An Online Fair Resource Allocation Solution for Fog Computing

Jia He Sun, Salimur Choudhury, and Kai Salomaa

Fog computing is a complementary computing paradigm to the existing cloud computing. A fundamental problem of fog computing is how to allocate the computing resources of fog nodes when scheduling tasks that arrive in an online manner. Other than task completion speed metrics, fairness of resource allocation between competing users is also an important metric to consider. One such metric is Dominant Resource Fairness (DRF), a fairness scheme that guarantees four key qualities: incentivized sharing, strategy-proof, Pareto-efficiency, and envy free. This paper examines the multi-resource, multi-server, and heterogeneous task resource allocation problem from a DRF perspective. Four different types of tasks are considered: ordered/unordered, splittable/unsplittable. Three low complexity heuristics are proposed to maximize fairness between users. Results show that the proposed heuristics are at least comparable to three baseline scheduling algorithms in terms of task completion speed while achieving higher fairness between users.

***Index Terms*—fog computing, fair resource allocation, heuristic, online scheduling**

I. Introduction

In recent years, distributed computing infrastructures have evolved at an rapid pace. With it, the sheer quantity of computational devices that comprise the edge layer of these large scale networks is astounding [1]. As a result, the vast amount of data produced by the edge devices as well as the computational needs required by the edge devices have become severe bottlenecks in large scale networks that rely on cloud computing. Fog computing is an one of several emerging complementary computing paradigms that aim to address current limitations in cloud computing [2]. As displayed in Fig. 1, fog computing introduces an intermediary layer between the cloud layer and the edge layer named the fog layer. This fog layer is comprised of many fog nodes that provide data processing and analysis for the edge devices with reduced latency and increased quality as well as data filtering and load management for the cloud layer. Overall, the implementation of the fog layer aims to provide better service to the end devices while reducing the burden on the cloud layer by managing data analysis and computational requests at an intermediate level.

To accomplish this, part of the fog layer's job is to receive and process computational service requests put forward by edge devices. In this infrastructure, computational service requests can be observed by several fog nodes. Each request can be decomposed into a set of tasks and each fog node can be decomposed into a set of resources. Consequently, the task scheduling problem is one of the main focuses of current research in the fog computing field [3].

As shown in Fig. 2, resource allocation and task scheduling algorithms can be categorized as either static or dynamic. In static scheduling, information regarding every incoming task are made available to the system. On the other hand, in dynamic scheduling, the resource requirements of each task are not known until its arrival. Due to the dynamic nature of the numerous edge devices in a fog environment, this paper's focus is dynamic scheduling algorithms, more specifically, real-time scheduling algorithms where the tasks arrive to the system in a continuous manner. In this paper, we will call this online scheduling.

There are two main approaches to online scheduling, the first is termed completely reactive scheduling where tasks are scheduled as they arrive based on a set of predefined rules or heuristics [4]. This approach easily handles randomly arriving tasks but often performs far from the optimal due to the reactive nature of the algorithm [5]. The second approach is predictive-reactive scheduling where an initial schedule is constructed for existing tasks and then revised when certain events occur such as new task arrivals or machine breakdowns. Predictive-reactive scheduling usually performs better than completely reactive scheduling, but due to the dynamic nature of the fog computing environment where large amount of tasks are continuously arriving, it is inappropriate to adopt the predictive-reactive scheduling approach [6] [7].

There many works regarding online scheduling techniques in various settings, Ouelhadj and Petrovic give a detailed survey of such methods [5]. Many of these scheduling schemes focus on some measure of task completion time (makespan, lateness, etc). However, resource fairness is an objective that is often critical in computing environments. Allocating resources with respect to fairness and efficiency is a fundamental problem in the design of fog computing environments. Maintaining fairness ensures that a balanced allocation of resources amongst the tasks where no tasks are starved at the expense of another. Furthermore, resource fairness also encourages maximizing utilization of all available resources. Maintaining resource fairness is an area quite heavily explored during the development of cloud computing several years ago. In 2009, Microsoft published the "Quincy" scheduler that maps a scheduling problem to a graph data structure that computes the optimal schedule that maximizes a global cost function that includes locality and resource fairness [8]. In 2010, Zaharia *et al.* proposed an algorithm called "Delay Scheduling" to address the conflict between locality and fairness for the 600-node Hadoop cluster at Facebook [9].

This paper was submitted in enter date 2022.

J. H. Sun is with the School of Computing, Queen's University, Kingston, ON K7L 3N6 Canada (email: 20jhhs@queensu.ca).

S. Choudhury is with the Department of Computer Science, Lakehead University, Orillia, ON L3V 0B9 Canada (email: salimur.choudhury@lakeheadu.ca).

K. Salomaa is with the School of Computing, Queen's University, Kingston, ON K7L 3N6 Canada (email: ksalomaa@cs.queensu.ca).

