


Article

# Dynamic Fault-Tolerant Workflow Scheduling with Hybrid Spatial-Temporal Re-Execution in Clouds

Na Wu , Decheng Zuo and Zhan Zhang \*

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China; wuna@ftcl.hit.edu.cn (N.W.); zuodc@hit.edu.cn (D.Z.)

\* Correspondence: zz@ftcl.hit.edu.cn

Received: 25 March 2019; Accepted: 24 April 2019; Published: 5 May 2019



**Abstract:** Improving reliability is one of the major concerns of scientific workflow scheduling in clouds. The ever-growing computational complexity and data size of workflows present challenges to fault-tolerant workflow scheduling. Therefore, it is essential to design a cost-effective fault-tolerant scheduling approach for large-scale workflows. In this paper, we propose a dynamic fault-tolerant workflow scheduling (DFTWS) approach with hybrid spatial and temporal re-execution schemes. First, DFTWS calculates the time attributes of tasks and identifies the critical path of workflow in advance. Then, DFTWS assigns appropriate virtual machine (VM) for each task according to the task urgency and budget quota in the phase of initial resource allocation. Finally, DFTWS performs online scheduling, which makes real-time fault-tolerant decisions based on failure type and task criticality throughout workflow execution. The proposed algorithm is evaluated on real-world workflows. Furthermore, the factors that affect the performance of DFTWS are analyzed. The experimental results demonstrate that DFTWS achieves a trade-off between high reliability and low cost objectives in cloud computing environments.

**Keywords:** cloud computing; workflow scheduling; fault-tolerant; re-execution

## 1. Introduction

In recent years, scientific workflow has been applied widely as a new paradigm of data analysis and scientific computation [1]. Scientific Workflow can be regarded as an orchestration form of the relationships between computational tasks and data transmissions [2]. However, the growing computational complexity and data size of scientific workflows increase the demand for a high-performance execution environment [3]. Deploying and running workflows in clouds is an effective solution for the advantages of cloud environments, such as flexible service provision, high cost efficiency and guaranteed Quality of Service (QoS) [4].

Scientific workflows are usually composed of heterogeneous components, including components inherited from legacy applications and newly developed. Clouds offer a customized execution environment for these components through virtualization technology. In addition, workflows can be deployed and executed in clouds that provide a virtually infinite resource pool in a pay-as-you-go manner [5]. In this way, workflows can acquire and release cloud resources on-demand to achieve a cost-effective operating mode. These advantages enable clouds to become a preferred execution environment for scientific workflows.

However, there are also challenges for workflows running in clouds because failures occur more frequently due to the complexity and dynamic of both workflow and cloud [6]. Failures tend to interrupt the continuous execution and significantly affect the performance of workflows, especially for large-scale, long-running ones [7]. To provide customers with seamless experience, it is critical to make scientific workflow scheduling fault-tolerant.

Because many scientific workflows are time-aware, their successful completion depends not only on the correct computational results, but also on the time instants at which these results become available [8]. A deadline limitation is set for a workflow that is a time by which the workflow must complete the processing step and produce the computational result [9]. Without an effective fault-tolerant scheduling scheme, failures will cause deadline-aware workflows cannot complete on time. In this situation, the QoS is severely affected, although the results might be obtained after the deadline. Therefore, fault-tolerant workflow scheduling becomes necessary and important because it makes workflows be able to successfully complete before the deadline.

In failure-prone environments, an effective workflow scheduling approach should not only able to map tasks on to heterogeneous resources, but to deal with failures throughout workflow executions as well. In this paper, we propose a dynamic fault-tolerant workflow scheduling with hybrid spatial-temporal re-execution, called DFTWS. To improve the time efficiency of workflow scheduling, DFTWS separates the scheduling process into initial resource allocation and online workflow scheduling. Meanwhile, DFTWS dynamically implements spatial and temporal re-execution schemes according to the task property and the failure type to achieve a balance between reliability and cost.

The main contributions of this paper are as follows. First, all static information of a workflow is abstracted and calculated in advance, including a set of time attributes and the critical path (CP). Second, a budget division method is designed to divide the total workflow budget into appropriate quotas for all tasks, which are used to rent suitable VM instances. Third, DFTWS scheduling approach is applied through the real-time execution of workflows, which dynamically employs spatial re-execution (SRE) and temporal re-execution (TRE) scheme in different failure scenarios. SRE scheme is adopted for critical tasks or tasks with permanent failures to maintain high reliability, while TRE scheme for non-critical tasks with transient failures to improve cost efficiency.

The remainder of the paper is organized as follows. The related works are reviewed in Section 2. Section 3 describes the preliminaries of fault-tolerant workflow scheduling, including cloud system, workflow model and fault tolerance schemes. Section 4 introduces the proposed DFTWS algorithm. Section 5 presents the experiments on the performance of DFTWS using real-world scientific workflows, and analyze the factors that influence the performance of DFTWS. The conclusions of the paper are given in Section 6.

## 2. Related Work

Scientific workflow scheduling is one of the important issues in cloud computing, which has been widely studied in recent years [10]. Some surveys on workflow scheduling have been made. For instance, the taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments are summarized in [7]. It details the objects, problems, techniques and solutions involved in workflow scheduling. Masdari et al. conduct a comprehensive analysis on workflow scheduling schemes [11]. Additionally, it makes a category of workflow scheduling schemes according to the types and objectives, including metaheuristic-based [12–14], heuristic [15–19] and hybrid metaheuristic and heuristic [20–22] scheduling. A total of 23 scheduling schemes are analyzed and classified based on the proposed classification criteria. These works provide an outline of the problems and solutions of workflow scheduling.

To focus on development and use of scientific workflows, scientific workflow management system is proposed. The Kepler system addresses many of the core requirements and provides support for Web service-based workflows and Grid extensions [23]. Pegasus is an automatic execution simulation and data analysis platform for workflows, which has been widely used by many researchers and originations for more than a decade. Besides the reliable, scalable workflow simulation environment, Pegasus also offers a batch of real-world workflow models and a set of synthetic workflow generators [24]. The simulation platform and workflow dataset provide convenience and benchmark for the researchers on workflow scheduling.

According to the phase of creating and executing scheduling scheme, workflow scheduling can be divided into two types: static and dynamic [25]. Static workflow scheduling completes resource planning on the compile phase, which generally requires preliminary information of workflow and cloud resources. Fard et al. present a general framework and heuristic algorithm for multi-objective static scheduling of scientific workflows in heterogeneous computing environments [26]. Rodriguez et al. propose a static cost-minimization, deadline-constrained heuristic for scheduling a scientific workflow application in a cloud environment [9]. The static scheme requires task information in advance, including their sizes, service demands and estimated execution cost. It is easier for a scheduler to adapt a static scheme. The dynamic scheme makes and executes online scheduling decisions according to the real-time performance of workflows. Shi et al. present a dynamic constraint workflow scheduling algorithm to meet the requirements of the bi-criteria scheduling issue, which uses a two-phase heuristic algorithm for multiple-optimal solution [27]. However, it calls for a longer running time. An elastic resource provisioning and task scheduling mechanism is proposed to perform scientific workflows and submitted at unpredictable time in cloud. It is designed to finish as many high-priority workflows as possible by considering the budget and deadline aspects, which lead to a high cost [28]. Dynamic scheme is more suitable for executing on-demand variable workflows because it doesn't need as much information as static one. However, dynamic scheme becomes more complex since it has to monitor and update the real-time system information [29].

Redundancy is one of the commonly used fault-tolerant mechanisms, which can be realized either in space or in time [30]. Spatial redundancy means providing additional resources to execute the same task on duplication or replication of resources to provide resilience. For example, parallel execution is a typical spatial redundancy scheme. Parallel execution is running a task on multiple resources simultaneously to guarantee a viable result, which results in a high spatial cost [31]. By contrast, temporal redundancy relaxes time constraint to provide more time for re-executing the failed task on the original resources [32]. Re-execution can be either done on another resource or the same resource, that is SRE and TRE, respectively.

