DATA: Throughput And Deadline-Aware Genetic Approach for Task Scheduling in Fog Networks

Arya Motamedhashemi, Bardia Safaei, Amir Mahdi Hosseini Monazzah, and Alireza Ejlali

***Abstract*—Fog devices in fog computing frameworks are responsible for fetching and executing the tasks submitted by the deployed resource-constraint embedded edge devices. Based on the availability of resources, tasks are offloaded to the virtual machines hosted by the fog devices. These tasks may then get scheduled to guarantee a number of efficiency-related metrics. While throughput has a decisive impact on the timely execution of tasks, the appropriate utilization of this metric has not been considered in the existing mechanisms. In this letter, we first discuss the proper use of this objective in the fitness function of meta-heuristic algorithms. Then, we explain that adopting throughput by the fitness functions in the form of two conventionally used weighted-sum, and fractional techniques may ignore solutions with a better guarantee ratio. Consequently, we propose a novel approach called DATA to be replaced with these two old approaches. DATA is a throughput, and deadline-aware task scheduling mechanism for time-sensitive fog frameworks, which its fitness function utilizes genetic optimization by encoding the solutions into chromosomes. It uses single gene mutation and two-point crossover. In this approach, two populations are considered to search the problem space. The main population is evaluated based on the guarantee ratio, while the helper population is evaluated based on the throughput. Furthermore, the helper population uses weighted-sum. The initial population is generated randomly by the uniform distribution, to provide a load-balancing. Based on our extensive evaluations, the selected solution by DATA provides the highest guarantee ratio, while having the lowest possible makespan.**

***Index Terms*—Fog Computing, Task Scheduling, Offloading, Throughput, Genetic Algorithm, Weighted-Sum, Fractional.**

I. INTRODUCTION

FOG computing frameworks are composed of embedded edge devices, fog nodes, and ultimately cloud servers. In contrast with the cloud, while edge and fog devices have computationally limited resources, they provide lower End-to-End *(E2E)* latency [1]. Due to the existing resource constraints, the embedded edge devices are obliged to offload and schedule their tasks to the upper layers of the tiered network for execution. In fog computing, task scheduling is referred to as the mapping of tasks to the deployed Virtual Machine *(VM)* hosted by the fog devices, and cloud data centers. According to Fig. 1, this mapping takes place on a broker node, whose primary duty is to receive a job, decompose it into a set of tasks, and assign them to the VMs, according to the availability of their resources. Meanwhile, finding the optimal assignment in terms of a single or more metrics is an NP-Hard problem. Optimization techniques are used to find the optimal or near-optimal solution. Two approaches could be broadly seen in the previous assignment strategies: 1) Heuristic, and 2) Meta-Heuristic. Meta-heuristic algorithms are mostly used in dynamic environments, where we need to explore the search space efficiently to find the optimal or near-optimal solution for the intended problem. Some of the most commonly used optimization approaches include 1) Ant-Colony, 2) Particle Swarm, 3) Genetic Algorithm, and 4) Simulated Annealing.

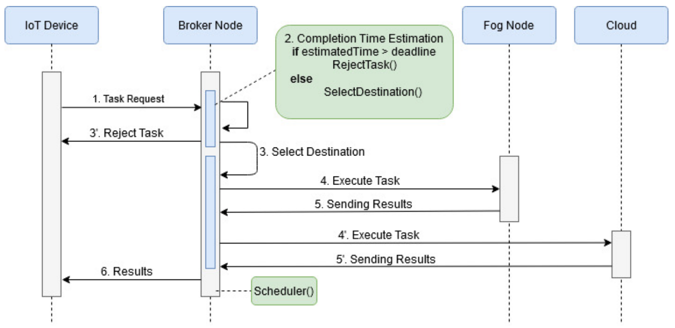


Figure 1: Task scheduling in edge/fog computing frameworks.

In time-sensitive fog applications, while throughput has a decisive impact on the guarantee ratio and the timely execution of tasks, the appropriate utilization of this metric has not been considered in the literature. Furthermore, all of the recent studies convert their objective functions into a single scalar value either with fractional, or weighted-sum techniques. In this letter, we show that using the aforementioned methods discards solutions with the highest guarantee ratio. Accordingly, we propose DATA; a throughput, and deadline-aware scheduling mechanism to be replaced with these old approaches. The main goal of DATA is to execute as many tasks as possible in a shorter makespan (by considering throughput). Its fitness function utilizes genetic optimization by encoding the solutions into chromosomes. It uses single gene mutation and two-point crossover. In this approach, two populations are considered to search the problem space. The main population is evaluated based on the guarantee ratio, while the helper population is evaluated based on the throughput, without adding complexity to the algorithm. In this novel technique, the helper population itself uses the weighted-sum method. The initial population is generated randomly by the uniform distribution, to provide a load-balancing.

