## THE DETAILS OF THE OFFLOADING AND SCHEDULING ALGORITHMS

In this section, we introduce the details of the task offloading and scheduling algorithms supported in EdgeWorkflow currently. In the edge computing environment, given the three different layers of resources, viz. end device layer, Edge server layer and cloud server layer, a task offloading strategy is required first to determine which level or levels of resources will be used for task computation, before the task scheduling algorithm determines the order of tasks running on the allocated resources.

### 1 TASK OFFLOADING ALGORITHM

The strategies of task offloading in the edge computing environment can be generally divided into three types, viz. local computation, full offloading and partial offloading, as shown in Fig. 15 [1]. In local computation, all tasks are executed on user's end device. This strategy is suitable for small task load or when other edge resources are inaccessible. In full offloading, all tasks are offloaded wholly to either edge servers and/or cloud servers. This strategy is not suitable when the size of data transmitted among resources is excessive. In partial offloading strategy, tasks can be offloaded partially to the edge servers and/or the cloud servers. This strategy is useful for tasks with complicated computation. For example, the end device can do simple data pre-processing and then offload the complicated computation tasks with pre-processed data to the edge server or the cloud servers. It has been reported that the overall computation time for the complicated task can be reduced [2]. Given the complexity of many workflow tasks, EdgeWorkflow adopts the partial offloading strategy.

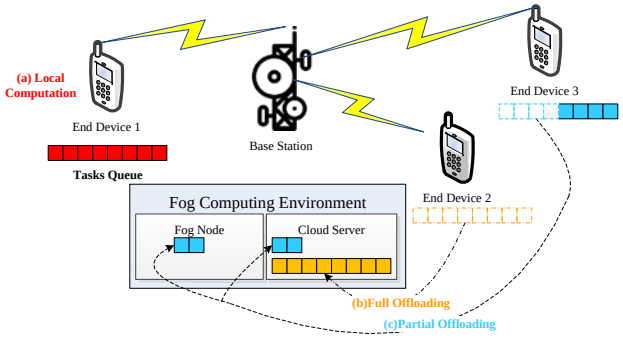


Fig. 15. Task offloading in Edge Computing

In EdgeWorkflow, the partial offloading strategy can generate specific task offloading decisions according to the main performance metric used such as time, energy and cost under the deadline constraint. Therefore, the offloading strategy can be divided into three different types. Firstly, when the main performance metric is execution time, the strategy of task offloading is shown in Formula 10. The task will not be offloaded if the task execution time on the end device is shorter than the time for offloading it to the Edge server or Cloud server for execution. Otherwise, the task will be offloaded.

$$D_i^{\text{time}} = \begin{cases} 0 & t_i^e < t_i^o \\ 1 & t_i^e > t_i^o \end{cases} \tag{10}$$

where $D_i^{\text{time}}$ is the offloading decision given the performance metric of time, $t_i^e$ is the execution time at the local end device for task $i$, and $t_i^o$ is the time for offloading task $i$ to the edge server or cloud server.

Secondly, when the main performance metric is energy consumption, the strategy of task offloading is shown in Formula 11. Under the deadline constraint, the task will not be offloaded if the energy consumption for executing the task on the end device is smaller than the energy consumption for offloading it to the edge server or cloud server for execution. Otherwise, the task will be offloaded.

$$D_i^{energy} = \begin{cases} 0 & E_i^{\text{end}} < E_i^{\text{offload}} \\ 1 & E_i^{\text{end}} > E_i^{\text{offload}} \end{cases} \quad \text{s.t.} \quad T_{\text{makespan}} < deadline \tag{11}$$

where $D_i^{\text{energy}}$ is the offloading decision given the performance metric of energy consumption, $E_i^{\text{end}}$ is the energy consumption at the local end device for task $i$, and $E_i^{\text{offload}}$ is the energy consumption for offloading task $i$ to the edge server or cloud server.

