IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 26, 2019, accepted June 19, 2019, date of publication June 27, 2019, date of current version July 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2925429

A Scheduling Method of Moldable Parallel Tasks Considering Speedup and System Load on the Cloud

JIANMIN LI[1], YING ZHONG[1], AND XIN ZHANG[2]

[1]School of Computer and Information Engineering, Xiamen University of Technology, Xiamen 361005, China
[2]School of Ophthalmology and Optometry, School of Biomedical Engineering, Wenzhou Medical University, Wenzhou 325035, China

Corresponding authors: Jianmin Li (lijianmin2006@sina.cn) and Xin Zhang (mywork@nuist.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61806173 and Grant 61672442, in part by the Joint Funds of Scientific and Technological Innovation Program of Fujian Province under Grant 2017Y9059, in part by the Science and Technology Planning Project of Xiamen/Quanzhou City under Grant 3502Z20183055 and Grant 2017G030, in part by the Natural Science Foundation of Zhejiang Province under Grant LQ18F020005, and in part by the Science and Technology of Wenzhou under Grant S20170008.

ABSTRACT The moldable parallel task (MPT) is a kind of parallel task that their sub-tasks hold the resources exclusively, which has been widely used in different areas. Our paper focuses on the scheduling of moldable tasks when every sub-task supports time-slice. The time-slice is a consecutive time that the sub-task holds the resources exclusively. After every time-slice, the sub-task can be canceled, suspended, or continued. We give the model of MPTs and propose a scheduling method for MPTs: MC (a heuristic scheduling method supporting time-slice model on the cloud). The simulation results show that, even under a forecast accuracy of system load under 90% and 95%, MC reduces average waiting time and average execution time; at the same time, MC has a lower value in the percentages of unfinished tasks.

INDEX TERMS Moldable parallel task, speedup, parallelism, scheduling method.

I. INTRODUCTION

Cloud has been widely used in different aspects and it brings convenience for many enterprises [1]. Cloud computing takes infrastructure, platform, and software as services and supports a pay-as-you-go model for geographically distributed consumers [2]. There are three kinds of service providing methods in Cloud: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service). Many cloud platforms have been built by different companies, such as Microsoft Azure, Amazon EC2, Google App Engine and Aneka [3]. Researchers also have done much work on the Cloud, including the scheduling methods [4], security, virtualization [5], load balance [6], and so on [2], [7]–[10].

The application with big data always spends much time on computing and data transmission. Parallelization is the most important way to shorten the execution time of applications with big data. It divides one problem into some sub-problems that would be parallel executed. Parallel tasks are widely used in the Cloud [8]. Most of past work used DAG (Directed acyclic graph) to model parallel tasks and given different scheduling methods based on the DAG. Those scheduling methods are mainly through intelligent selecting the execution route to achieve scheduling targets, such as the QoS (Quality of Service) requirement, minimizing energy consumption, maximizing output and profit and so on. Different to the prior work, we focus on the scheduling of the MPT considering the system load and the speedup of MPT. Especially, the sub-task supports the time-slice. The time-slice is a continuous period and the sub-task holds resources exclusively during a time-slice. Only at the end of every time-slice, the allocated resources can be dropped. Our paper gives attention to the scheduling of MPTs when those tasks have nonlinear speedup and they support the time-slice.

The originality and novelty of this work is emphasized by the following contributions:

1) We give a MPT model supporting time-slice under the Cloud environment;
2) We give the reference parallelism from the system load;
3) We propose a schedule method for the MPTs based on the time-slice based on the reference parallelism;

The associate editor coordinating the review of this manuscript and approving it for publication was Jeonghwan Gwak.

4) Comparisons are given to test our method to check the performance in average waiting time, average execution time, and percentages of unfinished tasks.

The rest of the paper is organized as follows. Section II provides a literature review of scheduling models and scheduling methods for parallel tasks. Section III gives the system model. Section IV addresses our proposed system architecture, MPTs models and so on. Section V describes a scheduling method for MPT in the Cloud. Section VI describes the simulation performance of our proposed methods and existing methods. We conclude our study and further work in Section VII.

II. RELATED WORK

Parallel tasks always reduce the execution time and they are widely used in different areas, such as weather forecasting, stoichiometric calculation, and so on. The data explosive growth makes the parallel task more important that shortens the execution time of applications with big data.

Most of past researchers used DAG (Directed-Acyclic Graph) to model tasks, and they proposed methods from the scheduling of the DAG. Each task is a DAG having a set of subtasks (i.e., nodes) with precedence constraints (i.e., directed edges). The scheduler must complete the execution of all its sub-parallel-tasks by some specified deadlines. They tried to smartly select the execution route to reduce execution time. Pathan *et al.* [11] proposed two-level preemptive GFP (Global Fixed-Priority scheduling) policy: a task-level scheduler first determines the priority of ready tasks and a subtask-level scheduler decides the execution orders of subtasks. Yue *et al.* [12] used a dynamic DAG scheduling method considered the critical path and depth to enhance the scheduling performance. Bhatti *et al.* [13] proposed LeTS (Locality-aware Task Scheduling) for homogeneous multi-core systems. The LeTS heuristic took account of locality and load balancing to reduce the execution time of target applications. Wang *et al.* [14] proposed CPLMT (Cloud Parallel scheduling based on the Lists of Multiple Attributes) for the parallel tasks, which consider the requirements of jobs to different attributes: bandwidth, memory, computing speed, and so on. Li [15] focused the problem of non-clairvoyant scheduling of independent parallel tasks on multiple multicore processors. For a single multicore processor, they derived an asymptotic worst-case performance bound for a non-clairvoyant offline scheduling algorithm, called LTF (Largest Task First). Generally, those methods mainly tried to select of the execution route to reduce the execution time.

Besides reducing execution time of tasks, other researchers also tried to consider others scheduling targets, such as the cost [16], QoE (Quality of Experience), energy consumption [17], reliability [18], load balance, and so on. Our research mainly focuses on the execution time, so we do not introduce them here.

Different to above mention methods, some researchers also give some scheduling methods just from the flexibility of parallel tasks. The parallel tasks can be classified into three kinds according to the flexibility [19], [20]: rigid, moldable, and moldable parallel. The rigid parallel tasks have a parallelism in a set of constants. The parallelism of MPTs even can be changed during execution. For the MPT, the parallelism cannot be changed once it has begun. Ye *et al.* [21] presented a constant competitive online algorithm by applying a black-box reduction from the moldable task scheduling to the rigid task scheduling. Focused on DVFS (Dynamic Voltage and Frequency Scaling) technology, parallelization, real-time scheduling and resource allocation techniques, Zahaf *et al.* [22] defined the scheduling of MPTs as the optimization problem of an INLP (Integer Non-linear Programming) problem. Wu and Loiseau [23] used a greedy algorithm to satisfy multiple targets of MPTs: social welfare maximization, machine minimization, minimize weighted completion time of tasks. Chen [24] presented an iterative method for improving the performance of scheduling MPTs, which aimed to find a feasible schedule with the minimization of makespan. Those methods always consider the DAG of tasks, and try to select execution route to ensure their scheduling targets. Javanmardi *et al.* [25] and Shojafar *et al.* [26] proposed FUGE for scheduling problem in the cloud, which is based on fuzzy theory and a GA (Genetic Algorithm). FUGE aimed to perform optimal load balancing considering execution time and cost. They modified the SGA (Standard Genetic Algorithm) and use fuzzy theory to devise a fuzzy-based steady-state GA in order to improve SGA performance in term of makespan. Besides that, FUGE algorithm assigns jobs to resources by considering multiple attributes of VM (Virtual Machine), such as processing speed, VM memory, VM bandwidth, and the job lengths.