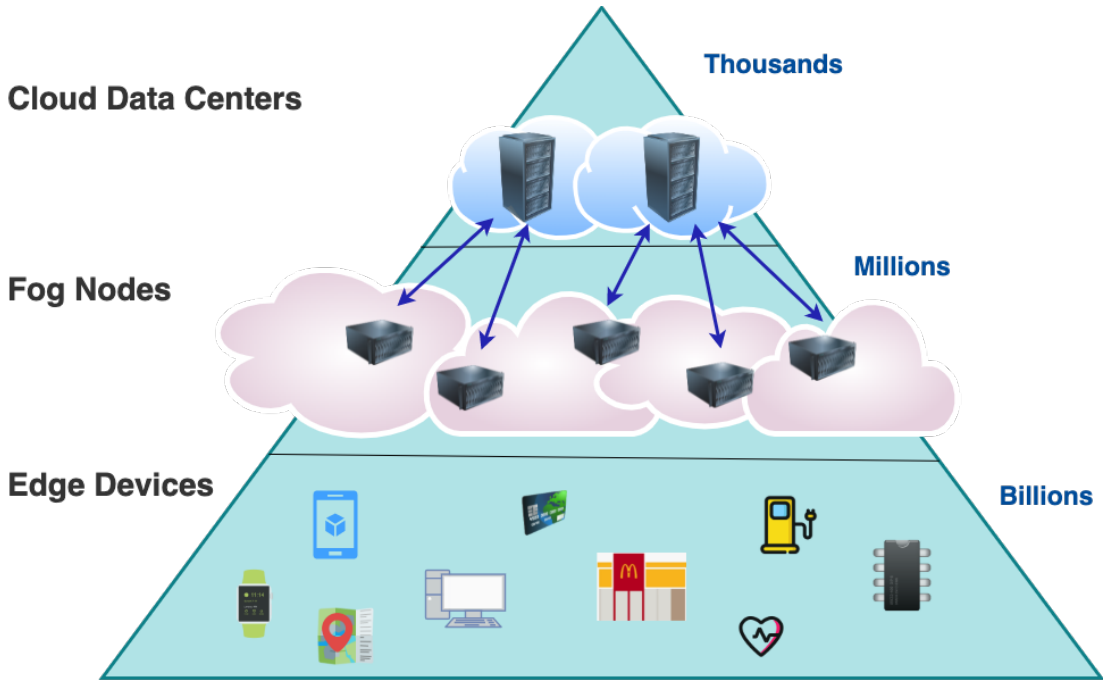

Fig. 1. Fog Computing Infrastructure Diagram

However, since fog computing is an area that has only recently emerged, resource fairness within a fog environment is a much less explored topic. In 2018, Zhang *et al.* proposed Fair Task Offloading (FTO) scheme that minimizes task delay and energy consumption while maintaining fairness between network nodes [10]. In 2019, Mukherjee *et al.* proposed a scheduling policy that maximizes the number of tasks completed before their deadline while maintaining fairness by keeping both high priority and low priority task queues stable [11]. However, the both of these scheduling schemes only maximize fairness between different fog nodes instead of between participating users. This is also true for the Quincy scheduler and the Hadoop Delay Scheduling algorithm.

Fairness between users who request tasks can be evaluated by "Dominant Resource Fairness" (DRF), an index developed by Ghodsi *et al.* in 2011 [12]. The DRF scheme is a multi-resource generalization of max-min fairness. According to the DRF scheme, in a multi-resource environment where tasks have heterogeneous resource demands, a user's dominant share is the maximum global share that the user has been allocated of any resource. In essence, the DRF scheme aims to have all users have equal dominant share values in any allocation. In their publication defining the DRF scheme, Godhsi *et al.* considered many other fairness schemes including Asset Fairness and Competitive Equilibrium from Equal Incomes (CEEI). Asset Fairness aims to equalize the aggregate resource value allocated to each user assuming that equal shares of different resources are worth the same. That is, 1% of resource 1 is equivalent to 1% of resource 2. CEEI, the preferred method of fair resource allocation in microeconomic theory, initially allocates $\frac{1}{n}$ of each resource to each user and subsequently, each user can trade their resources with other users. Godhsi *et al.* found that the DRF scheme is the only fairness scheme that satisfies the following four key qualities:

- Sharing Incentive: every task's allocation is not worse off than that obtained by evenly dividing the entire resource pool
- Strategy-proof: tasks cannot get better allocation by lying about their requirements
- Pareto Efficiency: all available resources are allocated subject to satisfying the other properties, and without preempting existing allocations
- Envy Free: no tasks prefers the allocation of another task

Asset Fairness violates the sharing incentive quality and CEEI violates the strategy-proof quality. All of the above qualities are desired in a fog computing environment and would increase the Quality of Service (QoS) for users. So far, it remains a difficult problem to design an online resource allocation scheme while maintaining multi-server multi-resource fairness. In 2019, Bian *et al.* tackles this problem and proposed "FairTS", a fair online task scheduling scheme that uses DRF as their fairness scheme [7]. FairTS uses a Reinforcement Learning (RL) approach to minimize average task slowdown while maximizing the minimum dominant resource share of each task. However, as a machine learning based algorithm, FairTS falls behind heuristics in terms of computation speed, a much desired attribute in fog computing environments. Furthermore, FairTS is designed for a simplified model where all resources are located in one computer/server. However, in a realistic fog setting, task requests would be sent to be processed by a collection of fog servers. In 2014, Wang *et al.* proposed "Dominant Resource Fairness with Heterogeneous Servers" (DRFH), which is a generalization of DRF for multiple heterogeneous servers [13].

DRFH preserves the strategy-proof, Pareto efficiency, and envy free qualities of DRF. The sharing incentive strategy is

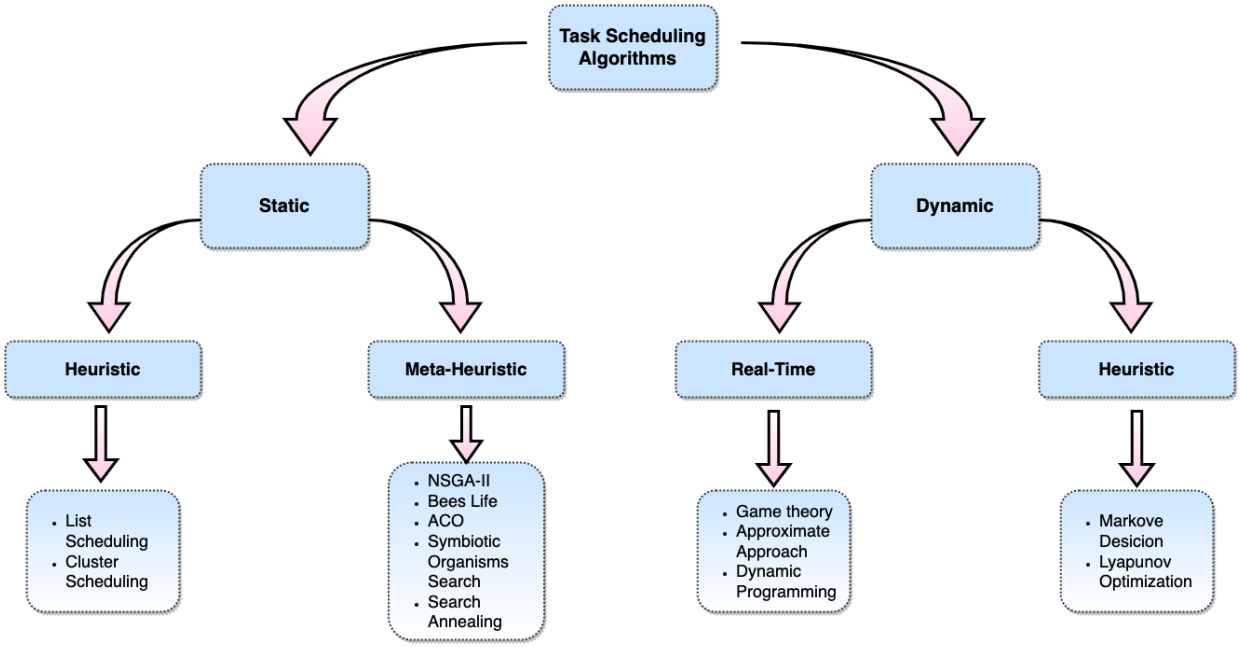

Fig. 2. Taxonomy of Scheduling Algorithms

not well defined in a multi server setting as there is an infinite number of ways to evenly divide the resource pool. In 2017, Östman proposed a Distributed Dominant Resource Fairness (DDRF) scheme that uses a gradient network topology overlay to create a dynamic directed sorted graph based on their users' resource share. The proposed scheme allows multiple servers to allocate resources in parallel to achieve faster allocation time [14]. However, similar to the proposed DRFH scheme, the environment considered is also very basic. It only considers the case where each user will have an infinite amount of divisible homogeneous tasks. In fact, after the proposal of the DRF scheme in 2011, its assumption of divisible tasks is a heavily discussed topic as it contradicts the Leontief preferences [15].

