2014 IEEE 28th International Conference on Advanced Information Networking and Applications

Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds

Deepak Poola*, Saurabh Kumar Garg‡, Rajkumar Buyya*, Yun Yang†, and Kotagiri Ramamohanarao*

*Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computing and Information Systems, The University of Melbourne, Australia
deepakc@student.unimelb.edu.au,{raj,rao}@csse.unimelb.edu.au

† Faculty of SET, Swinburne University of Technology, Melbourne, Australia
yyang@swin.edu.au

‡ Department of Computing and Information Systems, University of Tasmania, Hobart, Australia
Saurabh.Garg@utas.edu.au

***Abstract*—Dynamic resource provisioning and the notion of seemingly unlimited resources are attracting scientific workflows rapidly into Cloud computing. Existing works on workflow scheduling in the context of Clouds are either on deadline or cost optimization, ignoring the necessity for robustness. Robust scheduling that handles performance variations of Cloud resources and failures in the environment is essential in the context of Clouds. In this paper, we present a robust scheduling algorithm with resource allocation policies that schedule workflow tasks on heterogeneous Cloud resources while trying to minimize the total elapsed time (makespan) and the cost. Our results show that the proposed resource allocation policies provide robust and fault-tolerant schedule while minimizing makespan. The results also show that with the increase in budget, our policies increase the robustness of the schedule.**

***Index Terms*—Workflows; Cloud; Robustness; Fault-Tolerance; Scheduling;**

I. Introduction

Cloud computing offers virtualized servers, which are dynamically managed, monitored, maintained, and governed by market principles. As a subscription based computing service, it provides a convenient platform for *scientific workflows* due to features like application scalability, heterogeneous resources, dynamic resource provisioning, and pay-as-you-go cost model. However, Clouds are faced with challenges like performance variations (because of resource sharing, consolidation and migration) and failures (caused by outages and faults in computational and network components).

The performance variation of Virtual Machines (VM) in Clouds affects the overall execution time (i.e. makespan) of the workflow. It also increases the difficulty to estimate the task execution time accurately. Dejun et al. [7] show that the behavior of multiple "identical" resources vary in performance while serving exactly the same workload. A performance variation of 4% to 16% is observed when Cloud resources share network and disk I/O [3].

Failures also affect the overall workflow execution and increase the makespan. Failures in a workflow application are mainly of the following types: task failures, VM failures, and workflow level failures [11]. Task failures may occur due to dynamic execution environment configurations, missing input data, or system errors. VM failures are caused by hardware failures and load in the datacenter, among other reasons. Workflow level failures can occur due to server failures, Cloud outages, etc. Prominent fault-tolerant techniques that handle such failures are retry, alternate resource, checkpointing, and replication [22].

Workflow management systems should handle performance variations and failures while scheduling workflows. Workflow scheduling maps tasks to suitable resources, whilst maintaining the task dependencies. It also satisfies the performance criteria while being bounded by user defined constraints. This is a well known NP-complete problem [12].

A schedule is said to be robust if it is able to absorb some degree of uncertainty in the task execution time [5]. Robust schedules are much needed in mission-critical and time-critical applications. Here, meeting the deadline is paramount and it also improves the application dependability [9]. Robust and fault-tolerant workflow scheduling algorithms identify these aspects and provide a schedule that is insensitive to these uncertainties, by tolerating variations and failures in the environment up to a certain degree. Robustness of a schedule is always measured with respect to another parameter such as makespan, schedule length, etc. [5]. It is usually achieved with redundancy in time or space [9] i.e. adding slack time or replication of nodes.

In this paper, we present a robust and fault-tolerant scheduling algorithm. The proposed algorithm is robust against uncertainties such as performance variations and failures in Cloud environments. This scheduling algorithm

This work is partly supported by Australian Research Council Linkage Project LP0990393

1550-445X/14 $31.00 © 2014 IEEE
DOI 10.1109/AINA.2014.105

efficiently maps tasks on heterogeneous Cloud resources and judiciously adds slack time based on the deadline and budget constraints to make the schedule robust. Additionally, three multi-objective resource selection policies are presented, which maximize robustness while minimizing makespan and cost.

The **key contribution** of this paper is a robust and fault-tolerant scheduling algorithm with three multi-objective resource selection policies. This paper also presents two robustness metrics and a detailed performance analysis of the scheduling algorithm using them.

The outline of the paper is as follows: Section II discusses the related works in this area. Section III presents the system model. In section IV, the proposed approach is discussed. Section V details experimental setup with the analysis and results. Finally in section VI conclusions and future directions are discussed.

II. Related Work

Current workflow scheduling on Clouds mostly focuses on homogeneous resources [16]. One of the early attempts of exploiting the heterogeneous types of resources is presented by Abrishami et al. [2]. They do not consider budget constraints and their scheduling algorithm does not consider failures or performance variations.