Considering different values of speedups and the related consumed resources, some new scheduling methods are proposed from the view of speedup of parallel tasks. Suter *et al.* [27] proposed FFDH (First Fit Decreasing Height)-stretch for scheduling of parallel tasks and took it as a 0-1 integer linear programming problem. They used a three-step algorithm to solve the problem: parallel degree allotment, task scheduling and frequency assignment. Marchal *et al.* [29] proposed a practical speedup model for graphs of MPTs, which takes trade-off between tractability and accuracy. They proposed model-optimized variants of the existing algorithms PROPMAPPING and FLOWFLEX, and proved them are 2-approximation algorithms. Hao *et al.* [30] gave an example of speedups under various parallelisms, and based on the speedup, they gave a scheduling method according to the system load and the speedup under different kinds of parallelisms. Map-Reduce also widely used in the scheduling of parallel tasks. Chen *et al.* [31] proposed a new MapReduce Scheduler-BGMRS. The BGMRS utilizes the Bipartite Graph modeling and it can obtain the optimal solution of the deadline-constrained scheduling problem by transforming the scheduling problem into a well-known graph problem: minimum weighted bipartite matching.

Different to prior research, we pay attention to the scheduling of MPTs for the case when they support the time-slice.

Most of past work focuses on parallel tasks which exclusive hold the resources until all tasks have been finished. The time-slice gives the task the chance to be canceled, suspended, and resumed after every end of the time-slice. The parallel sub-tasks is denoted as some MPTs (Cloudlets in Cloudsim [2], [9]) and each task (Cloudlet) has the same execution time-one time-slice. Most of past work ignores the fact that some tasks need much times to execute them, the time-slice gives us a chance to divide one big task into some small sub-tasks. This brings convenience for scheduling tasks. There are many cloud simulation tools, such as Cloudsim [32]–[34], GreenCloud, CloudSched, MDCSim, iCanCloud, and DCSim. Those tools help us to evaluate our proposed method.

III. THE SYSTEM MODEL

Figure 1 is physical structure of a Cloud. Every Host has many PEs (Processing Elements) and a data center has many hosts. Datacenter broker decides how to allocate resources in the Cloud. From Figure 1, the total processing ability of the system TP is:

$$TP = \sum P(N, MN, L_{M,N}) \tag{1}$$

where,

-N is the number of data centers;

-MN is the number of hosts in every data center (every data center may have different numbers of hosts);

-$L_{M,N}$ is the number of PEs in every host (every host may have different numbers of PEs).

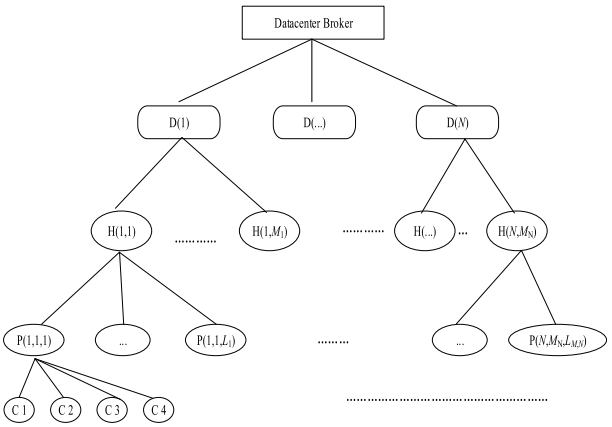

FIGURE 1. Physical structure of cloud.

The computing ability is always denoted by MIPS/s (Million Instructions Per Second).

Figure 2 describes the user view of the Cloud. The user only has right to execute the task on the VMs assigned to him according to the scheduling policy. A Datacenter always handles several hosts and a host always manages multiple VMs. The parameters of a host include: (1) attributes of resources: computing speed, memory, storage; and (2) the provisioning policy (scheduling method) for allocating processing cores (or PEs) to VMs. The Host provides different

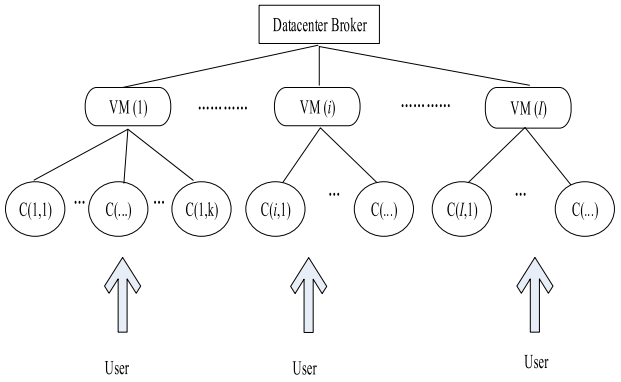

FIGURE 2. User view of cloud.

parameters (memory, hard disk, CPU, etc.) to VMs. The host has two kinds of resource sharing policies for allocating cores to VMs: space-shared and time-shared [2]. In this paper, we assume that there is only one task on a computer node at any time, in other words, those resources are belonging to time-shared.

Fig. 1 and Fig. 2 address the holistic architecture of the system. In fact, the datacenter broker manages the system. It reduces or adds a new VM to the system according to the system load (sometimes, it shuts down some VMs to reduce the energy consumption when it has a lower system load). At the same time, if it finds that the system is overloaded, it adds some hardware to reduce load. And we assume the system has a relative high load, so we do not consider shutting down VMs in our system.

IV. MOLDABLE PARALLEL TASK SUPPORTING TIME-SLICE MODEL

In this section, first, we introduce the MPT model that supports time-slice. Here, we give an example of MPTs-WRF (Weather Research and Forecasting Model) [29]. WRF is widely used in the weather forecast. From Figure 3, we find that the maximum available parallelism of WRF is 48. If the parallelism is less than 48, the execution time would be shortened with the increase of the parallelism; when there are more resources (more than 48) for the task, the execution time would be prolonged. The reason is that the system needs more time to exchange data between different resources when the MPT has a large parallelism (than a constant). Like most of parallel tasks, WRF [29] has a nonlinear speedup.

According to the execution time of WRF under different parallelisms, we model the MPT T_i as:

$$T_i = \left\{ \left(A_i D_i, P_i < pl_{i,j}, e\ t_{i,j} > \right) | 1 \le j \le P_i \right\} \tag{2}$$

$pl_{i,j}$ is the parallelism of the ith tasks of jth parallelism, and $et_{i,j}$ is the execution time of ith tasks when the parallelism is $pl_{i,j}$.A_i and D_i is the arrival time and the deadline of task T_i. For task T_i, suppose that the maximum parallelism is P_i. If the number of assigned resources is less than P_i, with the increase of the number of resources, the execution time would

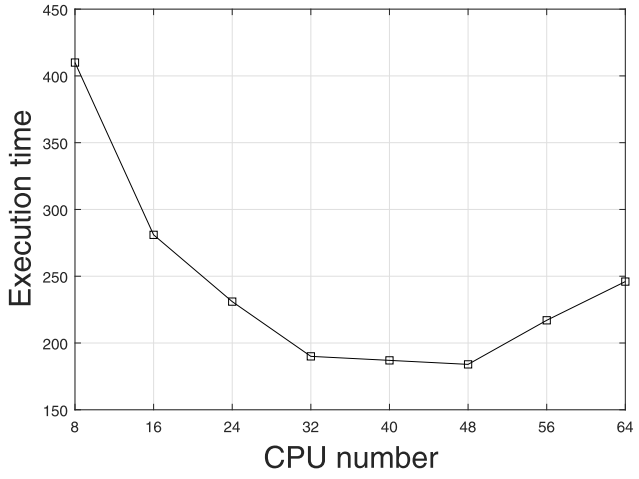

FIGURE 3. Execution time of WRF under different numbers of CPUs.

be reduced; if the assigned resources number is more than P_i, with the improvement of the number of assigned resources, the execution time would be delayed. The reason is, with the increase of the number of assigned resources, more time and more bandwidth need to be consumed between different resources. When the parallelism of the task is $pl_{i,j}$, the relative execution time is $et_{i,j}$.