To make full use of the advantages of dynamic fault tolerance, DFTWS separates the scheduling process into initial resource allocation and online fault tolerant scheduling. The initial resource allocation scheme can be decided based on the information of workflows and cloud resources before a workflow is executed, and fault handling strategies can be carried out during execution process. Moreover, SRE and TRE are adopted as the fault-tolerant schemes with the aim of improving reliability and resource utilization simultaneously.

### 3. Preliminaries

In this section, we introduce cloud system, workflow model and the fault-tolerant schemes used in this paper. To provide a clear description, the major notations used throughout of this paper are summarized in Table 1.

**Table 1.** Major notations.

Notation	Definition
$t_i$	the $i$ -th task
$ET(t_i)$	Execution time of $t_i$
$CT_{ij}$	communication time between $t_i$ and $t_j$
$CP$	critical path
$BDT(t_i)$	budget quota of $t_i$
$R(t_i, SRE)$	reliability of $t_i$ with spatial re-execution (SRE) scheme
$R(t_i, TRE)$	reliability of $t_i$ with temporal re-execution (TRE) scheme
$VMT(t_i)$	type of VM selected by $t_i$
$pr(VMT(t_i))$	price of $VMT(t_i)$

Table 1. Cont.

Notation	Definition
$dur(VMT(t_i))$	length of service time of $VMT(t_i)$
$cost(t_i, SRE)$	cost of $t_i$ with SRE scheme
$cost(t_i, TRE)$	cost of $t_i$ with TRE scheme
$cost(G)$	total cost of workflow $G$
$IST(t_i)$	Instance start time of $t_i$
$EEST(t_i)$	The earliest execution start time of $t_i$
$LEST(t_i)$	The latest execution start time of $t_i$
$EEET(t_i)$	The earliest execution end time of $t_i$
$RET(t_i)$	Resilient execution time of $t_i$
$FP(t_i)$	Failure probability of $t_i$
$AIST(t_i)$	Actual instance start time of $t_i$
$AEST(t_i)$	Actual execution start time of $t_i$
$AEET(t_i)$	Actual execution end time of $t_i$
$FOT(t_i)$	failure occurrence time of $t_i$
$FRT(t_i)$	failure recovery time of $t_i$
$OVM(t_i)$	Original VM to host $t_i$
$RVM(t_i)$	redundant VM to take over the failed $t_i$
$DM$	deadline multiplier
$FR$	failure rate

### 3.1. Cloud System

Cloud systems generally consist of amounts of distributed heterogeneous resources. Through virtualization technology, a physical machine (PM) can typically hold multiple virtual machines. Virtual machine (VM) is the basic processor unit in cloud systems.

To meet various user requirements, cloud service providers offer different types of VMs at varying prices. For example, Amazon EC2 provides a set of compute optimized C5 VM instances that are suitable for compute-intensive workloads, and there are six specific types as shown in Table 2. Similarly, we define a set of VM types  $VMT = \{VMT_1, VMT_2, \dots, VMT_p\}$  for our cloud system model, which has  $p$  types of VMs.

Table 2. Compute optimized C5 VM instances in Amazon EC2.

Model	vCPU	Processing Capacity (gigaFLOPS (GFLOPS))	Mem (GiB)	Storage	Price (USD/h)
c5d.large	2	384	4	1 × 50 NVMe SSD	0.096
c5d.xlarge	4	768	8	1 × 100 NVMe SSD	0.192
c5d.2xlarge	8	1536	16	1 × 200 NVMe SSD	0.384
c5d.4xlarge	16	3072	32	1 × 400 NVMe SSD	0.768
c5d.9xlarge	36	6912	72	1 × 900 NVMe SSD	1.728
c5d.18xlarge	72	13,824	144	1 × 900 NVMe SSD	3.456

### 3.2. Workflow Model

A workflow is composed by a set of tasks according to their dependencies. Workflow structure indicates the temporal relationship between tasks, which is typically represented by Directed Acyclic Graph (DAG). Each task of a workflow is represented by a node, and each control dependency or data between tasks is represented by a directed edge in the DAG.

In this paper, the DAG of a workflow is defined as a tuple  $G = (T, M)$ .  $T = \{t_1, t_2, \dots, t_n\}$  is the set of nodes in  $G$ , where  $t_i$  denotes the  $i$ -th task. Each task is a set of instructions. The size of a task is measured by millions of instructions  $MI_i$ . The execution time  $ET_i$  of task  $t_i$  depends on its task size and the processing capacity of the VM hosts it.  $M$  is the adjacency matrix of  $G$ ; each entity of  $M$  is a directed edge, which means the precedence of a pair of tasks. The weight of an edge represents the communication time  $CT_{ij}$  from node  $t_i$  to  $t_j$ , which depends on the data size and bandwidth.

$CT_{ij} = \text{datasize}_i / B$ , where  $B$  is the bandwidth between the VMs hosting  $t_i$  and  $t_j$ . If there is no edge from node  $t_i$  to  $t_j$  or they are allocated on the same VM instance,  $CT_{ij} = 0$ :

$$M = \begin{bmatrix} e_{11} & \cdots & e_{1n} \\ \vdots & \ddots & \vdots \\ e_{n1} & \cdots & e_{nn} \end{bmatrix} = \begin{bmatrix} CT_{11} & \cdots & CT_{1n} \\ \vdots & \ddots & \vdots \\ CT_{n1} & \cdots & CT_{nn} \end{bmatrix}. \quad (1)$$

In a given  $G$ , the node without any predecessor is denoted as the entry node  $t_{\text{entry}}$  and the node without any successor is denoted as the exit node  $t_{\text{exit}}$ . There may be more than one entry node and exit node in a workflow. To ensure the uniqueness of the entry node and exit node while not affecting the scheduling, two pseudo nodes  $t_{\text{entry}}$  and  $t_{\text{exit}}$  are added to the beginning and the end of  $G$ , respectively. Since the pseudo nodes have no real tasks, their ETs are 0. The edges between the pseudo and original entry nodes and the original and pseudo exit nodes are zero-weight, thereby, their CTs are 0 as well.

### 3.3. Fault Tolerance Schemes

For scientific workflows running in cloud environments, task failures are inevitable. There are multiple failure causes. Generally, a task failure is caused by the failure of the VM or PM in which the task is executed. Moreover, task failures can occur for some specific reasons, such as unavailable resources, overutilization of resources, execution time exceeds than threshold value and so on [33]. If a task is not able to complete due to its internal or external reason, it is defined as a task failure in this paper.

Two types of failure are considered in this paper, transient and permanent failures. The major difference between them is failure duration. Transient failures (i.e., fail-recovery) are those that occur temporarily and for a short period of time, while a permanent failure (i.e., fail-stop) is one that continues to exist until the failure component is restarted or repaired. A fault-detection mechanism, such as fail-signal or acceptance test [34,35], is used to detect failures.

Task failures might prolong the makespan of scientific workflows, which not only lead to the increase of workforce and time, but also is the potential risk of service-level agreement (SLA) violation. Fault tolerance is one of the most effective fault handling methods. Among various fault-tolerant strategies, re-execution is a common and cost-effective scheme for improving the reliability of workflow scheduling. Moreover, there are two ways to implement re-execution: re-executing on other resources and re-executing on the recovered original resources, which are SRE and TRE. Their comparison is summarized in Table 3.

