

Distributed and Application-aware Task Scheduling in Edge-clouds

Li Lin[†] Peng Li[‡] Jinbo Xiong[†] Mingwei Lin[†]

[†]College of Mathematics and Informatics, Fujian Normal University, China

[‡]School of Computer Science and Engineering, The University of Aizu, Japan

*Corresponding author: Li Lin (llfjz@163.com)

Abstract—Edge computing is an emerging technology which places computing at the edge of the network to provide an ultra-low latency. Computation offloading, a paradigm that migrates computing from mobile devices to remote servers, can now use the power of edge computing by offloading computation to cloudlets in edge-clouds. However, the task scheduling of computation offloading in edge-clouds faces a two-fold challenge. First, as cloudlets are geographically distributed, it is difficult for each cloudlet to perform load balancing without centralized control. Second, as tasks of computation offloading have a wide variety of types, to guarantee the user quality of experience (QoE) in terms of task types is challenging. In this paper, we present Petrel, a distributed and application-aware task scheduling framework for edge-clouds. Petrel implements a sample-based load balancing technology and further adopts adaptive scheduling policies according to task types. This application-aware scheduling not only provides QoE guarantee but also improves the overall scheduling performance. Trace-driven simulations show that Petrel achieves a significant improvement over existing scheduling strategies.

Index Terms—Edge computing, computation offloading, edge-clouds, task scheduling.

I. INTRODUCTION

Edge computing is an emerging technology [1], which performs data analytics and storage close to the data source (i.e., mobile devices) to reduce the network latency. This computing paradigm has attracted great attention from both academic and industry. As a typical use case, computation offloading [2], which migrates computing from mobile devices to the cloud, can now use the power of edge computing. The computation offloading over edge computing is also known as mobile edge computing [3]. By the deployment of edge-clouds, which are clusters of small servers (e.g., cloudlets [4]) nearby mobile devices, the end user can benefit from the ultra-low latency and perform computation offloading in edge-clouds.

However, given a large number of end users asking for computation offloading services, the task scheduling in edge-clouds is a two-fold challenge. First, cloudlets in edge-clouds are geographically distributed, and task requests from end users dispersed to cloudlets are naturally unbalanced. Unlike the cloud, which usually employs monolithic schedulers that have centralized scheduling policies and complete control over all resources, cloudlets schedule tasks alone and concurrently. Without the global view of the overall cloudlets in edge-clouds, it is difficult for each cloudlet to perform load balancing strategies.

Secondly, mobile applications have a wide variety of types, and it is challenging to guarantee the user quality of experience (QoE) in terms of task types. Generally speaking, people are sensitive to the response latency when offloading latency-sensitive applications (e.g., augmented reality); however, they are more tolerant to the delay with latency-tolerant applications (e.g., deep neural net) but pay attention to if these applications would complete within specific latency bounds. Then, it puts forward a problem that how to schedule tasks efficiently in terms of task types, which is unsolved in the task scheduling for edge-clouds.

To solve the above challenges, in this paper, we propose Petrel, a distributed and application-aware task scheduling framework for edge-clouds. Unlike the monolithic schedulers in the cloud, Petrel is a lightweight scheduler deployed on each cloudlet running automatically. Petrel implements a simple *sample-based* load balancing strategy, which employs the technology “the power of two choices” [5]. With this technology, for load balancing, Petrel randomly probes two cloudlets in edge-clouds and selects the cloudlet with the less load to place the task. This way has been proved to be effective under limited information [6] and can also reduce the scheduling overhead significantly.

Furthermore, Petrel implements an application-aware scheduling algorithm, which adapts different scheduling policies in terms of task types. Specifically, for latency-sensitive tasks, Petrel uses a “greedy” policy to find the cloudlet which has the minimum completion time for the tasks; whereas, for latency-tolerant tasks, Petrel adopts a policy of “best effort” scheduling. In the “best effort” scheduling, if there are idle resources on the cloudlet, Petrel then performs the task assignment; otherwise, it delays the task for a while but with the latency bound guarantee, which we call *delay scheduling*. The application-aware scheduling achieves a significant performance improvement which is substantiated in our analysis and experiments.

In this paper, our contributions can be summarized as follows:

- we propose Petrel, a distributed task scheduling framework for computation offloading in edge-clouds.
- Petrel uses a sample-based strategy for load balancing to reduce the scheduling overhead.
- Petrel implements an application-aware scheduling algorithm to improve the overall performance.

