}(w_i x_i)) \quad (1)$$

the input of each neuron is represented by x with (n+1) input dendrites (x_0….. x_n) and the axon $y(x)$ computes the output. In order to supply a node activation the weighted signals are initialized with (w_0….. w_n). The activation function is the identity function and is represented by g which computes the weighted sum of the inputs from other neurons and outputs it. The aim is to be able to produce the output within desired accuracy matching the input pattern [15].

After the historical execution logs are examined by the History Analyzer to generate estimates for the required resources by finding similar task with sufficient accuracy, once predictions are generated, the complete list will be forwarded to the Resource Selector to determine the optimal resources to execute the task. Algorithm 6 forms the core of FResS, which is used to generate rum time prediction for new tasks on the available resources.

The process of identifying the resource id dependent on five parameters: delay, price, previous execution time, memory requested, and submission time. Workloads can be modelled at any level but it is found that modelling at a user level is more realistic, this is because users of fog networks tend to repeat same tasks using the same data [12] [16]. Therefore, the results are interesting since prediction of required resources of the new task will be made using past execution parameters.

The proposed algorithm combines the static and random algorithms with the historical resource selection to overcome their disadvantages, e.g. in certain situations there is a possibility of selecting the same resource repeatedly by the Task Scheduler, creating an uneven load distribution. Consequently, the presence of the feedback loop as shown in Figure 4 overcome the severity of this problem.

Data: user preferences and task log history
Input: available resource list
Input: new task submission description file
Result: run time prediction for the resource list
Start
While Taskset <> empty
do
  process request for Task(s) k_i;
  if (Ki submitted by new user/device) then;
    Resource Selector send available resources to Task Scheduler;
    Task Scheduler send user preferences to Task Manager;
    if (Task Manager found optimal resource(s));
      send back optimal resource(s) to Task Scheduler;
      execute k_i on the optimal resource;
  else if (Ki submitted by existing user/devise) then;
    NN predict required resources for k_i;
    Send back predicted resource(s) to execute k_i;
    execute k_i on the predicted resource;
  else
    execute k_i on the preliminary fog;
  update Task Manager log history;
  select next task;
  end
end

Algorithm 6: Making predictions for run time

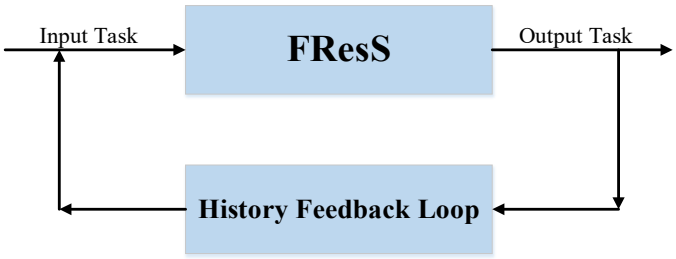


Figure 4: History as a feedback loop in the scheduling process

On the other hand the history management algorithm (algorithm 7) works separately and is responsible for keeping the history up to date. It means that it stores new history records after successful executions and removes older history records which is updated continuously after every execution and subsequently used by ANN to make predictions for future tasks. Subsequently, the increasing number of the historical executions records will increase the accuracy of predictions [17][18].

3.4. History Manager Module

The History Manager is interfaced with the Task Scheduler and history database, it is responsible for keeping the history data up to date. History updates are performed by using algorithm 7. The History Manager Module controls the storage and removal of

execution records as required. Algorithm 7 performs history management by storing completed tasks data in the database as soon as they are received. After this step, a query is sent to the database to retrieve the size of the node related to the new task, just stored. If the size of this node is less than, or equal to, the maximum history limit then no action is taken. Otherwise, if the size of this node is larger than the maximum, then the oldest member of this node is deleted. This module gathers different QoS data e.g. delay, cost, previous execution time and memory requested etc.