In practice, users will usually have indivisible tasks that are also heterogeneous. In [9], Zaharia *et al.* revealed the resource usage profiles of tasks in a 2000-node Hadoop cluster at Facebook over one month (October 2010) and showed that the requested tasks have very different resource requirements. Furthermore, there may be other requirements and characteristics for each user such as ordering and splittable-ness. A user's tasks can be ordered where they must be completed in a task chain. A user's tasks can also be splittable where they can be allocated more than their required amount of resources to be completed faster. This paper will consider the fair resource allocation problem under these more realistic conditions.

The main contributions of this paper can be summarized as follows:

- Formulate the multi-resource, multi-server, and heterogeneous task resource allocation problem as a fairness maximizing problem using the DRF scheme.
- Propose three low complexity heuristics for different types of tasks: ordered/unordered, splittable/unsplittable.
- Evaluate the proposed heuristics against three baseline scheduling algorithms.

The rest of this paper is organized as follows: we will describe the system model, formulate the problem, propose our solution, analyze the proposed solution, display simulation results, and conclude the paper.

II. System Model

In a fog computing system, n users will be connected to k nearby fog node servers $S = \{1, ..., k\}$. Each server contributes m resources $R = \{1, ..., m\}$. Each server j has resource capacity $c_j = \{c_{j1}, ..., c_{jm}\}$ where each element is represented as a fraction of the total amount of said resource in the entire system. That is, for every resource, the total capacity of all servers added together is 1. This paper will consider a time step system, where a set of users will arrive to the system at the beginning of each time step in an online manner. Each user $n = 1, 2, ...$ will have a set of heterogeneous tasks that need to be completed. In this set, each task $i = 1, 2, ...$ is characterized by:

- Arrival time t_i^{arr}
- Time steps required for computation l_i (assuming resource demands are met)
- Resource demand vector $D_i = (D_{i1}, ..., D_{im})^T$ where each D_{ir} is a fraction over the total amount of resource r in the entire system (assumed to be non-negative)
- Task i's global dominant resource $r_i^* := \arg\max_{r \in R} D_{ir}$
- Normalized resource demand vector $d_i = (d_{i1}, ..., d_{im})^T$ where $d_{ir} = D_{ir}/D_{ir_i^*}$ for each resource r

Under the DRF scheme, each user is characterized by its *global dominant share*.

Definition 1: Under the DRF scheme, the *global dominant share* of user i under allocation A_i is defined as:

$$G_i(A_i) := \sum_{j \in S} \min_{r \in R} \{A_{ijr}/d_{ir}\} \quad (1)$$

where for each server j and user i, $A_{ij} = (A_{ij1}, ..., A_{ijm})^T$ is the allocation vector and A_{ijr} is the share of resource r allocated to user i on server j represented as a fraction over the total amount of resource r in the entire system. And, $A_i = (A_{i1}, ..., A_{ik})$ is the allocation matrix for user i.

In other words, it is how much of the system's resource has been given to the most limiting resource of that user summed over all existing servers. An example case will be given in the next subsection to further explain the global dominant share definition.

To summarize the system model, this is a time step based system where at each time step, the system operates as follows:

1) A set of users arrive to the system (may be an empty set). Each user will have a set of tasks to be completed (assumed to be non-empty). Information regarding these tasks are not known to the system until their arrival.
2) A user's tasks can be ordered or unordered, splittable or unsplittable. We will consider all four possible cases. We will also assume tasks are non-preemptive, that is, they cannot be stopped once they have started computing.
3) The system will then allocate a portion of resources on any amount of servers to each user (new and existing) based on the DRF scheme. This means that existing users can have their allocation changed.
4) After the resources have been allocated, the tasks will be run for one time step using the resources that they have been assigned.
5) Any users who have completed all their tasks will now leave the system.
6) Repeat for next time step.

A. Example Case

To illustrate the idea behind the DRF scheme, a simple example with 2 servers and 2 users is given. The setting is as follows:

- Server 1: 5 CPU 11 memory / server 2: 11 CPU 5 memory
- Server 1 capacity vector: (5/16, 11/16) / server 2 capacity vector: (11/16, 5/16)
- User 1's task requirements: 1 CPU 0.5 memory / user 2's task requirements: 0.5 CPU 1 memory
- Assume both users have an infinite number of these tasks
- User 1's resource demand vector: (1/16, 1/32) / user 2's resource demand vector: (1/32, 1/16)
- User 1's normalized resource demand vector: (1, 1/2) / user 2's normalized resource demand vector: (1/2, 1)

From the normalized resource demand vectors, we can see that CPU is user 1's global dominant resource and memory is user 2's global dominant resource. Optimally, it is obvious that user 1 should be allocated all of server 2 and user 2 should be allocated all of server 1. Following this allocation, user 2's allocation is $(\frac{5}{16}, \frac{11}{16})$. User 2's global dominant share is $\min\{\frac{5/16}{1/2}, \frac{11/16}{1}\} = \frac{10}{16}$, which is associated with the CPU resource. Similarly, user 1's global dominant share is $\frac{10}{16}$. Notice that the two users' global dominant shares are the same and also the maximum that they can be. So, the system would allocate all of server 1 to user 2 and let it run 10 tasks and allocate all of server 2 to user 1 and let it run 10 tasks as well. This will continue until a new user arrives and the resources will be reallocated. This allocation results in equal global dominant share and $\frac{15}{16}$ or around 94% resource utilization.

Instead of using global dominant share, there exists a simpler way to extend the DRF scheme to our multi-server environment by applying the DRF scheme to each server individually. However, it is easy to show that this approach is naively inappropriate using the same example case as above.

Following this approach, we would consider the two servers separately. In server 1, both users' dominant resource is CPU. So, to maximize the minimum dominant share, both users will receive half of the available CPU which would be 2.5 each. Similarly, in server 2, both users' dominant resource is memory, then they will both receive half of the available memory resource which is also 2.5 each. So, user 1 will be able to run 2.5 tasks on server 1 and 5 tasks on server 2. User 2 will be able to run 5 tasks on server 1 and 2.5 tasks on server 2. Both users will be able to run 7.5 tasks each. This is far worse than the solution found previously. Furthermore, the resource utilization for the system is $\frac{22.5}{32}$ or around 70%, which is also evidently much worse than the previous solution.

III. Problem Formulation

The fairness maximizing optimization problem is defined as follows:

$$\underset{A}{\text{maximize}} \quad \min_{i \in T} G_i(A_i) \tag{2a}$$

$$\text{subject to}$$

$$\sum_{i \in T} A_{ijr} \leq c_{jr}, j = 1, ..., k, r = 1, ..., m, \tag{2b}$$

$$A_i \in \{0, 1\}^{mk}, i \in T \tag{2c}$$

Recall that $A_i = (A_{i1}, ..., A_{ik})$ is the allocation matrix for user i and $G_i(A_i) := \sum_{j \in S} \min_{r \in R}\{A_{ijr}/d_{ir}\}$ is the *global dominant share* of user i. To summarize, this optimization problem aims to maximize the minimum global dominant share among all users such that no server's resource capacities are violated.

