Taming System Dynamics on Resource Optimization for Data Processing Workflows: A Probabilistic Approach

Amelie Chi Zhou, Weilin Xue, Yao Xiao, Bingsheng He, Shadi Ibrahim, *Member, IEEE*, and Reynold Cheng, *Member, IEEE*

Abstract—In many data-intensive applications, workflow is often used as an important model for organizing data processing tasks and resource provisioning is an important and challenging problem for improving the performance of workflows. Recently, system variations in the cloud and large-scale clusters, such as those in I/O and network performances and failure events, have been observed to greatly affect the performance of workflows. Traditional resource provisioning methods, which overlook these variations, can lead to suboptimal resource provisioning results. In this article, we provide a general solution for workflow performance optimizations considering system variations. Specifically, we model system dynamics as time-dependent random variables and take their probability distributions as optimization input. Despite its effectiveness, this solution involves heavy computation overhead. Thus, we propose three pruning techniques to simplify workflow structure and reduce the probability evaluation overhead. We implement our techniques in a runtime library, which allows users to incorporate efficient probabilistic optimization into existing resource provisioning methods. Experiments show that probabilistic solutions can improve the performance by up to 65 percent compared to state-of-the-art static solutions, and our pruning techniques can greatly reduce the overhead of our probabilistic approach.

Index Terms—Cloud dynamics, resource optimization, data processing workflows

1 Introduction

In many data-intensive applications, data processing jobs are often modeled as workflows, which are sets of tasks connected according to their data and computation dependencies. For example, Montage workflow [1] is an astronomy-related big data application, which processes sky mosaics data in the scale of hundreds of GBs. Large companies such as Facebook, Yahoo, and Google frequently execute ad-hoc queries and periodic batch jobs over petabyte-scale data based on MapReduce (MR) workflows [2]. Those data-intensive workflows are usually executed in large scale systems (e.g., high performance computers for scientific applications and public/private clouds for industrial applications) and *resource provisioning*, which determines the size and type of resources to execute workflow tasks, is an important optimization factor to the performance of workflows. However, due to the complex workflow structures, resource provisioning has been a challenging problem for data processing workflows, and has been widely studied by existing work [3], [4], [5].

In many large-scale systems, variations have become the norm rather than the exception [6], [7]. The variations can be caused by both hardware and software reasons. For example, in supercomputer architectures, the variation in power and temperature of the chips can cause up to 16 percent performance variation between processors [8]. In cloud environments, the network and I/O performances also show significant variations due to the resource sharing between multiple users [7]. Job failures have been demonstrated to be variant and follow different kinds of probability distributions for different systems (e.g., HPC, cluster and cloud) [6]. These variations, which have been ignored by most existing optimization methods, raise new challenges to the resource provisioning problem of workflows. In this paper, we discuss the resource provisioning problem of workflows in the cloud as an example, aiming at proposing a general solution to incorporating system variations for performance optimization problems of workflows. Although the discussion is focused on the cloud, we conjecture that the observations can shed light on other systems such as shared clusters (see Section 7).

Why Consider Variations? Cloud providers often provide various types of instances (i.e., VMs) for users to select the most appropriate resources to execute workflow tasks. Most existing resource provisioning methods assume that the execution time of each task is static on a given type of VMs. However, this assumption does not hold in the cloud, where *cloud dynamics*, such as variations of I/O and network performances,

- *Amelie Chi Zhou, Weilin Xue, and Yao Xiao are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China. E-mail: chi.zhou@szu.edu.cn, 1910272018@email.szu.edu.cn, 369314724@qq.com.*
- *Bingsheng He is with the School of Computing, National University of Singapore, Singapore 119077, Singapore. E-mail: hebs@comp.nus.edu.sg.*
- *Shadi Ibrahim is with Inria, Univ. Rennes, CNRS, IRISA, 35042 Rennes, France. E-mail: shadi.ibrahim@inria.fr.*
- *Reynold Cheng is with the Department of Computer Science, University of Hong Kong, Hong Kong, China. E-mail: ckcheng@cs.hku.hk.*

Manuscript received 16 Nov. 2020; revised 15 May 2021; accepted 14 June 2021. Date of publication 22 June 2021; date of current version 9 July 2021.
(Corresponding author: Amelie Chi Zhou.)
Recommended for acceptance by R. Prodan.
Digital Object Identifier no. 10.1109/TPDS.2021.3091400

can result in major performance variation [7], [9] to large-scale data processing workflows. Traditional resource provisioning methods, which overlook these variance values, can lead to suboptimal resource provisioning results [8], [10]. We analyzed several common resource provisioning problems for workflows, and observed that the performance optimization goal is usually *nonlinearly* related to the cloud dynamics in I/O and network performance. Thus, traditional *static* optimizations (e.g., taking the *average* or *expected* performance as optimization input) can lead to suboptimal or even infeasible solutions. For example, to optimize workflow performance using checkpointing, we have to consider system failure dynamics (more details in Section 6). Using the production trace from Google Cloud [11] and performance trace from Amazon EC2, we found that a dynamics-aware method can obtain better performance optimization results than static method [6] for over 60 percent of the time.

Why Probabilistic Method? Existing studies propose various methods such as dynamic scheduling [5], [12] and stochastic modeling [13], [14] to address resource provisioning problems considering cloud dynamics. However, they either rely on accurate cloud performance estimation at runtime (e.g., dynamic scheduling) or involve complicated modeling and analysis and thus are hard to be generalized (e.g., stochastic methods).

In this paper, we study a systematic and effective way of incorporating cloud dynamics into resource optimization for workflows. We propose to model cloud dynamics as random variables and take their *probability distributions* as optimization input to formulate resource provisioning problems. This design has two main advantages. First, it enables probabilistic analysis [15] required by many problems with system randomness, such as designing fault-tolerant scheduling techniques for workflows in case of random system failures [16]. Second, it enables the derivation of probabilistic bounds [17] to guarantee the worst-case performance of applications, while the existing static methods only guarantee the average performance.

With the probabilistic representation of cloud dynamics, traditional static resource provisioning methods cannot be used directly. The main challenge is that using probability distributions as optimization input to resource provisioning problems can lead to a significantly high computation overhead due to the costly distribution calculations and complex structures of data processing workflows. There exist some optimization techniques to improve the efficiency of probabilistic methods in other fields, such as efficient query evaluations in probabilistic databases [18] and efficient probabilistic analysis in hardware design [15]. However, none of them has considered the special features of workflow structure and resource provisioning problems, which can help to more efficiently reduce the overhead of probabilistic resource provisioning of workflows.

Contributions. In this paper, we propose *Prob* to efficiently incorporate cloud dynamics into resource optimization for workflows. In *Prob*, we propose three simple yet effective pruning techniques to reduce the overhead of probabilistic optimizations. These techniques are designed based on the features of workflow structures and resource provisioning problems. First, we identify that calculating the makespan of a workflow is a common operation in many resource provisioning problems of workflows. Thus, we propose preprocessing pruning to reduce the overhead of this important calculation and hence reduce the overhead of probabilistic optimizations. Second, we propose workflow-specific optimizations using existing workflow transformation techniques to reduce the overhead of evaluating one instance configuration solution. Third, we propose a partial solution evaluation method and adopt an existing pruning technique [18] to reduce the overhead of comparing multiple solutions.

We develop a runtime library that includes all the pruning techniques of *Prob*. Users can implement their *existing* resource provisioning methods using *Prob* APIs to incorporate probabilistic optimizations, in order to improve both the effectiveness and efficiency of the existing methods. We introduce budget-constrained scheduling and fault tolerance problem as two examples of resource optimization problems for workflows. We compare *Prob* with state-of-the-art static algorithms using real-world workflows on two real cloud platforms and with simulations. Experimental results demonstrate the effectiveness and efficiency of our probabilistic approach. Specifically, *Prob* improves workflow performance by 2–65 percent compared to the static algorithms for the budget-constrained scheduling problem and by 5–30 percent for the fault tolerance problem. The pruning techniques of *Prob* bring significant reduction to the overhead of our probabilistic approach (e.g., up to 567x speedup to the Monte Carlo (MC) method). As a result, it takes less than one second to complete the optimizations for a Montage workflow with more than 10,000 tasks.

Goals and Non-Goals. The primary goal for *Prob*, and the focus of this paper, is proposing an efficient interface for existing resource provisioning methods to easily incorporate probabilistic optimizations, rather than proposing a new resource provisioning technique. In order to show the generality of *Prob*, we use two common resource provisioning problems of workflows as use cases and discuss how *Prob* can improve the effectiveness of the existing solutions to both use cases.

The rest of this paper is organized as follows. Section 2 presents the background and preliminaries. Section 3 shows the effectiveness of probabilistic optimizations. Section 4 introduces pruning techniques for reducing the optimization overhead of *Prob*. We introduce the implementation details of *Prob* in Section 5 and extend the cloud dynamics in Section 6. We evaluate the proposed techniques in Section 7. We summarize related work in Section 8 and conclude this paper in Section 9.

2 Preliminaries

2.1 Data Processing Workflows

A data processing workflow (a job) can be described as a directed acyclic graph (DAG) [19]. A vertex in the DAG represents a task in the workflow while an edge represents the data dependency between two tasks. A task in a workflow performs certain data transformation to its input data. We adopt an existing approach [20] widely used for data-intensive task execution time estimation, and calculate the task execution time as the sum of the CPU, I/O and network time. We define a virtual entry vertex and a virtual exit vertex in a workflow. The entry vertex does nothing but stages input data while the exit vertex saves output results.