Data: workload history
Input: execution log of completed job
Input: task submission description file
Result: history updated
initialization;
store record to the history database;
if size of this cluster is more then the size limit then
  delete the oldest record;
end

Algorithm 7: Update history

3.5. Complexity of Algorithms

Algorithms 6 and 7, presented above, form the core of FResS. The predictions algorithm (algorithm 6) is used to generate run time predictions for new tasks on the available resources. On the other hand the history management algorithm (algorithm 7) works separately and is responsible for keeping the history up to date. It means that it stores new history record after successful executions and removes older history records.

The complexity of predictions for algorithm 6 is of $O(n)$ where n is the size of available resources. On the other hand, complexity for history management algorithm 7 is of the $O(1)$. These algorithms will become part of the task submission service. The history management algorithm 7 can be executed offline, after the task is finished, so it's speed is not very critical to the overall execution time because it is not contributing to the overall delay. Algorithm 6, on the other hand, will contribute to the total delay in the execution time.

3.6. Fog Workflows and FResS

Predictions are utilized by the task management system to select resources to execute incoming workflows consisting of many tasks. These workflows can be very simple, consisting of only one task, as shown in Figure 5, or can be very complex, consisting of many interdependent tasks, as shown in Figure 6. These tasks can have dependencies which restrict them to execute in a certain sequential order or execute concurrently. Predictions for each task within a workflow are generated separately.

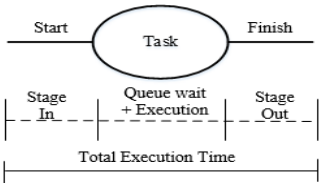
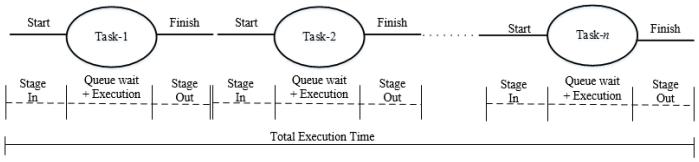


Figure 5: Workflow consisting of only single task



Quality of Service (QoS) refers to the level of performance and service that a given user will experience at a certain point in time, when starting a certain operation on a certain instance of a service. QoS support refers to the possibility of a certain level of required performance being available from a certain resource or not. QoS is very important to fog applications which are run collaboratively in real time. QoS requirements can be mentioned in terms of cost, delay, scalability, fault tolerance, etc.

Normally QoS requirements for these workflows are applied to the entire workflow. To achieve the best timing performance it is recommended that the workflow is partitioned in such a way that parallel activities finish at the same time. Because of the global nature of the fog, locality will also play its role to achieve best QoS results. Sometimes if the user wants to see intermediate results, then QoS can also be mentioned at the task level. Achieving QoS either at task level or workflow level boils down to the task or activity level. To achieve QoS at the workflow level requires that tasks are managed properly. Incorrect management of these tasks can damage QoS at a very high level.

The proposed FResS model is responsible for the overall execution of workflows. These executions, in certain situations, may also require advance reservation and co-allocation. The FResS performs its task comparatively easily when accurate run time predictions are available for tasks within a certain workload. Advance reservation is a technique in which resources are reserved for a certain task to start at a certain time and execute for certain duration. On the fog, advance reservation will be expensive so its accuracy is of prime importance. With the predictions already available, the FResS can reserve resources for accurate time duration, resulting in an efficient usage of resources.

4. A Multi-Level Load Balancing technique in Fog Computing

Internet traffic growing exponentially and congested with thousands and millions of devices [20], quality services should be provided with higher availability by the service provider. With the increase number of services in fog and IoT computing the network load is expected to increase which will create a new challenge to the researchers and network designer. The aim is to achieve better performance in terms of the total time to execute the tasks load balancing to accomplish better QoS, in addition to increasing the number of task acceptance. The fog computing attempts to integrate multiple distributed and heterogeneous resources, which are normally under separate administrative domains resulting in under-utilization or over-utilization. A multi-level load balancing approach has been proposed to provide better load balance. As mentioned above in algorithm 6 user preferences will be considered for the incoming task. Therefore, satisfying the below metrics and parameters optimally will led to achieve better performance of the fog system.

