Fault-Tolerant Sheduling Algorithms in Clouds

Zhengfang Xin
A0206957U
Electrical and Computer Engineering
National University of Singapore
Singapore, Singapore
e0427425@u.nus.edu

Yicheng Lai
A0206573E
Electrical and Computer Engineering
National University of Singapore
Singapore, Singapore
e0427401@u.nus.edu

***Abstract*—Cloud computing is increasingly become a new norm because of its efficiency, flexibility and strategic edge. However, the reliability and availability are two major concerns in clouds. Therefore, four recent fault tolerant scheduling algorithms are introduced in this report to solve these problems in clouds.**

Keywords—fault-tolerant, scheduling algorithm, clouds

I. INTRODUCTION

Cloud computing is widely deployed in nowadays. As for users, the resource on clouds is no limited and "pay as you go" mode with huge efficiency and flexibility. The security of users' data guarded by providers is out of users' concerns. The maintenance of machines are also handled by professional staffs. However, as for providers, there are a lot of real-time services based on cloud computing, e.g., real-time communication, real-time surveillance and real-time navigation, and they must guarantee correctness without missing deadline. Scheduling algorithms are deployed to guarantee availability of clouds system and fault tolerance strategies are deployed to guarantee safety and timeliness of clouds system. Cloud computing is special because virtualization technology are applied. Virtualization offers clouds the possibility to dynamically reduce or expand resource for maximum utilization. Because of this quality, fault tolerant scheduling algorithms in clouds are designed differently. We will discuss some recent scheduling algorithms for clouds, FESTAL[1], RFTR[2], FASTER[3] and CEFT[4], in this report.

We introduce the papers with uniform notions in this report.

The cloud heterogeneous hosts are represented by infinite set $H = \{h_1,h_2,...\}$, and each of them has different processing power P_i that is measured by Million Instructions Per Second (MIPS). The active hosts set is characterized by $H_a = \{h_1,h_2,...,h_{|Ha|}\}$, $H_a \in H$, while $H_s = H - H_a$ represents the hosts in the sleep status. The virtual machines (VMs) on the host are modeled by a set $V_i = \{ v_{i1},v_{i2},...,v_{|V_i|}\}$ with the different processing power P_{ij}, where the first subscript i indicates the corresponding host.

The rest parameters are task related. The independent and non-preemptive tasks denoted by a set $T = \{t_1,t_2,...,t_n\}$. Each task has three attributes, task size t_{s_i} measured by Million *Instruction (MI), arrival time* t_{a_i} *and deadline* t_{d_i}. *The classic primary-backup (PB) model is applied in which each task has two copies just in case task failure, primary copy* t_i^P *and backup copy* t_i^B. *The tasks* executed on VMs not on directly on hosts. $vm(t_i)$ indicates which virtual machine the task executed on. Similarly, $h(t_i)$ denotes the corresponding task. The ready time on vm_{kl} of the task t_i is $r_{kl}(t_i)$, and the execution time of the task t_i on vm_{kl} is $e_{kl}(t_i) = t_{s_i}/P_{kl}$. The finish time f_i^P and start time s_i^B decide the status of backup copy. If $f_i^P > s_i^B$, the status of backup $st(t_i^B)$ is passive, otherwise, the status is active.

II. ALGORITHMS

A. *FESTAL*

FESTAL is abbreviation of Fault-tolerant Elastic Scheduling algorithms for real-time Tasks in clouds. FESTAL is scheduling algorithms based on PB model, and it has four components, Function scaleUpResources(), Function scaleDownResources(), Primaries Scheduling in FESTAL and Backups Scheduling in FESTAL.

The scaleUp function and scaleDown support FESTAL to adopt for the dynamic feature of clouds, and scheduling frameworks are core components of FESTAL to apply fault-tolerant scheduling algorithm on clouds.

1) Function scaleUpResources()

When there is no available existing execution unit for task, the scaleUp function will be activated. The available host will be searched firstly in active hosts (line 1~line 8). If there is no active host that can accommodate task, then sleep hosts will be searched to accommodate task (line 9~line 21). If it still fails, task will be rejected.

Algorithm 1 Funcation scaleUpResources()

```
1: Select a kind of newVm satisfying the r_new(t_i) + ts_i/P_new + delay < d_i;
2: Sort H_a in a decreasing order by the remaining MIPS;
3: for h_k in H_a do
4:     if h_k can accommodate newVm then
5:         create newVm on h_k;
6:         return newVm
7:     end if
8: end for
9: for h_s in H_s do
10:    Migrate the VM with minimal MIPS to other hosts with the fault-tolerant requirements: 1. The primary and the backup of one task can not allocate on the same host 2. t_i^P and t_j^P can not on the same host if t_i^B and t_j^B overlap.
11:    if h_s can accommodate newVm then
12:        Create newVm on h_s;
13:        return newVm
14:    end if
15: end for
16: Turn on a host on h_new in H_s;
17: if the MIPS of h_new satisfies newVm then
18:    Create newVm on h_new
19:    return newVm
20: else
21:    return NULL
22: end if
```

2) Function scaleDownResources()

FESTAL have function to recycle the low utilization resource. The resource cannot activate all the time. Therefore, FESTAL defines the cancel time T_{cancel} for VMs and the utilization threshold value U_{low} for hosts.

