Job Scheduling on Edge and Cloud Servers

Xiao Chen
Department of Computer Science
Texas State University
San Marcos, TX 78666
xc10@txstate.edu

***Abstract*—In edge-clouds, servers are placed at the edge of the network so that mobile devices can get the jobs done with low latency. One fundamental and critical problem in edge-cloud systems is how to place independent jobs on the edge and cloud servers with the objective to minimize the makespan. In this paper, we propose a general model for this problem and put forward an algorithm called Cloud-Edge (CE) scheduling for two servers. We prove that the competitive ratio of the algorithm is upper-bounded by two and no other online algorithm can do better than it in the worst case. We conduct extensive simulations to verify the competitive ratio of our proposed algorithm. Simulation results confirm the correctness of our theoretical analysis regardless of the communication overhead.**

***Index Terms*—cloud, competitive ratio, makespan, mobile edge computing, online scheduling**

I. Introduction

With the development of cloud computing, more and more mobile applications offload computation-intensive jobs to remote cloud data centers [10]. Although such operation could substantially enhance the capability of mobile devices, a long communication delay is inevitable. To mitigate the problem, people place servers at the edge of the network to be closer to the users so that mobile devices can obtain computing results with low network latency. However, the servers at the edge are resource constrained compared with those in the cloud. So a remote cloud is needed to support the edge resources so that mobile devices can offload more computation demanding jobs to the cloud via the Internet. Thus, the paradigm that combines the resources at the edge and the cloud, called *edge-clouds*, also known as edge computing, is drawing more and more attention from the researchers and developers [12], [13].

Fig. 1 shows an edge-clouds environment. As we can see from the figure, some servers are placed at the edges. We call them *edge servers*. And some are put in the remote cloud. We call them *cloud servers*. If a job is executed on an edge server, we can ignore the communication latency. The total response time of the job is the job's running time. But if a job is offloaded to the remote cloud server, the total response time will be the job's running time plus the back-and-force communication delay, which we call the *overhead* of the job.

In an edge-cloud network, a fundamental and critical problem is how to schedule the in-coming jobs to the edge and cloud servers. Some work on scheduling assumed a First-Come-First-Serve scheme [11], [18] and some others schedule jobs to achieve load balancing [16], [17]. However, most of the studies assumed that the arrival of the jobs follows some known distribution, so that they can use statistical model to design an efficient strategy. In reality, job release may not follow a known distribution. Tan et al. [15] proposed a model where the jobs are generated in arbitrary order, but their results are based on the speed augmentation model.

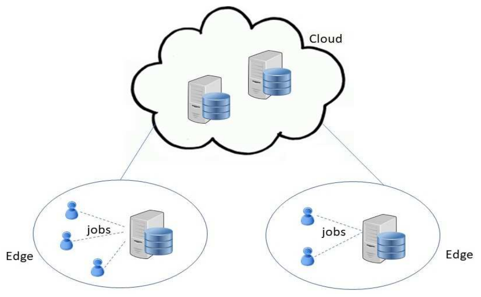


Fig. 1. Edge-clouds environment

In this paper, we investigate general online job scheduling algorithms in edge-clouds without the assumption of stochastic process and speed augmentation model. Our objective is to minimize the *makespan* (i.e., the length of the schedule, or equivalently the last job completion time). An input to this problem consists of a sequence of jobs of different *sizes* (or *running times*) and their corresponding overhead if they are assigned to the cloud servers. Each job has to be assigned to one of the edge or cloud servers. There are no dependencies between the jobs and no release times. The job scheduling problem in edge-clouds is related to the classic m-machine scheduling problem [5]. Even the special case of $m = 2$ is NP-hard. In order to analyze the situation further, in this paper, we focus on two servers: one edge server and one cloud server. We leave the extension to more than two servers in the future research.

We use *competitive ratio* as the metric to evaluate our online algorithm, defined as the largest possible ratio of the makespan achieved by the online algorithm and the offline optimal makespan for any possible set of jobs and overhead. It represents the worst-case scenario of the online algorithm.