Because some MPTs have a large value in the execution time, even they are paralleled. So, it is important to support the time-slice. Figure 4 addresses an example of the MPT that supports time-slice. The task has 3∗3 cloudlets (sub-tasks) and is denoted as $c_{i,j}(1 \leq i, j \leq 3)$. A small rectangle means a cloudlet. The black rectangles are cloudlets that have not been completed. The time-slice gives the cloudlet a chance to get the error not until all cloudlets have been finished. It also helps programmer to find the error point during a time-slice. During every time-slice, the cloudlet holds the resource exclusively, until all cloudlets (having same value in i) have been finished. In Figure 4, though those cloudlets $(c_{i,3}, 1 \leq i \leq 3)$ have been completed just after the third time-slice, those cloudlets also hold the three resources until the end of the third time-slice. In fact, the time-slice of tasks is the same and it always is given by the system. In this paper, we assume that the time-slice is sl. Figure 4 also gives us the way to model MPTs on Cloudsim. A task can be denoted by a

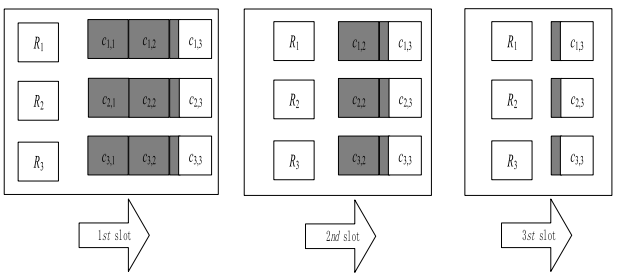

FIGURE 4. Moldable parallel tasks supporting time-slice.

matrix of parallel cloudlets. The cloudlet in the same column must be executed synchronized.

From Figure 4, we can take a task as many small cloudlets. In Figure 4, we can see there are 3∗3 cloudlets in the task. So, under the time-slice is SL, we give the model of the task T_i (when the parallelism is $pl_{i,j}$) as (l is the selected parallelism, m is the identifier of the time-slice):

$$T_i = \{c_i^{l,m} | l \in [1, pl_{i,j}], \quad m \in [1, \lfloor et_{i,j}/SL \rfloor]\} \tag{3}$$

When the task begins to execute, we assume that the deadline of those cloudlets as $d_i^{l,m}$:

$$d_i^{l,m} = D_i - m * SL \tag{4}$$

V. A SCHEDULING METHOD FOR MOLDABLE PARALLEL TASKS ON CLOUD

In the section, we assume that all parallel tasks have a same scope of normalized parallelism [0, 1]. Without considering the deadline, we assume that all parallel tasks are executed on the same parallelism and the system ensures having enough resources to execute all tasks-we call it as reference parallelism (The reference parallelism is not the selected parallelism of tasks).

A. A METHOD DECIDING THE REFERENCE PARALLELISM

In this section, we solve the problem to decide the reference parallelism of those MPTs. In the first step, we do not consider the deadline of tasks and suppose that all parallel tasks are executed under the same reference parallelism.

First, we know that different parallel tasks have different ranges of speedups. Hence, we need to normalize them and to make all of them have the normalized parallelism in the same range of [0, 1].

In formula (2), we assume task T_i has a parallelism of P_i. We set all those parallelisms in the scope of [0, 1], and we use SP_i to denote the selected parallelism used in the execution. So, the selected parallelism RP_i is:

$$RP_i = SP_i * P_i \tag{5}$$

Suppose $et_{i,temp}$ and $RP_{i,temp}$ is the execution time and the parallelism when the parallelism of task is $temp$. The consumed resources CS_i^{temp} of the tasks under the selected parallelism ($temp = RP_i$) is:

$$CS_i^{temp} = RP_{i,temp} * et_{i,temp} \tag{6}$$

Considering the time-slice, according to formula (3), the consumed resources is CSJ_i^{temp}:

$$CSJ_i^{temp} = RP_{i,temp} * \lfloor et_{i,temp}/sl \rfloor * sl \tag{7}$$

Now, without considering the deadline of tasks, we assume that all tasks have the same normalized parallelism (with the same scope in [0, 1]), in other words, those tasks have the same value in SP_i. We call it as the reference parallelism and suppose that it is AP. According to the formulas (1) and (7) (suppose $RP_{i,temp} = AP$):

$$TP = \sum CSJ_i^{temp} \tag{8}$$

TP (Formula 1) is the total resources provided by the Cloud.

From formula (8), we get the average normalize parallelism, and according to formula (5), we get the selected parallelism for every task, and suppose it is APL_i.

We must point out that, because of the difference in deadlines of tasks, we just get a reference parallelism, the selected parallelism may be different to *AP*. And in section 5.2, we give a detail introduction to schedule resources according to the reference parallelism.

B. A METHOD FOR SCHEDULING RESOURCES

First of all, our scheduling method assumes that all tasks are executed on the reference normalized parallelisms. Then, we check two requirements: (1) deadline: we add resources to the task to shorten the execution time only if we have saved enough resources before the scheduling; (2) resource fragment: whether we have a method to reduce more resource fragment than current scheduling. The saved resources (SR_i) refer to the difference between the assigned resources (CSJ_i^{temp1}, *temp*1 is the normalized parallelism) and the resource under reference parallelism *temp*2 (CSJ_i^{temp2}: *temp2* is the normalized parallelism, and the reference parallelism is *AP*):

$$SR_i = CSJ_i^{temp1} - CSJ_i^{temp2} \tag{9}$$

We must point out that, SR_i has a time influence factor TF_i, suppose the time is T:

$$TF_i = \begin{cases} 0 & others; \\ SR_i * \left(\frac{D_i - T}{D_i - A_i}\right) & A_i \le T \le D_i. \end{cases} \tag{10}$$

The number of total saved resources (*TSR*) is:

$$TSR = \sum_{i=1}^{N} (TF_i * SR_i) \tag{11}$$

Algorithm 1 describes the details of the scheduling method. In general, there are two main steps in the scheduling. First, we assume that the task is executed under the reference parallelism (APL_i). We need checking the finish time whether is smaller than the deadline. If it is, we check the parallelism from 1 to APL_i, to select the parallelism which has the smallest resource fragment (first step, lines 8 $\sim$ 17). If it is not, we check the parallelism from APL_i to the maximum parallelism (MAP_i), under the permission of the saved resources before the scheduling, we select the parallelism which first satisfies the requirement to the deadline (second step, lines 19 $\sim$ 25).

We give a detailed explanation of Algorithm 1 here. In line 1, *TSR* records the saving resources. Sometime, we schedule some tasks with a smaller parallelism than the reference parallelism. According to Amdahl's law [34], we would save some resources when the task is executed under a lower parallelism. *lr* (line 2, Algorithm 1; same in the following) is the number of resources that do not have been assigned. Lines 3 $\sim$ 4 calculate the total saving resources which is the gap between the consumed resource and the resources under the reference parallelism. Line 6 gets the reference parallelism (APL_i) according to *AP*. Line 7 checks the resources whether satisfy the system requirement. If it is satisfied, we calculate the finished time (line 8) and the resource fragment (line 9). Line 10 checks whether the execution time satisfies the requirement of the deadline.

If the deadline has been satisfied (lines 10 $\sim$ 18, we check that the parallelism is changed from 1 to the reference parallelism (APL_i) (lines 11 $\sim$ 18), to select the scheduling that

Algorithm 1: *SRP*(N, *AP*) //Schedule MPTs Under Reference Parallelism, N is the Number of Tasks in the Scheduling List, *AP* is the Reference Parallelism

```
1.  TSR = 0; //TSR records the saving resources.
2.  lr = N; // lr is the number of resources that do not have
    been assigned
3.  For i = 1 : N
4.      TSR = TSR + Σ_{i=1}^{N} (TF_i * SR_i);
5.  For i = 1 : N
6.      Get the reference parallelism (APL_i) according to
        the reference parallelism AP;
7.      If (lr ≥ APL_i) // there are enough resources for the
        task to be executed with parallelism APL_i.
8.          Get the finish time of the task (ft);
9.          Get the resource fragment minfr suppose that the
            reference parallelism is AP;
10.         If ft < D_i
11.             For sl = 1 : APL_i // only check the
                parallelism is less than AP
12.                 Get the finish time ft1 when the
                    parallelism of the task is APL_i;
13.                 If ft < D_i
14.                     Get the resource fragment rf when
                        we suppose that the parallelism is sl;
                        If rf ≤ minrf
15.                         minrf = rf;
16.                         selp = sl;
17.         else
18.             For sl = APL_i : MAP_i // only checking the
                normalized parallelism is more than AP
19.                 Get the finished time ft1 when the
                    parallelism of the task is sl;
20.                 If ft < D_i
21.                     Get the total saved resources as
                        formula (11);
22.                     If TSR > 0;
23.                         selp = sl;
24.                         break;
25.     SL_i = sl; // Assigning sl resources to the ith task; in
        other words, the parallelism of the task is sl.
```

has the smallest value in resource fragment (lines 13 $\sim$ 18). If the deadline has not been satisfied (lines 19 $\sim$ 25), we check that the parallelism which is changed from the reference parallelism to the maximize parallelism (lines 19 $\sim$ 25), and get the parallelism that which first satisfies the deadline as the selected parallelism (lines 21 $\sim$ 25). Line 26 assigns sl resources to the ith task.