T_cancel of the corresponding VM is updated as follows:

1. When a primary copy t_i^P is scheduled on the vm_{kl}, the cancel Time $vm_{kl}.T_{cancel} = \max\left(f_i^P + T_{idle}, T_{cancel}\right)$ (Fig. 1.(a))
2. When a backup copy t_i^B is scheduled on the vm_{kl}, the cancel Time $vm_{kl}.T_{cancel} = \max\left(f_i^P + T_{idle}, T_{cancel}\right)$ (Fig. 1.(a)); If the primary copy t_i^P fails, $T_{cancel} = \max\left(f_i^B + T_{idle}, T_{cancel}\right)$. (Fig. 1. (b))

The VMs on a host will try to migrate when the processing power of the host is lower than the utilization threshold U_{low}. After migration success, the host will reserve idle power by going to sleep status.

Algorithm 2 Funcation scaleDownResources()

```
for VM v_kl in the cloud system do
  if it reaches the time v_kl.T_cancel then
    Remove v_kl from h_k and cancel if;
    if h_k.utilization <= U_low then
      offTag ← TRUE;
      for v_kl in h_k do
        minTag ← FALSE;
        for h_i in H_a except h_k do
          if h_i can accommodate v_kl & the migration meets fault-tolerant requirements:1. The primary and the backup of one task can not allocate on the same host 2. t_i^P and t_j^P can not on the same host if t_i^B and t_j^B overlap. then
            migTag ← TRUE;
            break;
          end if
        end for
        if migTag == FALSE then
          offTag ← FALSE;
          break;
        end if
      end for
      if offTag then
        Migrate VMs in h_k to destination hosts;
        Switch h_k to sleep status and remove it from H_a;
      else
        Give up the migration operation;
      end if
    end if
  end if
end for
```

3) Primaries Scheduling in FESTAL

The principle of scheduling primaries is "As early as possible". Because the backup copies are always behind the primary copies. Moreover, evenly scheduling primaries on hosts can maximize the number of the overlap for backup copies to reserve resource.

In Algorithm 3., it firstly chooses top $\alpha\%$ hosts with minimum number of primary copies in active hosts (line 1~line 2). The host will be chosen if it has minimum finish time without missing deadline (line 3~line 14). FESTAL will continue to search next top $\alpha\%$ hosts when no available hosts at current time (line 15 ~line 20). If it still fails, the scaleUp function will be activated to search candidate in the sleep hosts (line 21~line 23). Finally, FESTAL allocates t_i^P to the suitable VM, otherwise, rejects it (line 24~line 29).

Algorithm 3 Primaries Scheduling in FESTAL

```
1: Sort H_a in an increasing order by the count of scheduled primaries;
2: H_candidate ← top α% hosts in H_a;
3: find ← FALSE;EFT ← +∞;v ← NULL
4: while !all hosts in H_a have been scanned do
5:   for h_k in H_candidate do
6:     for v_kl in h_k.VmList do
7:       Calculate the earliest finish time EFT_kl(t_i^P);
8:       if EFT_kl(t_i^P) < d_i then
9:         find ← True;
10:        if EFT_kl(t_i^P) < EFT then
11:          EFT ← EFT_kl(t_i^P);
12:          v ← v_kl
13:        end if
14:      end if
15:    end for
16:  end for
17:  if find == FALSE then
18:    H_candidate ← next top α% hosts in H_a;
19:  else
20:    break;
21:  end if
22: end while
23: if find == FALSE then
24:  v ← scaleUpResources();
25:  if v != NULL then
26:    Allocate t_i^P to v;
27:    Update the T_cancel of v;
28:  end if
29: end if
30: if find == TRUE then
31:  Allocate t_i^P to v;
32:  Update the T_cancel of v;
33: else
34:  Reject t_i^P;
35: end if
```

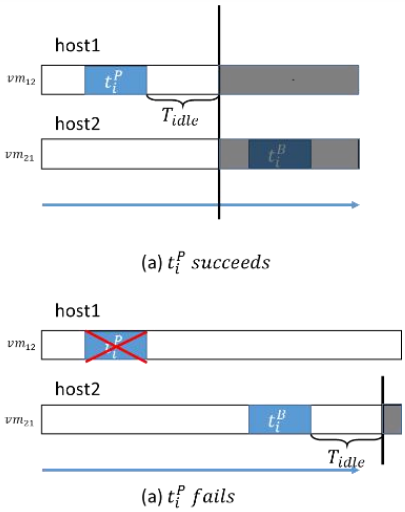


Figure 1. Illustrations of updating T_{cancel}

4) Backups Scheduling in FESTAL

The backups scheduling follows the rule of "as late as possible". Because backups should be passive scheme as much as possible to avoid conflict among backups when they overlap. Secondly, the system should schedule the primary on the VM with maximum T_{cancel} to fully utilize the processor time of T_{idle}. Finally, scheduling on the hosts with few primaries can help reliability of the system.