**Table 3.** The comparison of re-execution schemes.

Re-Execution Scheme	Characteristics
SRE	<ol style="list-style-type: none"> <li>1. Re-execute failed tasks on other resources</li> <li>2. Suitable for permanent faults</li> <li>3. Applied to high timeliness or critical tasks</li> <li>4. Low time cost, high space cost</li> </ol>
TRE	<ol style="list-style-type: none"> <li>1. Re-execute failed tasks on the original resources</li> <li>2. Suitable for transient faults</li> <li>3. Applied to low timeliness or non-critical tasks</li> <li>4. High time cost, low space cost</li> </ol>

#### 3.3.1. Failure Model

In general, the occurrence of a failure is supposed to follow a failure density function  $f(t)$ , which describes the relative likelihood for the failure to occur at a given time:

$$\begin{cases} f(t) = F'(t) = \frac{dF(t)}{dt}, \\ \text{s.t.} \quad f(t) \geq 0, \\ \int_{-\infty}^{+\infty} f(\tau) d\tau = 1, \end{cases} \quad (2)$$

where  $F(t)$  is the failure distribution function, which describes a real-valued random variable  $t$  with a given failure density function  $f(t)$  and will be found at a value less than or equal to  $t$ . The failure distribution function  $F(t)$  is defined by

$$\begin{cases} F(t) = P(-\infty \leq t) = \int_{-\infty}^t f(\tau) d\tau, \\ 1 - F(t) = P(t \leq +\infty) = \int_t^{+\infty} f(\tau) d\tau, \\ F(t_b) - F(t_a) = P(t_a < t \leq t_b) = \int_{t_a}^{t_b} f(\tau) d\tau, \\ \text{s.t.} \quad \lim_{t \rightarrow -\infty} F(t) = 0, \\ \lim_{t \rightarrow +\infty} F(t) = 1. \end{cases} \quad (3)$$

A failure expectation distribution value  $E(t)$  of a continuous failure is a value that describes the weighted average of all possible failure time values that admits probability density function  $f(t)$ , which can be calculated by

$$E(t) = \int_{-\infty}^{+\infty} \tau \cdot f(\tau) d\tau. \quad (4)$$

Poisson, exponential, Weibull, long-normal and uniform distribution are typical failure distribution functions [7]. In this paper, the occurrence of a failure is assumed to follow a Poisson distribution, which is commonly used in related works [36–41]. The reliability of a task in unit time  $t$  is denoted by

$$R(t) = e^{-\lambda t}, \quad (5)$$

where  $t$  is the constant failure rate per time unit for a task.

### 3.3.2. Cost Model

In commercial clouds, the cost model is determined by the payment mode that generally charges users according to the total service time of VMs. The execution cost of workflow is the sum of the execution cost of all tasks:

$$\text{cost}(G) = \sum_{i=1}^n \text{cost}(t_i). \quad (6)$$

For a task without failures,  $\text{cost}(t_i)$  is calculated by

$$\text{cost}(t_i) = \text{pr}(\text{VMT}(t_i)) \times \text{dur}(\text{VMT}(t_i)), \quad (7)$$

where  $\text{pr}(\text{VMT}(t_i))$  and  $\text{dur}(\text{VMT}(t_i))$  are the unit price and service time of the VM instance runs task  $t_i$ .

For a task with SRE scheme,  $\text{cost}(t_i)$  is calculated by

$$\text{cost}(t_i, \text{SRE}) = \text{pr}(\text{OVMT}(t_i)) \times \text{dur}(\text{OVMT}(t_i)) + \text{pr}(\text{RVMT}(t_i)) \times \text{dur}(\text{RVMT}(t_i)), \quad (8)$$

where  $\text{OVMT}(t_i)$  and  $\text{RVMT}(t_i)$  are the original and redundant VMT of task  $t_i$ ,  $\text{dur}(\text{OVMT}(t_i))$  is the duration of task  $t_i$  on the original VM instance before it fails, and  $\text{dur}(\text{RVMT}(t_i))$  is the duration of task  $t_i$  after it restarts on the redundant VM instance.

For a task with TRE scheme,  $\text{cost}(t_i)$  is calculated by

$$\text{cost}(t_i, \text{TRE}) = \text{pr}(\text{OVMT}(t_i)) \times \text{dur}(\text{OVMT}(t_i)), \quad (9)$$

where  $OVMT(t_i)$  is the original VMT of task  $t_i$ , because a task with TRE scheme remains on the same VM instance. However, the duration of task  $t_i$  is prolonged due to the occurrence and recovery of a transient failure.

#### 4. DFTWS Algorithm

This section introduces the proposed DFTWS algorithm, which includes three phases: Static DAG Information Calculation, Initial Resource Allocation and Online Scheduling. In the first phase, the static information of a DAG is calculated based on the information of the submitted workflow and cloud resources. Appropriate VM instances are assigned to each task according to the budget quota in the second phase. The last phase is to dynamically schedule the workflow with hybrid SRE and TRE during the real-time execution.

##### 4.1. Static DAG Information Calculation

In this phase, a group of static node information is firstly calculated, including time attributes and failure probability. Then, the critical path is identified based on the static node information.

##### 4.1.1. Static Node Information Calculation

Fault-tolerant workflow scheduling involves a number of important time attributes of tasks. To facilitate initial resource allocation and online scheduling, the time attributes are calculated in advance.

- **Average Execution Time (AET):** The average execution time of a task on all available VMTs:

$$\begin{aligned} AET(t_i) &= \frac{\sum_{VMT_k \in VMT} ET(t_i, VMT_k)}{|VMT|} \\ &= \frac{\sum_{VMT_k \in VMT} \frac{tasksize(t_i)}{VMT_k}}{|VMT|}, \end{aligned} \quad (10)$$

where  $ET(t_i, VMT_k)$  is the execution time of task  $t_i$  on the VM type of  $VMT_k$ . VMT is the set of available VM types, and  $|VMT|$  is the number of VM types. In most research works on deadline-constrained workflow scheduling, the most powerful host is assigned for a task to minimize its  $ET$ . It helps to achieve a significantly reduce on the makespan of workflows, but calls for a tremendous cost, especially for large-scale workflows. In our model, the  $ET$  of each task is estimated with  $AET$  in this phase, from a comprehensive perspective of cost and deadline.

- **Instance Start Time (IST):** the VM instance of task  $t_i$  should start once there is a predecessor task finish execution because it should be ready to receive the data handed down at that time:

$$IST(t_i) = \min_{t_p \in pred(t_i)} EEET(t_p), \quad (11)$$

where  $pred(t_i)$  is the set of predecessors of task  $t_i$ , and  $EEET(t_p)$  is the earliest execution end time of task  $t_p$ .

- **The Earliest Execution Start Time (EEST):** Once all data needed by  $t_i$  are received from its predecessors, it can start execution. Therefore,

$$\begin{aligned} EEST(t_i) &= \max_{t_p \in pred(t_i)} (EEET(t_p) + CT_{pi}) \\ &= \max_{t_p \in pred(t_i)} (EEST(t_p) + ET(t_i) + CT_{pi}). \end{aligned} \quad (12)$$

- **The Earliest Execution End Time (EEET):** EEET is the end time of a task that starts at its EEST:

$$EEET(t_i) = EEST(t_i) + ET(t_i). \quad (13)$$



- **The Latest Execution Start Time (LEST):** LEST is the latest start execution time of  $t_i$  on the assumption that none of its successors will fail and the entire workflow can just finish at the given deadline  $DDL$ :

$$LEST(t_i) = \max_{t_s \in succ(t_i)} (LEST(t_s) - CT_{is} - ET(t_i)), \quad (14)$$

where  $succ(t_i)$  is the set of successors of  $t_i$ . LEST is calculated in reverse order from  $t_{exit}$  to  $t_{entry}$ . Because  $ET(t_{exit}) = 0$ ,  $LEST(t_{exit}) = DDL$ .