Robust and fault-tolerant scheduling in workflows has been an active area of research with significant amount of work done in the area of Grids, clusters and distributed systems. Research in robust and fault-tolerant scheduling encompasses numerous fields like job-shop scheduling [15], supply chain [10], and distributed systems [11], [18], [19]. Many scheduling techniques have been employed to develop robust workflows. Dynamic scheduling or reactive scheduling reschedules tasks when unexpected events occur [10]. Trust based scheduling predicts the stability of a schedule by incorporating a trust model for resource providers [20]. Stochastic based approaches model uncertainty of system parameters in a non-deterministic way, which aid heuristic decision making [17], [19]. Robust schedule has also been developed using fuzzy techniques, where task execution times are represented by fuzzy logic, which is also used to model uncertainty [8].

Shi et al. [18] present a robust scheduling for heterogeneous resources using slack to schedule tasks. Task slack time represents a time window within which the task can be delayed without extending the makespan and it is intuitively related to the robustness of the schedule. They present a ϵ-constraint method where robustness is an objective and deadline is a constraint. This scheduling algorithm tries to find schedules with maximum slack time without exceeding the specified deadline. They do not consider a Cloud environment and also do not consider any cost models.

To the best of our knowledge, there has been no study in workflow scheduling algorithm for Clouds maximizing robustness, and minimizing makespan and cost at the same time. Also there are very few works which schedule workflow tasks on heterogeneous Cloud resources. This study tries to address these shortcomings.

III. System Model

The description of the system model, important definitions, assumptions, and the problem statement are discussed further in this section.

The **Cloud environment** in our system model has a single datacenter that provides heterogeneous VM/resource types, $VT = \{vt_1, vt_2, ..., vt_m\}$. Each VM type has a specific configuration and a price associated with it. The configuration of VM type differs with respect to memory, CPU measured in million instructions per second (MIPS) and OS. Each vt_i has a $Price(vt_i)$ associated with it, charged on an unit time basis (e.g. 1 hour, 10 minutes, etc.). A static VM startup/boot time is assigned to all VMs, which influences the start time of the task.

Uncertainties: We have considered two kinds of uncertainties, task failures and performance variations of VMs. Performance variations in the system arise due to factors like the datacenter load, network delays, VM consolidation, etc. Due to the performance variation of a VM, the execution time of a task increases or decreases by a value y. Here, y is a random variable with a certain probability distribution with a mean value of zero. The actual execution time (AET) of a task is calculated as $AET(t_j) = e_j(1 + y)$, where e_j is the expected execution time of task t_j.

A **Workflow** can be represented as a Directed Acyclic Graph (DAG), $G = (T, E)$, where T is a set of nodes, $T = \{t_1, t_2, ..., t_n\}$, and each node represents a task. Here, E represents a set of edges between tasks, which can be control and/or data dependencies. Each workflow is bounded by a user defined deadline D and budget B constraints. Additionally, each workflow task t_j has a task length len_j given in Million Instructions. We assume all tasks to be CPU intensive and model task execution time accordingly. Models for data or I/O intensive tasks can also be incorporated to estimate task execution without affecting the scheduling algorithm. Task length and the MIPS value of the VM are used to estimate the execution time on a particular VM type. We also account for data transfer times between tasks. The data transfer time between two tasks is calculated based on the size of the data transferred and the Cloud datacenter internal network bandwidth.

Makespan, M, is the total elapsed time required to execute the entire workflow. The deadline D is considered as a constraint where the Makespan M should not be more than the deadline ($M \leqslant D$). The makespan of the workflow is computed as following:

$$M = finish_{t_n} - ST, \tag{1}$$

where ST is the submission time of the workflow and $finish_{t_n}$ is the finish time of the exit node.

Total Cost, C, is the total cost of the workflow execution, which is the sum of the price for the VMs used to execute the workflow. Each VM type has a price associated with it, depending on its characteristics and type. The price of each VM is calculated based on its type and the duration of time it was provisioned. The duration of the time is calculated based on the number of hours a VM executes, from the time of its instantiation, until it is terminated or stopped. The time duration is always rounded to the next full hour (e.g. 5.1 hours is rounded to 6 hours). It is important to mention that multiple tasks can execute in a VM depending on the schedule. Moreover, to execute the entire workflow, multiple VMs of different types can be used. Therefore, the total execution cost, C, is the sum price of all the VMs of different types used in the workflow execution. Additionally, there is a budget B as a constraint, such that the total cost should be less than the budget ($C \leqslant B$).

Robustness of a schedule is measured using two metrics. The first metric is *robustness probability*, R_p, which is the likelihood of the workflow to finish before the given deadline [18], which can be formulated as below:

$$R_p = (TotalRun - FailedRun)/(TotalRun), \quad (2)$$

where $TotalRun$ is number of times the experiment was conducted and $FailedRun$ is number of times the constraint, $finish_{t_n} \leqslant D$ was violated. This equation is based on the methodology offered by Dastjerdi et al. [6].