Theorem 1: This problem is NP-hard

Proof. Consider the bin packing problem:

$$\underset{x,\, y}{\text{minimize}} \quad \sum_{i=1}^{N} y_i \tag{3a}$$

$$\text{subject to}$$

$$\sum_{i=1}^{N} x_{ji} = 1, \qquad j = 1, \ldots, K, \tag{3b}$$

$$\sum_{j=1}^{K} w_j x_{ji} \leq B y_i, i = 1, \ldots, N, \tag{3c}$$

$$y_i \in \{0, 1\} \qquad i = 1, \ldots, N, \tag{3d}$$

$$x_{ji} \in \{0, 1\} \qquad j = 1, \ldots, K, i = \ldots, N \tag{3e}$$

Now, consider a single resource instance of our resource allocation problem where items and bins correspond to tasks

and servers, respectively. The single resource corresponds to the size limitations in the bin packing problem, that is, the size of a item w_j corresponds to the resource requirement of a task and the capacity of a bin B corresponds to the resource capacity of a server. Then, the bin packing problem exactly corresponds to a single resource instance of our problem where each user has exactly one task. The bin packing problem (decision version) asks if all items can fit into A bins where A is a given constant. Notice that fitting all tasks onto A servers in our resource allocation problem would produce a unique objective value which would be the global dominant share of the smallest task. This value is unique since if any task was to remain unassigned to a server, the minimum global dominant share would be 0. Therefore, the following questions are equivalent:

- Can all tasks be assigned onto A servers?
- Can there be such a task assignment onto A servers where the minimum global dominant share is $b_j = \min_{i \in K} b_i$?

Therefore, we can see that the bin packing decision problem can be reduced to a decision instance of the formulated optimization. Since the bin packing problem is known to be NP-hard, our formulated resource allocation problem is NP-hard as well.

IV. Proposed Solution

As mentioned previously, there are different types of tasks that can be sent to the system. A user's tasks can be *splittable* which means they can be allocated an integer multiple of its resource requirements to complete proportionally faster (two times the required resources means two times the completion speed) or *unsplittable* which means that they complete at the same speed as long as its requirements are satisfied. A user's tasks can also be *ordered* where each task must be completed in a specific order or *unordered* where the tasks can be completed in any order and at the same time. For ordered tasks, it is likely that data must be transferred between the tasks, as such they must be completed on the same server since it will be inefficient and expensive to offload tasks of the same chain onto different servers. Furthermore, a task chain cannot be interrupted once it has started. In this paper, we will consider four different cases:

1) all users have unsplittable and unordered tasks
2) all users have splittable and unordered tasks
3) all users have unsplittable and unordered tasks
4) all users have splitaable and ordered tasks

A. Unsplittable and Unordered

In this section, we will assume all users have a set of unsplittable and unordered tasks. This means that tasks from the same user need not be on the same server since they are unordered. This is the least restricted case where tasks can be scheduled on any server and in any order. It is also meaningless to assign a task more than its required resource since it does not increase its completion speed. Algorithm 1 is a proposed heuristic to schedule tasks in this scenario.

At each time step, Algorithm 1 repeatedly picks the user with the lowest global dominant share and schedules its most resource demanding task until there are no remaining tasks left or there is no more resources available. By picking the user with the lowest global dominant share (line 5), we aim to maximize the minimum global dominant share across all users, which is the objective of our optimization problem. Instead of picking any task, the most resource demanding tasks are picked first to further minimize the variance between the global dominant shares (line 6). Consider the following example:

Algorithm 1 Unsplittable and Unordered

```
1:  for each time step do
2:      new users arrive
3:      available tasks ← all unassigned tasks of every user
4:      while there are available tasks do
5:          current_user i ← user with the smallest global dominant share G_i(A_i)
6:          current_task j ← available task of current_user i with the largest resource demands ∑ d_j
7:          if no server can fit current_task j then
8:              remove current_task j from available tasks
9:          else
10:             assign current_task j to best fit server
11:             update remaining server resources and available tasks
12:         end if
13:     end while
14: end for
```

- 1 server with 10 units of a single resource
- 2 users each with 4 tasks with the following resource demands: (1, 1, 1, 4)

If we schedule the tasks based on some arbitrary order say: (1, 1, 1, 4). Then, the result would be one user getting 7 units of the resource and the other user getting only 3. However, if the most resource demanding tasks are picked first, then the result would be both users getting 5 units of the resource which means equal global dominant shares. Hence, it would be the optimal solution since it maximizes the minimum global dominant share. From this example we can see that scheduling the most resource heavy tasks can be beneficial.

After picking the most resource heavy task, we assign to it to the best fit server. In this case, the fitness of assigning task i to server j is defined by $F(i,j) = ||D_i/D_{i1} - c_j^*/c_{j1}^*||$ where c_j^* is the vector representing the remaining resources of server j. Then in line 10 of Algorithm 1, task i's best fit server is characterized by the server who produces the smallest $F(i,j)$ value. In other words, the best fit server is the server whose remaining resource vector is the most similar to its resource demands. This ensures that CPU heavy tasks are assigned to CPU heavy servers, for example.

B. Splittable and Unordered

In this section, we will assume all users have a set of splittable and unordered tasks. Tasks from the same user need not be on the same server since they are unordered. Tasks can be allocated an integer multiple of their required resources to decrease completion time. Splittable tasks add

a new dimension to the resource allocation problem since it must be considered if a resource is better used to speed up an existing task or to begin a new task. Furthermore, since it is now possible to allocate more than the required resources to a task, we can get closer to equal global dominant shares than in the previous case. Algorithm 2 is the proposed heuristic for the splittable and unordered case.

Algorithm 2 Splittable and Unordered

1: **for** each time step **do**
2: new users arrive
3: available tasks ← all unassigned tasks of every user
4: **while** there are available tasks **do**
5: current_user i ← user with the smallest global dominant share $G_i(A_i)$
6: current_task j ← available task of current_user i with the largest resource demands $\sum d_j$
7: **if** no server can fit current_task j **then**
8: remove current_task j from available tasks
9: **else**
10: assign current_task j to best fit server
11: allocate nD_j resources where n minimizes $Var_i(G_i(A_i))$ and $n < l_j$
12: update remaining server resources and available tasks
13: **end if**
14: **end while**
15: **end for**

Due to the similarities of the two cases, Algorithm 2 works similarly to Algorithm 1. Once again, at each time step, the system repeatedly picks the user with the lowest global dominant share (line 5) and schedules its most resource demanding task (line 6) until there are no remaining tasks left or there is no more resources available. Also, the best fit server mentioned in line 10 is the same characterization as in Algorithm 1 where the server with the most similar remaining capacity vector is chosen.