Based on an extensive set of experiments conducted in the iFogSim simulator [2], the selected solution by DATA has provided the highest guarantee ratio while having the lowest possible makespan. As a result, while the makespan is kept relatively identical to fractional, and weighted-sum, DATA improves the throughput in fog networks by up to 8%.

Arya Motamedhashemi, Bardia Safaei, and Alireza Ejlali are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. e-mail: {motamedhashemi,bardiasafaei,ejlali}@sharif.edu.

Amir Mahdi Hosseini Monazzah is with the school of computer engineering, Iran University of Science and Technology (IUST), Tehran, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, e-mail: monazzah@iust.ac.ir.

II. System Model and Formulation

In this letter, let J represent our job or union of m tasks ($Task_i, i \in \{1, 2, ..., m\}$) for execution. These tasks are assumed to have soft deadlines (d_i), independent and non-preemptive. The total completion time of these tasks consists of three segments, including the transmission time for offloading the tasks, their processing time on a VM hosted by a fog node f_j, and the transmission of their results from a fog device f_j to the user device [3]. The uplink transmission of $Task_i$ with a size $TSize_i$ to a fog node is obtained by (1).

$$T_{i,j}^{uplink} = \frac{TSize_i}{R_{i,j}} \tag{1}$$

where $R_{i,j}$ is an indication for the wireless transmission rate. Since the provided results may not have the same size as the submitted $Task_i$, we assume $RSize_i$ to be the size of the task execution result in bits returned by a fog node. Accordingly, the downlink transmission time would be obtained by (2).

$$T_{i,j}^{downlink} = \frac{RSize_i}{R_{i,j}} \tag{2}$$

Depending on the length of the task, and the processing capability of fog node f_j in terms of MIPS (denoted with PC_j), the execution time of $Task_i$ on f_j is computed as (3).

$$T_{i,j}^{execution} = \frac{Length(Task_i)}{PC_j} \tag{3}$$

By adding these three values, the total completion time for a single task $Task_{i,j}^{Completion}$ could be calculated. By adding the completion time of all of the assigned tasks to fog node f_j, the completion time for the intended task sequence $T_j^{Completion}$ will be calculated. Finally, the makespan, which is the period between the instance when our task set J is submitted to the broker to the instance when the last task is completed could be obtained by (4) [4].

$$M_J = Max\{T_j^{Completion}\}, j \in \{1, 2, ..., n\} \tag{4}$$

In real-time fog applications, in addition to makespan, two other performance metrics must be also taken into consideration: 1) guarantee ratio, and 2) throughput. According to (5), guarantee ratio of task set J is obtained by dividing the number of completed tasks within their deadlines (d_i) by the total number of tasks (N_{total}) [5].

$$GuaranteeRatio_{Job_i} = \frac{N_{completed}}{N_{total}} \tag{5}$$

On the other hand, throughput indicates the number of tasks that are completed within their deadlines (d_i) in a unit of time.

$$Throughput = \frac{N_{completed}}{M_J} \tag{6}$$

III. Weakness of Fractional and Weighted-Sum

The main challenge in real-time applications is to achieve the highest guarantee ratio possible in less time period. In order to do so, we can conclude that it is necessary to deal with the throughput. Fig. 2, shows a number of possible solutions in the search space when using throughput as the fitness function. Solutions are named Si and are depicted with lines. Accordingly, the slope of each line indicates the value of its fitness function. The important issue, which could be concluded from this figure is that the slope alone cannot be accounted for the best solution. As Fig. 2 states, solution $S1$ has the greatest slope while solutions $S2$ and $S3$ have done more tasks. This means that if we conduct the selection of the best solution based on having the greatest throughput, we might end up missing better solutions such as $S2$ and $S3$. This challenge applies to both, fractional, and weighted-sum approaches if used for the task scheduling fitness function.

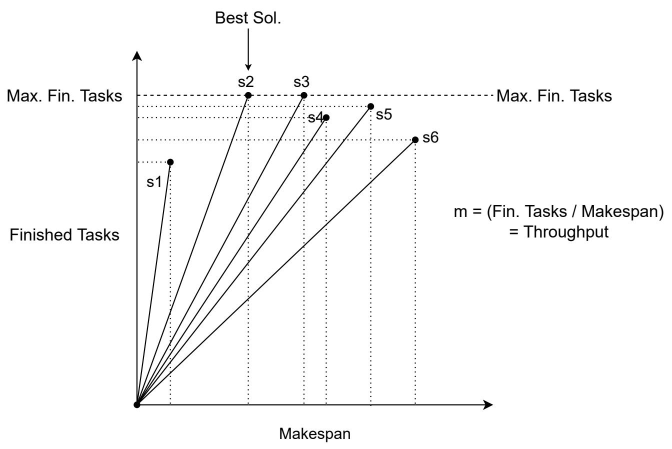


Figure 2: Possible solutions of the task scheduling problem.