The differences of our work from others and the key contributions of our work are as follows:

- we formulate a job scheduling problem in edge-clouds under a general model using two servers.
- we propose an online algorithm called *Cloud-Edge* (CE) scheduling with a competitive ratio bounded by 2. We show that no other online algorithm can do better than CE scheduling in the worst case.

- we conduct extensive simulations to validate the competitive ratio of our algorithm. The experimental results confirm the correctness.

The rest of the paper is organized as follows: Section II references the related work. Section III defines the problem. Section IV presents our algorithm and the proof of the competitive ratio. Section V describes the simulations we have conducted, and the conclusion is in Section VI.

II. Related Work

A. Cloud and Edge Computing

There have been quite a lot of studies on job placement problems to balance the resource utilization and response time among the servers in edge-clouds. Urgaonkar et al. [17] formulated the workload scheduling problem as a Markov Decision Process problem and adopted some Lyapunov optimization technique to solve this problem. Tong et al. [16] divided the edge-clouds into different levels and presented a heuristic algorithm to dispatch the workload within this hierarchical architecture. The above load balancing schemes adopt stochastic optimization, where they assume that the job release process follows a certain distribution. However, in practice, the jobs released from applications can be in arbitrary order and times. Tan et al. [15] proposed a model where the jobs are generated in arbitrary order. Their goal is to minimize the total weighted response time over all the jobs. They derived the first online job dispatching and scheduling algorithm in edge-clouds and proved that it is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for any constant $\epsilon \in (0,1)$ based on the speed augmentation model. In our study, we work on a general model without the assumption of job release distribution and speed augmentation model.

B. Classic Online Machine Scheduling Problem

The job placement problem in edge-clouds is related to the classic m-machine scheduling problem. The m-machine scheduling problem is one of the most widely-studied problems in computer science [5]. Even the special case of $m = 2$ is NP-hard. Here we only address online problems and solutions because, comparing with offline problems, they are more practical in real world because we need to make immediate decisions when complete information is not available.

The first proof of competitiveness of an on-line scheduling algorithm was given by Graham in 1966 [8]. He proposed a deterministic greedy algorithm called *List Scheduling* (LS). The studied model has m identical machines and a sequence of jobs characterized by their running times. The objective is to minimize the makespan. The LS algorithm is simple: whenever a new job arrives, we place it on the machine with the smallest load. The competitive ratio of LS is $2 - \frac{1}{m}$ [8], [9]. If we have two identical machines, we put the new job on the machine with smaller load and the competitive ratio is $\frac{3}{2}$. Several algorithms have obtained better worst-case ratio than LS. Galambos and Woeginger [7] improved the upper bound to $2 - \frac{1}{m} - e_m$, where e_m goes to 0 as m goes to infinity. Bartal et al. [3] developed a heuristic with worst-case ratio of $2 - \frac{1}{70} \approx 1.986$. The current best result is due to Albers [1]. The author gave an algorithm which achieves a worst-case ratio of at most 1.923 for all m. However, Faigle et al. [6] have shown that no deterministic on-line algorithm can have a worst-case ratio smaller than $2 - \frac{1}{m}$ for $m = 2$ and $m = 3$. Thus, there is no deterministic on-line scheduling algorithm with worst-case ratio lower than $\frac{3}{2}$. The latter result can be obtained by considering the instance where the first two items are $a_1 = a_2 = 1$ and the third item may be either $a_3 = 0$ or $a_3 = 2$. For $m > 4$ they gave a lower bound of $1 + \frac{1}{\sqrt{2}}$. Bartal et al. [4] improved this lower bound to 1.837 for m large enough. All of these algorithms assume that the machines are identical.

If the machines are not identical, there are two variants. In the variant of *uniformly related machines*, the ith machine has speed $v_i > 0$. If a job with running time t is scheduled on it, its processing takes time $\frac{t}{v_i}$. Another variant is *unrelated machines*, where the vector of speeds is possibly different for each job. Aspnes et. al. in [2] found that for related machines the competitive ratio of LS is asymptotically $\Theta(\log m)$ and for unrelated machines the competitive ratio of LS is n.