4.1. Metrics

- Cost: IoT and fog users are interested in the best performance at the lowest cost, but on the other hand the resource owners are interested in the overall system throughput. Therefore, the efficient algorithm should consider the cost parameters by scheduling the incoming task to a resource based on the cost.
- Response time: is the total execution time taken by the task to be completed, it starts from submitting the task by the user or device and end with receiving the completed task from the service provider. A smaller response time is always desirable.
- Delay: Researchers and service providers stress on the importance of guaranteeing acceptable delay, since this is an important metric in terms of QoS, the proposed model adopted the tenant maximum acceptable delay (TMAD) [21]. TMAD sorts the incoming tasks to a higher priority task or lower priority by considering different prioritization of users based on their important for the system according to the subscription plan each user pay for.
- Scalability: bottlenecks in fog computing system arises from the huge number of users, resources, and services, therefore to be able to support scalability, and extensibility, this is an important metric which determine if the system is able to achieve better load balancing with a restricted number of resources.
- Fault Tolerant: the larger the system, the more frequent failures result, IoT and fog systems should provide the ability to perform correctly in case of failure by having a backup

5. Simulation Results

The performance of the proposed solution is evaluated in this section by carrying out some experiments using CloudSim [21], selected for its flexibility and wider acceptance by the cloud and fog community. The proposed model performance was evaluated by comparing it with existing scheme and highlighting the differences and advancement. IoT devices were simulated as geographically distributed nodes. The fog network represented as a graph and storage sites were modeled as a set of nodes. A computing power, storage capacity, narrow and broad bandwidth, and memory have been configured to all nodes to be close to reality. The number of tasks, and capacity of storage nodes were varied to show the complexity of the proposed model and simulate different scenarios. The simulator defined 2 cloud sites, 5 fog sites, and 150 IoT devices, 5000 task requests, and 500 to 2000 mb/sec the connectivity of the bandwidth.

Scheduling with predictions helps the scheduler to select the best resource(s) for a certain task and it can be compared with two other techniques i.e. random and static. In a random selection the scheduler selects any resource from the qualifying resources. Qualifying resources are those which meet the minimum requirement criteria for a certain task. On the other hand static selection is one in which the resource with the highest number crunching speed is picked. In a real world scenario, number crunching speed can be based on any popular benchmark. Experimentation was conducted with CloudSim, which defines resource capability in term of Million Instructions Per Second (MIPS), hence in the static resource selection, resource with the highest MIPS rating will be selected from the qualifying resources.

In a deadline based scheduling scheme the function of the scheduler is to meet a deadline specified by the user. It is quite possible that more than one resource can meet this deadline, established by comparing the deadline with the predicted time. In such cases the scheduler can select the fastest resource, which is referred to as history based selection. It is expected that history based resource selection will create an uneven load distribution on the fog system. There are two other options: (1) select the fastest resource from the selected fastest resources, (2) select any resource randomly from the qualifying list. These options were given the name of history plus static and history plus random.

As mentioned in section 3.2, it can be seen that in all, there are five scheduling algorithms, i.e. History, Random, Static, History plus Random, and History plus Static. These algorithms were evaluated using simulation. There are two aspects of the performance of the scheduling algorithms: (1) the execution of tasks within a minimum possible time, (2) the distribution of loads evenly among the available resources. The first experiment was designed to compare the performance of these algorithms in terms of the total execution time to complete the same tasks. In this case the same tasks were submitted to the fog by using each of the five algorithms, one by one, and the total time to complete these tasks was recorded. This experiment was repeated for the tasks created by all workloads, one by one. Results were plotted in the form of bar graphs for each workload based tasks, separately, as shown in Figure 7.