The second metric is the *tolerance time*, R_t, which is the amount of time a workflow can be delayed without violating the deadline constraint. This provides an intuitive measurement of robustness, expressing the amount of uncertainties it can further withstand.

$$R_t = D - finish_{t_n}. \quad (3)$$

Assumptions: Data transfer cost between VMs are considered to be zero, as in many real Clouds, data transfer inside a Cloud datacenter is free. Storage cost associated with the workflow tasks is assumed to be free, since storage costs have no effect on our algorithm. The datacenter is assumed to have sufficient resources, avoiding VM rejections due to resource contention. This is not a prohibitive assumption as the resources required are much smaller than the datacenter capacity.

Problem Statement: The problem we address in this work is to find a mapping of workflow tasks onto heterogeneous VM types, such that the schedule is robust to the uncertainties in the system keeping the makespan and cost minimal, while executing within the given deadline and budget constraints.

IV. Proposed Approach

In this section, our algorithm and policies are presented. Before presenting the algorithm, some important definitions are detailed. The *critical path* of a workflow is the execution path between the entry and the exit nodes of the workflow with the longest execution time [1]. Critical path determines the execution time of the workflow. The *critical parent (CP)* of t_j is the parent t_p, whose sum of start time, data transfer time and execution time is maximum among other parent nodes.

Algorithm 1: FindPCP(t)

```
//Determine the PCP and allocate a VM for it.
input : task t
while t has unassigned parent do
    PCP ← null, t_j ← t
    while there exists an unassigned parent of t_j do
        add critical parent t_p of t_j to PCP
        t_j ← t_p
    call AllocateResource(PCP)
    for t_j ∈ PCP do
        marks t_j as assigned
        call FindPCP(t_j)
```

The *partial critical path (PCP)* of node t_j is a group of tasks that share a high dependency between them. PCP is determined by identifying the unassigned parents. Unassigned parent is a node that is not scheduled or assigned to any PCP. Further, PCP is created by finding the unassigned critical parent of the node and repeating the same for the critical parent recursively until there are no further unassigned parents. Algorithm 1 details the procedure to find the PCP of a node. Partial critical paths can be scheduled on a single resource, optimizing time and cost [1]. This algorithm decomposes the workflow into smaller groups of tasks, which helps in scheduling. PCPs of a workflow are mutually exclusive, i.e., each task can be in only one PCP.

For every PCP, the best suitable VM type with a robustness type is selected. The robustness type defines the amount of slack that will be added to the PCP execution time. It dictates the amount of fluctuation in the execution time a PCP can tolerate. Four types of robustness that can be associated with a PCP are defined: 1) *No robustness*: this robustness type does not add any slack time to the execution time of a PCP. 2) *Slack* : this robustness type adds a predefined limit of time for the PCP execution time i.e. it can tolerate fluctuations in execution time up to a defined limit. 3) *One Node Failure*: in this robustness type, the largest execution time among the PCP nodes is added to the PCP execution time. This robustness type provides sufficient slack time to handle the failure of the task with the largest execution time in the PCP. 4) *Two Node Failure*: here, the execution time of the largest two nodes is added to the PCP execution time; this is done only when the PCP consists of at least three nodes. PCP with this robustness type can tolerate up to two task failures. Four robustness types up to two node failures are proposed. However, robustness types with higher number of node failures can also be developed.

Algorithm 2 details the selection of a VM type and its associated robustness type. An exhaustive solution set, $SS = \{s_1, s_2, ..., s_{m*l}\}$ is generated, where m is the number of VM types and l is the number of robustness types. The solution set SS consists of solutions with every possible robustness type for every VM type defined. Each solution, $s_i = \{vt_i, RT_i, PCPc_i, PCPt_i\}$, consists of a robustness type (RT_i), PCP cost ($PCPc_i$) and PCP execution time ($PCPt_i$) for VM type vt_i. As m and l are generally not large, the time and space required are reasonable.

The solution set SS is reduced based on deadline and budget constraints into a smaller set of feasible solutions. The deadline constraint D is evaluated by adding the PCP execution time of the chosen instance and robustness type with top level and bottom level.

$$TopLevel + PCPt + BottomLevel \leqslant D, \tag{4}$$

where $TopLevel$ of PCP is the sum of execution times of nodes on the longest path from the entry node to the first node of PCP. $BottomLevel$ of PCP is the sum of execution times of nodes on the longest path from the end node of the PCP to the exit node.