Resource provisioning for a workflow in the cloud decides the number and types of cloud instances required for executing the workflow. Performance is an important optimization metric for resource provisioning of data-intensive workflows in the cloud. The performance (i.e., makespan) of a workflow is usually highly affected by the resource provisioning decisions. Many resource provisioning methods have been proposed to optimize either the performance of workflows in the cloud [3], [6] or the monetary cost/energy efficiency of workflows with performance constraints [4], [5], [21].

2.2 Cloud Dynamics Terminology

The cloud has many dynamic factors causing system variations, such as the performance dynamics in I/O and network and system failure dynamics, which are common in the cloud and decisive to the execution of data processing workflows. Also, as will be shown in Section 2.3, the two types of dynamics are correlated, which further complicates the performance optimizations of workflows.

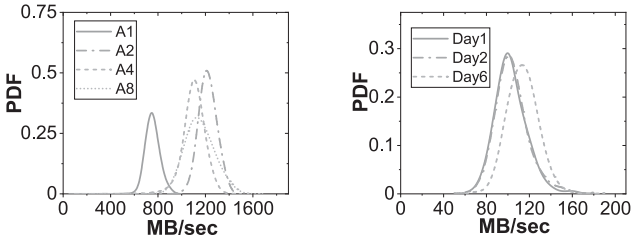
Using I/O (network) performance dynamics as an example, we formally define cloud dynamics as below.

Definition 1. *We view the I/O (network) bandwidth assigned to a running task as a random variable X. Then, the I/O (network) bandwidth dynamics can be described with a probability distribution $f_X(x)$, which represents the probability of $X = x$.*

The above definition greatly changes the formulation and the way of solving resource provisioning problems for workflows in the cloud. Consider an example of calculating the *expected* I/O time t of a task, given that the I/O data size is d. With the static definition of I/O bandwidth as a scalar value b ($b = \sum_x x \cdot f_X(x)$), we have $t = \frac{d}{b}$. With our dynamic definition on I/O bandwidth, the value of t is also dynamic. Define the I/O time of the task as a random variable (r.v.) T, then the probability distribution of T can be calculated as $f_T(t) = f_X(\frac{d}{t})$, where $f_X(x)$ is the probability distribution of I/O bandwidth. The expected value of T is $\sum_x \frac{d}{x} \cdot f_X(x)$ and can be different from the result of the static method (i.e., $\frac{d}{\sum_x x \cdot f_X(x)}$). We will show the effectiveness of the probabilistic model on improving resource provisioning results with case studies.

2.3 Features of Cloud Dynamics

Our previous study on the I/O and network performances of Amazon EC2 instances has revealed the spatial and temporal features of cloud dynamics [22]. Specifically, we have demonstrated that 1) the probability distributions of performances of different types of instances usually have similar patterns with different parameters and 2) the probability distributions of the cloud performance are stable within a short time period. To illustrate these features, we present the sequential I/O and network performances of Windows Azure instances measured during seven days in April 2018 in Fig. 1. More details about the measurements can be found in Section 5. Fig. 1a shows that, the I/O performances of the four types of instances have unignorable variations and the variations follow similar distributions. Fig. 1b shows the distributions of network performance between two A4 instances during different time of the measurement. It is clear that network performance distribution in a short period of time (e.g., one day) is more stable than that in a longer period (e.g., five days). These observations mean



(a) Sequential I/O performance (b) Network performance of A4

Fig. 1. (a) Spatial and (b) temporal features of the I/O and network performance distributions of Windows Azure instances.

that it is reasonable and also feasible to model and predict cloud performance using probabilistic distributions.

In large-scale cloud environments, failures have become the norm rather than the exception, and thus are decisive to the performance optimizations of cloud applications. Failure dynamics is usually more complicated than performance dynamics, due to the different causes of failure events. For example, system failures can be caused by hardware and software reasons such as failing motherboard and bad code. For cloud environments, the dynamic prices of cloud instances can also cause failures (e.g., out-of-bid events of spot instances). In this study, we mainly focus on the former type of failures that are most common in different systems. Specifically, we study the cloud dynamics in system failures using a production trace from Google Cloud [11], [23] which contains the resource usage information of a large cluster (11k machines) in 29 days. The consecutive intervals of failure events are often modeled using an exponential distribution, where the mean of the distribution is denoted as the Mean Time Between Failures (MTBF). Existing study [6] has found that different types of cloud tasks (e.g., with different priorities) often have different distributions of failure intervals. Fig. 2a shows the failure interval distributions of four types of tasks, where each type represents tasks with similar lengths. Specifically, type-1, type-2, type-3 and type-4 tasks have average execution time around 3, 10, 20 and 40 minutes, respectively. The mean of the four distributions range from 71, 203, 393 to 647 seconds, meaning that the MTBF of a task is correlated to its execution time. This can be further verified by Fig. 2b, which shows the correlation between task execution time and MTBF obtained from twelve types of tasks in the Google trace. Thus, for tasks with performance variations, the failure distributions of the task also vary, which makes the performance optimization of cloud workflows much more complicated.

3 Probabilistic Approach is Needed

As a motivating example, we present a budget-constrained scheduling problem for workflows, which involves I/O and network performance dynamics in the cloud. We first present an *existing* solution [10] under static performance notions and then discuss our solution considering cloud dynamics. We show that cloud dynamics can greatly affect the optimization effectivenesses.

3.1 Budget-Constrained Scheduling

Cloud providers usually offer multiple instance types with different capabilities and prices. In this problem, we aim to select a suitable instance type for each task in a workflow to

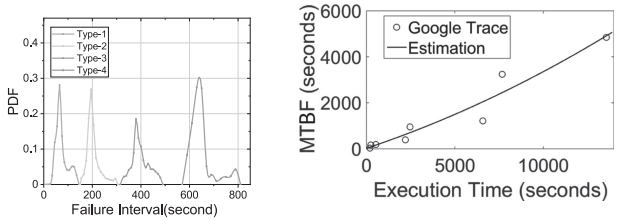


Fig. 2. Failure dynamics in Google trace: (a) failure interval distributions of four types of tasks; (b) relationship between task execution time and MTBF.

minimize the workflow execution time while satisfying the budget constraint.

Consider a workflow with N tasks running in a cloud with K types of instances. The optimization variable of this problem is vm_{ij}, meaning assigning instance type j ($j = 0, 1, \ldots, K-1$) to task i ($i = 0, 1, \ldots, N-1$). The value of vm_{ij} is 1 (i.e., task i is assigned to instance type j) or 0 (otherwise). We denote the execution time of task i on instance type j using r.v. T_{ij} with a probability distribution $f_{T_{ij}}(t)$. We denote the workflow execution time (i.e., makespan) with r.v. T_w and calculate its distribution $f_{T_w}(t)$ using the execution time distributions of tasks on the critical path (denoted as CP). The user-defined budget constraint is denoted as B, which includes the instance rental cost and networking cost. The unit time rental price of instance type j is denoted as U_j. The networking cost of task i transferring intermediate data to its child tasks is denoted as C^i_{net}. $E[X]$ denotes the expected value of a r.v. X. Formally, we formulate the problem as following.

$$\min E[T_w] = \min E\left[\sum_j \sum_{i \in CP} T_{ij} \times vm_{ij}\right] \tag{1}$$

$$s.t. \quad \sum\nolimits_i \left\{ \sum\nolimits_j E[T_{ij}] \times vm_{ij} \times U_j + C^i_{net} \right\} \leq B \tag{2}$$

$$\sum\nolimits_j vm_{ij} = 1, \forall i \in 0, \ldots, N-1 \tag{3}$$

3.2 A Static Solution

The budget-constrained scheduling problem has been studied by a number of existing studies [3], [4], using either heuristics or model-based methods to minimize the optimization goal in Equation 1. However, most of them assume static task execution time during the optimization. In this subsection, we introduce an existing static method [10] which formulates resource provisioning problems as search problems, and adopt generic search or more efficient A^* search to search for a good solution. We choose this algorithm for its generality. We briefly present the behavior of the search algorithm as below (also see Algorithm 1).

- State representation: A state s in the solution space is modeled as a N-dimensional vector, where s_i stands for the instance type assigned to task i. vm_{ij} equals to 1 if $j = s_i$ and 0 if otherwise.
- Initial state: All tasks are assigned to instance type 0 initially.
- State traversal: The search process is like a BFS search. A state s on the lth level of the search tree transits to another state by incrementing the lth dimension of s.
- State evaluation: Each found solution is evaluated using Equation 1 for the optimization goal and Eq. (2) for the optimization constraint. After evaluating a solution, we compare it with the best found solution and keep the one which has a better optimization goal while satisfying the budget constraint at the same time.
- Pruning rule: If a state s has violated the budget constraint, we do not further traverse the descendants from s because the states on the branch must have higher monetary cost than state s (assuming instance type 0 is the cheapest).