The rest of this paper is organized as follows. Section II reviews some background and related work about the task scheduling in edge-clouds. Section III presents an overview of the system architecture. Section IV builds the model of task scheduling and objective. Section V introduces the distributed and application-aware task scheduling algorithm. Section VI evaluates Petrel with other existing strategies based on trace-driven simulations, followed by conclusions in Section VII.

II. BACKGROUND AND RELATED WORK

In this section, we will review the background and related work.

A. Edge Computing and Computation Offloading

Edge computing is now a compelling paradigm which extends the cloud computing to the edge of the network. The origins of edge computing can be traced to the development of the Internet of Things (IoT) and 5G networks [3], in which sensor data are ingested and analyzed at the edge for the low latency. However, as edge computing has various advantages, it enables a new breed of services and applications. As a typical scenario, computation offloading over edge computing, also known as mobile edge computing (MEC), is an emerging computing paradigm that mobile devices migrate parts of their computation to the edge nodes in the vicinity. This paradigm offers a low end-to-end latency, which is critical for latency-sensitive applications.

To utilize the power of edge computing, a large number of edge nodes are required to be deployed in the proximity to end users. These edge nodes are heterogeneous and different in the form factors. However, the most notable paradigm of the edge node is “cloudlet” [4], a small data center nearby end users. Clusters of cloudlets, which are interconnected but without centralized nodes, become a kind of “edge-clouds”. Specifically, we use the word “edge-cloud” to represent this infrastructure of cloudlets in this paper.

B. Task Schedulers

The research of task scheduling in edge-clouds is fresh. As cloudlets are geo-distributed, scheduling on cloudlets is naturally decentralized. Rashidi *et al.* [7] have proposed a dynamic cloudlet selection policy, which uses the neuro-fuzzy inference system to place tasks. This policy is robust to the limited scheduling information. Shi *et al.* [8] have presented an adaptive probabilistic scheduler, which can optimize the energy consumption of tasks with time-constrained. Zhao *et al.* [9] have introduced a cooperative scheduling mechanism over edge-clouds and the cloud. The above works propose sophisticated algorithms to place tasks on cloudlets across the edge-clouds but are unaware of the characteristic of task types, which has a vital impact on the task scheduling. By contrast, Petrel implements a lightweight load balancing strategy and further an application-aware task scheduling algorithm.

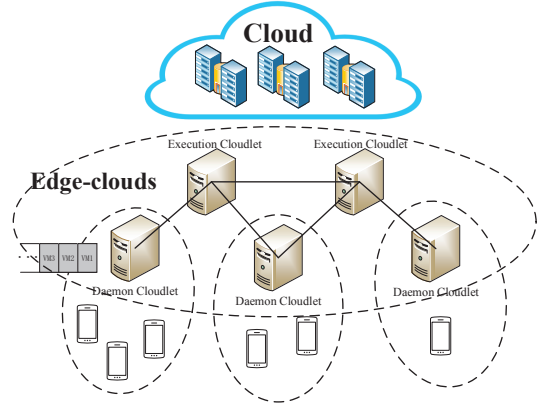


Fig. 1: System architecture

III. SYSTEM ARCHITECTURE

Fig. 1 illustrates the architecture of Petrel. It is a typical multi-tiered architecture, including mobile devices, edge-clouds, and the cloud. Thanks to the deployment of edge-clouds over cloudlets which are well-connected with each other, mobile devices can migrate their computation to these cloudlets for low latency. We introduce the concept of “daemon cloudlet”, which has the lowest delay with mobile devices in the vicinity, and it plays the role of a computing proxy for the mobile devices. Ideally, an offloading task is served by its daemon cloudlet; however, too many tasks can cause a long queueing time affecting the performance, and so a daemon cloudlet would send tasks to other execution cloudlets for load balancing. Petrel copes with the problem that whether a task would be executed on a daemon cloudlet or other execution cloudlets and which execution cloudlet to choose. Notice that daemon cloudlets and execution cloudlets are relative, a cloudlet can be a daemon cloudlet for some mobile devices but an execution cloudlet for others.