In recent studies, most of the fitness functions are weighted-sum [6], and only a few of them use fractional fitness functions [3]. Nevertheless, we enumerate the cases that using either the weighted-sum or fractional fitness functions is not compatible with throughput. For better understanding, assume that G represents guarantee ratio, M indicates makespan, M_n indicates normalized makespan to the N_{total}, and T stands for throughput. In the case of fractional, we have $T = G/M_n$. Accordingly, the four following cases are possible:

1) $T_1 > T_2$; $G_1 > G_2$ and $M_1 = M_2$
2) $T_1 > T_2$; $G_1 = G_2$ and $M_1 < M_2$
3) $T_1 > T_2$; $G_1 > G_2$ and $M_1 > M_2$ (if happens)
4) $T_1 > T_2$; $G_1 < G_2$ and $M_1 < M_2$ (if happens)

According to $T = G/M_n$, the fractional approach is simply the slope of each line representing the solutions, which is actually the throughput. In this situation, the solution with a higher slope (e.g., $S1$) takes precedence over a solution with a better G (e.g., $S2$), which is incorrect. Therefore, by using the fractional approach, having a higher throughput, is not an indication for objective satisfaction. On the other hand, in the case of weighted-sum, since the guarantee ratio is already normalized, we only need to normalize makespan by using M_i/M_{max} or M_{min}/M_i. A weighted-sum fitness function that uses guarantee ratio and makespan as its objectives is defined as $(\alpha \times G) - ((1-\alpha) \times M_n)$. Accordingly, there are situations, where we may choose a solution with a lower guarantee ratio over a solution with higher guarantee ratio, because it achieved a lower makespan. As a result, the weighted-sum approach may incorrectly choose $S1$ over $S2$ due to the fact that the weighted-sum value of $S1$ may become higher than $S2$.

Therefore, we need to find solutions that provide the maximum G and have the highest slope possible. In the case of real-time tasks, where we have more than 2 objectives, one possible solution is to use a Genetic-based approach to be replaced with fractional or weighted-sum. In this approach, a main population based on the guarantee ratio, and a helper

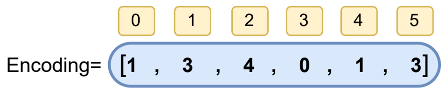


Figure 3: Chromosome encoding in assigning tasks to VMs.

population based on the throughput can be considered, where the helper population employs weighted-sum. This is the reason why DATA has been proposed in this letter.

IV. DATA Scheduling Mechanism

The proposed scheduling mechanism in this letter employs the genetic algorithm to optimize the objective function. This optimization technique assumes an initial population or particles, which represent the solution to the problem. These solutions are first created randomly in a uniform manner. In our problem, each chromosome represents a solution, which is encoded as Fig. 3. It is simply an array that has the same length as the tasks list. Each index corresponds to a task in the task list with the same index (top), and the value of the array at each index represents the VM id (bottom).

In the structure of the genetic algorithm, there are two probabilistic operations, known as mutation and crossover. In mutation, the value of a single gene (a VM that is assigned to a task) will change to a random value. In the crossover, two parents (two solutions, or chromosomes) exchange parts (some of their mapping) and form a new offspring (a new solution or mapping). At the end of each iteration, the fittest chromosomes survive and a portion of the new population is selected for the next iteration (natural selection). Mutation can lead to diversity and crossover makes the population have fit and similar chromosomes. The mutation and crossover rates must be determined before the algorithm gets started. We assume the initial population is randomly created from a uniform distribution and the VM capacities do not differ significantly. Therefore, we can also assume that we have a considerable degree of load-balancing. Therefore, all we need to focus is to optimize the throughput to satisfy the highest guarantee ratio possible within less makespan. Therefore, our observation must be focused on the guarantee ratio. We keep a helper population that is evaluated based on throughput, and a main population that is evaluated based on guarantee ratio. At the end of the algorithm, a population is reported (if greater than one). According to Algorithm 1, DATA compares the chromosomes of the two population lists. In the main list, we have the highest guarantee ratios and in the helper list, we have the highest throughputs. If the guarantee ratio of the top chromosome from the main list is greater than the helper list, then we compare this chromosome with the elements of its list to see whether we can find another chromosome with the same guarantee ratio and less makespan. If found, the new chromosome is the best solution since it has the highest guarantee ratio and less makespan. If not, the top chromosome is selected as the best answer. Otherwise, the top chromosome of the helper list is selected as the best answer, since its guarantee ratio is greater or equal than the guarantee ratio of the main list while having lower makespan.