Algorithm 1
Static (left) and Probabilistic (right) Search Algorithms for Budget-Constrained Scheduling on Workflow W

1:	Preserve the best solution $CurBest$;	Preserve the best solution $CurBest$;
2:	P = **PreProcessing(**$W, entry, exit$**)**;	P = **PreProcessing(**$W, entry, exit$**)**;
3:	... ▷ State traversal	... ▷ State traversal
4:	Given a solution s, assign instance configurations in s to each task in W;	Given a solution s, assign instance configurations in s to each task in W;
5:	Initialize workflow execution time as $time = 0$;	Define workflow execution time distribution $Time$;
6:	**for each** $path$ in P **do**	**for each** $path$ in P **do**
7:	**TaskBundling(TaskClustering(**$path$**))**;	**TaskBundling(TaskClustering(**$path$**))**;
8:	Initialize the length of $path$ as $t = 0$;	Define the length distribution of $path$ as T;
9:	**for each** task tk on $path$ **do**	**for each** task tk on $path$ **do**
10:		**if** tk is the first task in $path$ **then**
11:		$T = T_{tk}$;
12:	$t = t + t_{tk}$;	**else** T = **Add(**T, T_{tk}**)**;
13:	**if** $t \geq time$ **then**	**if** $path$ is the first path in P **then**
14:	$time = t$;	$Time = T$;
15:		**else** $Time$ = **Max(**$Time, T$**)**
16:	... ▷ Evaluate the expected cost	... ▷ Evaluate the expected cost
17:	**if** $cost \leq B$ **then**	**if** $cost \leq B$ **then**
18:	**if** $time < CurBest.time$ **then**	**if Compare** $(CurBest.Time, Time) > 0.5$ **then**
19:	$CurBest = s$;	$CurBest = s$;
20:	... ▷ Go back to state traversal	... ▷ Go back to state traversal

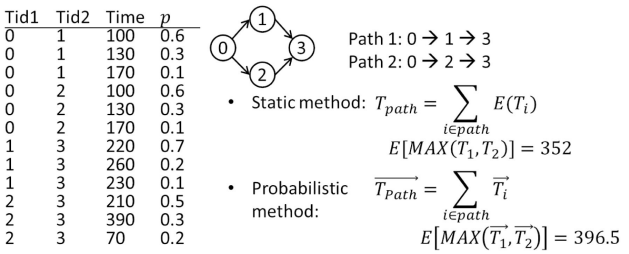


Fig. 3. An example of probabilistic optimizations.

- Termination: The search process terminates if the entire solution space has been traversed or a predefined number of iterations is reached. On termination, the feasible state with the best optimization goal is returned as the final result.

3.3 A Probabilistic Method

To incorporate cloud dynamics into the static method, we have made two efforts. First, we take the I/O and network performance dynamics as input to estimate the execution time of a task. As a result, the task execution time T_{ij} is a random variable with a probability distribution calculated from the I/O and network performance distributions. Second, whenever T_{ij} is used in the search process, we perform probabilistic calculations on distributions, e.g., adding task execution time distributions and finding the maximum execution time distribution of multiple paths (i.e., finding the critical path) to evaluate the optimization goal in Equation 1, and comparing the evaluation metric distributions of two found solutions to find a good solution.

The distribution addition with two independent r.v., e.g., $Z = X + Y$, can be calculated as below.

$$f_Z(z) = \int_{-\infty}^{\infty} f_Y(z - x) f_X(x)\, dx. \tag{4}$$

Finding the maximum distribution of two independent r.v., e.g., $Z = max\{X, Y\}$, can be calculated as below. $F_X(x)$ and $F_Y(y)$ are the cumulative distribution functions (CDFs) of X and Y, respectively.

$$f_Z(z) = \frac{d}{dz} F_Y(z) F_X(z) = F_Y(z) f_X(z) + F_X(z) f_Y(z). \tag{5}$$

We adopt the following definition to compare two evaluation metric distributions. Given two independent r.v. X and Y, we have $X > Y$ if $P(X > Y) > 0.5$, where

$$P(X > Y) = \int_{-\infty}^{\infty} \int_{y}^{\infty} f_X(x) f_Y(y)\, \mathrm{d}x\, \mathrm{d}y. \tag{6}$$

Fig. 3 gives a concrete example to show the difference between static and probabilistic optimizations. Consider the budget-constrained scheduling problem for a simple workflow with four tasks. The execution time distributions of the tasks have been calculated using I/O and network performance distributions, as shown in the table. There are 2 paths in the workflow. To optimize the workflow performance, we need to first identify the critical path and then schedule tasks on the critical path to more powerful instances. We make the following two observations.

3.3.1 Probabilistic optimization is more effective

With the traditional static method, the expected lengths of edges are used for evaluations and path 2 is returned as the critical path with a length of 352. With our probabilistic method (will be introduced in details in Section 4), we first calculate the length distributions of the two paths and then calculate the maximum of the two distributions. Given the length distributions of path 1 and 2 (denoted as $\vec{T_1}$ and $\vec{T_2}$, respectively), we calculate the expected length of the critical path to be 396.5. The probability of path 2 being the critical path is only 0.41. Theoretically, this observation is supported by the following lemma.

Lemma 1. *Given two independent r.v. X and Y, for the r.v. $Z = max\{X, Y\}$, we have $E[Z] \geq max\{E[X], E[Y]\}$.*

Proof. According to Equation 5, we have

$$\begin{aligned} E[Z] &= \int_{-\infty}^{\infty} z f_Z(z)\, \mathrm{d}z = \int_{-\infty}^{\infty} z[F_Y(z) f_X(z) + F_X(z) f_Y(z)]\, \mathrm{d}z \\ &= \int_{-\infty}^{\infty} z \int_{-\infty}^{z} f_Y(y) f_X(z)\, \mathrm{d}y\, \mathrm{d}z + \int_{-\infty}^{\infty} z \int_{-\infty}^{z} f_X(x) f_Y(z)\, \mathrm{d}x\, \mathrm{d}z \\ &\geq \int_{-\infty}^{\infty} y \int_{-\infty}^{x} f_Y(y) f_X(x)\, \mathrm{d}y\, \mathrm{d}x + \int_{-\infty}^{\infty} y \int_{x}^{\infty} f_X(x) f_Y(y)\, \mathrm{d}x\, \mathrm{d}y \\ &= \int_{-\infty}^{\infty} y f_Y(y)\, \mathrm{d}y \int_{-\infty}^{\infty} f_X(x)\, \mathrm{d}x = E[Y]. \end{aligned} \tag{7}$$

Similarly, we have $E[Z] \geq E[X]$. Thus Lemma 1 is proven. □

Lemma 1 proves that, for parallel structured dataflows, the static method using average task execution time as input always *under-estimates* the expected workflow execution time. Thus, the evaluation results of static and probabilistic approaches are different, and ignoring cloud dynamics can lead to inaccurate or even incorrect optimization results.

3.3.2 Probabilistic optimization is costly

A straightforward way to implement probabilistic optimization is to use the sampling-based Monte Carlo (MC) approach, which calculates all possible results using input probability distributions. For example, to obtain the sum distribution of Equation 4, we perform M times of MC calculations. In each calculation, we randomly sample values from the discretized distributions $f_X(x)$ and $f_Y(y)$ to get a possible result of the sum. After the M times calculations, we can create the sum distribution using the M calculated sum results. Thus, the time complexity of adding two task execution time distributions is $O(M)$ and for adding n tasks is $O((n-1)M)$. Note that M is usually very large to achieve good accuracy. In our experiments for the budget-constrained scheduling problem, we set M to be 10,000 for the MC method. In this case, the probabilistic addition operation is 10,000 times more costly than the static operation. For a workflow with complex structure and a large number of tasks, the time complexity for calculating the workflow execution time distribution is high. Similarly, with the MC approach, the time complexity of calculating the maximum distribution is $O(M)$ and of comparing two evaluation metric distributions is $O(M^2)$. Thus, the computation overhead of probabilistic optimization is prohibitively high due to the large M, complex workflow structures and costly distribution comparisons.

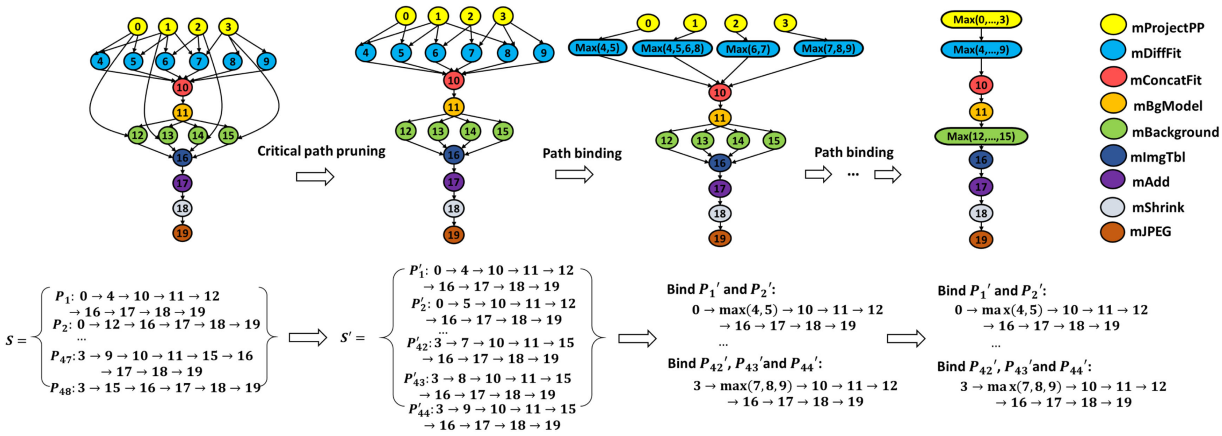


Fig. 4. An example of pre-processing for Montage workflow.

In summary, probabilistic distributions improve the effectiveness of workflow optimizations in the dynamic cloud environment while at the same time causing a large optimization overhead. This motivates us to develop an effective and efficient approach for resource provisioning problems of workflows in the cloud.

4 Efficient Probabilistic Optimizations

4.1 Optimizations in a Nutshell

In this paper, we propose a probabilistic optimization approach named *Prob* for incorporating cloud dynamics into workflow optimizations. As discussed above, *Prob* has a large overhead. In this section, we introduce three simple yet effective pruning stages to improve the efficiency of *Prob*, including pre-processing, workflow-specific and distribution-specific optimizations.