At last, when the main performance metric is task execution cost, the strategy of task offloading is shown in Formula 12. Under the deadline constraint, the task will not be offloaded if the cost for executing the task on the end device is smaller than the cost for offloading it to the edge server or cloud server for execution. Otherwise, the task will be offloaded.

$$D_i^{\text{cost}} = \begin{cases} 0 & C_i^{\text{end}} < C_i^{\text{offload}} \\ 1 & C_i^{\text{end}} > C_i^{\text{offload}} \end{cases} \quad \text{s.t.} \quad T_{\text{makespan}} < deadline \tag{12}$$

where $D_i^{\text{cost}}$ is the offloading decision given the performance metric of task execution cost, $C_i^{\text{end}}$ is the execution cost at the local end device for task $i$, and $C_i^{\text{offload}}$ is the execution cost for offloading task $i$ to the edge server or cloud server.

## 2 TASK SCHEDULING ALGORITHM

The current algorithm library of EdgeWorkflow includes six task scheduling algorithms: MinMin, MaxMin, FCFS, RoundRobin, PSO and GA [3]. Among them, MinMin, MaxMin, FCFS and RoundRobin algorithms are representative heuristics based algorithms, while PSO and GA are widely used metaheuristics based algorithms [3]. Here are the brief introductions to the six algorithms.

① MinMin Heuristic

The main idea of MinMin algorithm is selecting the minimum unassigned task and assigning it to the best processor, where the minimum task refers to the task with minimum completion time and the best processor can complete that task at the earliest time [4]. This algorithm proceeds in several iterations. At each iteration, a task-to-processor assignment is decided based on a two-step procedure. In the first step, MinMin computes the completion time of each unassigned task over

each processor to find the earliest completion time as well as the best processor which completes the processing of that task at the earliest time. In the second step, the task with the minimum completion time is selected from all unassigned tasks then assigned to its best processor found in the first step. After that, the selected task is removed from further consideration in the remaining iterations. The iterations will continue until all tasks are assigned. The advantage of MinMin heuristic is to ensure the makespan of all tasks stay minimum, but the disadvantage is that the processors with higher capacity are always in working state, while other processors keep idle. In the Edge Computing environment, Cloud servers have higher processing capacity than any other processors. Therefore, MinMin algorithm will assign most of the tasks to Cloud servers, which hardly takes the advantage of Edge Computing.

② MaxMin Heuristic

MaxMin differs from MinMin in the task selection strategy adopted in the second step of the task-to-processor assignment procedure, while the first step of MaxMin is similar to MinMin [5]. Unlike MinMin, which selects the task with the minimum completion time, MaxMin selects the task with the maximum completion time and then assigns it to the best processor found in the first step. Then the selected task is removed from further consideration in the remaining iterations. The iterations will continue until all tasks are assigned. The advantage of MaxMin algorithm is to ensure the assignment of tasks with maximum completion time in earlier iterations, but it will keep the tasks with minimum completion time waiting. Similar to MinMin, MaxMin will assign most of the tasks to Cloud servers, which also hardly takes the advantage of Edge Computing.

③ FCFS Heuristic

FCFS (First Come First Served) heuristic provisions VMs simply in the order of the tasks' arrival, without considering any other factors [6]. The advantages of FCFS heuristic are simple, easy to understand and implement, but the obvious shortcomings are that the average execution time of each task will be increased when small tasks come later than large ones. In such a case, the makespan of the batch of tasks will be increased significantly.

④ RoundRobin Heuristic

RoundRobin is widely used where time slices are assigned to each process in equal portions and in a circular order [7]. The basic idea of this heuristic is that all tasks are scheduled in a ready queue according to the FCFS policy, then the first task in this queue will get a quantum (its allowance of CPU time), and the task will be interrupted if it is not completed by the end of its quantum, after that this task will be sent to the tail of the ready queue waiting for its next quantum. The task is resumed next time when it becomes the first one in the queue. If the task terminates or changes its state to waiting during its allocated time quantum, the quantum will be delegated to other tasks. One of the advantages of RoundRobin heuristic is the fairness since each task will get the same quantum. However, inappropriate size of time quantum will reduce the performance and efficiency of the system. For example, if the time quantum is so big that every single task will be completed within a quantum, then the RoundRobin will deteriorate to FCFS. On the contrary, if the time quantum is too small, each task will require more quanta which will cause increased frequency of task switching, so that the overhead of system management will be increased as well as the CPU efficiency will be reduced.