To support a standard execution environment of computation offloading, cloudlets in edge-clouds are built on virtual machines (VMs). VMs based resource provisioning [10] is a widely-used technology to set up the executing environment for cloudlets quickly. When a computation offloading executed, a mobile device migrates its method execution to a VM associated with uploading data about the method. Then when the cloudlet finishes, it returns results to the mobile device. Every daemon cloudlet makes scheduling decisions automatically for task requests from mobile devices in the vicinity. The cloud in Petrel is backup for task executing but not a centralized scheduling node.

IV. SCHEDULING MODEL

In this section, we build a model for the task scheduling in edge-clouds, where the “task” refers to a computation offloading from a mobile device to a remote server. A task can be executed on mobile devices, cloudlets in edge-clouds or the cloud. It is a multi-cloudlets scheduling problem, and we define the problem in the following.

A. Task Completion Time

The set J contains tasks from mobile devices; the task arrivals are independent and identically distributed (i.e., i.i.d). For each task i in J , the completion time of the task on the mobile device, the cloudlet, and the cloud are defined as:

$$T_i = \begin{cases} T_i^{mobile}, & \text{the mobile device;} \\ T_i^{cloud}, & \text{the cloud;} \\ T_i^{cloudlet}, & \text{the cloudlet.} \end{cases} \quad (1)$$

The completion time denotes the time span which a specific platform completes the task.

When a task runs on the mobile device, the completion time is equal to the task execution time R_i^{mobile}

$$T_i^{mobile} = R_i^{mobile}, \quad (2)$$

where R_i^{mobile} depends on the hardware of the mobile device.

If a task is decided to be executed in the cloud, and its completion time is calculated by

$$T_i^{cloud} = R_i^{cloud} + \frac{D_i}{B_{cloud}} + RTT_{cloud}. \quad (3)$$

R_i^{cloud} is the task execution time in the cloud; D_i indicates the data volume of uploading and downloading, and B_{cloud} is the bandwidth; RTT_{cloud} is the network delay. The part of $\frac{D_i}{B_{cloud}} + RTT_{cloud}$ denotes the communication time between the mobile device and the cloud. In general, the cloud is considered to have unlimited resource capacity, and so a task arriving in the cloud is immediately served with the time R_i^{cloud} .

The case of executing tasks on cloudlets is complicated. Cloudlets in edge-clouds can be seen as a network topology $G = (V, E)$, where the set V denotes the connected cloudlets. In Petrel, a daemon cloudlet has the lowest latency with mobile devices in the vicinity. If a task is assigned to its daemon cloudlet $v_d \in V$, the completion time is

$$T_i^{cloudlet} = R_i^{v_d} + W_i^{v_d} + \frac{D_i}{B_{v_d}} + RTT_{v_d} \quad (4)$$

Unlike the cloud, cloudlets have limited resource, and the task should contend for running including the waiting time $W_i^{v_d}$. As a result, the completion time contains the task execution time $R_i^{v_d}$ on the $v_d \in V$, the waiting time $W_i^{v_d}$, and the communication time $\frac{D_i}{B_{v_d}} + RTT_{v_d}$. Generally, cloudlets have larger bandwidth B_{v_d} and lower delay RTT_{v_d} than the cloud, and so the smaller of the communication time. In particular, for load balancing, a daemon cloudlet would send the task to other execution cloudlets, for example, the cloudlet $v_e \in V$. If the task i is assigned to the cloudlet v_e , and its completion time is calculated by:

$$T_i^{cloudlet'} = R_i^{v_e} + W_i^{v_e} + \frac{D_i}{B_{v_e}} + RTT_{v_d} + RTT_{v_e} \quad (5)$$

In (5), it has an additional RTT_{v_e} comparing to (4). As in Petrel, the mobile device first connects to its daemon cloudlet which decides how to place tasks; if the task is finally assigned to other execution cloudlets, e.g., v_e in this example, the daemon cloudlet will redirect the task to v_e with the additional delay RTT_{v_e} .

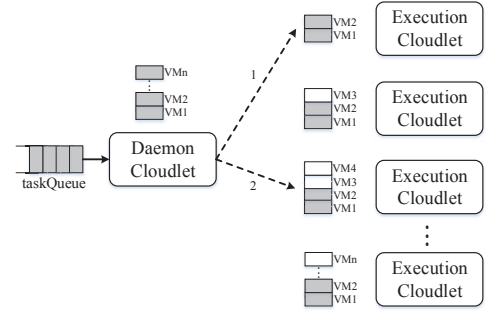


Fig. 2: An illustration of task scheduling in Petrel. If the daemon cloudlet has no idle VMs, it will probe two other execution cloudlets and pick up the one with the less load.