First, calculating the makespan of a workflow is a common operation in many resource provisioning problems of workflows. Thus, we propose pre-processing pruning to reduce the overhead of this important calculation and hence reduce the overhead of probabilistic optimizations. This stage is an offline optimization stage, as a workflow only needs to be optimized once and for all. Second, we propose workflow-specific optimizations using existing workflow transformation techniques to reduce the overhead of evaluating one instance configuration solution. Third, we propose two pruning techniques to reduce the overhead of comparing multiple solutions. The latter two stages are called at the runtime of solution search.

4.2 Pre-Processing

Due to cloud dynamics, the execution time of a workflow is represented as a random variable. To obtain the probability distribution of the random variable, we first decompose a workflow into a set of paths starting from the entry task to the exit task of the workflow. The execution time distribution of the workflow is the maximum of execution time distributions of all paths in the set. We further propose a *critical path pruning* and a *path binding* technique to reduce the number of paths in the set and hence reduce the overhead of calculating workflow execution time distribution. In Fig. 4, we use the structure of a real-world scientific workflow named Montage [24] as an example to demonstrate the effectiveness of the two pruning techniques.

Given a workflow, we can easily enumerate all paths in the workflow using depth-first search in $O(V + E)$ time, where V and E are the number of vertices and edges in the workflow, respectively. Assume all paths from the entry task to the exit task are stored in a set S. For example, in Fig. 4, we initially have 48 paths in S. We propose two techniques to reduce the size of S.

4.2.1 Critical Path Pruning

This technique eliminates the paths that are impossible to be the critical path from S. For example, in Fig. 4, as the execution time of path P_2 has to be shorter than that of path P_1, it is clear that P_2 is impossible to be the critical path. Similarly, path P_{48} is impossible to be the critical path due to path P_{47}. Thus, we can eliminate such paths from S and reduce the size of S from 48 to 44. We denote the set of paths after pruning as S'.

The critical path pruning follows the following rule: given any two paths P_1 and P_2 between the entry and exit tasks of a DAG, if the set of nodes in P_1 is a subset of that in P_2, P_1 is not the critical path. We order all paths in S according to their lengths, and iteratively compare the longest path with the ones shorter than it using the above rule. The worst-case time complexity of this technique is $O(|S|^2 \times V)$.

4.2.2 Path Binding

Given S' in Fig. 4, we need in total 352 distribution additions to evaluate path distributions and 43 distribution maximum operations to obtain workflow execution time distribution. However, we find that when evaluating P'_1 and P'_2 separately, many of the distribution additions are repetitive. By binding P'_1 with P'_2, we can reduce the overhead of 16 distribution additions to 8 distribution additions and one operation on calculating the maximum distribution of task 4 and 5. We can apply the binding to all paths in S' and reconstruct them to a single path as shown in S''. With S'', the overhead of calculating workflow execution time distribution is reduced to 8 additions and 11 maximum operations. As the distribution addition and maximum operations have the same time complexity with the MC method, path binding reduces the overhead by 95 percent in this example.

Formally, the path binding technique binds two paths with the same length L in the following way. We compare each node of one path with the node in the same position of another path in order. If the ith nodes ($i = 0, \ldots, L-1$) are the same, they are adopted as the ith node of the binded path. Otherwise, we bind the ith nodes of the paths as one node, the execution time of which is the maximum of the ith nodes of the two paths. Assume there are k binded nodes in the binded path, the calculations saved by binding the two paths is $L - k - 1$. We iteratively select two paths in S' with the best gain to bind until no gain can be further obtained or the largest number of iterations have been reached. This technique is very useful to data processing workflows such as MR jobs, where the input data partitions go through the same levels of processing (i.e., all paths have the same length). The worst-case time complexity of this pruning is $O(|S'|^2 \times L)$, where L is the average length of paths in S'.

Note that the pre-processing optimization only needs to be applied once at offline and the results can be reused for different resource provisioning problems of the same workflow structure.

4.3 Workflow-Specific Optimizations

After the pre-processing stage, evaluating a found solution, namely calculating the execution time distribution of a workflow according to the instance configurations indicated by the solution, is simplified to calculating the distributions of several paths. The workflow execution time distribution is calculated as the maximum distribution of all path distributions.

We decompose the calculation of workflow execution time distributions into two constructive operations, namely *ADD* and *MAX*. The *ADD* operation can be applied to the distributions of two dependent tasks while the *MAX* operation can be applied to the distributions of two parallel tasks or two paths from the pre-processing result. We define *ADD* and *MAX* as binary operators, which operate on two probability distributions at a time.

The calculation of *ADD* and *MAX* operations are introduced in Eqs. (4) and (5), respectively. As discussed in Section 3, a straight-forward implementation of *ADD* and *MAX* is to use the sampling-based MC method. However, this implementation is very costly due to complicated workflow structures and the large sampling size. Thus, we adopt two existing workflow transformation optimizations to reduce the overhead of *ADD* and *MAX*.

For many resource provisioning problems of workflows, existing studies have proposed various workflow transformations to simplify workflow structures, such as *task bundling* [25] and *task clustering* [26]. Both operations attempt to increase the computational granularity of workflows and reduce the resource provisioning overhead. In this paper, we discuss the two operations from the perspective of probabilistic optimizations.

4.3.1 *Optimizing* ADD *with task bundling*

Task bundling treats two pipelined tasks which have the same assigned instance type as one task, and runs them on the same instance sequentially. For example, in Fig. 4, we can bundle tasks 10 and 11 in P_1'' as one task when they have the same assigned instance type. Task bundling can increase instance utilization and reduce the amount of data transfer between dependent tasks. This technique gives us the opportunity to reduce the overhead of *ADD* operation. When applying the *ADD* operation on two tasks, if the tasks satisfy the requirement of task bundling, we can directly generate the resulted distribution using the I/O and network profiles of the two tasks without calculating their distributions separately.

4.3.2 *Optimizing* MAX *With Task Clustering*

In scientific workflows, tasks on the same level, using the same executable for execution and having the same predecessors are identified as *equivalent tasks*. In MR workflows, we can easily identify the map/reduce tasks on the same level as equivalent tasks. Task clustering groups two equivalent tasks assigned to the same instance type as one task and schedule them to the same instance for parallel execution. For example, in Fig. 4, tasks 12 to 15 in P_1'' can be identified as equivalent tasks and grouped as one task with three times of clustering. Note that, the four tasks cannot be viewed as equivalent tasks before the pre-processing (e.g., in S and S'), since they have different predecessors. Thus, our pre-processing optimization increases the possibility of further optimizations. The number of equivalent tasks to be scheduled on the same instance is determined by the resource capacity of the instance. The task clustering technique offers opportunity to reduce the overhead of *MAX* operation.

As equivalent tasks have the same (or similar) resource requirements and are executed on the same instance in parallel, we can use the execution time distribution of one of the equivalent tasks to represent the distribution of the clustered task. This can be verified with Equation 7, where $E[max\{X, Y\}] = E[X] = E[Y]$ when X and Y are the same random variable. However, we must consider the performance degradation of the clustered tasks due to resource contention. Thus, when applying the MAXoperation on two tasks, if they satisfy the requirement of task clustering, we use $f_X(\frac{x}{2})$ to represent the resulted distribution assuming $f_X(x)$ is the execution time distribution of one of the two tasks. With this optimization, we reduce the overhead of *MAX* from $O(M)$ to $O(1)$.

4.4 Distribution-Specific Optimizations

Given two resource provisioning solutions (i.e., two vectors of instance configurations for each task in the workflow), we need to evaluate each individual solution and compare their evaluation metric distributions to find a good solution. In the above, we have proposed to reduce the overhead of evaluating one solution. Thus, in this subsection, we mainly focus on reducing the overhead of solution comparisons to reduce the overhead of probabilistic optimizations. Specifically, we propose a partial solution evaluation technique and adopt an existing pruning in probabilistic database to reduce the overhead of solution comparisons.

4.4.1 *Partial Solution Evaluation*

As the purpose of solution evaluations is to compare their quality and find a good one, we propose a partial solution

evaluation technique. The idea is to avoid fully evaluating two solutions while guaranteeing the same solution comparison result. When evaluating two found solutions s and s', we only calculate the distributions of tasks with different configurations in s and s' (i.e., $\forall i$ where $s_i \neq s'_i$). We claim that comparing the partially evaluated distribution of s with that of s' gives the same result as comparing the fully evaluated distributions. This property is guaranteed by Lemma 2.

Lemma 2. *Given two r.v. X and Y, assume $X \geq Y$. Then we have $g(X, Z) \geq g(Y, Z)$ for any r.v. Z independent from X and Y, where $g(\cdot)$ is ADD or MAX.*

Proof. When $g(\cdot)$ is *ADD*, as $P(X \geq Y) > 0.5$, we have $P(X + Z \geq Y + Z) > 0.5$. When $g(\cdot)$ is *MAX*, denote $max\{X, Z\}$ as a r.v. A, we have

$$P(A \geq max\{Y, Z\}) = P(A \geq Y, Y \geq Z) + P(A \geq Z, Z > Y) \tag{8}$$

Further, we have

$$\begin{aligned} 1 = {} & P(A \geq Y, Y \geq Z) + P(A \geq Y, Y < Z) \\ & + P(A < Y, Y \geq Z) + P(A < Y, Y < Z) \end{aligned} \tag{9}$$

and

$$\begin{aligned} & P(A < min\{Y, Z\}) = P(A < Y, Y < Z) + P(A < Z, Y \geq Z) \\ & = P(A < Y, Y < Z) + P(Y \geq Z) - P(A \geq Z, Y \geq Z). \end{aligned} \tag{10}$$

Thus, we have the following:

$$\begin{aligned} & P(A \geq max\{Y, Z\}) + 1 - 1 \\ & = P(A \geq Y, Y \geq Z) + P(A \geq Y, Y < Z) \\ & \quad + P(A \geq Z, Z > Y) + P(A \geq Z, Z \leq Y) - P(A \geq Z, Z \leq Y) \\ & \quad + P(A \geq Y, Y \geq Z) + P(A < Y, Y \geq Z) + P(A < Y, Y < Z) - 1 \\ & = P(A \geq Y) + P(A \geq Z) + P(A < min\{Y, Z\}) - 1 \\ & \geq 0.5 + 1 + 0 - 1 = 0.5. \end{aligned} \tag{11}$$

Thus, Lemma 2 is proven. □

Assume we visit in total n solutions during the solution search process, the overhead of full solution evaluations would be $O(n \times (N-1) \times M)$, where N is the number of tasks in a workflow. With our partial evaluation technique, assume the average number of different configurations in a pair of solutions is N', the overhead of solution evaluations is reduced to $O(2n \times (N'-1) \times M)$. In many resource provisioning methods, such as the search method introduced in Section 3.2, adjacent solutions on the search tree only differ by a few configurations (i.e., $N' \ll \frac{N}{2}$). Thus, the partial evaluation technique can greatly reduce solution evaluation overhead. This pruning can be disabled when comparing two solutions where half of the tasks are assigned with different configurations.