In Algorithm 4., it firstly sorts candidate hosts by the principle mentioned before (line 1 ~ line 2). Then, the algorithm will search the most suitable VM in candidate hosts with largest latest start time (line 3 ~ line 21). If it fails, the scaleUp function will be activated. Finally, the system will allocate t_i^B on the target VM that found, otherwise reject t_i^B and t_j^P (line 22 ~ line 30).

Algorithm 4 Backups Scheduling in FESTAL

1: $H_{candidate} \leftarrow$ the hosts on which no primaries are scheduled;
2: $H_{primary} \leftarrow$ Sort $H_a - H_{candidate}$ in an increasing order by the count of scheduled primaries;
3: $find \leftarrow FALSE$; $v \leftarrow NULL$; $T \leftarrow MAX$; $LST \leftarrow 0$;
4: **while** !all hosts in $H_{primary}$ have been scanned **do**
5: **for** h_k in $H_{candidate}$ **do**
6: Calculate the latest start time $LST_{kl}(t_i^B)$;
7: **if** $LST_{kl}(t_i^B) + e_{kl}(t_i) \leq d_i$ **then**
8: $find \leftarrow True$;
9: **if** $v_{kl}.T_{cancel} < T \parallel v_{kl}.T_{cancel} == T$ & $LST_k l(t_i^B) > LST$ **then**
10: $T \leftarrow v_{kl}.T_{cancel}$;
11: $LST \leftarrow LST_{kl}(t_i^B)$;
12: $v \leftarrow v_{kl}$;
13: **end if**
14: **end if**
15: **end for**
16: **if** $find == FALSE$ **then**
17: $H_{candidate} \leftarrow$ next top $\alpha\%$ hosts in $H_{primary}$
18: **else**
19: break;
20: **end if**
21: **end while**
22: **if** $find == FALSE$ **then**
23: $v \leftarrow$ scaleUpResources()
24: **end if**
25: **if** $find == TRUE$ **then**
26: Allocate t_i^B to v;
27: Update the T_{cancel} of v;
28: **else**
29: Reject t_i^P; Reject t_i^B;
30: **end if**

5) Example

We define our setting of toy numerical example as follows:

TABLE I. TASK OF FESTAL

Task ID	Task Size (10^5 MI)	Deadline (s)	Arrival Time (s)
t_1	1	400	0
t_2	1	400	210
t_3	1	400	210
t_4	2	800	225

We have two hosts modeled as 2000 MIPS and 3000 MIPS, and three types of VMs with the processing power equivalent to 250 MIPS, 500 MIPS and 1000 MIPS. By following previous works, we define the required time 90 and 15 seconds for turning on a host and creating a VM.

The time of T_{idle} is 200 seconds and T_{cancel} is 400 seconds.