Budget Constraint is evaluated by the following equation:

$$PCPc \leqslant PCPb, \tag{5}$$

where PCPc is the total cost of the PCP. PCP Budget, $PCPb$, is the amount that can be spent on the PCP; this is decomposed from the overall budget according to the following equation,

$$PCPb = (PCPt/TT) * B, \tag{6}$$

where, TT is the total time of the workflow, which is calculated by adding the execution times of the tasks on the reference VM type, vt_{ref}. VM with the least MIPS value is considered as the reference type, vt_{ref}. $PCPt$ is the total execution time of the PCP on vt_{ref}. When $PCPb$ is less than LPr, which is the price required to execute on the cheapest resource, then $PCPb$ is assigned the value LPr.

A feasible solution set FS is created using these two constraints as outlined in Algorithm 2.

findBestSolution, method described in Algorithm 2, chooses the appropriate VM type vt_i for a PCP, based on the resource selection policy from the feasible solution set FS. The three resource selection policies used by this method are described in the following section.

A. Proposed Policies

In this section, three resource selection policies are explained. These policies select the best solution from the feasible solution set FS for each PCP. Each of them has three objectives, namely *robustness*, *time* and *cost* and the priorities among these objectives change for each of these policies. The description of the policies is given below.

Algorithm 2: AllocateResource(PCP)

```
//Allocate a suitable robust resource to the PCP
input  : PCP
output: Robust Resource for PCP
//Create Solution Set SS;
for Every Instance type do
    for Every Robustness type do
        Create Solution set with PCPt and PCPc
FS = null;
Calculate PCPb according to equation 6;
//Create a Feasible Solution Set FS;
for Every solution in SS do
    time = PCPt + TopLevel + BottomLevel;
    if time <= D and PCPc <= PCPb then
        Add to FS
//finds the best solution according to the chosen policy
RobustResource = findBestSolution(FS, Policy);
Assign every task in PCP to the RobustResource.
```

1. **Robustness-Cost-Time (RCT)**: The objective of this policy is to maximize robustness and minimize cost and makespan. This policy sorts the feasible solution set based on the robustness type, and among the solutions with the same robustness type, they are sorted in the increasing order of cost. Solutions with the same robustness type and cost are sorted with increasing order of time. The best solution from this sorted list is picked and the VM type with the associated robustness type is mapped to the tasks of the PCP. Solutions chosen by this policy have high robustness with a lower cost.
2. **Robustness-Time-Cost (RTC)**: RTC policy is similar to RCT policy described above with different priorities. This policy gives priority to robustness, followed by time and finally cost. This policy selects a solution that is robust with minimal makespan. Choices of RTC and RCT policies might have the same robustness type, but will vary with respect to the VM type they select. RTC policy selects a solution with high robustness with minimal makespan.
3. **Weighted**: With this policy users can define their own objective function using the three parameters (robustness, time and cost) and assign weights for each of them. Each value is normalized by taking the minimum and maximum values for that parameter. The weights are applied to the normalized values of robustness, time and cost, and based on these weights the best solution is selected. Weighted policy is a generalized policy, which can be used to find solutions according to user preferences.

Our algorithm with the chosen policy finds a suitable VM type associated with a robustness type for every PCP. Further, the algorithm allocates the PCP tasks on a VM of the chosen type. The resource allocator, first attempts to find a VM of the specified type among the running VMs.

If such a VM is found, the algorithm checks if its end time is less than the start time of the PCP. If this condition is satisfied, the algorithm allocates PCP tasks on this existing VM; otherwise a new VM is created to allocate the tasks. This reduces the number of VMs instantiated and also minimizes the makespan as new VMs take time to boot, which delays the schedule.

B. Fault-Tolerant Strategy

Checkpointing is employed in our algorithm as a fault-tolerant strategy. When a task fails, the algorithm resumes the task from the last checkpoint and checkpointing of tasks is done at regular intervals. The robustness type selected by the resource selection policy provides the necessary slack for the failed task. Additionally, checkpointing strategy helps to recover the task from the last checkpoint.

C. Time Complexity

Creating a solution set SS depends on the number of robustness types and VM types. The time complexity for creating such a set is $O(m.l)$, where m is the number of VM types and l is the number of robustness types. The time complexity for sorting and choosing the best solution based on the policy is $O(mlogm)$. The parameters m and l can take a maximum value of n, where n is the number of tasks. Therefore, the time complexity of AllocateResource is $O(n^2)$. The time complexity of FindPCP is $O(n)$ as the maximum number of times this method can be recursively invoked is equal to the number of tasks n. Hence, the overall time complexity of our algorithm is $O(n^2)$.

V. Performance Evaluation

A. Simulation Setup

CloudSim [4] was used to simulate the Cloud environment. It was extended to support workflow applications, making it easy to define, deploy and schedule workflows. A failure event generator was also integrated into the CloudSim, which generates failures from an input failure trace. Five types of workflow applications and two failure models are used in our simulation as described below.