B. The Scheduling Objective

To formalize the completion time on different platforms, we define an allocation vector $A = (\alpha_1, \alpha_2, \alpha_3)$ to indicate which platform a task is assigned to. In the vector A , α_1 denotes whether the task is executed on a mobile device with α_2 for the cloud and α_3 for the cloudlet; the value 1 represents yes, otherwise 0. For example, $A = (0, 0, 1)$ indicates that the task is executed on the cloudlet. Further, we use the task speedup as the metric indicating the benefit of the task executed on remote servers

$$Sp_i = \frac{R_i^{mobile}}{AT_i}. \quad (6)$$

Then, for all tasks in J , the objective of task scheduling is to maximize the following:

$$\overline{Sp}_i = (\sum_{i \in J} \frac{R_i^{mobile}}{AT_i}) / N, \quad (7)$$

i.e., the average task speedup, where N indicates the number of task. Specifically, for each task i , the Sp_i should larger than 1, which means the computation offloading should improve its performance on the mobile device.

V. DISTRIBUTED AND APPLICATION-AWARE TASK SCHEDULING

To solve the challenges of the task scheduling in edge-clouds, in this section, we propose a distributed and application-aware algorithm (DAA).

Fig. 2 illustrates the process of task scheduling in Petrel. Tasks are served using the principle of first come, first served (FCFS) based on VMs as one task at a time for each VM. Once a task is scheduled, Petrel first finds if there are idle VMs on the daemon cloudlet, if so, then the daemon cloudlet will execute the task; otherwise, Petrel will probe other execution cloudlets. As shown in Fig. 2, when the task at the head of *taskQueue* is scheduled, the daemon cloudlet finds itself has no idle VMs at the time; then it sends probe 1 and 2 and finds the probe 2 with the less load to assign the task.

The details of task scheduling are shown in Algorithm 1. For each *task_i* in *taskQueue*, if the daemon cloudlet has idle VMs, it will serve the task immediately (in line 2); otherwise, it performs the load balancing policy (in line 5) and adopts

Algorithm 1 Distributed and Application-aware Scheduling

Initialization: for each $task_i$ arrives at its daemon cloudlet $cloudlet_d$, we insert $task_i$ into the queue $taskQueue$; $Type_{sen}$ denotes the set of latency-sensitive tasks; $bound_i$ denotes the latency bound of $task_i$

- 1: **if** $cloudlet_d$ has idle VMs **then**
- 2: assign $task_i$ to $cloudlet_d$;
- 3: **else**
- 4: calculate the expected completion time t_i^d on $cloudlet_d$;
- 5: randomly probe two cloudlets in edge-clouds, and pick up the one, $cloudlet_e$, with the least loaded of the two cloudlets;
- 6: **if** $task_i \in Type_{sen}$ **then**
- 7: calculate the expected completion time t_i^e on $cloudlet_e$;
- 8: **if** $t_i^e < t_i^d$ **then**
- 9: assign $task_i$ to $cloudlet_e$;
- 10: **else**
- 11: assign $task_i$ to $cloudlet_d$;
- 12: **end if**
- 13: **else**
- 14: // $task_i$ is a latency-tolerant task
- 15: **if** $cloudlet_e$ has idle VMs **then**
- 16: assign $task_i$ to $cloudlet_e$;
- 17: **else**
- 18: calculate the expected completion time $t_i^{d'}$ on $cloudlet_d$ for a delay of time D ;
- 19: **if** $t_i^{d'} \geq bound_i$ **then**
- 20: assign $task_i$ to $cloudlet_d$;
- 21: **else**
- 22: delay $task_i$ for time D
- 23: **end if**
- 24: **end if**
- 25: **end if**
- 26: **end if**

different assignment strategies in terms of task types (in lines 6 to 25).

The “load” in Algorithm 1 indicates the task queue length with respect to the task expected completion time on a cloudlet. Specifically, the task expected completion time means the wall-clock time at which a cloudlet completes a task. In DAA, what we say the least loaded of two cloudlets means if a task is tentatively assigned to the two cloudlets, the one providing the earlier expected completion time has the less load, as the $cloudlet_e$ in the algorithm. We maintain a *PriorityQueue* to store the ready time for all VMs on a cloudlet, and so the head of the *PriorityQueue* is the earliest ready time for a task. Then, the task expected completion time is calculated based on the *PriorityQueue*.