The reasoning for this design is the same as in the previous case. The key difference is how to determine what integer multiple of the required resources to allocate to each scheduled task.

Recall that the DRF scheme's objective is to maximize the minimum global dominant share across all users, however, it is difficult to maximize this in each step of the allocation process since the lowest global dominant share user just needs to surpass another user's global dominant share to do so. However, the aim behind the maximization of the minimum global dominant share is to achieve a balanced resource share across all tasks. Therefore, the proposed algorithm allocates resource such that to minimize the variance of all global dominant shares (line 11). However, there is the requirement that a task j should not be allocated more than l_j times its resource requirements where l_j is its execution time.

Theorem 2: In the unordered and splittable case, a task j should not be allocated more than l_j times its resource requirements.

Proof. By allocating l_j times its resource requirements to task j, its execution time is reduced to 1 time step which is the lowest that it can be in our time step model. Increasing its allocation any further becomes a wasteful use of resources. Therefore, task j must not be allocated more than l_j times its resource requirements.

C. Unsplittable and Ordered

In this section, we will assume all users have a set of unsplittable and ordered tasks. Tasks from the same user must be on the same server.

Since this paper approaches the task scheduling problem from a fair allocation perspective, this case becomes trivial. In this case, it is meaningless to assign more than required resources to a user since tasks are unsplittable. Also, since each user has a task chain, a user will only need to run one unsplittable task at any given point. As such, there is no decision to be made from a fairness perspective as each user only requires a fixed amount of resources at any given moment and allocating more will not be beneficial to the user at all. Therefore, we will not propose a heuristic for this case.

D. Splittable and Ordered

In this section, we will assume all users have a set of splittable and ordered tasks. Tasks from the same user must be on the same server and a task chain cannot be interrupted. Tasks can be allocated an integer multiple of their required resources to decrease completion time. This case is rather complex since a task chain can include tasks with wildly different resource demands which makes it difficult to allocate without being wasteful. Furthermore, allocation can be changed during a task chain based on the task currently being executed. Overall, this means a lot more decisions will need to be made in order to achieve the most fair allocation. Algorithm 3 describes the proposed algorithm. Since, in this case, each user has exactly one task chain, we will use the terms "user" and "task chain" interchangeably. That is, user i and task chain i refer to the same thing.

** The best fit server for a given task chain is characterized by smallest $F(i,j) = ||z_i/z_{i1} - c_j^*/c_{j1}^*||$ for user i and server j where c_j^* is the vector representing the remaining resources of server j and z_i is the size characteristic of user i.

At each time step, if new users arrive, Algorithm 3 will first compute every new user's size characteristic which, for user i, is defined by $z_i = D_i^*$. D_i^* (line 7). This is the vector representing the largest demands of each resource in the entire task chain. For example, the largest demands of each resource in the task chain $(1,5),(5,1),(6,2)$ is $(6,5)$. Then, the size characteristic z_i for the user with this task chain would be $(6,5)$. In other words, if user i is allocated z_i resources, then it can execute its task chain in its entirety. Then, we reduce all users currently executing task chains down its lowest required amount of resources which is z_i for user i (line 10). This is to allow room for the new users to be allocated resources but at the same time not to interrupt any existing task chain. Then, the algorithm will select the user with the smallest global dominant share, however, since all new users have global

Algorithm 3 Splittable and Ordered

1: **for** each time step **do**
2: new users arrive
3: **if** new users is empty **then**
4: pass
5: **end if**
6: **for** user $i \in$ new users **do**
7: size characteristic $z_i \leftarrow D_i^*$
8: **end for**
9: **for** user $i \in$ existing users **do**
10: reduce A_i to z_i
11: **end for**
12: available users $\leftarrow$ all users
13: **while** available users is non empty **and** server resources are non empty **do**
14: current_user $i \leftarrow$ user with the smallest global dominant share $G_i(A_i)$ tie break by largest size characteristic z_i
15: **if** no server can fit current_user i **then**
16: remove current_user i from available users
17: **else**
18: assign current_user i to best fit server
19: allocate nz_i resources where n minimizes $Var_i(G_i(A_i))$ with $n < max_{j \in i} l_j$
20: **end if**
21: update server resources and available users
22: **end while**
23: **end for**

dominant share 0, this selection will have a tie break by largest size characteristic z_i (line 14). This is because the largest size characteristic typically denotes the most resource demanding users. We select the most resource demanding users to allocate first due to it being easier to allocate large task chains when there is more resource available. Then, we select the best fit server based on the size characteristic and assign the task chain to this server until its completion (line 18). The best fit server for a given task chain is characterized differently than for a single task. In this case, the fitness of server j for task chain i is defined $F'(i,j) = ||z_i/z_{i1} - c_j^*/c_{j1}^*||$ where c_j^* is the vector representing the remaining resources of server j and z_i is the size characteristic of task chain i. Then, the proposed algorithm allocates an integer multiple amount of resource such that to minimize the variance of all global dominant shares (line 19). However, there is the requirement that a user i should not be allocated more than $max_{j \in i} l_j$ times its size characteristic z_i. Repeat for next time step.

Theorem 3: In the splittable and ordered case, a task chain i should not be allocated more than $max_{j \in i} l_j$ times its size characteristic z_i.

Proof. Consider allocating $max_{j \in i} l_j$ times its size characteristic z_i amount resources to task chain i. Then, for each task j in task chain i, it will be executed on at $max_{j \in i} l_j$ times z_i amount of resources. By definition, $max_{j \in i} l_j \geq l_j$ and $z_j \geq D_j$. Hence, each task j in task chain i has its execution time reduced to less than or equal to 1 time step which is the minimum it can be. Therefore, it is wasteful to allocate more resources to this task chain. Therefore, a task chain i should not be allocated more than $max_{j \in i} l_j$ times its size characteristic z_i.

V. Algorithm Analysis

A key characteristic is that the proposed algorithms must be very low complexity. This is due to the highly dynamic nature of online scheduling in fog computing. Existing cloud clusters have server populations in the tens of thousands with even more users and task requests. As an augment to the cloud computing paradigm, fog computing is expected to be implemented at the same scale. Furthermore, these algorithms must be run at every time step, therefore the scheduling algorithms must not only be high effective but also highly efficient. Following, we will prove the termination and complexity for each of the proposed algorithms.

Theorem 4: Algorithm 1 Unsplittable and Unordered terminates and has complexity $O(n^2)$ where n is the number of available tasks.

Proof. Suppose there are n available tasks and m servers at each time step. Then, we will need to first find the user with the smallest global dominant share which takes $O(logn)$ time in the worst case. Then, we must select this user's task with the largest resource demands which takes $O(logn)$ time in the worst case. Then, we must find the best fit server for the selected task which takes m time since we must compute the best fit function for each server. In the worst case, we must do this for all n tasks. Then, the complexity of this algorithm is $O(n(m + logn))$. Without loss of generality, assume that $n > m$, then $O(n(m + logn)) = O(n^2 + nlogn) = O(n^2)$. Therefore, Algorithm 1 is $O(n^2)$. Furthermore, since n is finite, Algorithm 1 terminates.