Algorithm 1 just gives the method to obtain the parallelism of every task. Figure 4 shows that a task consists of many cloudlets. We can schedule those cloudlets sequential. But this may bring scheduling problem. We know that, to get an accurate speedup is very difficult, especially under the Cloud and the big data environment. Many tasks need much time (many time-slices) to execute them. If the task cannot be executed as our expectation before the deadline, this brings bad QoE to the user and reduces the system profit. So, we need an urgency-based scheduling method for cloudlets. Seeing from Figure 4, the last time-slice, if the task is not supposed as our expectation and has more instructions to be finished, we should give more time and resources to execute them. So, we give an emergence fact to every task, which would have a higher priority in the execution. In the next section, we would give a dynamic priority to every cloudlet.

C. A METHOD DECIDES THE PRIORITIES OF TASKS

As mentioned above, the factors deciding the priority of the task include: the time to deadline, the parallelism, the load of the system. Because of the dynamic of the load, we judge the load by the time of the end of every time-slice. For Figure 4, we can get the system load by the end of every time-slice.

We assume that every task is executed under the reference parallelism. Suppose that the time is nst, it is the end of the nth time-slice. We assume that every task average assigns the load to the assigned resources from the arrival time to the deadline. Algorithm 2 gives the method to calculate the system load of the time (the nstth time-slice).

Algorithm 2: sysload(nst) //Get the System Load of the nstth Time-Slice

1. $SJ = []$; //SJ are the tasks that arrive before the nth time-slice and have the deadline that is more than the nth time-slice, there are N tasks in SJ;
2. $nowt = nst * SL$;
3. **For** $i = 1 : N$
4. $\quad$ **If** $(D_i \geq nowt)$ and $(A_i \leq nowt)$
5. $\quad\quad$ Add D_i to SJ;
6. $sysload = 0$;
7. **For** $i = 1 : \text{len}(SJ)$
8. $\quad sysload = sysload + (APR_i * TAPR_i)/(N * (D_i - A_i))$;

We introduce Algorithm 2 in details in the following. SJ records the tasks that which arrive before the nth time-slice and the deadline is more than the nth time-slice (line 1). Lines 3 $\sim$ 5 get the task in SJ. $sysload$ is the system load of the time (nstth time-slice). APR_i and $TAPR_i$ are the reference parallelism and the relative execution time when it is executed under reference parallelism. In line 8, we get the system load. len(SJ) gets the number of tasks in SJ.$(APR_i * TAPR_i)/(N * (D_i - A_i))$ is the load given by ith task to the current time-slice.

We suppose that the priorities of cloudlets have a scope of [0, 1]. When task T_i comes, first according to Algorithm 1, we get the reference parallelism APL_i (according to the normalized reference parallelism AP), and the related working time is TAP_i (when the parallelism is APL_i), so, we need NL time-slices to execute the task (formula 12):

$$NL = TAP_i / SL \tag{12}$$

The task can be denoted as $APL_i * NL$ cloudlets. We suppose that the task at least needs to be completed before ANS time-slice and ∂ percentages of the execution time to its deadline. The priority of the cloudlets of different parallel tasks can be got as Algorithm 3:

Algorithm 3: Cloudpri (J_i, ANS, ∂)

1. $numsl = (D_i - A_i)/SL$;
2. **For** $m = 1 : APL_i$
3. $\quad$ **For** $n = 1 : NL$ // nth time-slice;
4. $\quad\quad numstd = (D_i - A_i)/SL - n$; // numstd is the number of time-slice of the cloudlet to the deadline
5. $\quad\quad \partial 1 = ((TAP_i - A_i) - (\text{NL-n}) * SL)/(D_i - A_i)$;
6. $\quad\quad$ **If** $(numstd > ANS)$ and $(\partial 1 < \partial)$
7. $\quad\quad\quad P_{m,n} = (NL - n)/(D_i - A_i)$;
8. $\quad\quad$ **else**
9. $\quad\quad\quad P_{m,n} = 1$;

The main step of Algorithm 3 is to get the priority of every cloudlet for T_i. Line 1 gets the number of cloudlets that between the arrival time (A_i) and the deadline (D_i). For every cloudlet, line 4 gets the time to deadline D_i; line 5 gets the percentage of the time to deadline. Line 6 checks the leaving time to the deadline whether it satisfies the absolute (deadline) and relative requirement (percentages) to the deadline. If it does, we set the priority as $(NL - n)/(D_i - A_i)$ (line 7); otherwise, we set it to 1 and so that it can get the resources as soon as possible.

D. A METHOD FOR SCHEDULING OF THE PARALLEL CLOUDLETS

As mentioned above, our system models the MPT as many small parallel sub-tasks. In every parallel task, there are many cloudlets (sub-tasks), and those cloudlets should be executed synchronized. We suppose that the execution time of every cloudlet is a time-slice SL, but not include the last cloudlets. Because it is difficult to forecast the execution time without errors, so we should pay more attention to the last cloudlet, especially when the time closes to the deadlines of tasks.

We model those tasks as:

$$CL = \{(clpr_i, cpl_i, cpr_i, cbh_i) | i \leq I\} \quad (13)$$

$$clet_i = (clpr_i, cpl_i, cpr_i, cbh_i) \quad (14)$$

where,

- I is the number of parallel cloudlets;
- $clpr_i$ is the priority of the cloudlet;
- cpl_i is the parallelism of the cloudlet;
- cpr_i are the prior tasks of the cloudlet, if the prior cloudlets have been scheduled, it equals 0;
- cbh_i is the succeed cloudlets of those cloudlets;

We use hbs_i to denote that the $clet_i$ whether has been scheduled. If it equals 1, it has been completed; otherwise, it has not been scheduled. We know that, if cpr_i does not equal 0, the task is not have been scheduled. We give a parameter $cprf_i$ to denote the task whether has been scheduled:

$$cprf_i = \begin{cases} 1 & cpr_i = 0; \\ 0 & cpr_i \neq 0. \end{cases} \quad (15)$$

There are four scheduling targets in our scheduling:

1) the total consumed resources (formula 16): which are the total resources that have been allocated to the cloudlets;
2) the total priority of all scheduled cloudlets (formula 17): which is the total priority of all scheduled cloudlet;
3) the number of finished cloudlets (formula 18);
4) the number of finished instructions (formula 19): which is the total number of instructions of all executed cloudlets.

The four targets are denoted as formulas (16) $\sim$ (19):

$$\text{Max} : \sum cpl_i * hbs_i * cpr_i \quad (16)$$

$$\text{Max} \sum cpl_i * hbs_i * clpr_i * cpr_i \quad (17)$$

$$\text{Max} : \sum clpr_i \quad (18)$$

$$\text{Max} : \sum clpr_i * pl_{i,1} * et_{i,1} \quad (19)$$

Especially, we calculate the number of instructions as the time when the cloudlet has the smallest parallelism as a parameter to judge the system performance. Suppose that I is the total number of resources (the total processing ability). The requirement is:

$$\sum cpl_i * hbs_i * cpr_i \leq I \quad (20)$$

Because hbs_i only has two values: 0 or 1. So, the scheduling problem becomes a 0-1 integer programming problem. And we assume that every target has the same weight in the paper.

Algorithm 4 gives details of scheduling those cloudlets. First, we would schedule those cloudlets that have a priority of 1. Then, for others cloudlets, we would schedule them by using a 0-1 integer programming.