1) Application Modeling: Five workflows (Montage, CyberShake, Epigenomics, LIGO and SIPHT) were considered. Their characteristics are explained in detail by Bharathi et al. [13]. These five workflows cover all the basic components such as, pipeline, data aggregration, data distribution and data redistribution. Three different sizes of these workflows are chosen, small (around 30 tasks), medium (around 100 tasks) and large (1000 tasks).

2) Resource Modeling: A Cloud model with a single datacenter offering 10 different types of VMs is considered. The characteristics of VMs are modeled similar to the Amazon EC2 instances (t1.micro, m1.small, m1.medium, m1.large, m1.xlarge, m2.xlarge, m2.2xlarge, m2.4xlarge, c1.medium, c1.xlarge). A charging period of 60 minutes is considered for these VMs, similar to the most prominent Cloud providers.

3) Failure Modeling: Two types of failure models are considered for our experiments. First, failures are simulated from failure traces (FT). Due to lack of publicly available Cloud specific failure traces, Condor (CAE) Grid failure dataset [21], available as a part of Failure Trace Archive [14] was chosen. Secondly, a failure model with 10% failure probability (FP) is considered, i.e., for each node there is 10% probability of failure based on uniform distribution. The failed nodes may fail again with the same probability until they complete their execution.

Each VM undergoes a performance variation, which affects the task execution time. We model the variance in the task execution time as a normal distribution $y = N(0, \sigma^2)$, where the standard deviation σ is 10% of the execution time of the task, as suggested by Dejun et al. [7]. They have analyzed and presented the performance variations of Amazon EC2 instances in their study.

4) Reference Algorithms: Two reference algorithms to compare our resource allocation policies are implemented. The first algorithm is a deadline constrained algorithm proposed by Abrishami et al. [2]. The IaaS Cloud Partial Critical Path (ICPCP) algorithm, similar to our algorithm, divides the workflow tasks into PCPs. ICPCP is non-robust algorithm bounded by a deadline constraint.

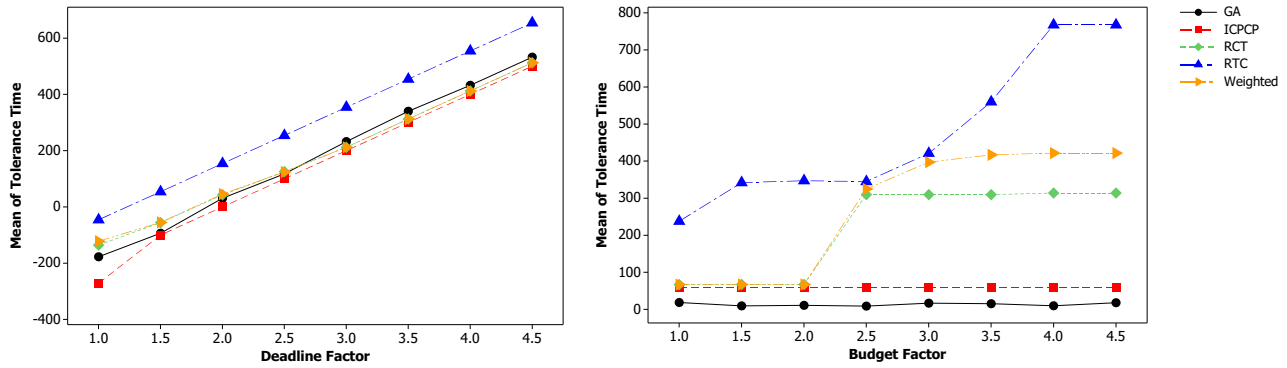
The second reference algorithm implemented is a robust bi-objective genetic algorithm (GA) [18]. This GA considers heterogeneous resources with the objective of maximizing the robustness by increasing the slack time between the tasks. This algorithm considers deadline as a threshold and verifies that the schedule does not violate the deadline. The fitness function, selection and mutation operators for the GA are implemented as described in [18]. The parameters of GA are set as follows: population size = 2000, cross over probability = 0.9 and mutation probability = 0.1, as defined by the authors. Maximum number of iterations is set to 800.

These algorithms are chosen for their similarity with our approach. ICPCP schedules tasks by grouping them into PCPs, similar to our algorithm and GA tries to maximize the slack time to be robust, which is the approach we adapt as well.

In this paper, we have executed the experiments for five workflow applications with two failure models. For each workflow, we varied the deadline with a fixed budget and also varied the budget keeping the deadline fixed to measure the performance with regards to robustness, makespan and cost. Each experiment was executed for 10 times, the mean of which is reported. For the weighted policy, the weights considered for robustness, time and cost are 0.5, 0.3 and 0.2 respectively. We intend to study the effect of varying weights in future. We present our analysis and results in the following section.