Theorem 5: Algorithm 2 Splittable and Unordered terminates and has complexity $O(n^2)$ where n is the number of tasks.

Proof. Suppose there are n tasks and m servers at each time step. Algorithm 2 performs similarly to Algorithm 1. The key difference is needing to find the variance of all users' global dominant share after the selection of a task. Computation of variance takes $O(n)$ time in the worst case. Then, the complexity of this algorithm is $O(n(m + n))$. Without loss of generality, assume that $n > m$, then $O(n(m + n)) = O(n^2 + n^2) = O(n^2)$. Therefore, Algorithm 2 is $O(n^2)$. Furthermore, since n is finite, Algorithm 2 terminates.

Theorem 6: Algorithm 3 Splittable and Ordered terminates.

Proof. Suppose there are n users and m servers at each time step. Algorithm 3 will compute the size characteristic z_i for all new users and reduce allocations A_i for all existing users which takes $O(n)$ time. Then, we will need to find the user with the smallest global dominant share tie broken by largest size characteristic which takes $O(logn)$ time in the worst case. Then, we must find the best fit server for the selected user which takes m time since we must compute the best fit function for each server. In the worst case, we must do this for all n users. Then, the complexity of this algorithm is $O(n(m + logn + n))$. Without loss of generality, assume that $n > m$, then $O(n(m + logn + n)) = O(nm + nlogn + n^2) =$

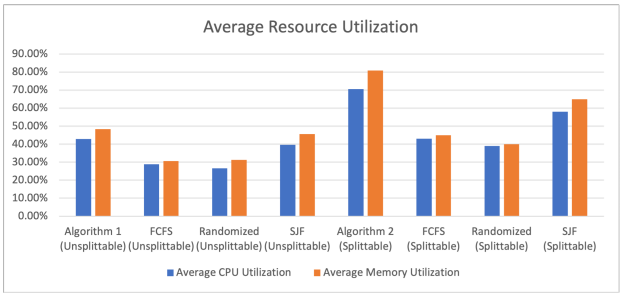

Fig. 3. Average Resource Utilization

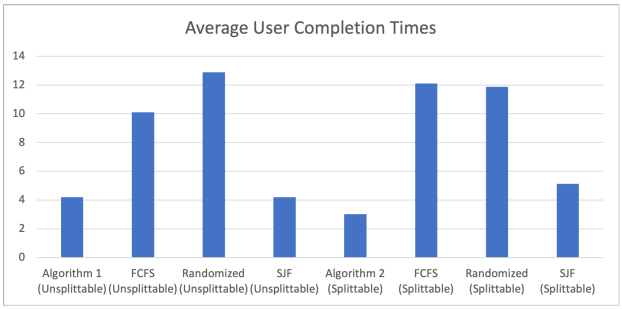

Fig. 4. Average Completion Time

$O(n^2)$. Therefore, Algorithm 3 is $O(n^2)$. Furthermore, since n is finite, Algorithm 3 terminates.

VI. Simulation Results

The simulation task data comes from Google cluster-usage traces. The traces contain resource demand/usage information of over 900 users (i.e., Google services and engineers) on a cluster of 12,000 servers. The server configurations of these servers are provided by Wang et al. in 2014 [13]. The tasks and servers used in our simulations are randomly extracted from the above data. Each time, we simulate 5 users who arrive based on a Poisson process with mean rate of 0.5 on 3 servers with 2 resource types (CPU and memory). Each user has a random amount of tasks.

To evaluate the performance of the proposed algorithms, three baseline algorithms will be used: First Come First Serve (FCFS), Randomized, and Shortest Job First (SJF). The first two algorithms are self explanatory. SJF is a very efficient algorithm used to minimize waiting time for scenarios where tasks' execution time is known. It works by scheduling the tasks with the shortest execution time first. In fact, it is optimal, in that for a given set of processes and their execution times it gives the least average waiting time for each process. While it is not optimal in our multi-server, multi-resource, online environment, it still provides a good benchmark. Since we have multiple servers, SJF will decide on which server to assign each task to using the same best fit characterization used in the proposed algorithms. Furthermore, to evaluate Algorithm 2 and Algorithm 3, a splittable version of all three baseline algorithms are used. In these versions, each task is allocated a random integer multiple of its required resources. The splittable versions of the baseline algorithms work the same otherwise. Each experiment was run 20 times for more generalized results since the input data are randomly selected.

A. Algorithm 1 and Algorithm 2

We will first evaluate the performance of Algorithm 1 and Algorithm 2. Algorithm 2 is evaluated with Algorithm 1 because they are very similar in design. Also, it allows us to observe the effect of having splittable tasks in the system.

1) Resource Utilization

Our first evaluation focuses on how well the algorithms utilize the available resources in the scheduling system. Fig. 3 shows the average resource utilization percentage of all simulated algorithms. We can see that Algorithm 1 slightly

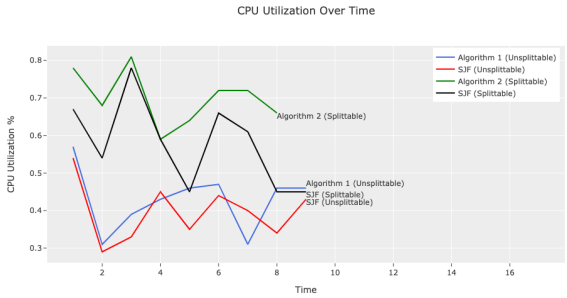

Fig. 5. CPU Utilization Over Time

outperforms SJF (unsplittable) and severely outperforms FCFS (unsplittable) and Randomized (unsplittable). Higher resource utilization correlates to faster completion times and Fig. 4 reflects this. Algorithm 1 completes tasks at a similar speed as SJF (unsplittable) and much faster than FCFS (unsplittable) and Randomized (unsplittable). Since Algorithm 1 is designed such that it maximizes the minimum global dominant share across all users, performing similar to SJF (unsplittable) which focuses on task completion speed can be considered a success.

In the splittable case, a similar result is shown. Algorithm 2 performs the best in terms of resource allocation followed by SJF (Splittable) and then by Randomized (Splittable) and FCFS (Splittable). This is mirrored in Fig. 4.

It is expected that Algorithm 2 outperforms all other algorithms, both splittable and unsplittable in terms of resource utilization. This is due to splittable tasks being able use more resources since the system can fill gaps of unused resources by allocating more to a splittable task, and consequently these splittable tasks will complete faster resulting in better performances in completion speeds as well. While Algorithm 1 performs similarly to SJF (Unsplittable), Algorithm 2 outperforms SJF (Splittable) by a rather significant margin.