We introduce Algorithm 4 in details in the following. First, we sort all cloudlets CL by the descending order of priorities of cloudlets (line 1, Algorithm 4; same in the following). lrs records the un-assigned resources in line 2.

Algorithm 4: SchCloudlet(I, NVM) //*NVM* is the Number of Resources

```
1.  Sort all cloudlets CL by the descending order of
    priorities;
2.  lrs = NVM; //lrs records the leaving resources
3.  bpos = 0
4.  While (clpr_i == 1) and (lrs − cpl_i ≥ 0)
5.      lrs = lrs − cpl_i;
6.      Allocate cpl_i resources to cloudlet clet_i;
7.  tar1 = ∑ cpl_i * hbs_i*cpr_i; // first target
8.  tar2 = ∑ cpl_i * hbs_i * clpr_i*cpr_i; // second target
9.  tar3= ∑ clpr_i; // third target
10. tar4 = ∑ clpr_i * pl_{i,1} * et_{i,1}; // fourth target
11. ∑ cpl_i * hbs_i*cpr_i ≤ lrs; // requirement
12. Zointp(tar1, tar2, tar3, tar4, lrs, hbs_i)
13. For i = 1:num(hbs)
14.     If hbs_i == 0
15.         Allocate cpl_i resources to the clet_i cloudlet;
```

Lines 4 $\sim$ 6 schedule cloudlets that must be scheduled first. Those cloudlets always have a priority of 1. Lines 7 to 10 are the four targets of our scheduling method. Our scheduling method includes maximizing four targets: the total consumed resources (line 7), the total priority of all scheduled cloudlets (line 8), the number of finished cloudlets (line 9), and the number of finished instructions (line 10). Line 11 is the requirement to the scheduling. In line 12, we used a 0-1 integer programming to schedule those cloudlets and the scheduling result is stored in hbs_i. Lines 14 $\sim$ 15 schedule resources as the result that we have got from line 13.

E. TIME COMPLEXITY

Suppose that the maximum parallelism of tasks is pl, the number of tasks is nt:

1) The complexity of Algorithm 1: for every task, to get the parallelism that which has the minimum resource fragment, so the complexity is O($nt * maxpl$);
2) The complexity of Algorithm 2: the complexity to get tasks in SJ is O(nt), then we can take tasks in SJ as a constant, so the complexity of Algorithm 2 is O(nt);
3) The complexity of Algorithm 3: for the time-slice (we take the number of time-slice of every task as a constant) of every task, Algorithm 3 tries to set the priority, so the complexity of Algorithm 3 is O(nt);
4) The complexity of Algorithm 4: Algorithm 4 would be scheduled at every end of the time-slice, so the number of related cloudlets in one time-slice is decided by the arrival rate, and we take it as a constant, so the complexity of Algorithm 4 is O(1);

So, the complexity of our scheduling method is:

$$O(nt * maxpl) + O(nt) + \text{O}(nt) + \text{O}(1) \quad = \text{O}(nt * maxpl)$$

VI. SIMULATIONS AND COMPARISONS

Algorithms 1 ∼ 4 work together to solve the scheduling problem in the cloud, called MC (a heuristic scheduling method supporting time-slice Model on Cloud). In this section, we evaluate our method-MC on different aspects, such as AET (Average Execution Time), AWT (Average Waiting Time) and PFT (Percentages of un-Finished tasks). We compare our method with FFDH (First Fit Decreasing Height), HEFTP (Heterogeneous Earliest Finish Time for Parallel tasks) and AFCFSP (AFCFS policy for moldable Parallel tasks).

A. SIMULATION ENVIRONMENT

We assume that there are four datacenters (clusters) in the simulation environment, and each of them has 10 hosts. There are 500 VMs computing nodes in every datacenter. The node in one cluster has the same processing speed. The processing ability of every node in different clusters is randomly changed from 80% to 120% of the standard computer node. We assume that the speedup of all tasks under different parallelisms are the same to WRF (In Figure 3), and the length of the Cloudlet (task) is obeyed uniform distribution in the range of [500, 9500] (s) (executed on the standard computer node). It is also the execution time (in seconds) when the task gets 8 computer nodes (VMs) in Figure 3. The deadline of every task is a random number in [0.75, 4] times of the execution time of the task when the task has the minimum parallelism. Those related parameters are listed in Table 1.

TABLE 1. Parameters used in the simulation.

Parameters	Scope	meaning
Datacenters	4	4 datacenters
Hosts/ Datacenters	10	Each datacenter has 10 hosts
VMs/ Datacenters	500	Each host has 500 VMs
Processing ability	[80%, 120%]	Changed from 80% to 120%
deadline	[0.75, 4]	A random in [0.75, 4] times of the execution time under minimum parallelism
Execution time	[500, 9500]	When executed on a standard VM
sl	360,480,720	Time-slice in the simulation is 360 (s), 480 (s), 720 (s).

It is difficult to attain the accurate execution time of tasks, especially under the dynamic cloud environment. So, it is difficult to get the system load. We give a method in Algorithm 2. Like most of forecast system load methods, it is impossible to forecast the system load very accurate. In the simulation, we check our scheduling method under different forecasting accuracies of the load. ''MC100'' means that the Cloud has 100% accuracy to the forecast system load. ''MC 95'' means that the Cloud has 95% forecasting accuracy to the system load. For example, for MC90, when the system has the load of 100, then the system forecast load is a random number in [90, 110]. Similar to ''MC 95'', ''MC 90'' means that the system has 90% accuracy of the forecast system load. In this paper, we would evaluate our scheduling method under 100%, 95% and 90% forecast accuracy of the system. λ is the average arrival rate which has different ranges under different values of the time-slice. We evaluate those methods when the time-slice (in seconds, same in the following) is 360, 480 and 720, the arrival rates are changed from 30 to 32.5 with a step of 0.5, from 40 to 45 with a step of 1, from 50 to 60 with a step of 2, respectively.

We compare our method with other three methods in the simulation. FFDH (First Fit Decreasing Height) [25]-stretch proves that the scheduling of parallel tasks is a 0-1 integer linear programming problem. It uses a three-step algorithm to solve the problem: parallel degree allotment, task scheduling and frequency assignment. In our simulation, the resource in the cluster has the same processing ability, so we need not to frequency assignment. Though HEFT (Heterogeneous Earliest Finish Time) is proposed for the non-parallel tasks, it has been extended to the scheduling of parallel tasks [35], [36], called HEFTP (Heterogeneous Earliest Finish Time for Parallel tasks). HEFTP always executes tasks in the shortest time, so it always allocates as many as possible resources to every task. AFCFS is widely used in many platforms and under many conditions. AFCFS always allocates tasks as the order of arrival time and allocates the least resources to ensure the task can be finished before its deadline, called AFCFSP (AFCFS policy for MPTs). We suppose that AFCFSP always allocates as small as possible in parallelisms to tasks while the execution time satisfies the deadline. In the simulation, we also extend the three methods to ensure supporting time slicing. We compare our method with FFDH, HEFTP and AFCFSP in the simulation.

B. COMPARISONS FROM DIFFERENT ASPECTS

In this section, we give comparisons from the AET, AWT and PFT. The horizontal axis of all figures represents the arrival rates under different values of time-slice.

1) COMPARISONS OF AET

Figures 5 ∼ 7 show that the AETs of different methods under different arrival rates when the time-slice (in seconds) is 360 (s), 480 (s), and 720 (s), respectively.

Generally, all methods have an increasing trend with the enhancement of arrival rates. AFCFSP always has the largest value in AET, followed by FFDH, HEFTP, MC90, MC95, and MC100. MC always has the smallest value in AET, even the forecast system load is 90% (MC90). With the increase of the forecast system load, the value of AET of MC reduces gradually.