4.4.2 Distribution Comparison Pruning

After solution evaluations (either partial or full evaluations), we repeatedly use Eq. (6) for distribution comparisons during the solution search process. The complexity of one distribution comparison is $O(M^2)$, which is very high due to the large sampling size M. Thus, we adopt an existing pruning technique [18] in probabilistic database to prune the unnecessary calculations of Equation 6. The basic idea is briefly described as follows.

Assume r.v. X (resp. Y) has the lower and upper bound of $X.l$ (resp. $Y.l$) and $X.r$ (resp. $Y.r$), respectively. With Eqs. (12) and (13), we can estimate the lower bound or upper bound of $P(X > Y)$. If the lower bound is greater than 0.5 or the upper bound is less than 0.5, we can prune the calculation of Eq. (6).

$$\text{If } X.l \leq Y.r \leq X.r, P(X > Y) \geq 1 - F_X(Y.r) \tag{12}$$

$$\text{If } X.l \leq Y.l \leq X.r, P(X > Y) \leq 1 - F_X(Y.l) \tag{13}$$

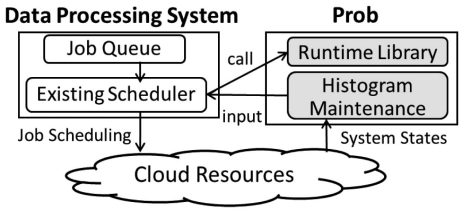


Fig. 5. System implementation.

5 Implementation Details

Fig. 5 shows how Prob can be used by existing data processing systems. We introduce two main implementation details of *Prob*.

First, *Prob* stores system states (e.g., cloud performance calibrations) and maintains the histograms (i.e., discrete distributions) of cloud dynamics. Those distributions are taken as input by the *Prob*-enabled system schedulers to find the optimal job scheduling solution. Specifically, we consider two types of system states, including 1) instance-related states such as price, CPU, I/O and network bandwidths of all instance types and 2) job-related states such as job start time, finish time and failures. System states are calibrated and updated periodically. We adopt the same method as in our previous work [22] to calibrate system states.

We maintain system state calibrations in the form of histograms. The number of bins in a histogram is carefully selected to balance the trade-off between optimization accuracy and overhead. Based on our sensitivity studies, we set this parameter to 200 by default. The histograms are used by *Prob* for probabilistic evaluations of solutions. We adopt a window-based prediction method to estimate the distributions of cloud dynamics in the future. The estimation simply assumes that the distribution of a dynamics factor in the current window is the same as that of the previous window. We select the window size which generates the shortest distance between distributions in two adjacent windows. According to sensitivity studies, we set this parameter to one day by default for cloud I/O and network performance distributions.

Second, *Prob* offers a runtime library which exposes the probabilistic operations (e.g., *ADD* and *MAX*) and pruning

TABLE 1
Description of APIs in the runtime library.

API	Parameter(s)	Return
$V(\vec{P}) \leftarrow PreProcessing(W, s, t)$	W: The workflow to be optimized s, t: The source and target node ID	$V(\vec{P})$: A set of paths $\vec{P}$ starting from s to t in the workflow W
$P' \leftarrow TaskBundling(P)$ $P' \leftarrow TaskClusterin(P)$	P: The path to be optimized	P': The path after optimization
$P_c \leftarrow Max(P_a, P_b)$ $P_c \leftarrow Add(P_a, P_b)$ $p \leftarrow Compare(P_a, P_b)$	P_a, P_b: Two probability distributions	P_c: The max/sum distribution of P_a and P_b p: Probability of $P_a > P_b$
$(P_1, P_2) \leftarrow PartialEval(s_1, s_2)$	s_1, s_2: Two found solutions	P_1, P_2: Evaluated distributions of s_1 and s_2, respectively

techniques designed for resource provisioning problems of workflows. Existing schedulers of data processing systems can call the APIs in the library when implementing their resource provisioning methods to incorporate probabilistic optimizations. The data processing system schedules jobs to the cloud resources according to the optimized scheduling solutions. There are three types of APIs in the library, as summarized in Table 1.

We take the budget-constrained scheduling problem as an example to show how users can modify their existing method using provided APIs. Algorithm 1 shows the modifications users need to make when incorporating probabilistic optimizations (right) into the existing search algorithm [10] introduced in Section 3.2 (left). Users mainly need to modify their state evaluation logics using our APIs (partial solution evaluation is omitted for clarity of presentation). Given a workflow W to optimize, users first call the PreProcessing() API to get an pruned set of critical paths P (Line 2). Given a found solution, users assign the instance configurations to each task of the workflow and call the TaskBundling() and TaskClustering() APIs to simplify the paths (Line 4-7). Note that, for fair comparison, we implement the pre-processing, task bundling and task clustering optimizations for both methods. The Add() and Max() APIs are used to evaluate the distributions of paths and the distribution of workflow (Line 8-15). Lines 17-19 show how to use the Compare() API to maintain the best found solution. With the APIs in the runtime library, users need little changes (highlighted with underlines) to their existing implementation in order to enable probabilistic optimizations.

6 Fault Tolerance Problem

Besides the I/O and network performance dynamics, many other dynamics exist and affect the resource provisioning results of data processing workflows in the cloud. For example, the job failure events in the cloud can be dynamic and follow different probability distributions for jobs with different lengths [6]. As failures can happen in the cloud very often [6], [27], it is important to explore fault tolerance techniques to ensure the execution reliability and correctness for workflows. In this section, we present a fault tolerance problem, which considers *the combined effect of cloud performance dynamics and dynamic system failures* to workflow executions. We study fault tolerance techniques based on the checkpointing/restart mechanism. Our goal is to determine the checkpoint interval for each task of a workflow in order to minimize the expected makespan of the workflow.

Problem Formulation. The optimization variable of this problem is the checkpoint interval V_i for each task i. The checkpoint cost is denoted as C_k and the recovery cost is denoted as C_r. We model the time interval between two failure events in task i as a random variable with a probability distribution $f_{F_i}(t)$. The expected value of r.v. F_i is called mean time between failure (MTBF), and it has been observed that MTBF of different tasks is correlated to the task execution time [6]. We perform checkpointing for each task of a workflow individually according to the optimized checkpoint interval. Once a task fails, it is restarted with the same configuration from the last checkpoint. We can formulate this problem as following. Note that, the formula involves two types of probability distributions, namely the execution time and failure interval distributions of each task.

$$\min E\left[\sum\nolimits_{i \in CP} T_i \times \left(1 + \frac{C_k}{V_i} + \frac{C_r \cdot V_i}{2F_i}\right)\right]. \tag{14}$$

A Static Solution. An extended Young's formula [6] has recently been proposed to optimize the checkpoint interval in a checkpointing/restart mechanism for cloud jobs. We use this method to calculate the optimal number of checkpoints ($x_i^* = T_i/V_i$) for a task i as in Formula 15, where $E(T_i)$ is the *expected* task execution time without failure and $E(Y_i)$ is the *expected* number of failure events occurred during the execution of the task (i.e., $Y_i = T_i/F_i$).

$$x_i^* = \sqrt{\frac{E(T_i) \cdot E(Y_i)}{2C_k}}. \tag{15}$$

The Probabilistic Method. With *Prob*, both T_i and Y_i in Equation 15 are represented as random variables. The probability distribution of Y_i can be calculated using $f_{F_i}(t)$ and $f_{T_i}(t)$. It is easy to observe that $E[Y_i] \neq E[T_i]/E[F_i]$, which means that the static method would lead to suboptimal fault tolerance solutions.