If there are no idle VMs on the daemon cloudlet, then the task scheduling strategies are task type dependent. If $task_i$ is a latency-sensitive task, we compare its expected completion time on $cloudlet_e$ with that on the daemon cloudlet $cloudlet_d$

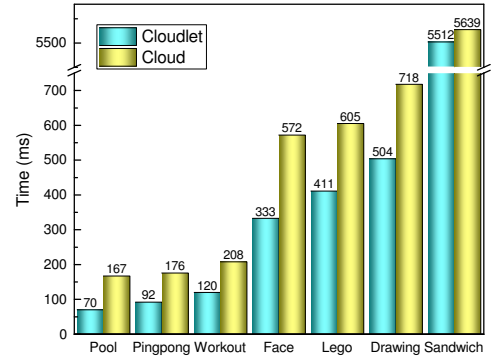


Fig. 3: The mean latency of benchmarks offloading to the cloudlet and the cloud

and assign the task to the faster one, as in lines 7 to 12. This comparison is necessary, as a task migrating to an execution cloudlet will cause an additional network delay shown in (5). As a result, this comparison guarantees the optimal latency of task executing. On the other hand, if $task_i$ is a latency-tolerant task, we first check if $cloudlet_e$ has idle VMs; if so, we perform the assignment; otherwise, we tentatively delay the task scheduling and calculate the expected completion time $t_i^{d'}$ in line 18. The $t_i^{d'}$ should satisfy the latency bound of $task_i$; if not, it will be assigned to $cloudlet_d$ with the expected completion time t_i^d . This process is what we call the “best effort” scheduling with the *delay scheduling* if there are no idle VMs.

VI. EVALUATION

In this section, we evaluate the performance of Petrel based on trace-driven simulations. First, we introduce benchmarks used in the simulations. Then, the experimental methodology is provided.

A. Benchmarks

Applications in computation offloading have various types and QoE demand [11], and we classify these applications into two categories: latency-sensitive applications and latency-tolerant applications. Then, we use the applications listed in the paper [12] as benchmarks, which are type of cognitive assistance applications [13]. Fig. 3 shows the mean latency when these benchmarks are offloaded to the cloudlet and the cloud, source from [12]. Notice that the latency on the cloudlet in this figure is considered as the *service time* in the following simulations; the service time indicates the time span if a task runs alone on the cloudlet. By further analysis, we find that the *Sandwich* uses a deep neural net approach which is compute-intensive and time-consuming, and people are more tolerant of the application; therefore, it can be seen as a latency-tolerant application. However, the other applications, employing some simple AI algorithms, need a tight interaction with users, and so they are latency-sensitive applications.

B. Methodology

To conduct the simulations, we build a trace-driven data set. The data set contains 200 offloading tasks; each task is

randomly selected from benchmarks in Fig. 3. There are 10 cloudlets in our edge-cloud, and each cloudlet has the number of virtual machine range from 1 to 10. Task arrivals on their daemon cloudlets are modeled as a Poisson process with the arrival rate λ . We set two different arrival rates $\lambda=1$ and $\lambda=2$ in our simulations, which means the mean time intervals between two tasks are 1 and 0.5 unit time respectively.

The latency illustrated in Fig. 3 can be seen as the time span when a task is offloaded from a mobile device to its daemon cloudlet, which is considered to have the lowest latency with mobile devices in the vicinity, and the RTT is usually smaller than 10ms. However, if a daemon cloudlet decides to send tasks to other execution cloudlets, it will cause an additional RTT from mobile devices to execution cloudlets according to (5), and this additional RTT usually ranges from 50-70ms [14]. We consider these RTTs in the construction of our data set.

1) *Comparing Algorithms*: For a comprehensive evaluation, we compare DAA with the following scheduling algorithms:

- **DaemonCloudletOnly**: all tasks are executed only on daemon cloudlets with the rule of FCFS. There is no load balancing strategy.
- **RoundRobin**: a simple load balancing algorithm. Tasks are distributed to cloudlets on the edge-cloud in a round robin way. This algorithm treats each cloudlet equally, without considering the benefit of daemon cloudlets.
- **GreedyScheduler**: a greedy scheduling algorithm. For each task, GreedyScheduler always finds an optimal cloudlet which has the minimum completion time to serve the task.
- **TwoChoices**: employing the “the power of two choices” strategy. In TwoChoices, for each task, it randomly probes two cloudlets and selects the one has the less load.