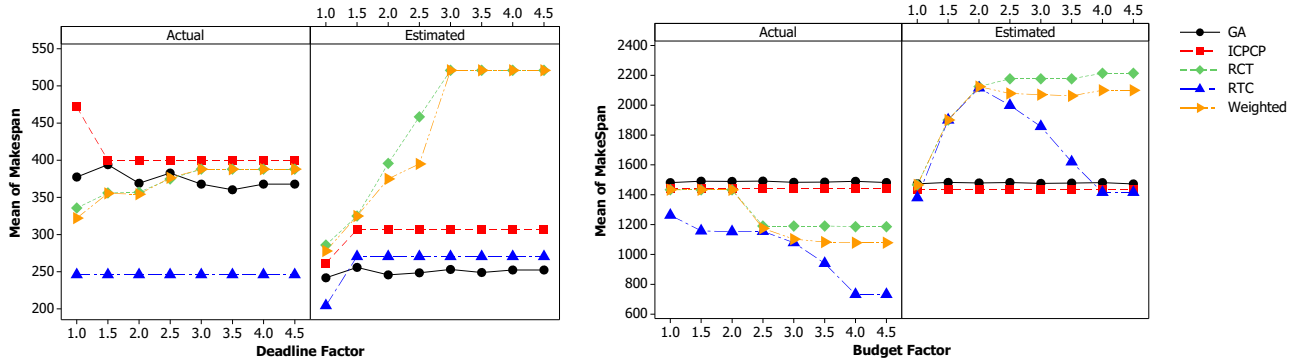
B. Analysis and Results

In this section, among the five workflows considered, we discuss the results of only CyberShake and LIGO work-



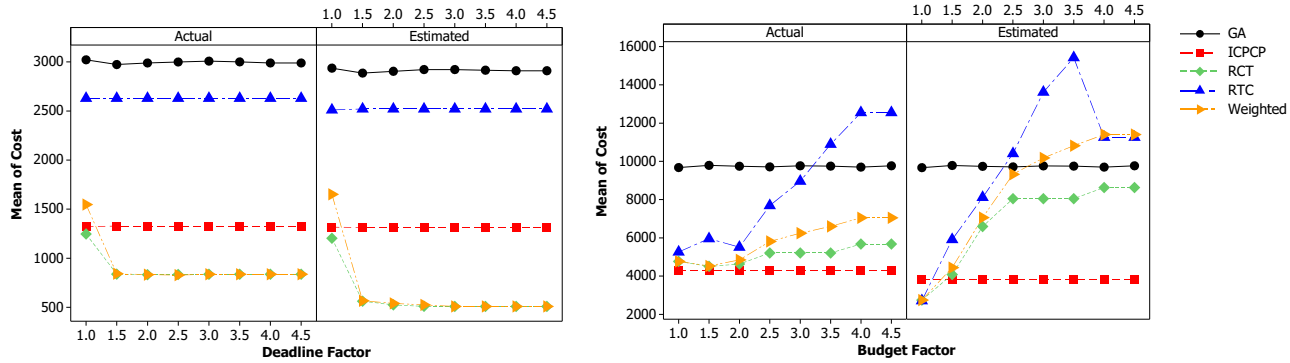
(a) Cybershake with fixed Budget and with FP failure model (b) LIGO with fixed Deadline and with FT failure model

Fig. 1. Effect on robustness with tolerance time R_t



(a) Cybershake with fixed Budget and with FP failure model (b) LIGO with fixed Deadline and with FT failure model

Fig. 2. Effect on makespan for large sized CyberShake and LIGO workflow



(a) Cybershake with fixed Budget and FP failure model (b) LIGO with fixed Deadline and with a FT failure model

Fig. 3. Effect on cost for large sized CyberShake and LIGO workflow

flows due to space restrictions. The CyberShake workflow uses the Probabilistic Seismic Hazard Analysis (PSHA) technique to characterize earth-quake hazards in a region and the LIGO Workflow detects gravitational waves of cosmic origin by observing stars and black holes [13]. We present two experiments considering large workflow types. In the first experiment, we vary the deadline with a fixed surplus budget for large CyberShake workflow and the failures are generated using the failure probability model (FP), with a 10% probability. In the second experiment, we vary the budget with a fixed strict deadline for large LIGO workflow and the failures are generated through the failure trace model (FT). Both these experiments are carefully devised to cover all combinations of deadline and budget, showing the performance of the algorithms under all conditions. For these experiments, we find the lowest makespan M_{low}, which is the time taken to execute on the most expensive VM. We also find the lowest cost C_{low}, which is the cost needed to execute on the cheapest VM. We introduce a deadline factor α similar to [2], based on which we vary the deadlines for workflows according to $\alpha.M_{low}$. We vary α from 1 to 4.5 with a step length of 0.5.

TABLE I
ROBUSTNESS PROBABILITY R_p OF LARGE MONTAGE WORKFLOW WITH FAILURE PROBABILITY MODEL (FP) FOR DIFFERENT POLICIES.