We give the average AET of different methods under different arrival rates in Table 2. The average AET (in seconds)

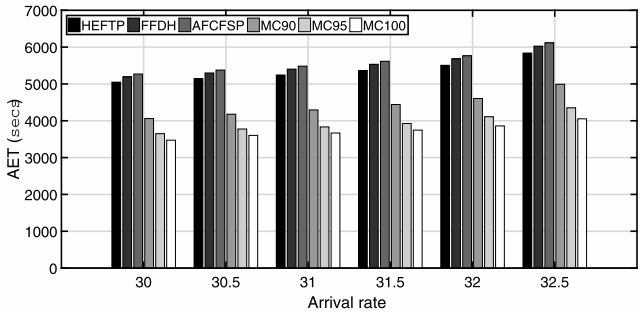

FIGURE 5. AET when $sl = 360$ (s).

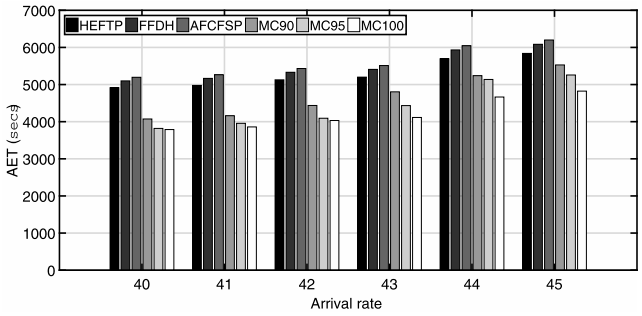

FIGURE 6. AET when $sl = 480$ (s).

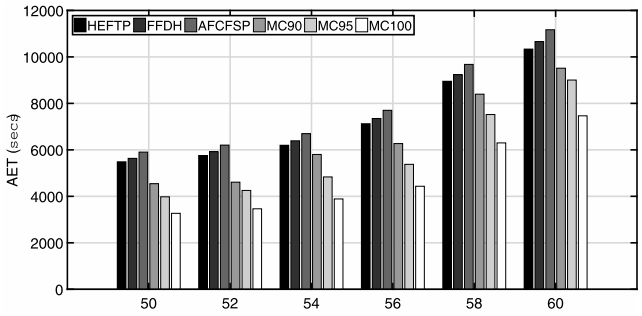

FIGURE 7. AET when $sl = 720$ (s).

of HEFTP, FFDH, AFCFSP, MC90, MC95 and MC100 are 5.9863e + 03, 6.1859e + 03, 7.8924e + 03, 5.2196e + 03, 4.7397e + 03 and 4.3523e + 03, respectively. MC always has a smallest value in AET even the accuracy of the forecasting system load is 90%. Compared to the AET of HEFTP, FFDH, and ASFCFSP, MC 90 reduces 766.7 (s), 966.3 (s) and 2672.8 (s), respectively; MC100 performs better and MC100 reduces 1634 (s), 1833.6 (s) and 3540.1 (s), about 37.54%, 42.13% and 81.34%.

MC always has the lowest value in AETs even we cannot obtain an accurate system load (MC90). MC performs best because it gives the reference parallelism, which ensures the maximum parallelism under the permission of the system load. As we known, a larger parallelism always makes a shorter execution time in the range of the acceptable system load. HEFTP gives better performance than FFDH and AFCFSP, because HEFTP always tries to make every task has the minimum execution time (maximum parallelism). Though AFCFSP always assigns tasks as soon as they come, but it gives the minimum parallelism to the task and that makes the task has the maximum execution time. FFDH has the same problem to AFCFSP.

TABLE 2. Average AET under different time-slice (e + 03).

sl (s)	HEFTP	FFDH	AFCFSP	MC90	MC95	MC100
360	5.36	5.52	5.60	4.43	3.94	3.73
480	5.29	5.50	5.61	4.71	4.45	4.21
720	7.31	7.54	7.55	6.52	5.83	5.11
Avg.	5.99	6.19	7.90	5.22	4.74	4.35

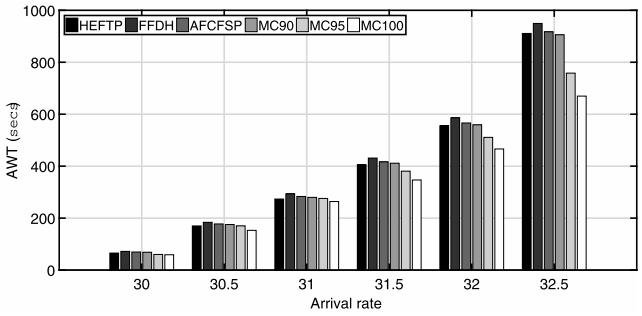

FIGURE 8. AWT when $sl = 360$ (s).

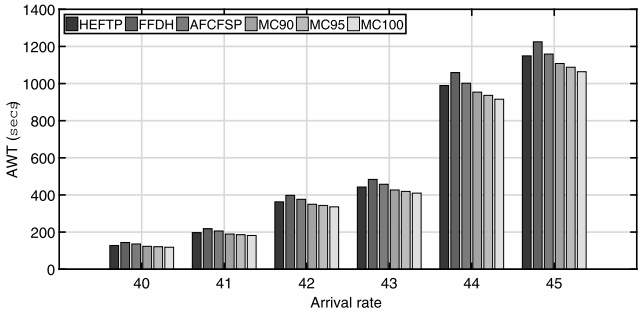

FIGURE 9. AWT when $sl = 480$ (s).

2) COMPARISONS OF AWT

Figures 8 ∼ 9 show that the AWT of different methods under different arrival rates when the time-slice (in seconds) is 360, 480 and 720, respectively. Generally, AWT of all methods are increasing with the enhancement of arrival rates. The order of AWTs from high to low is: AFCFSP, FFDH, HEFTP, MC90, MC95 and MC100. MC has a lowest value in AWT even for MC90.

Table 3 gives the average AWT of different time-slice. The average AWT (in seconds) of HEFTP, FFDH, AFCFSP, MC90, MC95, and MC100 are 1.0557e + 03 (s), 1.1172e + 03 (s), 1.0690e + 03 (s), 1.0476e + 03 (s), 1.0013e + 03 (s) and 961.2096 (s), respectively. MC has the smallest value in AWT. Compared to the average AWT of HEFTP, FFDH,

TABLE 3. Average AWT under different time-slice (e + 03).

sl (s)	HEFTP	FFDH	AFCFSP	MC90	MC95	MC100
360	0.3967	0.4193	0.4051	0.4001	0.3593	0.3264
480	0.5448	0.5879	0.5562	0.5252	0.5153	0.5042
720	2.2255	2.3445	2.2459	2.2176	2.1289	2.0530
Avg.	1.0557	1.1172	1.0690	1.0476	1.0013	0.9612

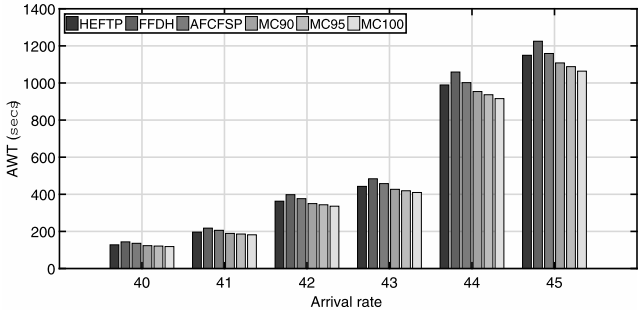

FIGURE 10. AWT when *sl* = 720 (s).

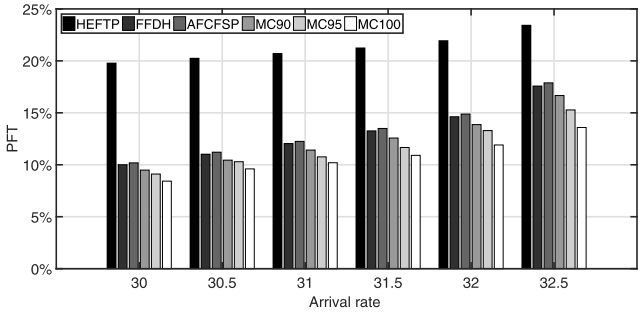

FIGURE 11. PFT when *sl* = 360(s).

AFCFSP, even AWT of MC90 reduces 81 (s), 696 (s) and 214 (s), about 0.77%, 6.64% and 2.24%.