Our proposed DAA in Algorithm 1, is a distributed scheduling algorithm, employing a sample-based load balancing technology and further an application-aware scheduling strategy.

C. Simulation Results

In this section, we evaluate the DAA algorithm in two metrics: the average weighted turnaround time (AWT) and the makespan for cloudlets.

1) *Average Weighted Turnaround Time*: The average weighted turnaround time is defined as:

$$AWT = \left(\sum_i^N \frac{T_i^{\text{turnaround}}}{T_i^{\text{service}}} \right) / N. \quad (8)$$

T_i^{service} is the *service time* as a task is served by a cloudlet alone; $T_i^{\text{turnaround}}$ denotes the turnaround time of a task, i.e., the *completion time* we define in Section IV-A; N is the number of tasks. Reviewing (7) in Section IV-B, the scheduling objective is to maximize the overall task speedups. In other words, by comparing (7) and (8), it can be seen that *the higher the average task speedup, the lower the average weighted turnaround time*.

Fig. 4 shows the comparison of these algorithms in terms of the average weighted turnaround time. As can be seen,

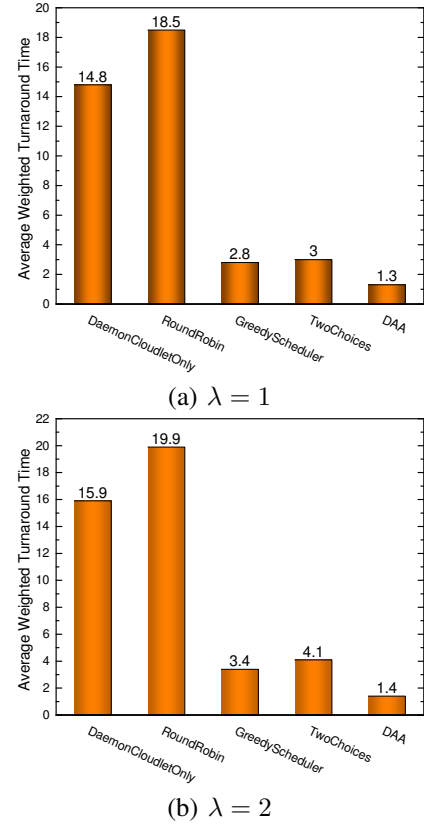


Fig. 4: The average weighted turnaround time

DaemonCloudletOnly and RoundRobin have the worst performance. Specifically, DaemonCloudletOnly restricts all tasks to be executed only on daemon cloudlets and employs no load balancing strategies, leading to poor performance on the less powerful cloudlets. Although RoundRobin uses a simple Round Robin load balancing, this strategy treats every cloudlet equally without considering the benefit of daemon cloudlets and the more powerful cloudlets. GreedyScheduler achieves a significant performance improvement, as it always assigns tasks to the cloudlet that has the minimum completion time. However, it incurs enormous scheduling overhead across cloudlets on the edge-cloud. On the contrary, the TwoChoices algorithm only samples two cloudlets when performing load balancing, but it gets similar performance with GreedyScheduler.

It can be seen that DAA has better performance than the other algorithms, and the gap is increasing with a more frequent task arrival ($\lambda = 2$). DAA adopts different strategies according to the type of tasks, as “greedy” for latency-sensitive tasks and “best effort” for latency-tolerant tasks. These strategies can conquer the drawback by always placing tasks greedily without considering the type of the task, which can cause the starvation of short tasks (latency-sensitive tasks) because of the long tasks (latency-tolerant tasks) executing, just as GreedyScheduler does. Moreover, DAA gets the task assignments with a lightweight load balancing strategy like TwoChoices does. Furthermore, we evaluate the performance