The problem we study in this paper is different from the above problems. In our problem, the finish time of a job is related to the communication overhead, which reflects the state of the network. Next we define our problem formally.

III. Problem Definition

Suppose there is a sequence of n jobs $A = \{a_1, a_2, \cdots, a_n\}(a_i > 0 \text{ for all } i)$, where each job is characterized by its running time a_i and has to be scheduled on one of the servers. To facilitate discussion, we denote the cloud server as P_1 and the edge server as P_2. If a job is scheduled on the edge server, there is no communication overhead for that job. We denote by $O = \{o_1, o_2, \cdots, o_n\}(o_i > 0 \text{ for all } i)$ the corresponding overhead of each job if it is placed on the remote cloud server. Our goal is to minimize the total length of the schedule, the makespan. In this paper, we study on-line algorithms where the algorithms do not have the access to the whole input sequence to reflect the realistic scenario. The algorithms learn the input piece by piece and react to the next job with only a partial knowledge of the input.

From the above definition, we take vectors A and O as inputs. How do we know the job size and overhead before we dispatch them? In order to do the estimation, we can use the idea in estimating the next CPU burst time in CPU scheduling algorithms [14]. Vector A is obtained in this way and so as O. We use O as an example here. To estimate the overhead of a particular job, our formula is: $\tilde{o}_{t+1} = \alpha o_t + (1-\alpha)\tilde{o}_t$, where $\tilde{o}_{t+1}$ is the predicted value of the next communication overhead at time $t+1$, o_t is the actual overhead at time t, and α is a weight in the range of [0,1]. For the first time, we do not know what the communication overhead of a particular job is, we can guess a reasonable number. After the job is sent to the server and the reply comes back, then we know the actual communication overhead. With the actual overhead and the guessed overhead, we can use the formula to estimate the next communication overhead for this kind of job.

IV. Our Solution

In this section, we present our heuristic solution to the defined problem. First, we show why the LS algorithm can no longer be used to solve the problem. Next we propose our online scheduling algorithm that has a competitive ratio of 2.

A. LS Algorithm not Working

The first intuition to solving our problem is to try the LS algorithm. However, the competitive ratio of LS will no longer be bounded. Here is why. Suppose we have jobs $A = \{1, 1\}$ and their corresponding overhead $O = \{1, L\}$, where L is a very large number. If we use LS, each time we place the new job on the server with a smaller load. The resulting schedule is $P_1 : \{L + 1\}$ and $P_2 : \{1\}$, which has a makespan of $L + 1$. The optimal schedule is $P_1 : \{\}$ and $P_2 : \{1, 1\}$. It has a makespan of 2. Thus, the competitive ratio of LS is $\frac{L+1}{2}$. Since overhead L can be infinite, the competitive ratio is not bounded in the worst case. This example tells us that, in order to come up with an online algorithm with a bounded competitive ratio, we need to take overhead into account since it plays an important role in the result.

B. Our Algorithm

In this section, we describe our proposed online algorithm CE scheduling to solve our defined problem. The algorithm is presented in Fig. 2. It schedules jobs one by one in a for loop. For each job, first it finds out which server is better by calling function Whos_better(). This function considers the current load on the two servers and the overhead of the incoming job. Once the better server is known, the algorithm calls put() function to place the job on the better server. The details of the two functions are explained below.