- **Resilient Execution Time (RET):** RET means the time range between EEET and LEST. RET indicates the urgency of a task. A smaller RET means that the task is more urgent to execute:

$$RET(t_i) = LEST(t_i) - EEET(t_i). \quad (15)$$

- **Failure Probability (FP):** Because the possible failure time values that admits probability density function  $f(t)$ , which can be calculated by Equation (16), the failure probability FP is in proportion to its active time.

$$\begin{aligned} FP(t_i) &= \int_{IST(t_i)}^{\max_{t_s \in succ(t_i)} (EEET(t_s) + CT_{is})} \tau \cdot f(\tau) d\tau \\ &\propto \max_{t_s \in succ(t_i)} (EEET(t_i) + CT_{is}) - IST(t_i). \end{aligned} \quad (16)$$

#### 4.1.2. Critical Path Identification

In workflow scheduling, the critical path (CP) is the path with the longest execution duration from  $t_{entry}$  to  $t_{exit}$  of a workflow, also named the longest path. CP is one of the most indicators for workflow scheduling. The makespan of a workflow is determined by its CP. The nodes along CP are critical nodes that are generally the most time-consuming tasks. Any delay or failure of critical nodes would directly impact the completion time of a workflow. Therefore, identifying CP is the basis of an efficient resource allocation mechanism and an effective fault tolerant strategy for workflows.

The critical predecessor of a task  $t_i$  is the predecessor of  $t_i$  that is the last one to finish data delivery to  $t_i$ , that is, the predecessor of  $t_i$  is task  $t_p$  whose  $EEET(t_p) + CT_{pi}$  is maximal. Once the critical predecessor of each task can be identified, the critical path of a workflow can be determined in a recursive manner.

#### 4.2. Initial Resource Allocation

In this subsection, we introduce how to allocate initial resources for all tasks, that is, how to select an appropriate VM type for each task. Due to the total cost limit, it is impractical to choose the most powerful type of VM for each task.

DFTWS takes RET as the reference in VMT selection because the failure or delay of tasks with different RET have different impacts on the overall makespan of a workflow. A smaller RET means a task is more urgent; thus, it needs a more powerful VM to complete execution quickly, which calls for a higher budget quota. Otherwise, a task with enough resilient time to execute, a lower budget quota should be assigned for it to rent an affordable VMT. We assume a total budget  $BDT$  is set for a workflow. The calculation of  $BDT(t_i)$  is as follows:

$$BDT(t_i) = \frac{1}{RET(t_i)} \times \frac{1}{\sum_{i=1}^n RET(t_i)} \times BDT. \quad (17)$$

Once the BDT of a task is obtained, we can select a suitable VMT for it according to the duration of the task and the prices of VMT candidates.  $VMT(t_i)$  is determined by

$$\begin{cases} VMT(t_i) \in \{VMT_1, VMT_2, \dots, VMT_p\} \\ \text{s.t. } \max_{t_s \in succ(t_i)} (EEET(t_i) + CT_{is} - AIST(t_i)) \times pr(VMT(t_i)) \leq BDT(t_i), \end{cases} \quad (18)$$



where the *s.t.* expression indicates that the total expense of  $t_i$  should be no more than its budget quota, which supports it running on  $VMT(t_i)$  for the period of task execution and data transmission. Because the VM instance of  $t_i$  cannot be shut down immediately after the execution finishes, it needs to deliver data to its successors as well. If there are multiple VMTs that satisfy Equation (18), the VMT with the highest price is selected for  $t_i$  because a higher price refers to a better performance for a VMT. This manner of VMT selection is to reduce execution time as much as possible within its budget quota. The pseudo code of initial resource allocation is shown in Algorithm 1.

---

**Algorithm 1** Initial Resource Allocation
 

---

**Require:** the workflow budget  $BDT$  and the VMT set  $\{VMT_1, VMT_2, \dots, VMT_p\}$ ;

**Ensure:** the initial resource allocation for each task

```

1: for  $i = 0$  to  $n + 1$  do
2:   calculate  $BDT(t_i)$  with Equation (17);
3:   if multiple  $VMT(t_i)$  satisfy Equation (18) then
4:     select the  $VMT(t_i)$  with maximum  $pr(VMT(t_i))$  for  $t_i$ ;
5:   end if
6: end for

```

---

#### 4.3. Online Scheduling

During workflow execution, DFTWS dynamically deploys FT scheme for each task. Algorithm 2 illustrates the pseudo code of online scheduling. The first step is failure judgment. If there is no failure occurs to the current task, it is executed on the designated VM by Algorithm 1. Otherwise, FT scheme selection should be performed. DFTWS assigns priority to a task which is a critical node or experiences a permanent failure because of their significant impact on the makespan of a workflow. Whether this is a critical node and whether there is a permanent failure that happens to this node are denoted as two conditions for FT scheme selection. The SRE scheme is applied as long as one of the above two conditions is met. Otherwise, the TRE scheme is performed.

As for the SRE scheme, because different failure occurrence times (FOT) have different impacts on the time attributes of a task; thereby, it needs different specific SRE operations, and results in different execution costs. There are three cases for SRE as follows:

- Case1 (lines 4–6):  $FOT(t_i) < IST(t_i)$ , the failure occurs before the instance of  $t_i$  starts. In this case, the original VM instance fails before  $t_i$  starts on it; thus, a redundant VM instance is allocated for  $t_i$ . None of the time attributes is impacted. The execution cost of  $t_i$  is calculated as follows:

$$cost(t_i) = pr(RVMT(t_i)) \times dur(RVMT(t_i)), \quad (19)$$

$$dur(RVMT(t_i)) = \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is} - IST(t_i)), \quad (20)$$

where  $RVMT(t_i)$  is the redundant VM takes over the failed task  $t_i$ .

- Case2 (lines 7–10):  $IST(t_i) \leq FOT(t_i) \leq \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is})$ , the failure occurs during the data transmission or execution of  $t_i$ . In this case, a redundant VM instance starts to take over  $t_i$  at  $FOT(t_i)$ . Because data transfer to  $t_i$  has started or finished on the original VM instance, data retransmission is needed. Therefore,  $IST(t_i)$ ,  $AEST(t_i)$ ,  $AEET(t_i)$  are impacted.  $IST(t_i)$  is delayed to  $FOT(t_i)$ .  $AEST(t_i) = IST(t_i) + \max_{t_p \in pred(t_i)} CT_{pi}$ .  $AEET(t_i) = AEST(t_i) + ET(t_i, RVMT(t_i))$ . The execution cost of  $t_i$  is calculated as follows:

$$cost(t_i) = pr(OVMT(t_i)) \times dur(OVMT(t_i)) + pr(RVMT(t_i)) \times dur(RVMT(t_i)), \quad (21)$$

$$dur(OVMT(t_i)) = FOT(t_i) - OIST(t_i), \quad (22)$$

$$dur(RVMT(t_i)) = \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is} - FOT(t_i)). \quad (23)$$

- Case3 (lines 11–13):  $\max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is}) < FOT(t_i)$ , the failure occurs after all data transfer from  $t_i$  to  $t_s$  finish. There is no harm to task  $t_i$ . The execution cost of  $t_i$  is calculated by

$$cost(t_i) = pr(OVMT(t_i)) \times dur(OVMT(t_i)), \quad (24)$$

$$dur(OVMT(t_i)) = \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is} - FOT(t_i)). \quad (25)$$

---

**Algorithm 2** Online scheduling
 

---

```

for  $i = 0$  to  $n + 1$  do
2:   if  $t_i$  fails then
      if ( $t_i$  is a critical node) or (this is a permanent failure) then
4:       if  $FOT(t_i) < IST(t_i)$  then
            start a redundant node for  $t_i$  at  $FOT(t_i)$ ;
6:       calculate  $cost(t_i)$  using Equations (19) and (20);
      else if  $IST(t_i) \leq FOT(t_i) \leq \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is})$  then
8:       start a redundant node for  $t_i$  at  $FOT(t_i)$ ;
            update  $AIST(t_i)$ ,  $AEST(t_i)$  and  $AEET(t_i)$ ;
10:      calculate  $cost(t_i)$  using Equations (21)–(23);
      else if  $\max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is}) < FOT(t_i)$  then
12:      start a redundant node for  $t_i$  at  $FOT(t_i)$ ;
            calculate  $cost(t_i)$  using Equations (24) and (25);
14:      end if
      else
16:      if  $IST(t_i) - rt \leq FOT(t_i) \leq \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is})$  then
            wait failure node to recovery and re-execute;
18:      update  $AEST(t_i)$  and  $AEET(t_i)$ ;
      else
20:      no TFT operation;
      end if
22:      calculate  $cost(t_i)$  using Equations (26) and (27);
      end if
24:      else
            no FT operation;
26:      end if
      end for

```

---

Likewise, different FOTs have different impacts on the time attributes of the task for the TRE scheme; thus, they need different specific TRE operations, and result in different execution costs. Unlike permanent failures, transient failures can automatically recover after a short period of time. Let  $rt$  denotes the length of time required by a transient failure to recovery. In addition, we assume a unified  $rt$  for all tasks, which is commonly used in the research of transient failures [42]. There are two cases for TRE as follows:

- Case 1 (lines 16–18):  $IST(t_i) - rt \leq FOT(t_i) \leq \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is})$ , the failure occurs during the execution or data transmission of  $t_i$ , that is, the transient failure recovers after the instance of  $t_i$  starts, and before all data transfer from  $t_i$  to  $t_s$  finish. In this case, the VM instance fails and then recovers, so task  $t_i$  requests data retransmission from all its predecessors.