Algorithm 1: The Optimization Approach of DATA

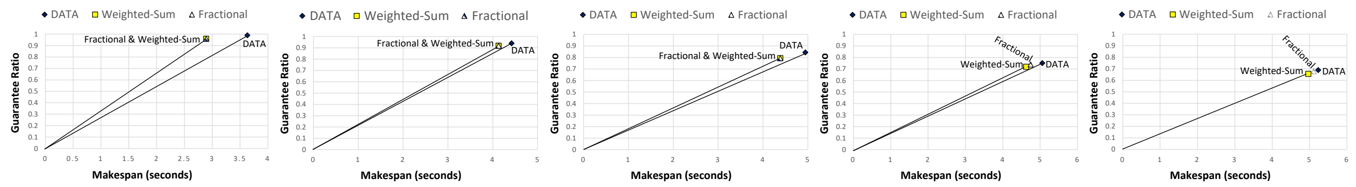
```
Data: $WeightedSum\ \alpha = 0.8$, $maxIteration = 50$,
      $mutationRate = 0.2$, $CrossOverRate = 0.7$,
      $initialPopulation = 10$.
1  mainPopulation = initialPopulation.clone();
2  helperPopulation = mainPopulation();
3  while iteration < maxIteration do
4      for each chromosome in mainPopulation do
5          if random() < mutationRate then
6              chromosome.mutate(randomIndex);
7          if random() < crossoverRate then
8              offspring = chromo-
                 some.crossover(nextRandomChromosome,
                 randomIndexStart, randomIndexEnd);
9              offspring.evaluate();
10             mainPopulation.add(offspring);
11         chromosome.evaluate();
12     for each chromosome in helperPopulation do
13         if random() < mutationRate then
14             chromosome.mutate(randomIndex);
15         if random() < crossoverRate then
16             offspring = chromo-
                 some.crossover(nextRandomChromosome,
                 randomIndexStart, randomIndexEnd);
17             offspring.evaluate();
18             helperPopulation.add(offspring);
19         chromosome.evaluate();
20     mainPopulation.SortBy(Fitness.guarantee);
21     mainPopulation = mainPopulation.slice(0,
         length×naturalSelectionPortion)
22     helperPopulation.SortBy(Fitness.throughput);
23     helperPopulation = helperPopulation.slice(0,
         length×naturalSelectionPortion)
24     iteration++;
25 if mainPopulation[0].guarantee >
     helperPopulation[0].guarantee then
26     SelectMinMakespanWithSameGuarantee(mainPopulation,
         mainPopulation[0].guarantee);
27 if mainPopulation[0].guarantee <=
     helperPopulation[0].guarantee then
28     Select helperPopulation[0];
```

V. Experimental Evaluation and Result Discussion

In order to conduct our simulations, we have exploited the iFogSim simulation environment [2]. The considered system consists of a cloud data center and one host. A total number of 5 VMs are allocated to the host of the data center. The VMs allocation policy is space-shared [7]. Furthermore, the VM execution policy on a host and the cloudlet scheduling policy is FCFS. Evaluations consist of five sets of tasks ranging from 100 to 500 million instructions and their deadlines are uniformly distributed in the interval of [1.5, 5.5] seconds.

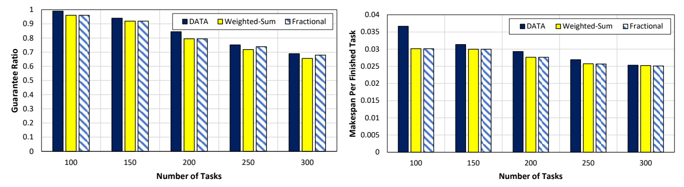
First, we compare the best solution provided by DATA, the weighted-sum approach, and the fractional approach. The selected best solution by every one of these approaches has been depicted in Fig. 4. Accordingly, DATA's selected solution has achieved a higher guarantee ratio (y-axis) compared to the other two approaches, despite the slope of the line for the weighted-sum and the fractional approaches being greater than that of DATA. This has been obtained in different scenarios with different numbers of cloudlets.