Deadline	Budget	ICPCP	GA	RCT	WGHT*	RTC
Strict	Strict	0.00	0.10	0.20	0.40	**0.70**
	Normal	0.00	0.10	0.20	0.70	**0.90**
	Relaxed	0.00	0.00	0.20	0.70	**0.90**
Relaxed	Strict	**1.00**	0.90	**1.00**	**1.00**	**1.00**
	Normal	**1.00**	0.90	**1.00**	**1.00**	**1.00**
	Relaxed	**1.00**	0.80	**1.00**	**1.00**	**1.00**

* WGHT is an abbreviation for the weighted policy.

Similarly we introduce a budget factor β and the budget is varied according to $\beta.C_{low}$. We vary β from 1 to 4.5 with a step length of 0.5.

The analysis of these experiments and its effect on robustness, makespan and cost are presented below.

1) Effect on robustness: Figure 1 presents the tolerance time R_t of Cybershake and LIGO workflows. Positive values of R_t represent robust solutions that have finished execution within the deadline even with failures and performance variations. Negative values represent schedules that have violated the deadline constraint. In Figure 1, we observe that the RTC policy has the highest mean tolerance time, conveying that the policy is not just robust but can withstand more failures. RTC policy outperforms other policies emerging as the most robust policy. We observe that robustness increases as deadline or budget increases. We observe in Figure 1(b) that tolerance time R_t of ICPCP and GA do not vary with increase in budget. This is because these algorithms do not take budget as an input and do not show any effect as the budget varies.

In 97.5% of the cases weighted policy outperforms ICPCP for CyberShake workflow as seen in Figure 1(a). Under strict deadline, weighted and RCT policies perform better than GA in 67.5% of the cases. Under relaxed deadline GA has a higher tolerance time than weighted and RCT policy in 72.5% of the cases, but the cost of execution for GA is 2.6 times higher than RCT and weighted policies. RCT and weighted policies tries to achieve a robust solution with minimal cost, even under a relaxed deadline we have a robust solution with costs much lower than GA.

In the LIGO workflow experiment, we see that our policies have higher tolerance time in comparison to ICPCP and GA algorithm as shown 1(b). We can also observe that the mean tolerance time increases with increase in budget for our policies unlike ICPCP and GA.

Table I presents the robustness probability R_p for the large Montage workflow with varying deadline and budget. This experiment was performed for all workflows. For space considerations we report only the large Montage workflow as it is the most complex and failures in its task nodes have high adverse effect on the makespan and cost. Other workflows show similar trends and have better results for our policies. This table provides a measure of robustness probability, R_p, which is the probability of a schedule being within the deadline. It can be seen that the RTC is the most robust policy and has the highest probability of being within the deadline. The robustness probability, R_p for weighted and RCT policies outperform GA and ICPCP. It can also be observed that our policies perform with high levels of robustness even under strict deadlines and budgets.

2) Effect on makespan: Figure 2 shows the effect on makespan for CyberShake and LIGO workflows. Figures 2 and 3 have graphs with two panels, where the actual panel represents schedules after execution with uncertainties and the estimated panel depicts schedules before execution without failures. Figures 2(a) and 2(b) show that makespan increases as deadline increases and makespan decreases as budget increases. Our policies have a higher makespan when the schedule is estimated in comparison to ICPCP or GA; however the actual makespan after failures and performance variations of resources is minimal for our policies. RTC provides schedules with smallest makespan under the scenarios of failures and performance variations because it chooses robust resources with least execution time. The average makespan of weighted policy is 19% lower than ICPCP for both CyberShake and LIGO workflows. The average makespan of RCT policy is 14% and 11% lower than ICPCP for CyberShake and LIGO workflows respectively.

In Figure 2(b), the estimated panel of the graph shows the working of the RTC policy. As the budget increases, the makespan increases steadily and then decreases steadily as shown. This is because as the budget increases, the algorithm has the flexibility to either add more slack time or choose an expensive VM. Therefore, with smaller increase in budget, the algorithm chooses inexpensive VMs and adds slack time based on the increases in budget, which increases the estimated makespan. With sufficient increase in budget, the algorithm chooses expensive VM resulting in the decrease of the estimated makespan. The actual panel of the graphs shows that the makespan of our policies are much lower than ICPCP and GA. The RTC policy gives lower makespan consistently, while the makespan of RCT and weighted policies increases slightly with increase in deadline and decreases with increase in budget.

3) Effect on cost: Figure 3 presents the effects on cost, Figure 3(a) shows that the cost decreases for RCT and weighted policies, as the deadline increases with a fixed budget and Figure 3(b) shows that cost increases for our policies as budget increases with a fixed deadline.