To demonstrate the effectiveness of our probabilistic method and show the complexity of the fault tolerance problem, we give a concrete example as shown in Fig. 6. Consider a simple task with execution time and failure

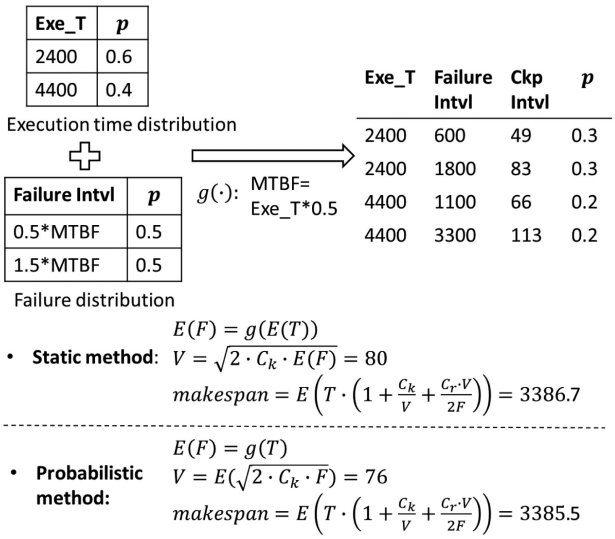


Fig. 6. An example of the failure tolerance problem.

distributions as shown in the figure. The distribution of failure interval is related to the MTBF, which in turn is correlated to task execution time [6]. We denote the correlation using function $g(\cdot)$. The checkpoint overhead is 2s. Using Eqs. (15) and (14), we can calculate the optimized checkpoint interval and further obtain the expected makespan of the task. According to the example, our probabilistic method generates shorter checkpoint interval and shorter expected makespan than the static method. For a real workflow with many tasks, our probabilistic method obtains even higher reduction on the expected makespan than static method. Also, due to the correlation of two types of system dynamics, the computation complexity of the probabilistic method for the fault tolerance problem is much higher than that of the budget-constrained scheduling. Consider the sampling-based MC method and set the sampling size of a probability distribution to M. Then the computation complexity of an ADD operation for the fault tolerance problem is M^2 and for the budget-constrained scheduling is only M.

7 Evaluations

We evaluate the effectiveness and efficiency of *Prob* using the two workflow optimization problems. To demonstrate the effectiveness of probabilistic optimizations, we compare existing state-of-the-art workflow optimization approaches with and without *Prob* incorporation. For efficiency, we compare the optimization overhead of *Prob* with the overhead of MC. We run the compared approaches on a server with 64GB DRAM and two 8-core Intel Xeon CPUs. Workflows are executed on real clouds or a cloud simulator.

7.1 Experimental Setup

7.1.1 Data processing workflows

We consider data processing workflows from scientific and data analytics applications, denoted as scientific and MR workflows, respectively. Fig. 7 shows the structure of the workflows.

The tested scientific workflows include the I/O-intensive Montage workflow and data-intensive CyberShake workflow. We create Montage workflows with 10,567 tasks each using

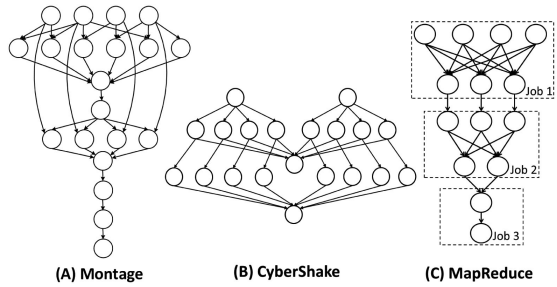


Fig. 7. Structures of experimented data processing workflows.

Montage source code. The input data are the 2MASS J-band images covering 8-degree by 8-degree areas retrieved from the Montage archive [28]. Since CyberShake is not open-sourced, we construct synthetic workflows with 1000 tasks each using workflow generator [24].

The MR workflows include two TPC-H queries, Q1 and Q9, expressed as Hive programs. Q1 is a relatively simple selection query, and Q9 involves multiple joins. Both queries have order-by and group-by operators. The input data size is around 500GB (the scale factor is 500) and is stored on the local HDFS. A Hive query is usually composed of several MR jobs. Q1 is composed of two MR jobs and Q9 is composed of seven MR jobs. We further experimented with the BigDataSort workload from BigDataBench [29] as a complementary big data workflow to Q1 and Q9 workflows. BigDataSort has 110 tasks and uses 100 GB data randomly generated from Wikipedia entries as input data.

7.1.2 Implementation details

For budget-constrained scheduling, we conduct experiments on both real clouds and simulator to study workflow optimization in a controlled and in-depth manner. For real clouds, we adopt both Amazon EC2 and Windows Azure, which are the most popular public clouds with different dynamics features. We use four types of instances with different prices and computational capabilities on each cloud, including m1.small, m1.medium, m1.large and m1.xlarge on Amazon EC2, and A1, A2, A4 and A8 on Windows Azure. On real clouds, we utilize an existing workflow management system named Pegasus [19] to execute scientific workflows, and deploy Hadoop and Hive to run the TPC-H queries. We use Spark to run the BigDataSort workload. We adopt an existing cloud simulator [10] designed based on CloudSim [30] to simulate the dynamic cloud environment. As CyberShake is not open-sourced, we use simulations for all CyberShake evaluations.

We run all fault tolerance evaluations with trace-driven simulations, using system performance traces of Amazon EC2 and performance traces adopted from [31], which simulate performance dynamics of HPC systems, to study the failure behaviors of different types of systems. We evaluate the average makespan of workflows executing on a virtual cluster of 50 instances using simulator. We use the performance trace of the m1.xlarge instances for Amazon EC2 evaluations. The I/O traces adopted from [31] are collected by measuring the I/O bandwidth given to one application when simultaneously running multiple applications in a

dedicated HPC environment, which is deployed on the Rennes site of Grid'5000 [32] (refer to [31] for more details on the experimental setting). Hereafter we refer to this setting as HPC system.

7.1.3 Parameter setting

In each experiment, we submit 100 jobs for each workflow with job arrival time in a Poisson distribution ($\lambda = 0.1$ by default), which is sufficiently large for measuring the stable performance. We present the detailed experimental settings for each workflow optimization problem as follows.

Budget-Constrained Scheduling. We compare *Prob* with two static algorithms named *Deco* and *Deco-W*, which optimize the performance of workflows using the expected and worst-case (90th percentile) of cloud dynamics distributions, respectively. The static algorithms adopt the search method [10] as shown in Algorithm 1. We set a loose budget and a tight budget as $\frac{B_{min}+3B_{max}}{4}$ and $\frac{3B_{min}+B_{max}}{4}$, respectively, where B_{min} and B_{max} are the expected cost of executing all tasks in the workflow on m1.small (A1) instances and on m1.xlarge (A8) instances, respectively. Given a budget, we run the compared algorithms for 100 times and compare the obtained average monetary cost and execution time. With different budgets, we can see clear trade-off between the cost and time optimization results.

Fault Tolerance Optimization. The makespan of a workflow includes the task execution time, checkpoint/recovery overhead and the time wasted on the rollback of task execution to its closest checkpoint. We set the recovery overhead to 1 second by default. For Amazon EC2, we vary the checkpoint overhead from 2, 10 to 50 seconds and vary the recovery overhead from 1 to 10 seconds to study the behavior of *Prob* under different system performances. For HPC system, we fix the checkpoint overhead to 50 seconds due to the large checkpoint overhead of parallel file systems in data-intensive clusters [33]. We study the failure events of tasks with different lengths in Google Trace [11] and correlate the MTBF of tasks with their lengths using polynomial regression as shown in Fig. 2b. We generate failure events for each task following Poisson distribution, where the λ parameter of the distribution is set according to the MTBF of tasks with similar execution time in the Google Trace. Assume no failure happens when checkpointing.

We compare *Prob* with the static Young's formula [6] to study the impact of cloud dynamics in performance and system failures. To further breakdown the impact of the two dynamics, we compare *Prob* with *ProbNP* and *ProbNF*, which are *Prob* without considering cloud dynamics in performance and failures, respectively. *ProbNP* uses the expected execution time of tasks and *ProbNF* uses the expected MTBF for the optimization.

7.2 Optimization Effectiveness

7.2.1 Budget-constrained Scheduling

Fig. 8 and 10 present the average execution time and monetary cost optimization results of the compared algorithms on Amazon EC2 and Windows Azure, respectively. The execution time results are normalized to those of *Prob* and the monetary cost results are normalized to the budget. In the following, the error bars show the standard deviation of the results.

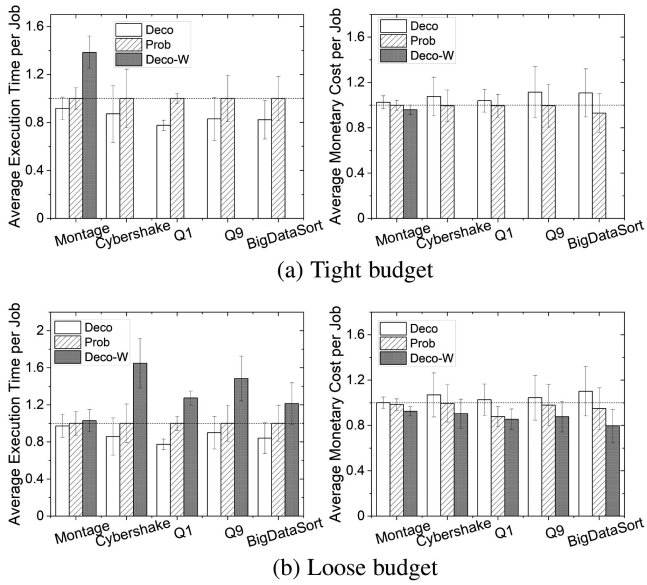


Fig. 8. Normalized results of budget-constrained scheduling under tight and loose budget on Amazon EC2.