Regarding the makespan, we expect that by increasing the number of executed tasks, the makespan will also increase. However, DATA's second objective was to make sure that this increase in the makespan is controlled by considering



(a) 100 Tasks (b) 150 Tasks (c) 200 Tasks (d) 250 Tasks (e) 300 Tasks

Figure 4: The fittest solution selected by each approach in case of having a different number of tasks.



(a) Analysis of guarantee ratio. (b) Execution time per finished task.

Figure 5: Performance analysis of DATA.

the throughput as its helper population's evaluation function. Accordingly, in DATA, we are not trying to build an archive of non-dominant solutions and compare the solutions in the main population to get our results. This approach may be quite complex and time-consuming in terms of algorithm complexity. However, in DATA we simply handle two populations without adding more complexity.

Fig. 5(a), explicitly compares the guarantee ratio of DATA with other approaches. We can observe that the weighted-sum approach has missed some of the chromosomes, which had a better guarantee ratio, because they had a higher makespan, and were not fit for the population. The fractional approach has also missed solutions that had a higher guarantee ratio and selected those solutions that only provide higher throughput (cases 3 or 4 in Section III). According to Fig. 5(a), as the number of tasks grows, the gap between the guarantee ratios is minimized. This is due to the fact that the deadlines are absolute and are randomly chosen for the tasks within the intervals. Therefore, by increasing the number of tasks, the miss-ratio grows, resulting in having close guarantee ratios. According to the considered system model in this letter, if in any solution, at least a task of a job cannot be executed before its deadline (independent of how high the guarantee ratio is), that job will not be completed. Nevertheless, DATA has improved the execution of real-time tasks from this perspective. Fig. 5(b), compares the average execution time per each finished task. In the case of 100 tasks, we observe a higher makespan per finished task in DATA. This is due to executing more tasks against the other approaches. As stated earlier, executing more tasks can increase the makespan. This trend also applies to other task numbers, but the difference gets smaller by submitting more tasks. In summary, DATA selects a solution with the highest guarantee ratio with the lowest possible makespan, especially in cases where there are more tasks submitted in the system.

A. Complementary Discussion Regarding Throughput

Based on the application, throughput could be defined in two ways: 1) Number of executed tasks within a fixed period, 2) Number of executed tasks finished within the makespan (not fixed). In the first case, the only metric affecting the evaluation would be the guarantee ratio. Thus, having a high throughput is equivalent to having a high guarantee ratio. In the second case, three scenarios are possible:

1) Makespan remains constant and throughput increases
2) Makespan decreases and guarantee ratio remains constant and throughput increases
3) Both Makespan, and guarantee ratio increase in a way that results in higher throughput

In the first scenario, we will be able to schedule and execute more tasks within the same makespan, which is beneficial; since more tasks are executed. In the second scenario, we would be able to schedule and execute the same number of tasks, but within a lower makespan, which is also beneficial; since the makespan is decreased. In the third scenario, we will be able to schedule, and execute more tasks; but executing more tasks could increase the makespan. Accordingly, only in situations, where we have high utilization or load balancing, we could be sure that an increase in makespan results in higher throughput. In summary, improving the throughput in real-time applications could affect both the guarantee ratio and makespan simultaneously. This is why DATA is a valuable scheduling mechanism for real-time applications.

VI. Conclusion

In this letter, we have shown that employing the well-known weighted-sum, and fractional approaches in the fitness function of task scheduling policies, consisting of guarantee ratio, and makespan, may not always give the best solution. Accordingly, DATA has been proposed, which is a genetic-based algorithm that utilizes a helper population to store chromosomes with the highest throughput, and a main population that contains chromosomes with the highest guarantee ratio. Evaluations have shown that DATA selects a solution with the highest guarantee ratio and the lowest possible makespan.

References

[1] R. Siyadatzadeh *et al.*, "Relief: A reinforcement learning-based real-time task assignment strategy in emerging fault-tolerant fog computing," *IEEE Internet of Things Journal*, 2023.
[2] H. Gupta *et al.*, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[3] S. Ghanavati *et al.*, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Transactions on Services Computing*, vol. 15, no. 4, pp. 2007–2017, 2020.
[4] M. Abdel-Basset *et al.*, "Energy-aware marine predators algorithm for task scheduling in iot-based fog computing applications," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 5068–5076, 2020.
[5] J. Fan *et al.*, "Deadline-aware task scheduling in a tiered iot infrastructure," in *Proceedings of the IEEE Global Communications Conference (GOLBECOM)*. IEEE, 2017, pp. 1–7.
[6] M. K. Hussein *et al.*, "Efficient task offloading for iot-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37 191–37 201, 2020.
[7] R. Buyya *et al.*, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proceedings of the International Conference on High Performance Computing & Simulation*. IEEE, 2009, pp. 1–11.