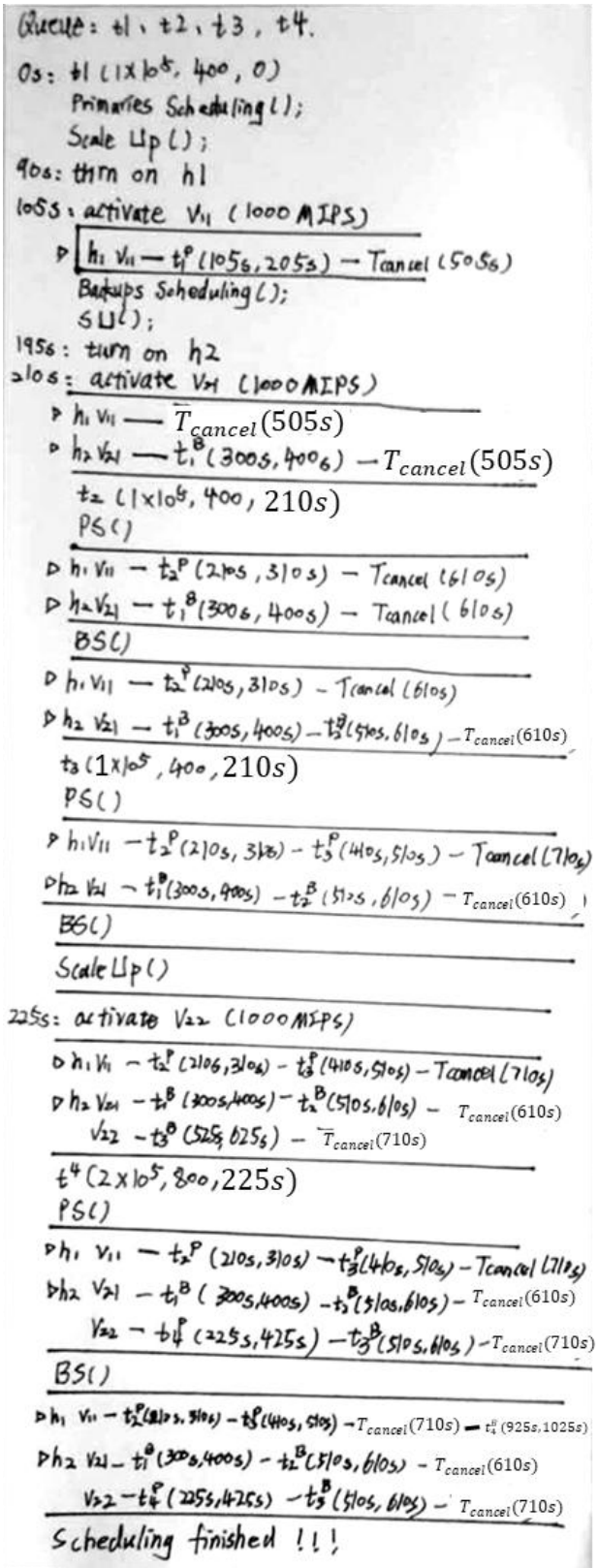


Figure 2. Hand-written example of FESTAL

6) Analysis

FESTAL is first fault-tolerant scheduling designed for clouds. FESTAL uses PB mode to solve the basic elastic scheduling problem by introducing scaleUp and scaleDown functions. All tasks in this algorithm assumed to be independent. However, there are still have specific dependent tasks in cloud applications. Because the sequence of tasks is fixed in FESTAL, the utilization still can be improved during the deallocation.

B. *RFTR*

RFTR is abbreviation of Real-time Fault-Tolerant scheduling algorithm with Rearrangement in clouds. RFTR is scheduling algorithms based on FESTAL[4], the main difference is that RFTR consider the dynamic issues in clouds. In most of scheduling algorithms, they did not explore the executing queue to maximize the utilization, while RFTR will rearrange the waiting sequence to utilize the released resource.

1) Rearrangement

RFTR follows the same rules and has all components in FESTAL[4]. The unique algorithm components in RFTR is Rearrangement triggered by finish of primary copies. The main idea is that some primary copies may queue behind backup copies. At the certain condition, the backup copies are likely to be canceled on vm_{kl} because of their primary executing successfully. Then, the primary copies waiting sequence can be rearranged on vm_{kl} to use vacant time of canceled backups.

Algorithm 1 Pseudocode of Rearrangement

```
for t_i^P on vm_kl do
    startTimeOrigin ← s_i^P;
    finishTimeOrigin ← f_i^P;
    Deallocate t_i^P from vm_kl
    finishTimeNew ← Schedulability(t_i^P, vm_kl);
    if finishTimeNew < finshishTimeOrigin then
        Allocate t_i^P on vm_kl;
        s_i^P ← finishTimeNew − e_jk(t_i);
        f_i^P ← finishTimeNew;
        Update the status of t_i^B;
    else
        Allocate t_i^P on vm_kl;
        s_i^P ← startTimeOrigin;
        f_i^P ← finishTimeOrigin;
    end if
end for
```

2) Example

We design a certain situation to explain how it works.

TABLE II. TASK OF RFTR (CONTINUE TO TABLE I)

Task ID	Task Size (10^5 MI)	Deadline (s)	Arrival Time (s)
t_5	1	600	225

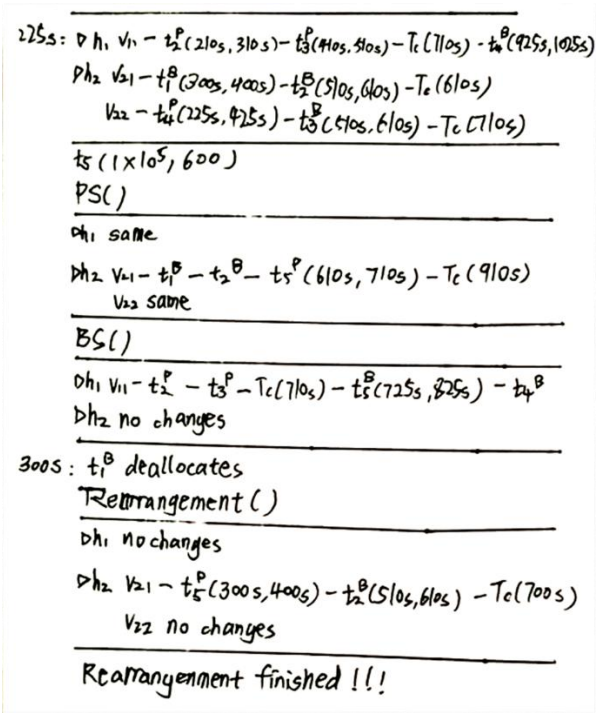


Figure 2. Hand-written example of RFTR

3) Analysis

In RFTR, it considers the limitation of FESTAL that waiting sequence of FESTAL is fixed. When some primary copies are finished successfully, the time slots can be reused to improve the utilization of resource. However, RFTR and FESTAL both only consider independent tasks in clouds.