Therefore, its  $IST(t_i)$  does not change,  $AEST(t_i)$  is delayed to  $FOT(t_i) + rt + \max_{t_p \in pred(t_i)} CT_{pi}$ , and  $AEET(t_i) = AEST(t_i) + ET(t_i, VMT(t_i))$ .

- Case 2 (lines 19–20):  $FOT(t_i) < AIST(t_i) - rt$  or  $\max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is}) < FOT(t_i)$ , the transient failure does not occur during the data transmission or execution of  $t_i$ , that is, the failure occurs and recovers before the VM instance of  $t_i$  starts, or after all data transfer from  $t_i$  to  $t_s$  finish. In this case, none of the time attributes are impacted.

Transient failure only affects the time attributes of  $t_i$ , whereas  $t_i$  remains running on the same VM instance. Therefore, the execution cost of  $t_i$  in these two cases of TRE scheme can be calculated as follows, which corresponds to line 22:

$$cost(t_i) = pr(VMT(t_i)) \times dur(VMT(t_i)), \quad (26)$$

$$dur(VMT(t_i)) = \max_{t_s \in succ(t_i)} (AEET(t_i) + CT_{is} - AIST(t_i)), \quad (27)$$

where  $AEET(t_i)$  is determined by the specific failure scenarios described above.

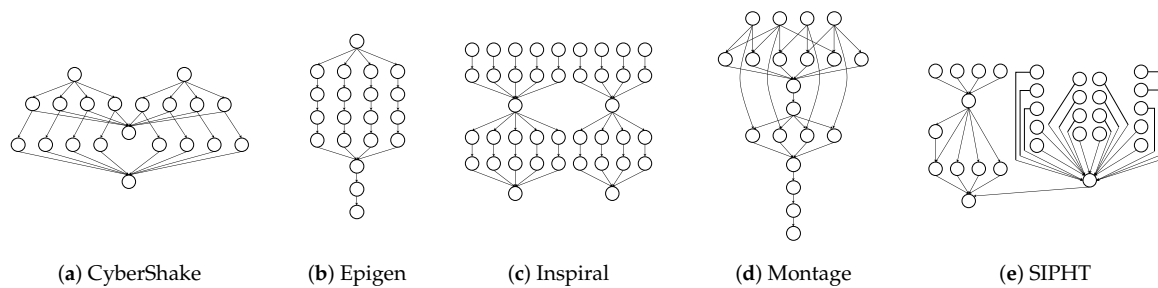
## 5. Experiment

In this section, we conduct a series of experiments on real-world workflows to evaluate the performance of DFTWS and analyze the impact factors on the performance.

### 5.1. Experimental Setup

To evaluate the performance of DFTWS, we adopt five typical types of workflows provided by Pegasus project [24], including CyberShake, Epigenomics, Inspiral, Montage and sRNA identification protocol using high-throughput technology (SIPHT), which are complex scientific workflows from different domains including geology, astronomy, physics, life sciences and biology. These workflow models have been widely used in workflow scheduling and performance evaluation.

Figure 1 shows the structures of these workflow models, and Table 4 lists their attributes, including the number of tasks (NoT), number of edges (NoE), task runtimes (TR) and average task runtimes (ATR).



**Figure 1.** The structure of real-world scientific workflows.

**Table 4.** The parameters of real-world scientific workflows.

Workflow	NoT	NoE	TR	ATR
CyberShake	100	193	[0.57, 164.74]	22.37
Epigenomics	100	122	[0.02, 23571.36]	4034.00
Inspiral	100	121	[4.25, 670.45]	210.24
Montage	100	233	[0.83, 13.85]	10.79
SIPHT	97	111	[0.06, 2891.24]	179.17

There is an array of metrics to measure the performance of fault-tolerant workflow scheduling algorithms, and each metric provides a different evaluation aspect. Among various metrics, we select two common used ones: reliability and FT cost ratio:

1. Reliability is defined as the ratio of successfully completed workflows over all submitted workflows within their DDLs.
2. FT cost ratio is defined as the percentage of FT cost over the total execution of a workflow with DFTWS, which further includes SRE and TRE cost ratio. Because workflows differ in the execution cost even if no failure occurs, it is more accurate to use FT cost ratio, rather than the absolute FT cost, to measure the performance of DFTWS and analyze the factors affecting the performance.

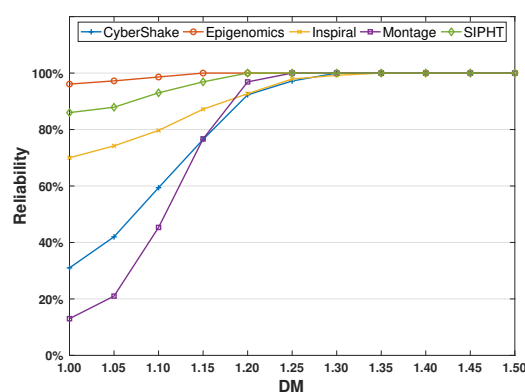
To achieve a detailed analysis of the performance of DFTWS, three major impact factors of DFTWS are analyzed, including deadline multiplier (DM), failure rate (FR) and workflow structure.

### 5.2. Impact of DM

DM refers to a given DDL divided by the minimum completion time of a workflow without failures that equals to the length of CP (CPL). Therefore,  $DM = DDL/CPL$ . A smaller DM means a stricter time limitation for workflow execution.

To evaluate the impact of DM, we conduct a group of experiments on these five workflows. In this group of experiments, DM increases from 1.00 to 1.50, and the increasing step is 0.05. FR is set as 0.1. Failures include permanent and transient failures. There are 10 subgroups, the ratio of transient failure increases from 0 to 100%, with the increasing step of 10%. The test results of these 10 subgroups are averaged to eliminate the impact the specific ratio of transient and permanent failures.

Figure 2 depicts the results of the reliabilities of all workflows with DFTWS with the increasing DM. It can be found that, as DM increases, the reliabilities of all five of these types of workflows turn out an uptrend. The reliabilities increase gradually as DM increase from 1.0 to around 1.25, and then become stable at 100% when DM is bigger than around 1.25. This indicates that the reliability is lower with a small DM, and rises as DM increases.