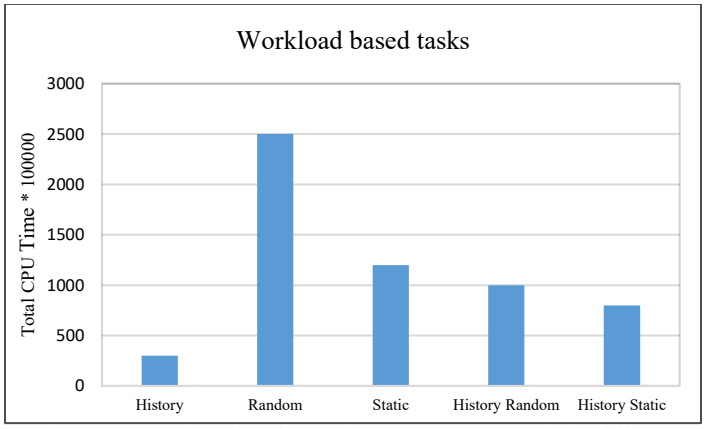


Figure.7 Comparison of total CPU time to execute 5000 tasks for different resource selection algorithms

Different scheduling algorithms were used to evaluate the performance of the predictor i.e. history, static, random, history plus static and history plus random. It was seen that overall execution time was minimum for the static and history based scheduler but it also created issues of load imbalance. History plus random and history plus static provided better load balance, but took larger overall time to finish the same tasks. It was concluded that selective task scheduling will create a load imbalance but a history based solution will operate to fix this imbalance because of the presence of a feedback loop. Random and static schedulers are missing this feedback loop, and hence lack the self-healing property of history based algorithms. Overall performance of FResS was found to be better when using the historical technique, or any hybrid technique that incorporated the historical technique

The accuracy of predictions was also compared with user provided run time estimates. Users can provide run time estimates while submitting tasks to the fog or IoT device. User run time estimates are recorded by all workloads. Given that predictions are

made, the next natural step is to determine the accuracy of these predictions. The accuracy of predictions is calculated by using (3) by the Performance Evaluator.

$$accuracy = \begin{cases} 1 & \text{if} \quad P_{RT} = R_T \\ P_{RT}/R_T & \text{if} \quad P_{RT} < R_T \\ R_T/P_{RT} & \text{if} \quad P_{RT} > R_T \end{cases} \tag{3}$$

Where, P_{RT} is predicted run time and R_T is actual run time.

The accuracy of predictions achieved by both the user and the system is presented in Table 1.

Table 1: Comparison of user's prediction accuracy and system's prediction accuracy

# No. of tasks	Average User % Accuracy	Average System % Accuracy	System 90% Confidence Interval	System 95% Confidence Interval	User 90% Confidence Interval	User 95% Confidence Interval
1000	5.9	73.0	81.4	83.6	2.0	2.0
2000	9.8	79.8	84.7	87.9	8	10
3000	15.3	86.9	88.8	90.7	14.7	16.5
4000	20.0	90.2	91.9	93.3	20.8	23.4
5000	27.1	95.0	96.2	96.9	35.0	39.0

From the results presented in Table 1, it can be seen that the smallest accuracy of 73% was achieved for the workloads of 1000 tasks. The largest accuracy of 95% was achieved for the workloads of 5000 tasks. The remaining workloads have also shown very high accuracy. The accuracy of predictions was also compared with user provided run time estimates while submitting tasks to the fog. User run time estimates are recorded, and as shown in the results in Table 1, the accuracy of user run time estimates lies between 5.9% and 27.1%. It can be concluded therefore that user accuracy is very poor when compared with the achieved accuracy from the predictor.

Another metric of interest, in the fog context, is the number of tasks falling within a certain confidence interval. This interval is defined by the Chebyshevs inequality theorem [22] and can be calculated when a sample's mean is being used as a predictor. On the fog where deadline based scheduling is expected, this confidence interval gives a better idea to the user about the expected task completion time. Chebyshevs' theorem states that the portion of data that lies within k standard deviations to the either side of the mean is at least $1 - \frac{1}{k^2}$ of any data set [22] [23] and it can be expressed as (3):

$$c = \left(1 - \frac{1}{k^2}\right) * 100 \tag{4}$$

Where c is referred to as the confidence interval and k is the number of standard deviations. The Performance Evaluator calculates the predicted tasks falling within the 90% confidence interval and 95% confidence interval. The results of both of these calculations are shown in Table 1. The predictor performed better again, since 81.4% to 96.2% tasks were completed within a 90% confidence interval and 83.6% to 96.9% tasks were completed within a 95% confidence interval. On the other hand, for user run time estimates, only 2.0% to 35.0% tasks were completed within a 90% confidence interval and 2.0% to 39.0% tasks were completed within a 95% confidence interval.