C. *FASTER*

FASTER, short for Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning, have established a real-time workflow fault-tolerant model on virtualized clouds, extending the PB model by incorporating the cloud characteristics. Task allocation and message transmission strategies are provided to fault-tolerant algorithm. Moreover, overlapping and VM migration mechanisms are applied to achieve both fault-tolerance and high resource efficiency.

The challenge for the previous works is that multiple computational instances' failures may result from a host's failure, then execution failure may happen when 2 copies of a task are allocated to the failed VMs. Furthermore, PB model in the VM migration should take extra constraints into consideration for more complicated conditions.

In this algorithm, it mainly considers dependent tasks, extra constraints like task execution precedence and data transmission order based on PB model, in order to effectively tolerate the faults. Therefore,suitable start time can be find to process as many tasks as possible, which could result in the high overall throughput.

As most processes are similar to FESTAL, the main features which would be discussed for this algorithm are as follows:

1) Model

Directed Acyclic Graph (DAG) is used to model the tasks. It defines **G = {T, E}**, where **T = {t1, t2, ..., tn}** which indicates a set of real-time tasks, assumed to be non-preemptive and **E**: a set of directed edges,which means the dependencies among tasks. $\mathbf{e_{ij} = (t_i, t_j)}$: task tj depends on the data or message generated by task ti. $\mathbf{p(t_i)}$ and $\mathbf{C(t_i)}$ denote its parents and children set respectively. each workflow G has an arrival time **a(G)** and a deadline **d(G).** for a task ti in T, $\mathbf{t_i = (a_i, d_i, s_i)}$ **= (arrival time, deadline time, task size)** , where task size can be measured by MI(Million of Instructions).

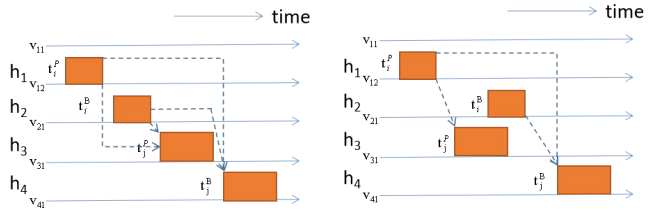
PB(primary-backup) model means each task has 2 copies and they are allocated to 2 different hosts for fault tolerance. It is universally used in the cluster and grid computing and effective in fault tolerance.

Unlimited number of physical hosts are represented by $\mathbf{H=\{h_1, h_2, ...\}}$, providing the hardware infrastructure. $\mathbf{H_a}$ indicates the active host and $\mathbf{P_k}$ means processing capability with respect to the CPU performance in MIPS. For each host hk, $\mathbf{V_k = \{v_{k1}, v_{k2}, ...\}}$, where $\mathbf{R_{jk}}$ is the ready time for VMs.

The fault model mainly focus on host failure which is independent and permanent. Host failures can be test by fault detection mechanism like the fail-signal and acceptance test.

2) Scheduling conditions of tasks' allocation and message transmission.

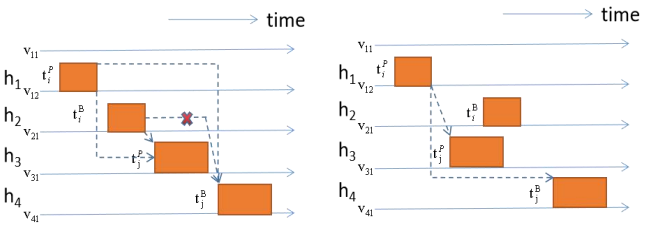
To guarantee the fault tolerance, FASTER have introduced 2 definitions of strong primary copy and Weak primary copy based on if t_j^P can be always executed when $h(t_j^P)$ is operational. Take Fig. C1 as example, t_j^P (a) is a strong primary copy which can receive message when h3 is operational, whereas t_j^P (b) is a weak primary copy.



(a) t_i^P is a strong primary copy (b) t_i^P is a weak primary copy

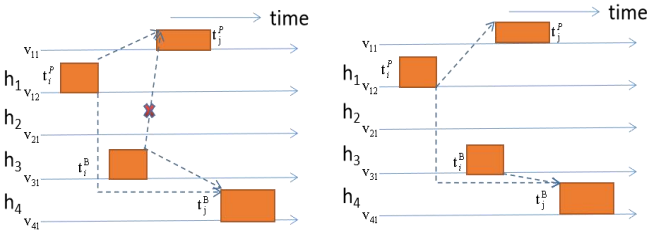
Fig. C1 Examples of strong primary copy and weakprimary copy. The dashed lines with arrows represent messages sent from parents to children.