In summary, the proposed algorithms performed similarly or better than their counterparts in terms of both resource utilization and task completion times.

Fig. 5 and Fig. 6 shows the resource utilization over the duration of one sample simulation. In these figures, we can observe the changes in resource allocation over each individual time step. The FCFS and Randomized algorithms are omitted as to not clutter the figures. We can see that while Algorithm 1 and Algorithm 2 perform better than their SJF counterparts on average, they may perform worse during specific time steps.

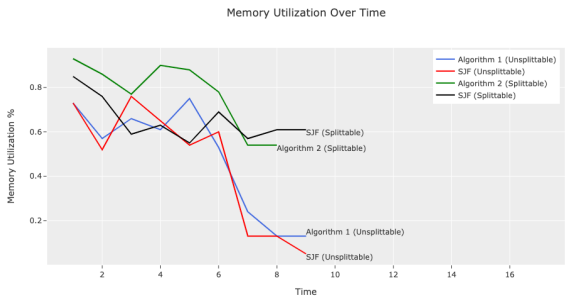

Fig. 6. Memory Utilization Over Time

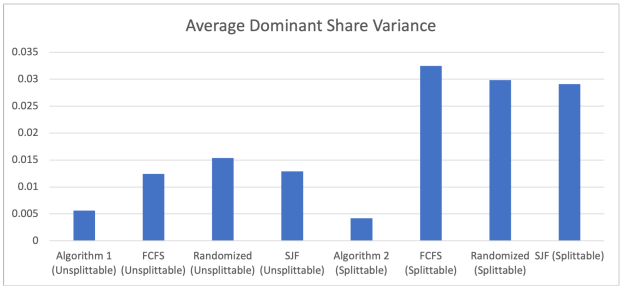

Fig. 7. Average Dominant Share Variance

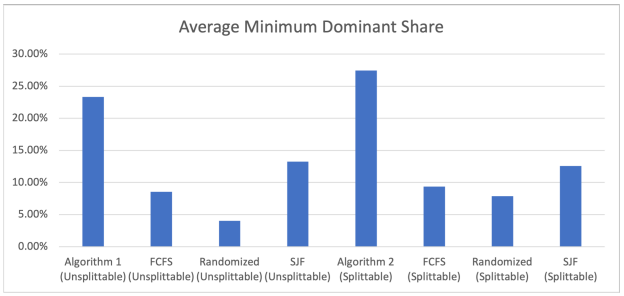

Fig. 8. Average Minimum Dominant Share

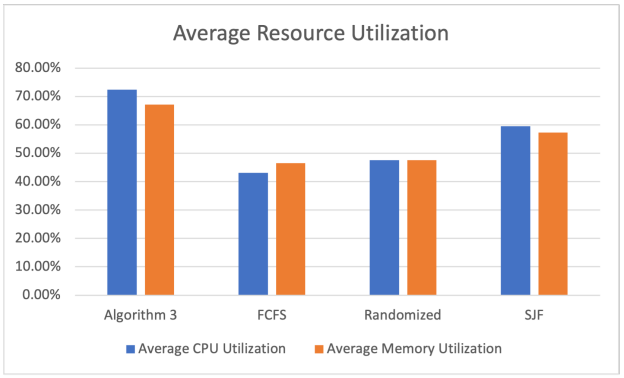

Fig. 9. Average Resource Utilization

This is due to the nature of online environments where users arrive continuously. Then, since Algorithm 1 and Algorithm 2 complete tasks faster than their SJF counterparts, they may have more downtime between the arrival of users. This would explain why sometimes the SJF algorithms have higher resource utilization but lower completion times overall.

2) Fairness

The second evaluation will focus on the main objective of this paper, fairness between users. Recall that the DRF scheme aims to maximize the minimum global dominant share across all users. Then, to evaluate this objective, we use two metrics: average variance of all global dominant shares, and average minimum dominant share. Fig. 7 and Fig. 8 summarize our results. Fig. 7 displays the variance of the global dominant shares of all users at every time step averaged throughout the simulations. Lower variance typically correlates to higher minimum global dominant share. Fig. 8 displays the minimum global dominant share at every time step averaged throughout the simulations. The higher the minimum global dominant share the better as per the DRF scheme. Between the two proposed algorithms, Algorithm 2 performs better than the nonsplittable cases in terms of the two metrics. This is due to the splittable-ness of its tasks allows it to have more freedom when allocating resources to users. It also performs significantly better than SFJ Splittable. We can see the two figures show similar results where Algorithm 1 and Algorithm 2 have better results than the other algorithms. This is to be expected as the other algorithms do not consider global dominant shares at all.

B. Algorithm 3

To evaluate the performance of the Algorithm 3, the same 3 baseline algorithms will be used: FCFS, Randomized, and SJF. However, since none of the three algorithms are meant for splittable tasks, they will allocate a random integer multiple of the required amount of resources per task chain. They will also pick servers using the best fit characterization. The simulations were run for 20 times for more generalized results since the input data are randomly selected.

1) Resource Utilization

Our first evaluation, once again, focuses on how well the algorithms utilize the available resources in the scheduling system. Fig. 9 shows the average resource utilization percentage of all simulated algorithms. Results show that the proposed Algorithm 3 outperforms all of the other algorithms by a significant margin. Consequently, Fig. 10 shows a similar result in completion times. In both metrics, SJF performs better than FCFS and Randomized but not as good as Algorithm 3.

Fig. 11 and Fig. 12 shows the resource utilization over the duration of one sample simulation. In these graphs, once again, we see that the baseline algorithms have higher resource allocations at specific time steps compared to the proposed Algorithm 3. Algorithm 3 executes users' task faster and the users will leave the system faster which results in relative downtime between the arrival of users where the available resources aren't being used.

2) Fairness

The second evaluation will focus on fairness between users. To evaluate this objective, we will once again use two metrics: average variance of all global dominant shares, and average