Large number of tasks were executed, first using the existing scheme then the prediction FResS model. The prediction model were generated using queries logs of large number of tasks from CloudSim to evaluate the performance of the proposed model, resource predictions from ANN were generated and used to execute the incoming tasks. The time taken by FResS and the current scheme to execute tasks was measured.

The stages of executing tasks on the fog are explained in the below section, which start with submitting the task and end with the retrieving results after completion. These stages comprise four major components and are given below:

1. **Stage in**: transferring the task(s) to executable resource.
2. **Waiting time**: waiting in the queue for its execution turn.
3. **Run time**: the task is assigned for execution, which marks the start of the run time.
4. **Stage out**: transferring the completed task(s) to the originating node.

Stage in also called stage one, starts with transferring the task from its originating node to the executing node. Once the destination node receives all the required data, then the task is forwarded to the local task manager, which in turns puts it in the waiting queue. The queue wait time represent the time spent in the queue for its execution turn, it depend on the load of the executing node. The run time in stage three marks the start of task execution. In stage four after execution is completed, the task is returned to the originating node which is called the stage out phase. The total turnaround time is given in (2).

$$T_{TRT} = T_{SIT} + T_{QWT} + T_{RT} + T_{SOT} \tag{2}$$

where:

T_{TRT} = *Total Run Time*

T_{SIT} = *Stage In Time*

T_{QWT} = *Queue Wait Time*

T_{RT} = *Run Time*

T_{SOT} = *Stage Out Time*

To evaluate FResS a number of experiments we designed. The purpose of these experiments was to evaluate overall time performance by varying the number of tasks. The tasks response time outlined in Figure 8 show that the FResS model outperform the existing scheme, and the results were promising where the tasks response time was decreased by 33%, as shown in Table 2 showing significant time savings.

Since the fog users have a limited network bandwidth to communicate among the system, such a high communication cost

is not acceptable. In such environment, nodes within a region is provided with broader bandwidth whereas nodes across region is provided with narrow bandwidth. Since many nodes from different regions try to connect to each other through a narrow bandwidth, causing congested network traffic which in turn will affect the task execution time which is an important factor, therefore, carrying out some experiments by varying and expanding the network bandwidths was important in order to simulate the real world environment, the bandwidth between sites was varied in a set of experiments to evaluate the performance of both models.

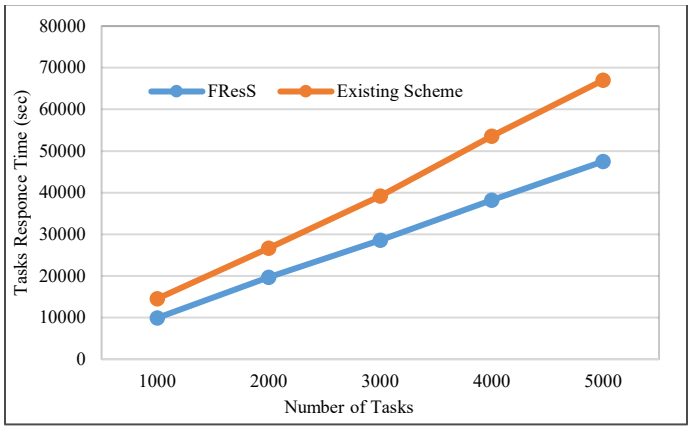


Figure 8: Comparative analysis of Response Time with no of tasks

Table 2. Simulation results: Task Turnaround Time using FResS and Existing scheme

# No. of tasks	Task Turnaround Time using existing scheme (Sec)	Task Turnaround Time using FResS (Sec)	Difference (Sec)
1000	14540	9920	4620
2000	26703	19710	6993
3000	39243	28608	10635
4000	53600	38214	15386
5000	67020	47512	19508
Total	201106	143964	57142
Average	28792	40221	33%