Cases when $h(t_i^P) \neq h(t_j^P)$ (Fig. C2) and $h(t_i^P) = h(t_j^P)$ (Fig. C3) are displayed below.



(a) t_i^P is a strong primary copy (b) t_i^P is a weak primary copy and t_j^B has no edge with t_i^B

Fig. C2 Examples of $h(t_i^P) \neq h(t_j^P)$



(a) t_i^P is a strong primary copy (b) t_i^P is a weak primary copy

Fig. C3 Examples of $h(t_i^P) = h(t_j^P)$

Based on the definitions and take into consideration various conditions and constraints. This paper[3] have put forward other 4 definitions, 2 lemmas, 7 propositions, 3 theorems and different cases for those theories. Those above include basic constraints dependent tasks in a DAG, VM migration and overlapping mechanisms, which would be used in the algorithm for judgment statements. Each theorem has been mathematically proved in the paper[3].

As the tasks are dependent, BB (backup-backup) overlapping can not be utilized where PB (primary-backup) overlapping is still valid.

3) Backward shifting method

Idle resources may cost some time and space which are not fully used. Then, task backward shifting is proposed to improve this case and execute more tasks earlier than they cost before. It defines a Backward Time Slack which represents how long the we could shift a task backward but don not cause any impacts. Some of the new tasks can be inserted to the idle time between two tasks.

To achieve this goal, the backward time slack for t_j^P can be calculated as:

$$bts_i^P = \min\left\{\min_{t_j \in C(t_i)}(s_j^P - tt_{ij}^{PP} - f_i^P), s_x - f_i^P\right\} \quad \text{(C.1)}$$

And t_j^B can be shifted as:

$$bts_i^B = \min\left\{\min_{t_j \in C(t_i)} c_j, s_x - f_i^B\right\},$$
$$c_j = \begin{cases} s_j^P - tt_{ij}^{BP} - f_i^B, & \text{if } f_i^B + tt_{ij}^{BP} \leq s_j^P \\ s_j^B - tt_{ij}^{BB} - f_i^B, & \text{if } f_i^B + tt_{ij}^{BP} > s_j^P \end{cases} \quad \text{(C.2)}$$

4) Main processes of FASTER

Different form FESTAL, in the primary and backup scheduling processes, dependent tasks are taken into account so that there are quite a lot judgements to divide various cases.

Taking the same example in algorithm A, we assume t1 = p(t2) which means task 1 is the parent task of t2. In other words, the execution of t2 depends on the output of t1 and have to start after t1. Considering the requirements such as start time and deadline, the process of scheduling t2 are displayed in the following part. Other steps are similar to FESTAL since only t2 is set to be the children task of t1.