Algorithm CE Scheduling

```
Requires: /* server set $P$; $P_1$ is the cloud server and $P_2$ is the edge server */
Inputs: a set of jobs $A = \{a_1, a_2, \cdots, a_n\}$ and their corresponding overhead $O = \{o_1, o_2, \cdots, o_n\}$
Output: a schedule of jobs on the cloud and edge servers
for $i = 1 : n$ do
    /* find which server of $P$ is better*/
    $P\_better$ = whos_better$(P_1, P_2, a_i, o_i)$;
    put($P\_better$, $a_i$, $o_i$);
end for
return the schedule on $P$;
```

Fig. 2. The CE Scheduling algorithm

The procedure of function Whos_better() is described in Fig. 3 and the conditions in the function come from the decision tree in Fig.4. The goal of the decision is to find out which server, P_1 or P_2, is better. There are two types of possible placement of job a_i, either on P_1 (shown on the top left in Fig. 4) or P_2 (shown on the top right). Since P_1 is the cloud server, we need to add the overhead when a job is placed on it. P_2 is the edge server. We do not need to consider overhead. Through the comparison of the loads on the two servers and the makespans of the two assignments, we identified four cases of conditions and the corresponding better server shown in the lower part of Fig. 4. The items in shade represent the larger item in each load comparison. We write up these conditions and put them in function Whos_better().

Function $better$ = **Whos_better**(P_1, P_2, a_i, o_i)

```
/* calculate the load on $P_1$ and $P_2$ */
$s_1 = sum(P_1)$; $s_2 = sum(P_2)$;
if $s_1 + a_i + o_i > s_2$ then
    if $s_1 > s_2 + a_i$ then
        $better = 2$;
    else
        if $s_1 + o_i <= s_2$ then
            $better = 1$;
        else
            $better = 2$;
        end if
    end if
else
    $better = 1$;
end if
```

Fig. 3. The whos_better() function

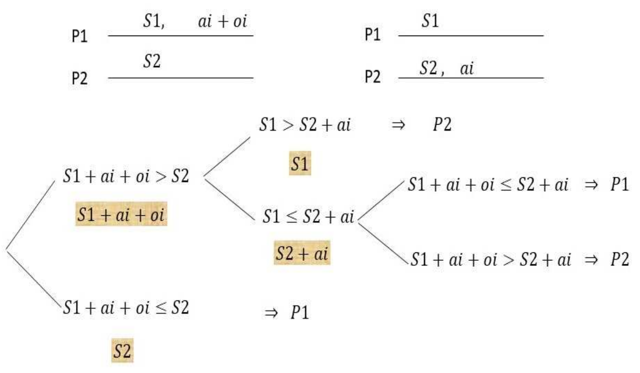


Fig. 4. Decision tree for whos_better()

Function put$(server, a_i, o_i)$

```
if $server$ is the cloud server then
    put $a_i + o_i$ on the cloud server;
else
    put $a_i$ on the edge server;
end if
```

Fig. 5. The put() function

Function put$(server, a_i, o_i)$ is presented in Fig. 5. It checks if the server is the cloud server, if so, it adds $a_i + o_i$ to the cloud server; otherwise, it adds a_i to the edge server.

Next, we calculate the competitive ratio of our proposed algorithm. We first show that the competitive ratio of our algorithm cannot be upper bounded by a number less than 2. Then we prove that any other online scheduling algorithm cannot do better than our proposed algorithm in the worst case. And finally, we demonstrate that the competitive ratio of our algorithm is bounded by 2.

Theorem 1: The competitive ratio of CE scheduling cannot be $\frac{C^H}{C^*} \leq B$, where $B < 2$.

Proof. We use proof by contradiction. Suppose the competitive ratio of CE scheduling is upper bounded by a number B, which is less than 2. Consider this example: jobs $A = \{a, a\}$ and their corresponding overhead $O = \{\epsilon, L\}$, where $\epsilon < a, L > a$. The schedule from CE scheduling is: $P_1 = \{\}$ and $P_2 = \{a, a\}$, which has makespan $2a$. The optimal schedule is: $P_1 = \{a + \epsilon\}$ and $P_2 = \{a\}$. Its makespan is $a + \epsilon$. The ratio of this example is $\frac{2a}{a+\epsilon}$. According to our assumption, $\frac{2a}{a+\epsilon} \leq B$. Now we pick an ϵ' less than $\frac{2a}{B} - a$. Because $B < 2$, a positive ϵ' exists. That means there exists an example where jobs $A = \{a, a\}$ and the corresponding overhead $O = \{\epsilon', L\}$. The ratio of this example is $\frac{2a}{a+\epsilon'} > \frac{2a}{a+\frac{2a}{B}} = B$. Thus the competitive ratio is greater than B. This is in conflict with our assumption. So the competitive ratio of CE scheduling cannot be upper bounded by a number less than 2. In addition, since there is always some communication overhead if the job is executed on the cloud server, so $\epsilon > 0$. Therefore, the competitive ratio of CE scheduling algorithm is strictly less than 2. □