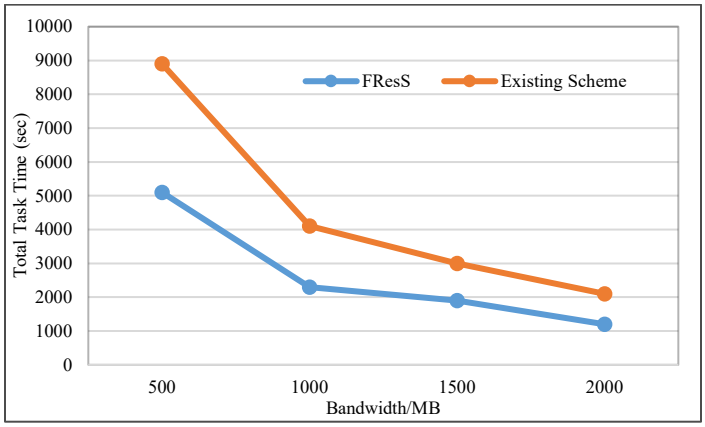


Figure 9: Total task time with varying bandwidth

Figure 9 shows the result of sets of experiments that were run on both narrow and broad bandwidth, as shown in the results, the FResS outperforms the existing scheme, even though setting broader bandwidth decreases the differences of task execution time, however, the difference is still significant in all scenarios. Consequently, it can be concluded that the cost and resource utilization are effectively considered in the proposed FResS model and outperform the existing scheme.

6. Conclusion and Future Work

to overcome the drawback of the cloud computing, the fog computing has emerged with the concept of sharing computational resources and information services by offloading the resources to the edge network to be closer to the devices that originate the requests rather than transporting the tasks to the distant cloud, thus reducing communication overhead, bandwidth consumption and latency. However, the complexity and dynamic nature of the fog systems demand a more coordination platform to handle such a large-scale numbers of resources, users and tasks requests. Therefore, the existing approaches, however, often exhibit a high cost in terms of response time and bandwidth consumption. These shortcomings were overcome by presenting a fully distributed FResS, which stores historical data related to a single user, which simplifies tasks management.

This paper proposed an extension to the resource selection service FResS based on neural network. The proposed prediction model minimize the total overhead of the task turnaround time of the incoming task by predicting and informing the system the required resources, and the database are updated constantly by storing the result of NN tool. The proposed prediction model is simple and shows comparatively lower overheads and provide high possible accuracy.

An evaluation strategy was devised to test five different algorithm used by the Resource Selector Module to provide a detailed performance analysis for each one. A History update algorithm was introduced to manage the history execution logs, as the size in IoT and fog environments are very limited. In addition, the experiments showed that the proposed prediction model distribute the load based on the user preferences. Another major strength of the proposed model is its effective resources utilization achieved by decreasing overall cost, response time, and bandwidth usage. In the future work, more metrics will be added to improve the load balancing, in addition to distributing the proposed model on the cloud layer. Moreover, more experiments will be carried out to define the size of the history records.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

The writer would like to thank the American University of the Middle East for its support.

References

[1] N. Mostafa, "Cooperative Fog Communications using A Multi-Level Load Balancing," 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC), Rome, Italy, 2019, pp. 45-51.doi: 10.1109/FMEC.2019.8795325.

[2] S. Zahara, I. Pratomo and D. S. Rahardjo, "Application and data level interoperability on virtual machine in cloud computing environment," 2015

1st International Conference on Wireless and Telematics (ICWT), Manado, 2015, pp.1-5. doi: 10.1109/ICWT.2015.7449238.

[3] V. B. Souza, X. Masip-Bruin, E. Marin-Tordera, W. Ramirez and S. Sanchez, "Towards Distributed Service Allocation in Fog-to-Cloud (F2C) Scenarios," 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, 2016, pp. 1-6. doi: 10.1109/GLOCOM.2016.7842341.