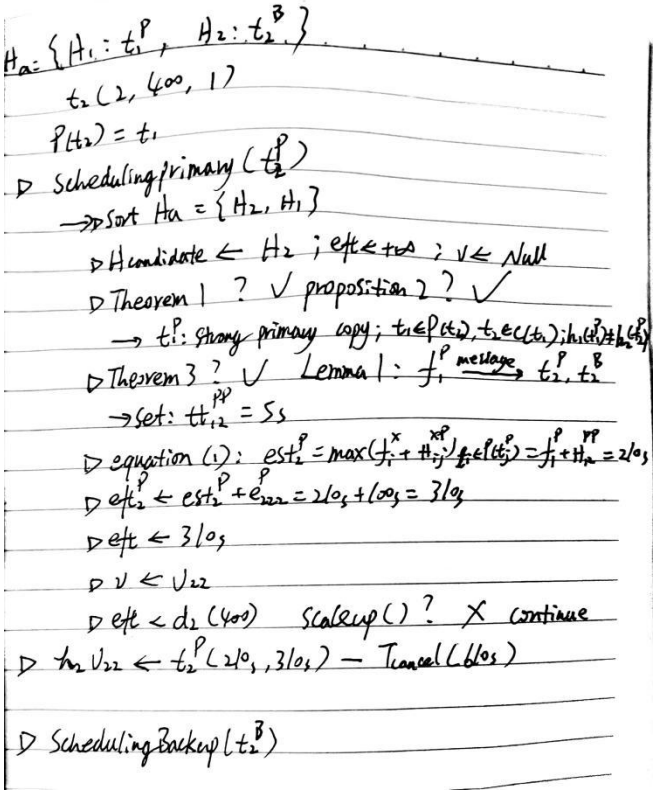


Fig. C4 Example of primary scheduling t2

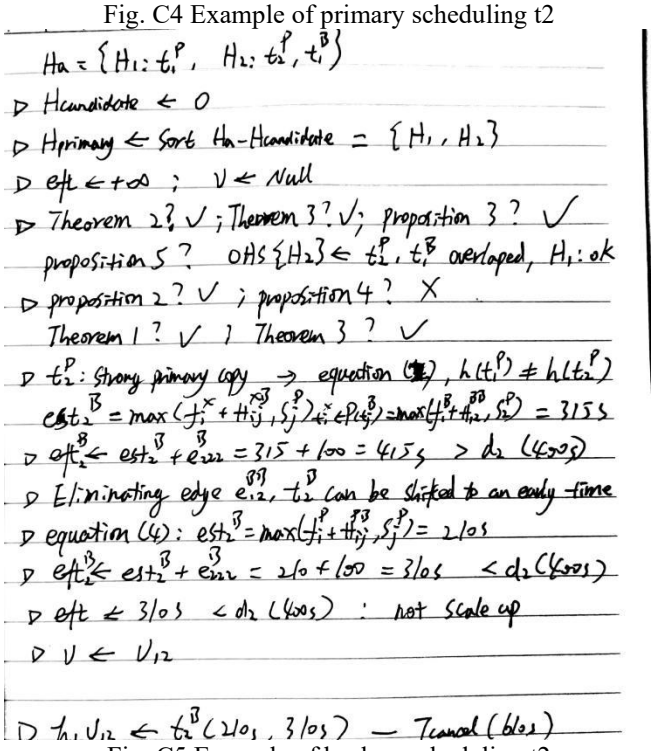


Fig. C5 Example of backup scheduling t2

Primary scheduling is shown in Fig. C4. A few judgements are added before computing the earliest start time such as Theorem 1, proposition 2, Theorem 3 and Lemma 1. As the first task is a strong primary copy, est is computed by equation (2). Then, t2 is primary scheduled

After following the steps to do backup scheduling (Fig. C5), judge the Theorem 1, 2, 3 and proposition 2, 3, 4, 5, we found that in host 2, the existing 2 task are overlapped, then H1 was chosen. Since the eft exceeded the deadline for t2 computed by equation (2), we apply equation (4) to eliminate the BB(backup-backup) edge so that t2-backup can be shifted to an early time. Finally, the result achieved the deadline.

Compared with the traditional works, strict restriction have been relaxed since one missing deadline of a task does not mean the hole workflow would miss its deadline. Hence, if a parent task can not meet the deadline, FASTER will strive to make the children tasks finish on time. If not possible, the workflow will then be rejected and the DAG will be reclaimed.

In this example, we set all the transfer time as 5 seconds.

5) Algorithm Performance

Compared with baseline algorithms and simulated on cloud platform, FASTER with random synthetic workflows are evaluated by Guarantee Ratio(GR), Host Active Time(HAT) and Ratio of Task time over Hosts time(RTH) which display the processing success, resource consumption and the resource utilization respectively.

Observing from the results in paper[3], FASTER shows the highest performance in RTH, lowest HAT and high GR.

Moreover, the impact of DAG count, arrival rate, deadline and task dependence have also been discussed about the performance. Normally, the resource utilization increases with the DAG count. When arrival interval time increases, idle resources tend to be more, resulting in the decreased resource utilization. However, with tight or loose deadline, the algorithms show different performances. High task dependence need more extra execution constraints, causing available tasks become less to VMs although there are some idle VMs. Therefore, resource utilization is reduced. Less resource can will be wasted if more tasks can be executed in parallel due to the lower RTH.

6) Analysis

Merits:

- a real-time workflow fault-tolerant model is established on virtualized clouds and the model is extended by the cloud characteristics incorporation.
- it innovatively defines strong primary copy and weak primary copy to analyse the dependent task allocation and message transmission.
- Idle resources are highly used by the backward shifting scheduling method
- vertical and horizontal scaling up for a burst of workflows and down (avoiding ineffective resource changes due to the fluctuated requests) process contributes to fast resource provisioning. Therefore, the resources could be dynamically changed in terms of the demand of workflows.

Pitfalls

- Only 1 failure is assumed on the hosts. More failures or other type of failures may not get such satisfied results.
- Only data dependent tasks are considered, but the relation among various tasks are more complicated in the real condition.

D. *CEFT*

The aim of Cost-Effective Fault-Tolerant Scheduling Algorithm (CEFT)[4] is to reduce the cost of execution while meeting other requirements like deadline constraints and fault tolerance. Cost-effective is realized by the algorithm discrete PSO (particle swarm optimization). Fault-tolerance is also based on PB model.

1) DPSO

The objective function to evaluate the average execution cost is

$$AvgCost = \frac{1}{n}\sum_{i=1}^{m} C_i(FT_i - ST_i) \quad \text{(D. 1)}$$

Where n: number of tasks, m: number of VMs selected, STi: the earliest start time on vmi, FTi: the latest finish time.