⑤ PSO meta-heuristic

PSO (Particle Swarm Optimization) is an intelligent meta-heuristics based algorithm. PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality [8]. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions found by other particles. This is expected to move the swarm toward the best solutions. The particle's position and velocity are calculated by Formulas 13 and 14 as follows.

$$v_{i+1} = w \cdot v_i + c_1 \cdot r_1 \cdot (p_{best} - x_i) + c_2 \cdot r_2 \cdot (g_{best} - x_i) \tag{13}$$

$$x_{i+1} = v_i + x_i \tag{14}$$

where $v_i$ is the velocity of particle $i$, $x_i$ is the position of particle $i$, $w$ is the inertia weight of particle, $c_1$ is the optimal self-learning factor for particle individuals, $p_{best}$ is the best individual position for particle $i$, $c_2$ is the optimal self-learning factor for particle swarm, $g_{best}$ is the best swarm position, and $r_1$ and $r_2$ are random numbers.

The fitness value is used to evaluate the quality of solutions generated by PSO based scheduling algorithm [8, 9]. We design the fitness function to evaluate the performance metric under given deadlines. The smaller the fitness value is, the lower the energy consumption and task execution cost of the solution are. The fitness value can be calculated by Formula 15.

$$fitness = \left( f_1 \times PM_{\text{sum}} \right) + \left( f_2 \times 10 \times PM_{\text{sum}} \times \frac{T_{\text{sum}}}{deadline} \right) \tag{15}$$

The fitness function consists of two parts: the first part is the performance metric under given deadlines $\left( f_1 = 1, f_2 = 0 \right)$; the second part is the performance metric beyond deadlines $\left( f_1 = 0, f_2 = 1 \right)$. $PM_{\text{sum}}$ represents the performance metric (energy consumption or task execution cost) of the solution. $T_{\text{sum}}$ is the total execution time of the workflow. *deadline* is the user specified time constraint for the workflow.

⑥ GA meta-heuristic

GA (Genetic Algorithm) is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). GA is commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. GA has the advantages of simple process, randomness and expansibility, but its disadvantages are complex programming, poor local search ability, slow search speed and easy to premature convergence. The fitness function for evaluating the quality of solutions generated by GA based scheduling algorithm is similar to the one for PSO described above, hence omitted here.

[1] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, “Sustainable Task Offloading in UAV Networks via Multi-Agent Reinforcement Learning,” *IEEE Transactions on Vehicular Technology*, 2021.

[2] P. Mach, and Z. Becvar, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” *IEEE Communications Surveys & Tutorials*, vol. PP, no. 99, pp. 1-1, 2017.

[3] X. Liu, L. Fan, J. Xu, X. Li, L. Gong, J. Grundy, and Y. Yang, "FogWorkflowSim: An automated simulation toolkit for workflow performance evaluation in fog computing." pp. 1114-1117.

[4] B. G. Batista, N. B. Morais, B. T. Kuehne, R. M. Frinhani, M. Dionisio Filho, and M. L. Peixoto, "Heuristic Performance Evaluation for Load Balancing in Cloud." pp. 593-600.

[5] H. Alquhayz, and M. Jemmali, “Max-Min Processors Scheduling,” *Information Technology and Control*, vol. 50, no. 1, pp. 5-12, 2021.

[6] M. H. Bin Kamilin, M. A. Bin Ahmadon, and S. Yamaguchi, “Multi-Task Learning-Based Task Scheduling Switcher for a Resource-Constrained IoT System,” *Information*, vol. 12, no. 4, pp. 150, 2021.

[7] J. C. Guevara, and N. L. da Fonseca, “Task scheduling in cloud-fog computing systems,” *Peer-to-Peer Networking and Applications*, vol. 14, no. 2, pp. 962-977, 2021.

[8] J. Ding, S. Schulz, L. Shen, U. Buscher, and Z. Lü, “Energy aware scheduling in flexible flow shops with hybrid particle swarm optimization,” *Computers & Operations Research*, vol. 125, pp. 105088, 2021.

[9] D. C. Hop, N. Van Hop, and T. T. M. Anh, “Adaptive particle swarm optimization for integrated quay crane and yard truck scheduling problem,” *Computers & Industrial Engineering*, vol. 153, pp. 107075, 2021.