Results on Amazon EC2. Compared to *Deco-W*, *Prob* is able to obtain better performance results while satisfying the budget constraint under all cases. Specifically, *Prob* reduces the expected execution time over *Deco-W* by 38 percent for Montage under the tight budget, and by 3, 65, 27, 48 and 21 percent for Montage, CyberShake, Q1, Q9 and BigDataSort, respectively, under the loose budget. *Deco-W* cannot find a feasible solution for CyberShake, Q1, Q9 and BigDataSort under the tight budget as these workflows are more data-intensive than Montage. The reason that *Prob* outperforms *Deco-W* is that *Deco-W* tends to over-estimate the monetary cost when searching for a good solution. Thus, in cases of tight budget, it is hard for *Deco-W* to find a feasible solution. For those data-intensive workflows, *Deco-W* is able to obtain a feasible solution only when we set it to use the cloud performance below 82th percentile of the dynamics distributions. In cases of loose budget, *Deco-W* may return scheduling solutions with cheaper instance types than *Prob* in order to satisfy the budget constraint, and thus result in poorer performance optimization compared to *Prob*. For example, for Q9 workflow under the tight budget, *Deco-W* estimates the expected monetary cost of the solution obtained by *Prob* 17 percent higher than that of *Prob*. To have a better idea on the distributions of the optimized workflow execution time, we further compared the 95th percentile of the optimization results obtained by *Prob* and *Deco-W*. Specifically, *Prob* reduces the 95th percentile of workflow execution time over *Deco-W* by 35 percent for Montage under the tight budget and by 11-56 percent under the loose budget.

Compared to *Deco*, *Prob* can better satisfy the budget constraint under all settings. In contrary to *Deco-W*, *Deco* tends to under-estimate the monetary cost of each found scheduling solution. For example, differences between the cost estimated by *Prob* and *Deco* during solution search are up to 3, 8, 4, 11 and 7 percent for Montage, CyberShake, Q1, Q9 and BigDataSort, respectively, under the tight budget. Underestimation of the cost leads to infeasible solutions due to

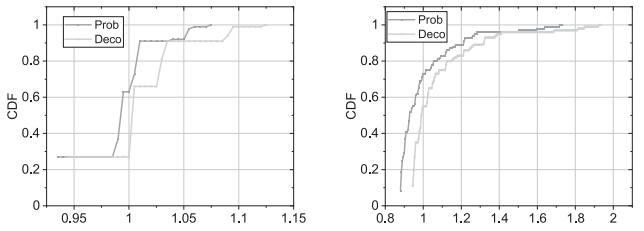


Fig. 9. Cumulative distributions of the normalized monetary cost of Prob and Deco on Amazon EC2 under the loose budget.

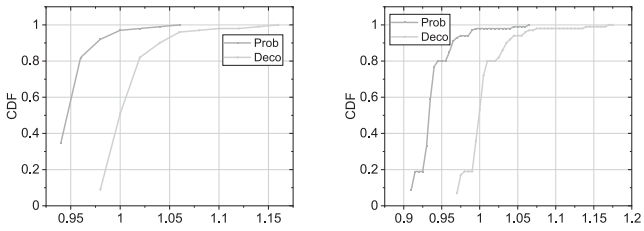


Fig. 11. Cumulative distributions of the normalized monetary cost of CyberShake optimized by Prob and Deco on Windows Azure.

budget violations. As shown in Fig. 8, *Deco* violates both the tight and loose budget in the experiment for all workflows. To further understand the impact of cloud dynamics, we show the cumulative distributions of the monetary cost results optimized by *Deco* and *Prob* for Montage and Cybershake under the loose budget in Fig. 9. For Montage, although the average monetary cost obtained by *Deco* is very close to the loose budget according to Fig. 8b, only around 27 percent of the executions actually satisfy the budget constraint (i.e., normalized cost ≤ 1). *Prob* satisfies the budget constraint with a much higher probability of 63 percent. When looking at the more data-intensive workflow CyberShake, differences between the two algorithms are more obvious. The budget violation of *Deco* leads to more serious consequences with CyberShake, as the actual monetary cost results are up to 94 percent higher than the budget while the cost are only up to 13 percent higher than the budget for Montage.

Results on Windows Azure. Similar to the results on Amazon EC2, *Prob* obtains better performance optimizations than *Deco-W* while satisfying the budget constraint. The difference is that, as the cloud performance of Window Azure is more stable than that of Amazon EC2, *Deco-W* is able to obtain feasible solutions for all workflows under the tight budget. Specifically, *Prob* reduces the expected execution time over *Deco-W* by 6, 23, 12, 43 and 33 percent under the tight budget, and 2, 13, 3, 19 and 23 percent under the loose budget for Montage, CyberShake, Q1, Q9 and BigDataSort, respectively. *Prob* obtains larger execution time reduction over *Deco-W* under the tight budget. This is because the cloud performances of cheap instances are usually more dynamic than expensive ones. When the budget is tight, the scheduling solution contains more cheap instance types and thus the monetary cost estimated by *Deco-W* is more distant from the real value.

As for *Deco*, it again leads to infeasible solutions even with the relatively stable performance of Windows Azure. We show the cumulative distributions of the monetary cost results optimized by *Prob* and *Deco* for CyberShake under the tight and loose budget in Fig. 11. Under both budgets, *Deco* can only satisfy the budget constraint at around 50 percent of the time while *Prob* satisfies the budget constraint with over 96 percent guarantee. Comparing Fig. 11b with Fig. 9b, we can also observe the different dynamicities of the two clouds, where the long tail of the monetary cost distribution goes up to twice the budget on Amazon EC2 while the maximum cost on Windows Azure is only 20 percent higher than the budget.

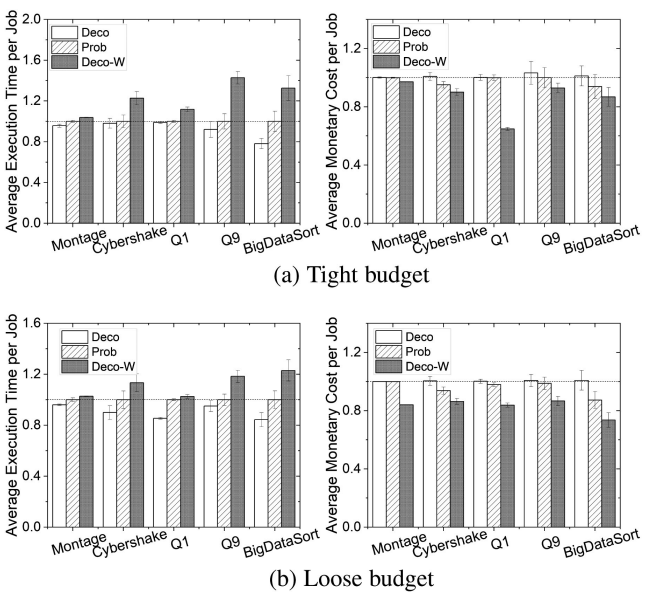


Fig. 10. Normalized results of budget-constrained scheduling under tight and loose budget on Windows Azure.

7.2.2 Fault Tolerance Optimization

Figs. 12 and 17 show the normalized average workflow makespan optimized by the compared algorithms simulating on Amazon EC2 and HPC system, respectively. All results are normalized to those of Young's formula [6].

Results on Amazon EC2. Compared to Young's formula, *Prob* reduces the average makespan of Montage, CyberShake, Q1, Q9 and BigDataSort by 5, 11, 9, 12 and 6 percent when the checkpoint overhead is 2 seconds, by 11, 19, 19, 22 and 13 percent when the checkpoint overhead is 10 seconds and by 20, 26 percent, 27, 30 and 23 percent when the checkpoint overhead is 50 seconds, respectively. We have the following observations.

First, *Prob* obtains better makespan results compared to Young's formula under all settings. To take a closer look at the results, we break down the makespan into workflow execution time, checkpointing time and rollback time. Fig. 13 shows the breakdown of the normalized makespan of a task on the first level of Montage workflow, where *Prob* obtains shorter rollback time and longer checkpointing time compared to Young's formula. This is mainly because *Prob* obtains shorter checkpoint interval and performs checkpointing more frequently than Young's formula. As the time saved on task rollback is higher than the time wasted on checkpointing, *Prob* obtains better makespan optimization result. Table 2 shows

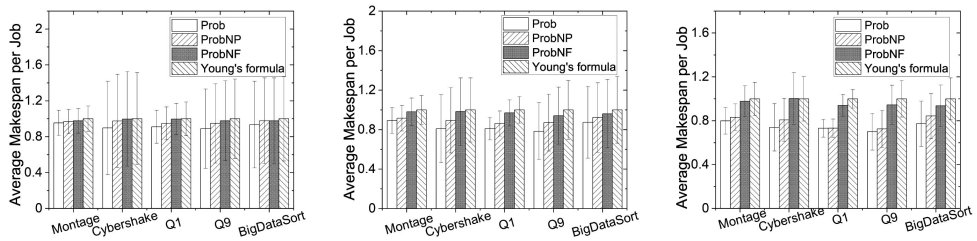


Fig. 12. Normalized optimization results for the fault tolerance problem on Amazon EC2.

the checkpoint interval optimized by *Prob* for the Montage workflow normalized to the results of Young's formula. The Montage workflow structure is reduced to five levels after the clustering and bundling optimizations, and *Prob* obtains shorter checkpoint interval for tasks on all levels of the workflow.

Second, when the checkpoint overhead gets larger, *Prob* obtains higher reduction on average workflow makespan over the compared algorithms. This is mainly because when the checkpoint overhead gets larger, the checkpointing is less frequently and there are more failures between every two checkpoints. Consider the example in Fig. 13, there are on average 0.5 and 2.1 failures happened between every two checkpoints optimized by *Prob* when the checkpoint overhead is 2s and 50s, respectively. As a result, when the checkpoint overhead gets larger, *Prob* saves more time on the rollback. Specifically, the rollback time optimized by *Prob* is 11 and 25 percent shorter than that of Young's formula while the checkpointing time of *Prob* is only 0.6 and 1 percent longer than that of Young's formula, when the checkpoint overhead is set to 2s and 50s, respectively.