By minimizing the cost function, cost-effective can be achieved and deadline constraints should be added

$$\begin{aligned} &min \quad AvgCost \\ &s.t. \quad f_i^P \le d_i, f_i^B \le d_i, \forall t_i \in T. \end{aligned} \quad \text{(D. 2)}$$

PSO is only used in the algorithm with continuous variables by the formulas below:

$$v_i^{t+1} = \omega_i v_i^t + c_1 r_1 (pbest_i - x_i^t) + c_2 r_2 (gbest - x_i^t). \quad \text{(D. 3)}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad \text{(D. 4)}$$

Then PSO can be converted to DPSO (Algorithm 1) to handle the discrete variables

Algorithm 1 Pseudocode of DPSO

```
Initialize particles with random positions and velocities
while termination criterion is not met do
  for particle i do
    if f(x_i) < f(pbest_i) then
      pbest_i ⟵ x_i
    end if
    if f(x_i) < f(gbest) then
      gbest ⟵ x_i
    end if
    Update x_i and v_i according to Eqs. (4) and (5)
  end for
end while
```

Positions and velocities are initialized randomly at first, then iterates as updating x and v to meet the best condition.

2) Scheduling Process

Since the main purpose is to minimize the execution cost, main body of the scheduling algorithm is similar to those traditional ones, in which, primary scheduling and backup scheduling is to find the earliest start time and earliest finish time. In CEFT, Fitness function (D. 5) is added to the process.

$$Fitness = AvgCost \times \frac{1}{1 + e^{-\sqrt{MR}}} \quad \text{(D. 5)}$$

Where MR indicates the deadline missing ratio.

Through this process, Fitness and the resource consumption R can be computed by the inputs Tasks, initial resource pool R_{init} and position coordinate x.

3) CEFT Algorithm

Since each VM type is of great importance to get the optimal fitness value, CEFT can select the proper number and type of VMs as the initial VM types are the same.

In every iteration, DPSO is applied first and then it calculate the average decreasing number of all VM types. The number of VM type would be increased by % when the number of VM type is less than the average, otherwise, decreased by 10%. Finally, the fitness value would be optimized.

4) Algorithm Performance

The performance of CEFT is evaluated with the impact of task count, task length and deadline. With the task count increasing, guarantee ratio(GR) stays stable since the initial resource pool is selected. CEFT turn out to cost lowest resources after utilizing the iteration method by reducing the needed number of VMs.

While ranging the tasks length, CEFT can still achieve high guarantee ratio but more tasks are rejected as continuously increasing the task length. However, compared with CEFTNI and CEFTNR, CEFT still performs best. Under different deadline, for example, it is obvious that more tasks are not able to finish as deadline become tight. CEFT can reduce cost and optimize resource consumption under these situations as well.

5) Analysis

Merits:

- Cost-effective algorithm can be utilized in other scheduling processes. The iteration can effectively reduce resource utilization given the sets of schedule, hosts and VMs.

Pitfalls:

- Initial resource pool is selected at first, hence, the resources is limited at the beginning which can not be dynamically adjusted such as the scale-up and scale-down process in other scheduling algorithms.

III. CONCLUSION

As many nodes will be deployed in clouds for a real-time workflow execution where deadlines should be satisfied, the reliability is of great significance and many improvements and researches have been proposed. Tasks on clouds are allocated to VMs instead of directly to physical hosts, VMs as basic computational instances should be allowed to migrate among multiple hosts. Therefore, two main issues - fault-tolerance and tasks allocation are focused in our research.

This essay have discussed 4 novel algorithms, FESTAL, RFTR, FASTER and CEFT which are popular and universally utilized in fault-tolerance scheduling for virtuallized clouds. These algorithms have achieved different goals under various considerations.

FESTAL is the first classical algorithm that apply dynamic resource utilization, PB fault-tolerant model to real-time elastic scheduling on virtualized clouds.

Based on FESTAL, RFTR proposed a rearrangement process to increase the resource utilization can reduce idle resources by rearranging the scheduler.

Then CEFT mainly dealt with the resource consumption problem, successfully reducing the cost by DPSO iterating process.

To go further, most merits of the above algorithms are included in FASTER which have achieved both stable fault-tolerance and effective resource utilization with dependent tasks constraints. FASTER introduced backward shifting to make full use of the idle resources and built foundation strategies to analyse dependent task allocation and message transmission conditions.

More complicated constraints for other conditions like multiple failures or more complex task dependence could be search deeper by the future works.

IV. References

[1] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015, doi: 10.1109/TC.2014.2366751.

[2] P. Guo and Z. Xue, "Real-time fault-tolerant scheduling algorithm with rearrangement in cloud systems," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chengdu, 2017, pp. 399–402, doi: 10.1109/ITNEC.2017.8284760.

[3] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-Tolerant Scheduling for Real-Time Scientific Workflows with Elastic Resource Provisioning in Virtualized Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3501–3517, Dec. 2016, doi: 10.1109/TPDS.2016.2543731.

[4] P. Guo and Z. Xue, "Cost-effective fault-tolerant scheduling algorithm for real-time tasks in cloud systems," in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, Chengdu, 2017, pp. 1942–1946, doi: 10.1109/ICCT.2017.8359968.

Appendix

Group Works

Name	Algorithms
Zhengfang Xin	FESTAL, RFTR
Yicheng Lai	FASTER, CEFT

Note: ALL pseudocodes and figures are made by latex and PPT ourselves.