For CyberShake workflow, we observe that our policies RCT and weighted have lower costs in comparison with ICPCP and GA. RTC policy has a 98% higher cost than ICPCP and 12% lower cost than GA, but has a 39.7% and 33.6% lower makespan than ICPCP and GA respectively. RTC policy chooses resources that are robust with a lower makespan; on the other hand RCT policy chooses a robust schedule with lower costs.

For LIGO workflow, as depicted in Figure 3(b) we see increase in costs for our policies as the budget increases, but we can also observe that the robustness increases and makespan decreases with increasing budget as shown in Figure 1(b) and 2(b). We can see that the costs of our policies are much lower than GA in most of the cases.

Experiments show that our policies consistently offer schedules with high robustness. RTC policy gives robust schedules with increase in costs but minimal makespan, and RCT policy provides robust schedules, which minimizes costs under relaxed deadline or increases costs under surplus budget. Under strict deadline or a stringent budget our policies behave comparable to ICPCP with respect to cost, yet provides a robust schedule with lower makespan. Our weighted policy in this experiment is tested with only one set of weights, which are comparable to our RCT policy and hence the results show similar trends. The users can use this policy according to their priorities and get schedules that are aligned to their priorities.

VI. Conclusions and Future Work

This paper presents three resource allocation policies with robustness, makespan and cost as its objectives. This is one of the early works in robust and fault-tolerant workflow scheduling on Clouds, considering deadline and budget constraints. The resource allocation policies judiciously add slack time to make the schedule robust considering the deadline and budget constraints. We test our policies with two failure models for five scientific workflows with two metrics for robustness. Results indicated that our policies are robust against uncertainties like task failures and performance variations of VMs.

Among the proposed policies presented, the RTC policy shows the highest robustness and at the same time minimizes makespan of the workflow. The RCT policy provides a robust schedule with costs marginally higher than the reference algorithms considered. The weights of the weighted policy can be varied according to the user priorities. Overall, our policies provide robust schedules with a minimal makespan. They also show that with increase in budget, our policies increase the robustness of the schedule with reasonable increase in cost.

Experiments were conducted with a failure probability of 10%; we plan to experiment with varying failure probabilities. In future, we like to experiment our system on various cost models (e.g. spot market) offered by the Cloud and their impact on the robustness of workflows.

References

[1] S. Abrishami, M. Naghibzadeh, and D.H.J. Epema. Cost-driven scheduling of grid workflows using partial critical paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1400 –1414, aug. 2012.

[2] S. Abrishami, M. Naghibzadeh, and D.H.J. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *ACM Commun*, 53(4):50–58, April 2010.

[4] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[5] L.C. Canon and E. Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.

[6] A.V. Dastjerdi and R. Buyya. An autonomous reliability-aware negotiation strategy for cloud computing environments. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012*, pages 284–291. IEEE, 2012.

[7] J. Dejun, G. Pierre, and C.H. Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In *Workshop on Service-Oriented Computing. ICSOC/ServiceWave 2009*, pages 197–207. Springer, 2010.

[8] C. Fayad, J.M. Garibaldi, and D. Ouelhadj. Fuzzy grid scheduling using tabu search. In *IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2007*, pages 1 –6, july 2007.

[9] F.C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, March 1999.

[10] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European journal of operational research*, 165(2):289–306, 2005.

[11] S. Hwang and C. Kesselman. Grid workflow: a flexible failure handling framework for the grid. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, 2003*, pages 126–137. IEEE, 2003.

[12] D.S. Johnson and M.R. Garey. *Computers and Intractability-A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[13] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682 – 692, 2013.

[14] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010*, pages 398–407. IEEE, 2010.

[15] V.J. Leon, S.D. Wu, and H.S. ROBERT. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[16] S. Pandey, D. Karunamoorthy, and R. Buyya. Workflow engine for clouds. *Cloud Computing*, pages 321–344.

[17] V. Shestak, J. Smith, H.J. Siegel, and A.A. Maciejewski. A stochastic approach to measuring the robustness of resource allocations in distributed systems. In *International Conference on Parallel Processing, 2006. ICPP 2006*, pages 459–470. IEEE, 2006.

[18] Z. Shi, E. Jeannot, and J.J. Dongarra. Robust task scheduling in non-deterministic heterogeneous computing systems. In *IEEE International Conference on Cluster Computing, 2006*, pages 1–10. IEEE, 2006.

[19] J. Smith, H.J. Siegel, and A.A. Maciejewski. *Robust resource allocation in heterogeneous parallel and distributed computing systems*. Wiley Online Library, 2009.

[20] M. Wang, K. Ramamohanarao, and J. Chen. Trust-based robust scheduling and runtime adaptation of scientific workflow. *Concurrency and Computation: Practice and Experience*, 21(16):1982–1998, 2009.

[21] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. In *11th IEEE/ACM International Conference on Grid Computing (GRID), 2010*, pages 65 –72, oct. 2010.

[22] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3):171–200, 2005.