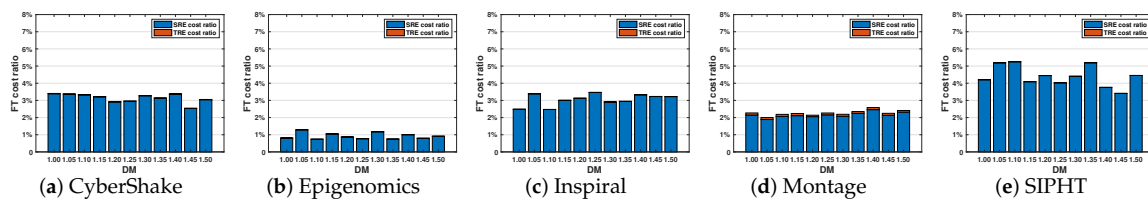


**Figure 2.** Reliabilities of workflows with the increasing deadline multiplier (DM).

This phenomenon is caused by the dynamic re-execution schemes of DFTWS. Unlike the parallel execution whether or not failures happen, the dynamic re-execution schemes act when there are failures actually occur. This is the reason why DFTWS is sensitive to a smaller DM, since there is not sufficient time for DFTWS to eliminate the influence of all failures. This suggests that a stricter DM has a more obvious impact on the performance of DFTWS on reliability, whereas DFTWS can maintain workflow reliability in a high level with a lenient limit of completion time. Although the timeliness of dynamic re-execution schemes are inferior to parallel execution, the costs of dynamic re-execution schemes are much lower than parallel execution.

The results of FT cost ratio of all workflows with DFTWS under an increasing DM and a fixed FR are depicted in Figure 3. In each subfigure, the overall height of a bar is the FT cost ratio with a specific value of DM, which further include a part of SRE cost ratio (the blue part) and a part of TRE cost ratio (the orange part).

As Figure 3 shows, the FT, SRE and TRE cost ratio of each workflow basically remains stable with the increasing DM. The FT cost ratios of CyberShake, Epigenomics, Inspiral, Montage and SIPHT are around 3.1%, 0.9%, 3.0%, 2.3% and 4.8%, respectively. This indicates that the FT cost ratio is not affected by DM because the change of DM does not affect the manner of failure handling of DFTWS.



**Figure 3.** Fault-tolerant (FT) cost ratio of workflows with the increasing deadline multiplier (DM).

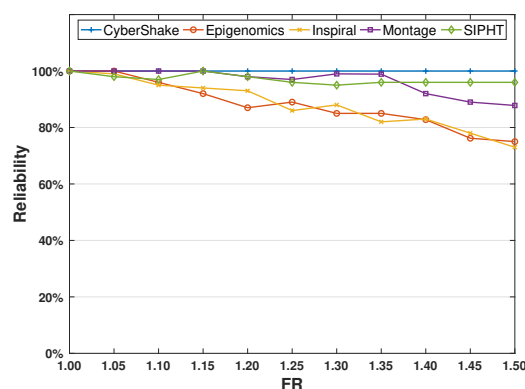
### 5.3. Impact of FR

To evaluate the impact of FR on the performance of DFTWS, we conduct another group of experiments on the five types of workflows. In this group of experiments, DM is set as 1.25. FR is set in  $[0, 0.1]$ , and the increasing step is 0.01. There are also 10 subgroups with the same setting and goal as designed in Section 5.2.

Figure 4 depicts the results of the reliabilities of all workflows with DFTWS with the increasing FR. It can be found that the reliabilities of all workflows decline as FR increases. This is because fewer tasks can be completed in the given DDL because more time is spent on handling failed tasks. Contrary to the strong uptrend of reliabilities with the increasing DM, a gentle downtrend is observed for all workflows, and the average reliability of each workflow is higher. This indicates that DFTWS can deal with the increasing FR well in order to maintain workflows in a higher reliability range.

The results of FT cost ratio of all workflows with DFTWS under an increasing FR and a fixed DM are depicted in Figure 5. It can be observed that the FT, SRE and TRE cost ratio of each workflow increase gradually with the increase FR because handling more failed tasks results in the rising FT cost while the execution cost remains constant. Therefore, the FT cost ratio rises as FR increases.

The average FT cost ratio is relatively lower of all workflows with DFTWS. DFTWS uses around 1.8%, 0.8%, 1.9%, 1.7% and 2.8% FT cost to tolerant 5% failures. This suggests DFTWS is a cost-effective fault-tolerant workflow scheduling approach.



**Figure 4.** Reliabilities of workflows with the increasing failure rate (FR).

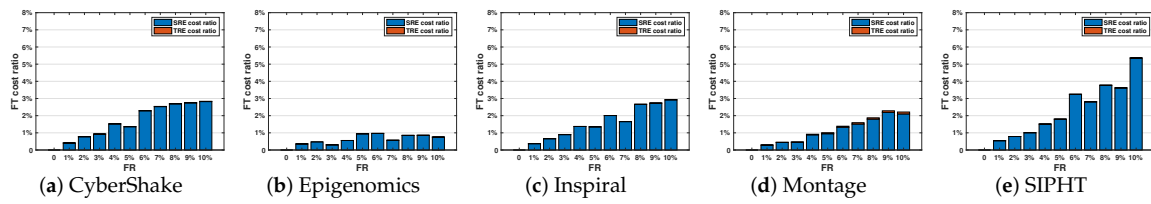


Figure 5. Fault-tolerant (FT) cost ratio of workflows with the increasing failure rate (FR).

#### 5.4. Impact of Workflow Structure

In Sections 5.2 and 5.3, it can be found that, in addition to the similar trend of reliabilities and FT cost ratio of all workflows, there are also some detailed differences among these workflows. For example, Epigenomics has a steady and the highest level of reliability, while the reliability of Montage is the lowest and shows the sharpest uptrend. Additionally, the average of FT cost ratio of Epigenomics is less than 1%, whereas SIPHT results in more than 4% the average of FT cost ratio under the same settings. Intuitively, the disparities should be caused by structural difference among these workflows.

To further analyze the impact of workflow structure on the performance of DFTWS, we conduct a group of comprehensive experiments on these workflows. Unlike the previous groups of experiments that focus on either DM or FR, both of them are considered simultaneously in this group. DM is set from 1.00 to 1.50, and the increasing step is 0.05. FR is set from 15% to 5%, and the decreasing step is 1%. The results of reliability and FT cost ratio of each workflow under increasing DM and decreasing FR are shown in Figures 6 and 7, respectively.

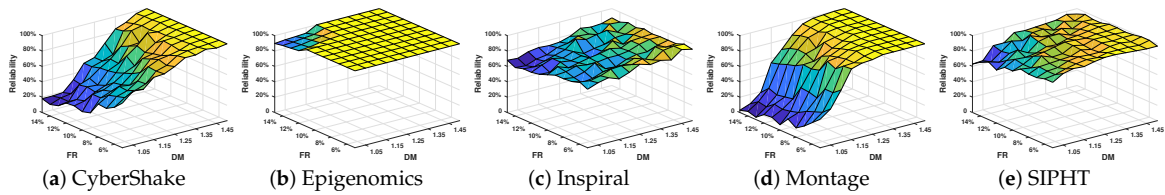


Figure 6. Reliability of real-world workflows with increasing DM and decreasing FR.

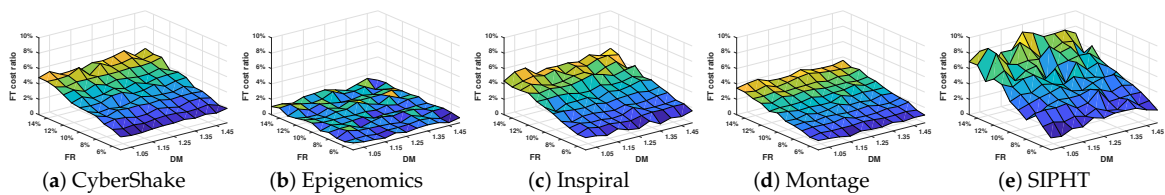


Figure 7. FT cost ratio of real-world workflows with increasing DM and decreasing FR.

Among all workflows, the reliability of Montage is the lowest and shows the sharpest uptrend, while Epigenomics has a steady and the highest level of reliability. This indicates that Montage is the most sensitive one to DM, whereas Epigenomics is the most stable one. Because  $DM = DDL/CPL$ , a smaller DM means the DDL is closer to CPL. If a path has a similar length to CPL, thus, it is more likely to be affected by a smaller DM as well. Moreover, more such paths mean more tasks will be affected by a smaller DM. In addition, a workflow with similar PLs is more vulnerable to a tight deadline because its CP is more likely to change. Whether the CP will change depends on the number of CPs, the location where the failure occurs and the time needed to handle the failure. Four cases are summarized as follows:

1. If a critical node fails and the workflow only has one CP, the CP does not change, but the CP length (CPL) is increased.