Next we prove that any other online scheduling algorithm cannot do better than CE scheduling in the worst case.

Theorem 2: Any online scheduling algorithm cannot do better than CE scheduling in the worst case.

Proof. We can still use the same example: jobs $A = \{a, a\}$ and $O = \{\epsilon, L\}$, where $\epsilon < a, L > a$. Using CE scheduling, the best choice for the first a is P_2 since it is the edge server without overhead. For the second a, we will still place it on P_2 because the makespan will be $2a$ rather than $a + L$ if we put it on P_1, and $a+L > 2a$ as $L > a$. Since any other online algorithm schedules jobs one by one and cannot change the allocation of the previous jobs, it will reach the same schedule and have makespan $2a$ in this example. So it cannot do better than CE scheduling in the worst case. □

Theorem 3: The competitive ratio of heuristic CE scheduling is $\frac{C^H}{C^*} < 2$.

Proof. We denote by C_i^* the optimum solution value for the job sequence $a_1, a_2, \cdots, a_i$. We will use induction. When $i = 1$, the assertion is true. Because both CE scheduling and the optimal will put the job on P_2, resulting a ratio of 1, which is less than 2. Now let $i > 1$, and suppose that the upper bound 2 is valid for all $i' < i$. Denote by s_1 and s_2 the load of P_1 and P_2, respectively, before assigning job a_i. According to assumption, $s_1 < 2C_{i-1}^*$ and $s_2 < 2C_{i-1}^*$.

When a new job a_i arrives, CE scheduling puts it on a better server determined by whos_better() function. In the function, there are four cases. Two of them identify P_1 as the better server and the other two return P_2 as better. First, let us look at the two cases where P_2 is better. These two cases are when $s_1+a_i+o_i > s_2$ && $s_1 > s_2+a_i$ are true and $s_1+a_i+o_i > s_2$ && $s_1 \leq s_2 + a_i$ && $s_1 + o_i > s_2$ are true. After a_i is placed on P_2, the load on P_2 becomes $s_2 + a_i$. We know the fact that $s_1 + s_2 + a_i \leq 2C_i^*$. If $s_1 > 0$, then $s_2 + a_i < 2C_i^*$. And if $s_1 = 0$, $s_2 + a_i = C_i^* < 2C_i^*$. So $s_2 + a_i < 2C_i^*$ in any case. On the other hand, the load on P_1 is still s_1. By assumption and a trivial fact, $s_1 < 2C_{i-1}^* \leq 2C_i^*$. Thus, the makespan $max(s_1, s_2 + a_i) < 2C_i^*$. The assertion is true.

Now let us look at the two cases when P_1 is better. In the first case, when $s_1+a_i+o_i > s_2$ && $s_1 \leq s_2+a_i$ && $s_1+o_i \leq s_2$ are true, job a_i is put on P_1. Then the load on P_1 becomes $s_1 + a_i + o_i$. From condition $s_1 + o_i \leq s_2$, we get $o_i \leq s_2 - s_1$. So $s_1 + a_i + o_i \leq s_1 + a_i + s_2 - s_1 = s_2 + a_i < s_2 + a_i + s_1 + o_i \leq 2C_i^*$ using the fact that $o_i > 0$ and $s_2 + a_i + s_1 + o_i \leq 2C_i$. The load on P_2 is still s_2, where $s_2 < 2C_{i-1}^* \leq 2C_i^*$. Thus, the makespan $max(s_1 + a_i + o_i, s_2) < 2C_i^*$. The assertion is true. In the second case, when $s_1+a_i+o_i \leq s_2$ is true, we put job a_i on P_1. Then the load on P_1 is $s_1+a_i+o_i$. From the condition, assumption, and the trivial fact, we get $s_1 + a_i + o_i \leq s_2 < 2C_{i-1}^* \leq 2C_i^*$. The load on P_2 is still the same. Again, the makespan $max(s_1+a_i+o_i, s_2) < 2C_i^*$. The assertion is true.