Generally, all methods have a little difference in AWT under the same scheduling condition (time-slice, average arrival rate). FFDH is a little higher because it schedules resource from a global optimization. which would enlarge the waiting time. MC has the smallest value in AWT because the priority in MC is changed with time, when the task has a high value in the waiting time, it enhances the priority of being executed. The accuracy of the forecasting execution time also has a positive effect to the AWT of MC.

3) COMPARISONS OF PFT

Figures 11 $\sim$ 13 show that the PFTs of different methods under different arrival rates when the time-slice is 360 (s), 480 (s) and 720 (s). All methods have an enhancement trend with the increase of arrival rates. The order of PFTs from high to low is HEFTP, AFCFSP, FFDH, MC90, MC95 and MC100. MC always has the smallest value in PFT, even MC90 also has better performance than other methods under the same condition.

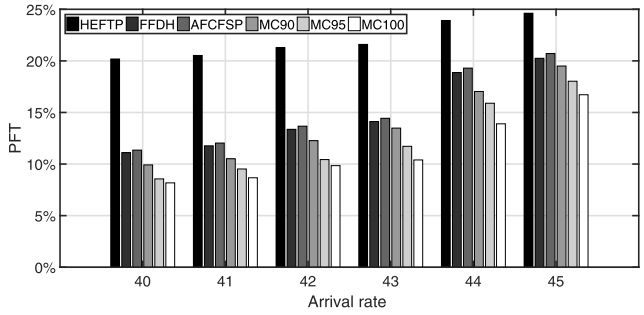

FIGURE 12. PFT when *sl* = 480 (s).

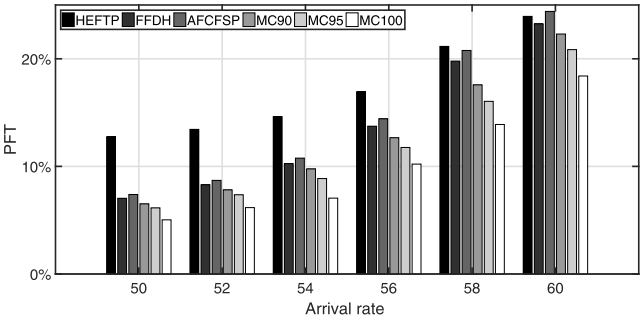

FIGURE 13. PFT when *sl* = 720 (s).

Table 3 gives the average PFTs of different methods. We find that MC always has the smallest values in PFTs. The average PFTs of HEFTP, FFDH, AFCFSP, MC90, MC95 and MC100 are 20.1267%, 13.9067%, 14.3267%, 12.99%, 11.9767% and 10.7267%, respectively. Compared to the PFT of HEFTP, FFDH, AFCFSP, MC100 reduces 9.4, 3.18 and 3.6 percentages, reduces by 87.63%, 29.65% and 33.56%.

MC has the smallest value in PFT because it considers the system load and the deadline. Those two aspects make every task has a higher probability to be executed before its deadline. HEFTP always makes every task has a smallest execution time, it would make every task has a larger parallelism and consume more resources. So, for the future coming tasks, they may not be completed before their deadline. FFDH always gets the first parallelism (Decreasing Height) that satisfies the deadline, and AFCFSP always favors the task with the smallest accepted parallelism, so, they have much difference in PFT. But, the two methods are ignoring the time to the deadline, so they do not perform as well as MC.

4) SUMMARY

Generally, MC has a smallest value in AET and AWT. At the same time, MC also has the lowest value in PFT. Even the forecast accuracy of the system load is 90%, MC also gets a better performance in AWT, AET and PFT. MC performs best because: (1) it considers the system load, and according to the system load, MC gives the reference parallelism, which ensures that every task has a minimum execution time from overall load situation; (2) the priority changed with time and

TABLE 4. Average PFT under different time-slice (%).

sl (s)	HEFTP	FFDH	AFCFSP	MC90	MC95	MC100
360	21.22	13.09	13.32	12.41	11.73	10.77
480	22.02	14.91	15.25	13.79	12.36	11.28
720	17.14	13.72	14.41	12.77	11.84	10.13
Avg.	20.13	13.91	14.33	12.99	11.98	10.73

TABLE 5. Comparison of various metrics.

Metrics	HEFTP	FFDH	AFCFSP	MC90	MC95	MC100
AET	5212	5412	5595	4521	4017	3821
AWT	401	429	437	398	387	360
PET (%)	19.34	12.02	12.25	11.67	11.21	10.94

related to the deadline, which makes every cloudlet has a chance to be executed if the cloudlet waits more time than others. HEFTP just considers the execution time of the scheduled task, and loses considering the system load. FFDH considers the optimizing scheduling and does not consider the deadline. On the contrary, AFCFSP just schedules tasks according to the arrival time. So, the three methods just schedule resources from some aspects.

The accuracy of system load also has a distinct influence to the scheduling result. Higher accuracy always brings better performance in AET, AWT and PFT. Compared to MC90 and MC95, MC100 reduces 867.3 (s) and 387.4 (s) in AET, reduces 86.3904 (s) and 40.0904 (s) in AWT, and reduces by 21.10% and 11.65% in PFT. The reason is that when we cannot get an accuracy forecast system load, it is difficult to decide the execution time. This has a negative effect to the scheduling. Therefore, MC90 and MC95 do not perform as well as MC100.

C. SIMULATION ON CLOUDSIM

In this section, the simulation is executed on the Cloudsim [2], [9], [38], [39]. In the simulation, there are four datacenters (clusters) and each of them has 10 hosts. There are 500 VMs computing nodes in every datacenter and each of them has the same processing speed. The processing ability of every node in different clusters is randomly changed from 80% to 120% of the standard computer node. Other parameters are the same to Table 1. Different to above mentioned simulation, we only evaluate our method when time-slice is 360 seconds and the arrival rate is 31 Per. hour. Table 4 shows that: (1) MC has the best performance in AET, AWT and PET; (2) for MC, MC100 always has the best performance in various metrics.

Compared to the last section, we also find those methods have the same trend in above section. AET, AWT and PET have the same trend to Fig. 5, Fig. 8 and Fig. 11 when the arrival rate is 31, though there is a little difference in values. So, the simulation results also prove the same trend of various methods in the above mentioned simulation.

VII. CONCLUSIONS

In this paper, we focus on the scheduling of MPTs under Cloud environment. First of all, we give the model of MPTs which supports time-slice, and get the reference parallelism according to the average system load. Then, according to the time to the deadline, a method is used to give priorities to the cloudlets. Finally, we give a scheduling method according to the reference parallelism and the priorities. Simulation results demonstrate that our proposed method has good performance in AET, AWT and PFT.

This paper assumes that the system works under a system load, as a future work, we hope that we can propose some scheduling methods that which can adaptive to the change of the system load. As the energy-aware scheduling [37] is a hot topic in the scheduling and Cloudsim supports the DVFS model in the simulation, we would do some work which considers the energy consumption on Cloudsim.

REFERENCES

[1] T. Welsh and E. Benkhelifa, "Perspectives on resilience in cloud computing: Review and trends," in *Proc. IEEE/ACS 14th Int. Conf. Comput. Syst. Appl.*, Oct./Nov. 2017, pp. 696–703.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.

[3] A. Touhafi, A. Braeken, A. Tahiri, and M. Zbakh, "CoderLabs: A cloud-based platform for real-time online labs with user collaboration," in *Proc. 2nd Int. Conf. Cloud Comput. Technol. Appl. (CloudTech)*, May 2017, pp. 317–324.

[4] A. Keivani, F. Ghayoor, and J.-R. Tapamo, "A review of recent methods of task scheduling in cloud computing," in *Proc. 19th IEEE Medit. Electrotech. Conf. (MELECON)*, May 2018, pp. 104–109.

[5] A. Kovari and P. Dukan, "KVM & OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE," in *Proc. IEEE 10th Jubilee Int. Symp. Intell. Syst. Inform.*, Sep. 2012, pp. 335–339.

[6] J. T. Mościcki and L. Mascetti, "Cloud storage services for file synchronization and sharing in science, education and research," *Future Gener. Comput. Syst.*, vol. 78, pp. 1052–1054, Jan. 2018.