2. If a critical node fails and the workflow has multiple CPs, the length of the CPs with the failed node are increased, and they are still the CPs. However, the CPs without the failed node turn into non-critical paths (NCPs).
3. If a non-critical node fails, the lengths of all the paths with this failed node are increased. If any of these paths becomes longer than the original CP, it turns into the new CP. The original CP turns into NPC.
4. If a non-critical node fails, but none of the prolonged paths with this failed node is longer than the original CP; both the CP and CPL do not change.

In general, if the distribution of PL is more even, there will be more tasks that are limited by a small DM; thus, the reliability will be lower. Meanwhile, if the distribution of PL is more even, it is more likely to change the CP of the workflow. CP change tends to disrupt the fault-tolerant schemes, leading to a lower reliability.

Based on the above analysis, we compute the standard deviation of path length  $std(PL)$  for each workflow. The results are in the 4-th column of Table 5. It can be found that  $std(PL)$  of Montage is the smallest, 1.91. This indicates that the lengths of all paths of Montage are close to each other. Therefore, Montage suffers the most serious impact of a small DM. In contrast, Epigenomics is basically not affected by a small DM because its  $std(PL)$  is the biggest.

**Table 5.** Structural parameters of five types of workflows. Number of paths (NoP); average path length (APL); standard deviation of path length ( $std(PL)$ ); number of critical tasks (NoCT); critical path length (CPL); the ratio of critical path length over total task runtime (CPL/TTR); standard deviation of task runtime ( $std(TR)$ ).

Workflow	NoP	APL	$std(PL)$	NoCT	CPL	CPL/TTR	$std(TR)$
CyberShake	96	166.06	38.93	5	228.29	0.1021	26.69
Epigenomics	24	228,470	3663.10	9	29,873.00	0.0741	7254.90
Inspirial	218	898.41	159.13	7	1332.80	0.0634	226.16
Montage	1920	69.89	1.91	10	70.72	0.0655	2.31
SIPHT	133	904.97	1170.51	6	4475	0.2575	521.51

As Figure 7 shows, SIPHT obtains the highest FT cost ratio, while Epigenomics achieves the lowest one under the same setting of FR. The same phenomenon can be found in Section 5.4 as well. The previous results show that SRE cost comprises most of FT cost. At the same time, failed critical task is the major cause of SRE cost. Therefore, the ratio of the length of CP (CPL) over the total task runtime (TTR) can indicate FT cost ratio. Thus, we calculate CPL/TTR as shown in the 7th column of Table 5. Indeed, CPL/TTR of SIPHT is the biggest, 0.2575. However, CPL/TTR of Epigenomics is not the smallest, which is close to Inspirial and Montage. Thus, what is the deeper reason?

To dig into the previous issue, we further analyze the formation of the CPs of Epigenomics, Inspirial and Montage. We calculate their standard deviations of runtimes  $std(TR)$  as shown in the last column of Table 5. The extremely large  $std(TR)$  of Epigenomics indicates that the distribution of runtime of Epigenomics is highly skewed. A very small minority of critical tasks account for the vast majority of CPL, although they have fewer advantages on numbers. However, under the fixed FR, the failure probabilities of these critical tasks with extremely large runtimes are much lower because of their low quantity proportion. Therefore, Epigenomics achieves a very low FT cost ratio among all workflows.

In addition, let's review Figures 3 and 5, it can be found that only Montage has a bigger TRE cost ratio (orange part of bars), while the TRE cost ratio of the other four workflows are quite small, which are barely visible. This is because the runtimes of all tasks of Montage are distributed much more evenly than the others; its  $std(TR)$  is only 2.31. In this situation, the non-critical tasks have similar runtimes with critical ones. Therefore, the TRE cost ratio caused by non-critical tasks becomes



a part to be reckoned with. This suggests that the proportion of TRE increases for the workflows with similar runtimes of tasks.

### 5.5. Experimental Summary

Based on the above experimental results and analysis, some summarizations are made. The reliability and cost of a workflow are affected by DM, FR and workflow structure. When workflow structure is not considered, DM is the main factor affecting the ability of reliability maintaining of DFTWS. A small DM remarkably weakens the ability of reliability maintaining of DFTWS while a lenient deadline can facilitate DFTWS to enhance the performance so that workflow reliability can be kept in a high range. The ratio of FT cost depends on FR. Moreover, DFTWS maintains high reliability for workflows with low FT cost ratio. This indicates that DFTWS achieves a good balance between the requirements of high reliability and low cost. From the perspective of workflow structure, different structures have different impacts on the performance of DFTWS. Moreover, we find that the distribution of PL, the distribution of TR and the ratio of CPL over total TR are the major factors that lead to the performance differences of DFTWS on all workflows.

## 6. Conclusions

In cloud environments, fault tolerance is significant for workflow scheduling. In this paper, we propose DFTWS, a dynamic fault-tolerant workflow scheduling approach with hybrid spatial-temporal re-execution. DFTWS first calculates the static DAG information, including calculating time attributes of each node, identifying the critical path and nodes. After that, DFTWS conducts initial resource allocation based on the static DAG information before executing workflow, and then makes real-time scheduling decisions dynamically based on failure type and task criticality throughout workflow execution. By separating resource allocation and online scheduling, DFTWS improves the time efficiency of fault-tolerant workflow scheduling. To achieve the trade-off between reliability and cost, DFTWS adopts hybrid SRE and TRE instead of parallel execution. Therefore, DFTWS guarantees the reliability and controls the cost of workflows simultaneously.

Comprehensive experiments are conducted on real-world workflows. We analyze the impacts of DM, FR and workflow structure on the reliability and cost of workflows with DFTWS. The experimental results demonstrate that DFTWS is an effective solution for fault-tolerant workflow scheduling.

According to the experimental analysis, it can be found that there are different performance metrics for assessing a fault-tolerant workflow scheduling approach, which are generally contradictory, like reliability and cost. Meanwhile, there are multiple factors that affect the performance of a fault-tolerant workflow scheduling approach. Therefore, no single fault-tolerant workflow scheduling approach is the best one for all type workflows under varying conditions. As future work, we plan to design a flexible fault-tolerant scheme selecting or combining the approach for workflow scheduling in Cloud computing environments.

**Author Contributions:** Conceptualization, N.W. and D.Z.; Data curation, N.W.; Funding acquisition, D.Z.; Investigation, N.W. and D.Z.; Methodology, N.W. and Z.Z.; Software, N.W.; Supervision, D.Z.; Validation, N.W. and Z.Z.; Writing—original draft, N.W. and D.Z.; Writing—review and editing, N.W. and Z.Z.

**Funding:** This work is supported by the National High Technology Development 863 Program of China under Grant No. 2013AA01A215.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Donoho, D. 50 years of data science. *J. Comput. Graph. Stat.* **2017**, *26*, 745–766. [[CrossRef](#)]
2. Yu, J.; Buyya, R.; Ramamohanarao, K. Workflow scheduling algorithms for grid computing. In *Metaheuristics for Scheduling in Distributed Computing Environments*; Springer: Berlin, Germany, 2008; pp. 173–214.