After discussing all the cases, we proved the theorem. □

V. Simulations

In this section, we conduct simulations to validate the competitive ratio of our proposed algorithm using a customized simulator written in Matlab.

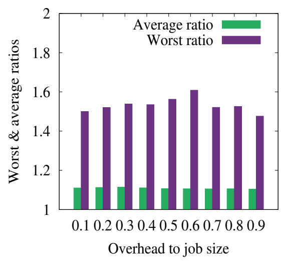
A. Metrics

We use two metrics in our simulations. One is *worst ratio*, which is the largest possible ratio of the makespan generated by the proposed online algorithm in our sample space and the offline optimal makespan. The other one is the *average ratio*, which is the average of all the ratios of the makespans produced by the online algorithm in our sample space and the offline optimal makespan. To obtain the optimal makespan in each data setting, we adopted the brute force method.

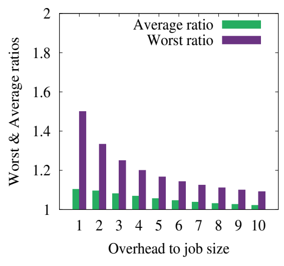
B. Experiment 1

In this experiment, we assume that the overhead of a job is related to its size. We tried five and eight jobs in our simulation. We randomly generated job sizes in the range of [1, 200]. For the overhead, we had two categories. In one category, the overhead of each job is a fraction of the job size. We varied the fraction from 0.1 to 0.9. In another category, the overhead of a job is a multiple of the job size. We enumerated the multiple from 1 to 10. For each parameter setting, we ran the algorithm $10,000$ times and then output the average and worst ratios of these samples.

The simulation results with different job numbers and overhead are shown in Figs. 6 and 7. In each parameter setting of all of these figures, it is obvious that the worst ratio is higher than the average ratio. And regardless of the value of the overhead, these two ratios are upperbounded by 2, which is consistent with our theoretical analysis.

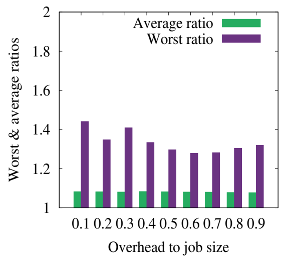


(a) Overhead a fraction of job size

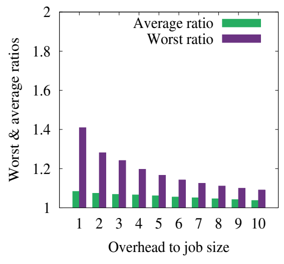


(b) Overhead multiple of job size

Fig. 6. Five jobs, overhead related to job size



(a) Overhead a fraction of job size



(b) Overhead multiple of job size

Fig. 7. Eight jobs, overhead related to job size

C. *Experiment* 2

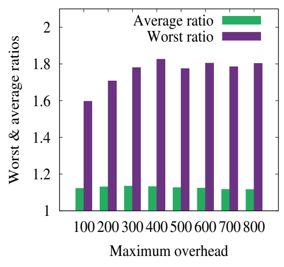
In this experiment, we assume that the overhead is not related to job size. We tried job numbers from 5 to 8. We randomly generated job sizes in the range of [1, 400]. The overhead of each job is randomly generated in the range of [1,X], where X starts from 100 to 800 with a step of 100. For each parameter setting, we ran the algorithm $10,000$ times and then output the average and worst ratios of these samples.