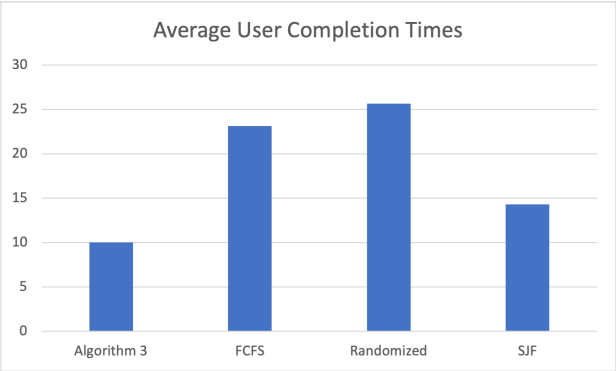

Fig. 10. Average Completion Time

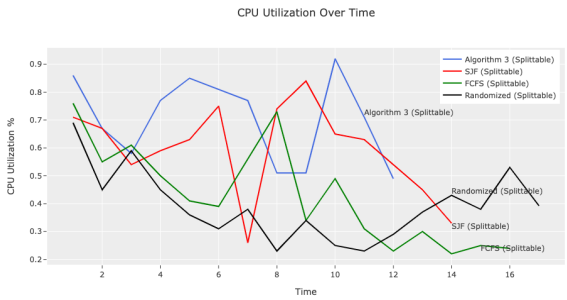

Fig. 11. CPU Utilization Over Time

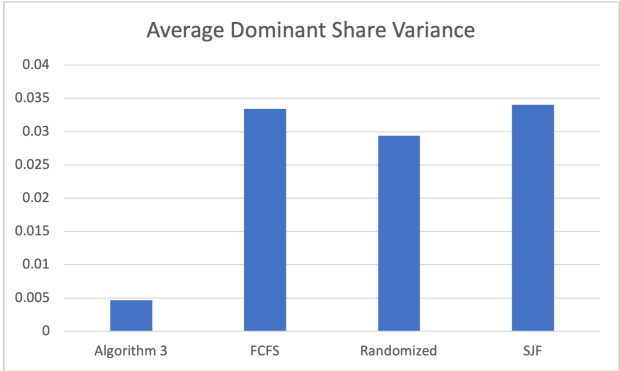

Fig. 13. Average Dominant Share Variance

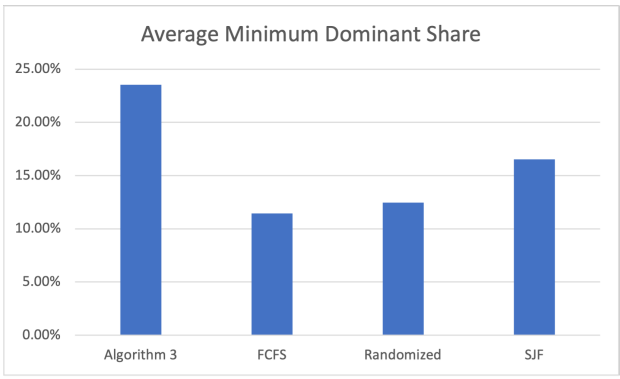

Fig. 14. Average Minimum Dominant Share

minimum dominant share. Fig. 13 and Fig. 14 summarize our results. Fig. 7 displays the variance of the global dominant shares of all users at every time step averaged throughout the simulations. Fig. 8 displays the minimum global dominant share at every time step averaged throughout the simulations. We see that Algorithm 3 performs the best out of all simulated algorithms. However, Algorithm 3 has extremely low global dominant share variance but this does not translate to average minimum dominant share as Algorithm 3 only slightly beats out SJF in average minimum dominant share. This is due to the nature of the ordered environment. Since each user's tasks are ordered, they cannot be completed in parallel and will hence stay in the system for longer. Then, there will be more users executing tasks in the system at any given time.

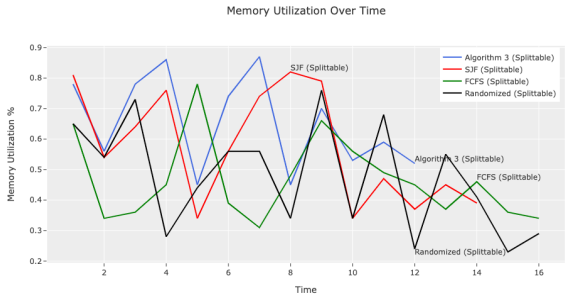

Fig. 12. Memory Utilization Over Time

VII. Conclusion

This paper examines the online resource allocation problem within fog computing under an environment that is multi-server, multi-resource, and includes heterogeneous tasks. This problem is approached from a fairness perspective where the DRF scheme is used to formulate a fairness maximizing optimization problem. Four different types of tasks are considered: ordered/unordered, splittable/unsplittable. Three low complexity heuristics are proposed and are evaluated against three baseline algorithms. Results show that the proposed algorithms perform similar to better in terms of task completion speed but significantly better in terms of user fairness.

A key limitation in the algorithms proposed is that they are not very generalizable. Realistically, not all requested tasks will be the same in terms of ordered/unordered and splittable/unsplittable. In the future algorithm that is able to handle an environment where there are tasks with a variety of characteristics. To do so, machine learning techniques would likely have to be used such as deep reinforcement learning.

However, currently, machine learning is used to produce highly specific models that are only equipped to handle some highly specific instances of an optimization problem. We are still quite far from being able to produce a highly generalizable machine learning model that can be used to solve several, or perhaps an entire class of, optimizations. Nevertheless, it is without a doubt that machine learning is emerging as a promising approach to solving optimization problems.

REFERENCES

[1] A. Markus, and A. Kertesz. "A survey and taxonomy of simulation environments modelling fog computing." *Simulation Modelling Practice and Theory 101* 2020.

[2] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia. "Fog computing: a comprehensive architectural survey." *IEEE Access 8* 2020, 69105-69133.

[3] P. Hosseinioun, M. Kheirabadi, S. R. K. Tabbakh, and R. Ghaemi. "aTask scheduling approaches in fog computing: a survey." *Transactions on Emerging Telecommunications Technologies* 2020, e3792.

[4] I. Sabuncuoglu, and M. Bayız. "Analysis of reactive scheduling problems in a job shop environment." E*uropean Journal of operational research 126.3* 2000, 567-586.

[5] D. Ouelhadj, and S. Petrovic. "A survey of dynamic scheduling in manufacturing systems." *Journal of scheduling 12.4* 2009, 417-431.

[6] Z. Wang, J. Zhang, and S. Yang. "An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals." *Swarm and Evolutionary Computation 51* 2019, 100594.

[7] 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)* 2019.

[8] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. "Quincy: fair scheduling for distributed computing clusters." *In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* 2009, 261-276.

[9] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling." *In Proceedings of the 5th European conference on Computer systems* 2010, 265-278.

[10] G. Zhang, F. Shen, Y. Yang, H. Qian, and W. Yao. "Fair task offloading among fog nodes in fog computing networks." *In 2018 IEEE international conference on communications (ICC)* 2018, 1-6.

[11] M. Mukherjee, M. Guo, J. Lloret, R. Iqbal, and Q. Zhang. "Deadline-aware fair scheduling for offloaded tasks in fog computing with inter-fog dependency." *IEEE Communications Letters 24* 2019, 307-311.

[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." *In Nsdi, vol. 11* 2011, 24-38.

[13] W. Wang, B. Li, and B. Liang. "Dominant resource fairness in cloud computing systems with heterogeneous servers." *In IEEE INFOCOM 2014-IEEE Conference on Computer Communications* 2014, 583-591.

[14] A. Östman, "Distributed Dominant Resource Fairness using Gradient Overlay." *In unpublished Master's thesis.* KTH Royal Institute of Technology 2017.

[15] D.C. Parkes., A. D. Procaccia, and N. Shah. "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities." *ACM Transactions on Economics and Computation (TEAC)* 2015, 1-22.