[7] P. K. Senyo, E. Addae, and R. Boateng, "Cloud computing research: A review of research themes, frameworks, methods and future research directions," *Int. J. Inf. Manage.*, vol. 38, no. 1, pp. 128–139, 2018.

[8] F. Facchinei, G. Scutari, and S. Sagratella, "Parallel selective algorithms for nonconvex big data optimization," *IEEE Trans. Signal Process.*, vol. 63, no. 7, pp. 1874–1889, Apr. 2014.

[9] M. Gherari, A. Amirat, M. R. Laouar, and M. Oussalah, "MC-Sim: A mobile cloud simulation toolkit based on CloudSim," *Int. J. Comput. Appl. Technol.*, vol. 57, no. 1, pp. 72–82, 2018.

[10] R. T. Zaidi, "Virtual machine allocation policy in cloud computing environment using CloudSim," *Int. J. Elect. Comput. Eng.*, vol. 8, no. 1, pp. 344–354, 2018.

[11] R. Pathan, P. Voudouris, and P. Stenström, "Scheduling parallel real-time recurrent tasks on multicore platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, Apr. 2018.

[12] S. Yue, Y. Ma, L. Chen, Y. Wang, and W. Song, "Dynamic DAG scheduling for many-task computing of distributed eco-hydrological model," *J. Supercomput.*, vol. 75, no. 2, pp. 510–532, 2017.

[13] M. K. Bhatti, I. Oz, S. Amin, M. Mushtaq, U. Farooq, K. Popov, and M. Brorsson, "Locality-aware task scheduling for homogeneous parallel computing systems," *Computing*, vol. 100, no. 6, pp. 557–595, 2018.

[14] Q. Wang, R. Hou, Y. Hao, and Y. Wang, "A parallel tasks Scheduling heuristic in the Cloud with multiple attributes," *Ksii Trans. Internet Inf. Syst.*, vol. 12, no. 1, pp. 287–307, 2018.

[15] K. Li, "Non-clairvoyant scheduling of independent parallel tasks on single and multiple multicore processors," *J. Parallel Distrib. Comput.*, to be published. doi: 10.1016/j.jpdc.2018.06.001.

[16] Y. Xin, Z.-Q. Xie, and J. Yang, *A Load Balance oriented Cost Efficient Scheduling Method for Parallel Tasks*. New York, NY, USA: Academic, 2017.
[17] K. Li, "Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment," *Future Gener. Comput. Syst.*, vol. 82, pp. 591–605, May 2017.
[18] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, and K. Li, "Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems," *J. Comput. Sci.*, vol. 26, pp. 344–353, May 2017.
[19] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing* (Lecture Notes in Computer Science), vol. 1291, no. 13. Cham, Switzerland: Springer, 1997, pp. 1–34.
[20] F. A. B. da Silva and I. D. Scherson, "Towards flexibility and scalability in parallel job scheduling," in *Proc. IASTED Conf. Parallel Distrib. Comput. Syst.*, Nov. 1999.
[21] D. Ye, D. Z. Chen, and G. Zhang, "Online scheduling of moldable parallel tasks," *J. Scheduling*, vol. 21, no. 6, pp. 647–654, 2018.
[22] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, and G. Lipari, "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms," *J. Syst. Archit.*, vol. 74, pp. 46–60, Mar. 2017.
[23] X.Wu and P. Loiseau, "Algorithms for scheduling malleable cloud tasks," 2018, *arXiv:1501.04343*. [Online]. Available: https://arxiv.org/abs/1501.04343
[24] C.-Y. Chen, "An improved approximation for scheduling malleable tasks with precedence constraints via iterative method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 1937–1946, Sep. 2018.
[25] S. Javanmardi, M. Shojafar, and D. Amendola, "Hybrid job scheduling algorithm for cloud computing environment," in *Proc. 5th Int. Conf. Innov. Bio-Inspired Comput. Appl. (IBICA)*. Cham, Switzerland: Springer, 2014, pp. 43–52.
[26] M. Shojafar, S. Javanmardi, S. Abolfazli, and N. Cordeschi, "FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," *Cluster Comput.*, vol. 18, no. 2, pp. 829–844, 2015.
[27] F. Suter, F. Desprez, and H. Casanova, "From heterogeneous task scheduling to heterogeneous mixed parallel scheduling," in *Lecture Notes in Computer Science*, vol. 3149, M. Danelutto, M. Vanneschi, and D. Laforenza, Eds. Berlin, Germany: Springer, 2004, pp. 230–237.
[28] H. Xu, F, Kong, and Q. Deng, "Energy minimizing for parallel real-time tasks based on level-packing," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2012, pp. 98–103. doi: 10.1109/RTCSA.2012.10.
[29] L. Marchal, B. Simon, O. Sinnen, and F. Vivien, "Malleable task-graph scheduling with a practical speed-up model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1357–1370, Jun. 2018.
[30] Y. Hao, L. Wang, and M. Zheng, "An adaptive algorithm for scheduling parallel jobs in meteorological Cloud," *Knowl.-Based Syst.*, vol. 98, pp. 226–240, Apr. 2016.
[31] C.-H. Chen, J.-W. Lin, and S.-Y. Kuo, "MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 127–140, Jan./Mar. 2018. doi: 10.1109/TCC.2015.2474403.
[32] H. Ouarnoughi, J. Boukhobza, F. Singhoff, and S. Rubini, "Integrating I/Os in cloudsim for performance and energy estimation," *ACM SIGOPS Operating Syst. Rev.*, vol. 50, no. 2, pp. 27–36, 2017.
[33] E. Rani and H. Kaur, "Study on fundamental usage of Cloudsim simulator and algorithms of resource allocation in cloud computing," in *Proc. 8th Int. Conf. Comput., Commun. Netw. Technol.*, Jul. 2017, pp. 1–7.
[34] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Dec. 2012, pp. 105–113.
[35] S. Pei, M.-S. Kim, and J.-L. Gaudiot, "Extending Amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation," *IEEE Embedded Syst. Lett.*, vol. 8, no. 1, pp. 26–29, Mar. 2017.
[36] R. Carvalho, G. Andrade, D. Santana, T. Silveira, D. Madeira, R. Sachetto, R. Ferreira, and L. Rocha, "Evaluating dynamic scheduling of tasks in mobile architectures using ParallelME framework," in *Proc. Int. Conf. Comput. Sci.*, 2018, pp. 744–751.
[37] D. Machovec, B. Khemka, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, M. Wright, M. Hilton, R. Rambharos, and N. Imam, "Dynamic resource management for parallel tasks in an oversubscribed energy-constrained heterogeneous environment," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, May 2016, pp. 67–78.
[38] S. K. Mishra, D. Puthal, B. Sahoo, S. K. Jena, and M. S. Obaidat, "An adaptive task allocation technique for green cloud computing," *J. Supercomput.*, vol. 74, no. 1, pp. 370–385, 2018.
[39] E. Barbierato, M. Gribaudo, M. Iacono, and A. Jakóbik, "Exploiting CloudSim in a multiformalism modeling approach for cloud based systems," *Simul. Model. Pract. Theory*, vol. 93, pp. 133–147, May 2019.

JIANMIN LI received the M.S. degree from Computer Science Department, Xiamen University, in 2009, and the Ph.D. degree from the Department of Automation, Xiamen University, in 2015. He is currently a Faculty Member with the School of Computer and Information Engineering, Xiamen University of Technology. His research interests include computer vision, machine learning, and pattern recognition.

YING ZHONG received the M.S. degree from the School of Software, Hunan University. She is currently an Advanced Experimenter with the Xiamen University of Technology. Her main research interests include data analysis and information retrieval technology.

XIN ZHANG received the M.S. degree in control theory and control engineering from the Zhejiang University of Technology, Hangzhou, Zhejiang, China, in 2006. She is currently pursuing the Ph.D. degree with the Nanjing University of Information Science and Technology. She is currently a Teacher with Wenzhou Medical University, Wenzhou, Zhejiang. Her main research interests include scientific computing, image processing, and pattern recognition.