The simulation results with different job numbers and overhead are shown in Figs. 8(a)-(d). In each parameter setting, the worst ratio is higher than the average ratio. And regardless of the value of the overhead, these two ratios do not exceed 2, which again matches our theoretical analysis.

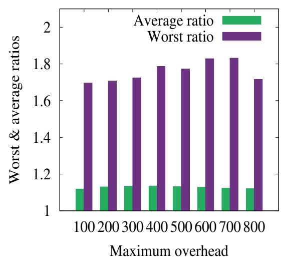
From both experiments, we confirm that the competitive ratio of our proposed algorithm is upper-bounded by 2.

VI. Conclusion

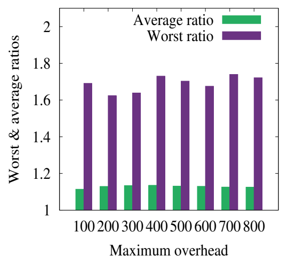
In this paper, we have investigated general online scheduling algorithms to place independent jobs on the edge and cloud servers with the objective to minimize the makespan. We have proposed the CE scheduling algorithm for two servers. We have proved that the competitive ratio of the algorithm is upper-bounded by two and no other online algorithm can do better than it in the worst case. We have also conducted extensive simulations to verify the competitive ratio of our proposed algorithm. Simulation results have confirmed the correctness of our theoretical analysis. In this paper, we just discussed the case of two servers. If there are more than two servers, the competitive ratio of 2 no longer holds. More detailed analysis will be our future work.



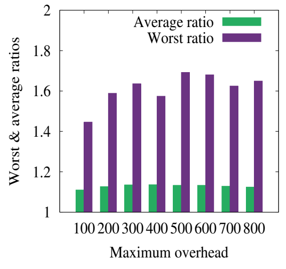
(a) Five jobs



(b) Six jobs



(c) Seven jobs



(d) Eight jobs

Fig. 8. Different numbers of jobs, overhead not related to job size

References

[1] S. Albers. Better bounds for online scheduling. In *Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 130–139, 1997.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 3(44):486–504, 1997.

[3] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Sys. Sci.*, (51):359 –366, 1995.

[4] Y. Bartal, A. Fiat, and Y. Rabani. A better lower bound for online scheduling. *Inform. Process. Lett.*, (50):113–116, 1994.

[5] D. Dwibedy and Rakesh Mohanty. Online scheduling with makespan minimization: State of the art results, research challenges and open problems. *ArXiv*, abs/2001.04698, 2020.

[6] U. Faigle, W. Kern, and G. Turán. On the performance of online algorithms for particular problems. *Acta Cybernetica*, (9):107–119, 1989.

[7] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM J. Comput.*, (22):349–355, 1993.

[8] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, November 1966.

[9] L. A. Hall and D. B. Shmoys. Approximation schemes for constrained scheduling problems. In *Proc. of 30th Ann. IEEE Symp. on Foundations of Computer Sci.*, pages 134–139, 1989.

[10] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau. Dynamic virtual machine management via approximate markov decision process. In *Proc. of INFOCOM*, 2016.

[11] M. Jia, J. Cao, and W. Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. In *Proc. of INFOCOM*, 2016.

[12] Y. Li and S. Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *Proc. of IEEE International Conference on Edge Computing (EDGE)*, pages 66–73, 2018.

[13] Z. Lu, N. Wang, J. Wu, and M. Qiu. IoTDeM: An IoT Big Data-oriented MapReduce performance prediction extended model in multiple edge clouds. *Journal of Parallel and Distriuted Computing*, 118, 2015.

[14] A. Silberschatz, G. Gagne, and P. B. Galvin. *Operating System Concepts*. Wiley, 10th edition, 2018.

[15] H. Tan, Z. Han, X. Y. Li, and F. Lau. Online job dispatching and scheduling in edge-clouds. In *Proc. of INFOCOM*, 2017.

[16] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *Proc. of INFOCOM*, 2016.

[17] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation*, 91:205–228, 2015.

[18] Y. Zhang, D. Niyato, and P Wang. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516–2529, 2015.