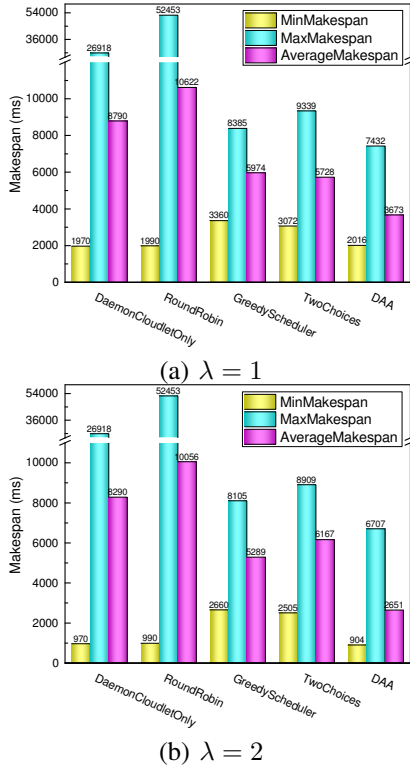


Fig. 5: The minimum, maximum, and average makespan of cloudlets

when all tasks are executed on the cloud, and the average weighted turnaround time is 1.6, which is also worse than DAA. It appears that task executed on cloudlets can achieve better performance than on the cloud if the right scheduling algorithms are adopted. Meanwhile, all user tasks executed on the public cloud seems unpractical in terms of resource consumption.

2) *Makespan*: Makespan is defined as the time when a cloudlet completes the last task; in other words, it is a metric of the throughput of the cloudlet [15]. Scheduling algorithms strive for the goal of minimizing the makespan and maximizing the throughput. Fig. 5 depicts the minimum (MinMakespan), the maximum (MaxMakespan), and the average makespan (AverageMakespan) of cloudlets. The average makespan indicates the mean value of all cloudlets in our edge-cloud. It can be seen that DaemonCloudlet and RoundRobin have the worst maximum and average makespan, as they perform no or inefficient load balancing strategies, which incurs so obviously unbalanced load among cloudlets. GreedyScheduler and TwoChoices improve the overall average makespan by distributing tasks efficiently, but sacrifice some cloudlets which have the lower load, as can be seen in the minimum makespan. Finally, the DAA algorithm achieves the best overall performance.

VII. CONCLUSION

In this paper, we present Petrel, a distributed and application-aware task scheduling framework for edge-clouds.

Petrel implements a sample-based load balancing for cloudlets in edge-clouds, which is simple but efficient and can reduce the scheduling overhead sharply. Furthermore, Petrel adopts different scheduling policies in terms of task types, as the “greedy” policy for latency-sensitive tasks but the “best effort” service for latency-tolerant tasks. The results of trace-driven simulations show that our proposed scheduling strategies achieve significant improvements over the existing scheduling algorithms.

VIII. ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under grant No. 61502103, 61872088, and 61872086, and Natural Science Foundation of Fujian Province under grant No. 2017J01737.

REFERENCES

- [1] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
- [3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing a key technology towards 5g,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.
- [5] M. Mitzenmacher, “The power of two choices in randomized load balancing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, Oct 2001.
- [6] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, “The power of two random choices: A survey of techniques and results,” *Combinatorial Optimization*, vol. 9, pp. 255–304, 2001.
- [7] S. Rashidi and S. Sharifian, “Cloudlet dynamic server selection policy for mobile task off-loading in mobile cloud computing using soft computing techniques,” *The Journal of Supercomputing*, vol. 73, no. 9, pp. 3796–3820, Sep 2017.
- [8] T. Shi, M. Yang, X. Li, Q. Lei, and Y. Jiang, “An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds,” *Pervasive and Mobile Computing*, vol. 27, pp. 90 – 105, 2016.
- [9] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, “A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing,” in *2015 IEEE Globecom Workshops (GC Wkshps)*, Dec 2015, pp. 1–6.
- [10] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, “Just-in-time provisioning for cyber foraging,” in *Proceeding of MobiSys*, 2013.
- [11] F. A. Silva, G. Zaicaner, E. Quesado, M. Dornelas, B. Silva, and P. Maciel, “Benchmark applications used in mobile cloud computing research: a systematic mapping study,” *The Journal of Supercomputing*, vol. 72, no. 4, pp. 1431–1452, Apr 2016.
- [12] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, “An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance,” in *Proceedings of SEC*, 2017, pp. 14:1–14:14.
- [13] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, “Towards wearable cognitive assistance,” in *Proceedings of MobiSys*, 2014.
- [14] S. Agarwal, M. Philipose, and P. Bahl, “Vision: The case for cellular small cells for cloudlets,” in *Proceedings of MCS*, 2014, pp. 1–5.
- [15] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen, and R. F. Freund, “Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems,” in *Proceedings of HCW*, April 1999, pp. 30–44.