[4] L. Ni, J. Zhang, C. Jiang, C. Yan and K. Yu, "Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets," in IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1216-1228, Oct. 2017. doi: 10.1109/JIOT.2017.2709814.

[5] J. Yao and N. Ansari, "Task Allocation in Fog-Aided Mobile IoT by Lyapunov Online Reinforcement Learning," in IEEE Transactions on Green Communications and Networking. doi: 10.1109/TGCN.2019.2956626.

[6] S. Bian, X. Huang and Z. Shao, "Online Task Scheduling for Fog Computing with Multi-Resource Fairness," 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), Honolulu, HI, USA, 2019, pp. 1-5. doi: 10.1109/VTCFall.2019.8891573.

[7] M. Mukherjee et al., "Task Data Offloading and Resource Allocation in Fog Computing With Multi-Task Delay Guarantee," in IEEE Access, vol. 7, pp. 152911-152918, 2019. doi: 10.1109/ACCESS.2019.2941741.

[8] F. Derakhshan, H. Roessler, P. Schefczik and S. Randriamasy, "On prediction of resource consumption of service requests in cloud environments," the 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, pp. 169-176, 2017.

[9] Z. Zhang, B. Bockelman, D. W. Carder and T. Tannenbaum, "Lark: Bringing Network Awareness to High Throughput Computing," 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, pp. 382-391, 2015. doi: 10.1109/CCGrid.2015.116

[10] S. Yashpal, S. Alok, " Neural Network in Data Mining", Journal of Theoretical and Applied Information Technology, 37-42, 2009.

[11] C. Krieger, "Neural Networks in Data Mining", technician report, 1996.

[12] S. Duggal, R. Chhabra , "Learning Systems and Their Applications: Future of Strategic Expert System". Issues in Information Systems, Vol. III, 2002.

[13] G. jha, "Artificial Neural Networks," International journal of computer science and issues, Indian Research Institute, PUSA, New Delhi, 2005.

[14] S. Nissen, "Implementation of a fast artificial neural network library (FANN)," Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003.

[15] D. Shanthi, G. Sahoo and N. Saravanan, "Designing an Artificial Neural Network Model for the Prediction of Thromboembolic Stroke," International Journals of Biometric and Bioinformatics (IJBB), Volume (3), 10-18, 2009.

[16] S. Chen, R. Lu, J. Zhang, "An Efficient Fog-Assisted Unstable Sensor Detection Scheme with Privacy Preserved", arXiv:1711.10190v1 [cs.CR] 28 Nov 2017.

[17] I. Rao and E. Huh, "A probabilistic and adaptive scheduling algorithm using system- generated predictions for inter-grid resource sharing," Journal of Supercomputer, 45, pp: 185-204, 2008.

[18] N. Moustafa, I. Al Ridhawi, and A. Hamza, "An Intelligent Dynamic Replica Selection Model within Grid Systems, " in Proc. 8th IEEE GCC conference on Towards Smart Sustainable Solutions, pp. 1-6, February 2015. doi: 10.1109/IEEEGCC.2015.7060061.

[19] "The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne," December. 2012, Available: http://www.cloudbus.org/cloudsim/.

[20] S. Mostafi, F. Khan, A. Chakrabarty, D. Y. Suh and M. J. Piran, "An Algorithm for Mapping a Traffic Domain Into a Complex Network: A Social Internet of Things Approach," in IEEE Access, vol. 7, pp. 40925-40940, 2019. doi: 10.1109/ACCESS.2019.2906647.

[21] B. Bhuyan, H. Sarma, N. Sarma, A. Kar and R. Mall, "Quality of Service (QoS) Provisions in Wireless Sensor Networks and Related Challenges", Wireless Sensor Network, Vol. 2 No. 11, pp. 861- 868, 2010. doi: 10.4236/wsn.2010.211104.

[22] W. J. Chu Using Chebyshev's Inequality to Determine Sample Size in Biometric Evaluation of Fingerprint Data, Forgotten Books, 2018.

[23] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," Journal of Parallel Distributed Computing, pp. 1007–1016, 2004.