3. Zhu, X.; Wang, J.; Guo, H.; Zhu, D.; Yang, L.T.; Liu, L. Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 3501–3517. [[CrossRef](#)]
4. Rao, J.; Wei, Y.; Gong, J.; Xu, C.Z. QoS guarantees and service differentiation for dynamic cloud applications. *IEEE Trans. Netw. Serv. Manag.* **2013**, *10*, 43–55.
5. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Gunho, L.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58. [[CrossRef](#)]
6. Chen, H.; Wang, F.Z.; Helian, N. Entropy4Cloud: Using Entropy-Based Complexity to Optimize Cloud Service Resource Management. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 13–24. [[CrossRef](#)]
7. Poola, D.; Salehi, M.A.; Ramamohanarao, K.; Buyya, R. A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments. In *Software Architecture for Big Data and the Cloud*; Elsevier: Amsterdam, The Netherlands, 2017; pp. 285–320.
8. Qin, X.; Jiang, H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Comput.* **2006**, *32*, 331–356. [[CrossRef](#)]
9. Rodriguez, M.A.; Buyya, R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans. Cloud Comput.* **2014**, *2*, 222–235. [[CrossRef](#)]
10. Zheng, Q. Improving MapReduce fault tolerance in the cloud. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–6.
11. Masdari, M.; ValiKardan, S.; Shahi, Z.; Azar, S.I. Towards workflow scheduling in cloud computing: A comprehensive analysis. *J. Netw. Comput. Appl.* **2016**, *66*, 64–82. [[CrossRef](#)]
12. Yaseen, S.G.; Al-Slamy, N. Ant colony optimization. *IJCSNS* **2008**, *8*, 351.
13. Verma, A.; Kaushal, S. Cost-time efficient scheduling plan for executing workflows in the cloud. *J. Grid Comput.* **2015**, *13*, 495–506. [[CrossRef](#)]
14. Cao, J.; Chen, J.; Zhao, Q. An optimized scheduling algorithm on a cloud workflow using a discrete particle swarm. *Cybern. Inf. Technol.* **2014**, *14*, 25–39.
15. Singh, L.; Singh, S. A survey of workflow scheduling algorithms and research issues. *Int. J. Comput. Appl.* **2013**, *74*, 21–28. [[CrossRef](#)]
16. Lin, C.; Lu, S. Scheduling scientific workflows elastically for cloud computing. In Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, Washington, DC, USA, 4–9 July 2011; pp. 746–747.
17. Wu, H.; Tang, Z.; Li, R. A priority constrained scheduling strategy of multiple workflows for cloud computing. In Proceedings of the 2012 14th IEEE International Conference on Advanced Communication Technology (ICACT), Washington, DC, USA, 5–10 July 2012; pp. 1086–1089.
18. Verma, A.; Kaushal, S. Deadline and budget distribution based cost-time optimization workflow scheduling algorithm for cloud. In Proceedings of the IJCA Proceedings on International Conference on Recent Advances And Future Trends in Information Technology (iRAFIT 2012), Patiala, India, 21–23 March 2012; iRAFIT (7). Volume 4, pp. 1–4.
19. Zhu, M.M.; Cao, F.; Wu, C.Q. High-throughput scientific workflow scheduling under deadline constraint in clouds. *J. Commun.* **2014**, *9*, 312–321. [[CrossRef](#)]
20. Yassa, S.; Chelouah, R.; Kadima, H.; Granado, B. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *Sci. World J.* **2013**, *2013*, 350934. [[CrossRef](#)]
21. Delavar, A.G.; Aryan, Y. A goal-oriented workflow scheduling in heterogeneous distributed systems. *Int. J. Comput. Appl.* **2012**, *52*, 27–33.
22. Shengjun, X.; Jie, Z.; Xiaolong, X. An improved algorithm based on ACO for cloud service PDTs scheduling. *Adv. Inf. Sci. Serv. Sci.* **2012**, *4*. [[CrossRef](#)]
23. Ludäscher, B.; Altintas, I.; Berkley, C.; Higgins, D.; Jaeger, E.; Jones, M.; Lee, E.A.; Tao, J.; Zhao, Y. Scientific workflow management and the Kepler system. *Concurr. Comput. Pract. Exp.* **2006**, *18*, 1039–1065. [[CrossRef](#)]
24. Deelman, E.; Vahi, K.; Juve, G.; Rynge, M.; Callaghan, S.; Maechling, P.J.; Mayani, R.; Chen, W.; Da Silva, R.F.; Livny, M.; et al. Pegasus, a workflow management system for science automation. *Future Gener. Comput. Syst.* **2015**, *46*, 17–35.

- [CrossRef]
25. Mandal, A.; Kennedy, K.; Koelbel, C.; Marin, G.; Mellor-Crummey, J.; Liu, B.; Johnsson, L. Scheduling strategies for mapping application workflows onto the grid. In Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), Research Triangle Park, NC, USA, 24–27 July 2005; pp. 125–134.
  26. Fard, H.M.; Prodan, R.; Barrionuevo, J.J.D.; Fahringer, T. A multi-objective approach for workflow scheduling in heterogeneous environments. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, Canada, 13–16 May 2012; pp. 300–309.
  27. Prodan, R.; Wiczeorek, M. Bi-criteria scheduling of scientific grid workflows. *IEEE Trans. Autom. Sci. Eng.* **2010**, *7*, 364–376. [CrossRef]
  28. Shi, J.; Luo, J.; Dong, F.; Zhang, J. A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud. In Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hsinchu, Taiwan, 21–23 May 2014; pp. 672–677.
  29. Alkhanak, E.N.; Lee, S.P.; Rezaei, R.; Parizi, R.M. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *J. Syst. Softw.* **2016**, *113*, 1–26. [CrossRef]
  30. Anghel, L.; Alexandrescu, D.; Nicolaidis, M. Evaluation of a soft error tolerance technique based on time and/or space redundancy. In Proceedings of the 13th Symposium on Integrated Circuits and Systems Design (Cat. No. PR00843), Manaus, Brazil, 18–24 September 2000; pp. 237–242.
  31. Hwang, S.; Kesselman, C. Grid workflow: A flexible failure handling framework for the grid. In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, WA, USA, 22–24 June 2003; pp. 126–137.
  32. Gao, Y.; Gupta, S.K.; Wang, Y.; Pedram, M. An energy-aware fault tolerant scheduling framework for soft error resilient cloud computing systems. In Proceedings of the Conference on Design, Automation & Test in Europe. European Design and Automation Association, Dresden, Germany, 24–28 March 2014; p. 94.
  33. Bala, A.; Chana, I. Intelligent failure prediction models for scientific workflows. *Expert Syst. Appl.* **2015**, *42*, 980–989. [CrossRef]
  34. Ghosh, S.; Melhem, R.; Mossé, D. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst.* **1997**, *8*, 272–284. [CrossRef]
  35. Manimaran, G.; Murthy, C.S.R. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Trans. Parallel Distrib. Syst.* **1998**, *9*, 1137–1152. [CrossRef]
  36. Sun, D.; Zhang, G.; Wu, C.; Li, K.; Zheng, W. Building a fault tolerant framework with deadline guarantee in big data stream computing environments. *J. Comput. Syst. Sci.* **2017**, *89*, 4–23. [CrossRef]
  37. Qiu, X.; Dai, Y.; Xiang, Y.; Xing, L. Correlation modeling and resource optimization for cloud service with fault recovery. *IEEE Trans. Cloud Comput.* **2017**. [CrossRef]
  38. Benoit, A.; Hakem, M.; Robert, Y. Multi-criteria scheduling of precedence task graphs on heterogeneous platforms. *Comput. J.* **2010**, *53*, 772–785. [CrossRef]
  39. Xie, G.; Zeng, G.; Li, R.; Li, K. Quantitative fault-tolerance for reliable workflows on heterogeneous IaaS clouds. *IEEE Trans. Cloud Comput.* **2017**. [CrossRef]
  40. Mei, J.; Li, K.; Zhou, X.; Li, K. Fault-tolerant dynamic rescheduling for heterogeneous computing systems. *J. Grid Comput.* **2015**, *13*, 507–525. [CrossRef]
  41. Chen, C.Y. Task scheduling for maximizing performance and reliability considering fault recovery in heterogeneous distributed systems. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 521–532. [CrossRef]
  42. Chen, H.C.; Hu, Y.; Lee, P.P.; Tang, Y. NCCloud: A network-coding-based storage system in a cloud-of-clouds. *IEEE Trans. Comput.* **2014**, *63*, 31–44. [CrossRef]