Comparing *Prob* with *ProbNP* and *ProbNF*, it obtains the shortest makespan for all workflows and the makespan reduction increases with the increase of checkpoint overhead. This means that considering both performance and failure dynamics is important to the optimization of workflow makespan. We take a closer look at the results of Q1 workflow. Tasks on the first level of the workflow have average execution time of 3239 seconds without failure. When the checkpoint overhead is 50 seconds, checkpoint interval optimized by *Prob* and *ProbNF* for the tasks are 185 and 285 seconds, respectively. As a result, the checkpoint time of each task optimized by *Prob* and *ProbNF* are 1746 and 1439 seconds, respectively. Although *Prob* leads to longer checkpoint time due to shorter checkpoint interval compared to *ProbNF*, it spends less time on failure recovery. Specifically, the average time spent on rolling back to the latest checkpoint optimized by *Prob* and *ProbNF* are 2662 and 4442 seconds, respectively.

ProbNP and *ProbNF* outperform Young's formula under all cases. Specifically, *ProbNP* reduces the average workflow makespan by 3-5, 8-14, and 17-27 percent over Young's formula when the checkpoint overhead is 2s, 10s and 50s, respectively. *ProbNF* reduces the average workflow makespan by 1-3, 2-6, and 0-6 percent over Young's formula when the checkpoint overhead is 2s, 10s and 50s, respectively. These results further show that *ProbNP* outperforms *ProbNF* under all cases. This shows that the failure dynamics play a more important role in the fault tolerance problem than performance dynamics. On one hand, this is because the performance dynamics of m1.xlarge instances are not very significant on Amazon EC2. When we use the performance of m1.small instances for simultion, *ProbNF* sometimes outperforms *ProbNP*. For example, when the checkpoint overhead is 2s, *ProbNF* reduces the average makespan of Cybershake workflow by 19 percent compared to *ProbNP*. On another hand, the failure dynamics affect the entire workflow makespan while performance dynamics only affect part of the makespan. For example, the I/O performance dynamics have impact only on the I/O time of a task. This observation can be further verified using the experimental results on HPC system as discussed below.

We further study the impact of recovery overhead on the effectiveness of *Prob*. Fig. 14 shows the normalized optimization results for the fault tolerance problem using Amazon EC2 trace when the recovery overhead is increased from 1 to 10 seconds and the checkpoint overhead is 2 seconds. Specifically, *Prob* reduces the average workflow makespan by 6, 13, 22, 13 and 8 percent for Montage, CyberShake, Q1, Q9 and BigDataSort, respectively, compared to Young's formula. The observation is that, *Prob* obtains even higher improvement over static method when the recovery overhead becomes larger. This is mainly because there are more failures happening for jobs optimized by Young's formula than that of *Prob*. Thus, the large recovery overhead has a higher impact on the Young's formula than on *Prob*.

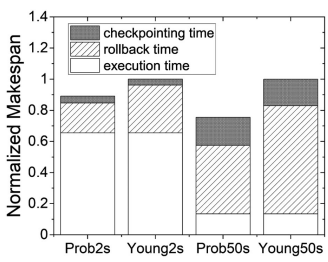


Fig. 13. Makespan breakdown of a Montage task when ckp. overhead is 2s and 50s.

TABLE 2
Checkpoint interval optimized by Prob for Montage tasks. All values are normalized to those of Young's formula.

ckpt overhead	level-1	level-2	level-3	level-4	level-5
2s	0.76	0.82	0.87	0.50	0.77
10s	0.75	0.75	0.82	0.50	0.76
50s	0.65	0.66	0.69	0.45	0.67

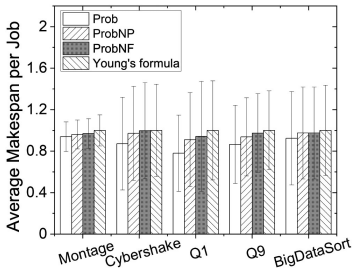


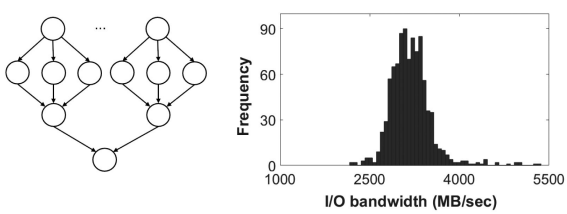
Fig. 14. Normalized optimization results for the fault tolerance problem on Amazon EC2 when the checkpoint overhead is 2s and recovery overhead is 10s.

Results on the HPC System. The evaluated HPC system is different from the cloud in two ways. First, as shown in Fig. 15b, we mainly consider the cross-application I/O performance dynamics of HPC system and its I/O performance is relatively more stable than the cloud I/O performance. Second, the aggregated I/O bandwidth per application on the HPC system can reach up to Gigbytes per second, which is much higher than that on the cloud. As a result, the performance dynamics factor has very little impact on the workflow makespan optimization on the HPC system. For example, as shown in Fig. 17, *Prob* (*ProbNF*) obtains similar results as *ProbNP* (Young's formula) for CyberShake, Q1 and Q9 workflows.

We introduce a new data-intensive workflow named PSMerge [34] to replace Montage and show the impact of performance dynamics on the HPC system. PSMerge is an astronomical workflow which preprocesses and updates data from the PS1 telescope to the database. PSMerge in total reads/writes 34PB of data. Thus, the I/O time of PSMerge occupies a large proportion of its overall makespan and the I/O dynamics of the system lead to unignorable performance dyanamics of the workflow. Fig. 16 shows the execution time distributions of the PSMerge and Cybershake workflows on the HPC system (no failure or checkpoint). Clearly, the execution time variation of PSMerge is much larger than that of Cybershake. As a result, *Prob* obtains the shortest makespan result among all compared algorithms for PSMerge, as shown in Fig. 17. Specifically, it reduces the average makespan by 10, 11 and 23 percent compared to *ProbNP*, *ProbNF* and Young's formula, respectively, for PSMerge. This again demonstrates the importance of considering all system dynamic factors in resource optimization problems for data processing workflows.

7.3 Optimization Efficiency

To study the efficiency of *Prob*, we compare the optimization results and overhead of *Prob* with those of naive



(a) PSMerge (b) I/O performance variation

Fig. 15. (a) PSMerge workflow structure and (b) the I/O performance variation of the HPC system deployed on Grid'5000.

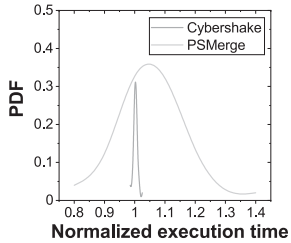


Fig. 16. Normalized execution time distributions of Cybershake and PSMerge workflows on HPC system.

sampling-based MC method (Section 3.3). We set the sampling size of MC to 10 thousands for the budget-constrained scheduling and set the size to 100 thousands for the fault tolerance optimization. Both use cases are evaluated using Amazon EC2 and all parameters are set as default if not otherwise specified. We have the following observations.

First, *Prob* is able to obtain comparable optimization results as MC for both optimization problems. We calculate the average difference of the optimized results between *Prob* and MC. If the difference is close to 0, it means that *Prob* obtains similarly good optimization results as MC. For the two optimization problems, the differences are 0.3 and 2.2 percent, respectively, which demonstrates the effectiveness of *Prob*. We can have the same observation by looking at the distributions of the optimization results obtained by *Prob* and MC. The result distributions of the two algorithms are almost identical, due to the fact that all optimizations in *Prob* can theoretically guarantee the input probability distributions maintained the same.

Second, the optimization overhead of *Prob* is much lower than that of MC. Table 3 summarizes the optimization overhead of *Prob* and the average speedup over MC. Overall, the overhead of *Prob* is within seconds for the budget-constrained scheduling and within minutes for the fault tolerance optimization, which is considerably low with the large sizes of workflows and high complexity of optimization problems. The overhead of *Prob* for the fault tolerance problem is larger than that of the budget-constrained scheduling, as the probabilistic optimization has to deal with two types of dynamics at the same time for the fault tolerance problem. In the following, we take a closer look at our pruning techniques to evaluate their individual effectiveness on reducing the overhead of *Prob*.

With the pre-processing operations, the number of paths enumerated in a workflow can be greatly reduced. For example, the pre-processing operation reduces the number of paths from 219,452 to one for the Montage workflow. The

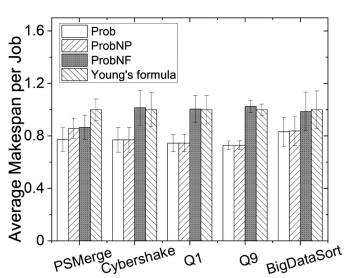


Fig. 17. Normalized optimization results for the fault tolerance problem on HPC system.

TABLE 3
Optimization overhead (sec) of *Prob* and the average speedup over MC for the budget-constrained scheduling problem (P1) and fault tolerance optimization problem (P2).

	Montage	CyberShake	Q1	Q9	BigDataSort	speedup
P1	0.79	0.82	1.32	2.04	0.1	482x
P2	48	223	79	424	63	80x

workflow-specific optimizations can reduce the path distribution evaluation overhead by up to 96 percent. The task bundling and clustering techniques have the benefit of not only reducing the overhead of *Prob* but also increasing the cost-efficiency of cloud instances. For example, with the same solution optimized by *Prob* for the budget-constrained scheduling problem, the expected monetary cost of Montage on Amazon EC2 between with and without workflow-specific optimizations differ by 18 percent. Finally, with our partial solution evaluation pruning, we reduce the solution searching overhead by up to 78 percent for budget-constrained scheduling problem.

7.4 Summary

With the above experiments using two resource provisioning use cases on real clouds, we have the following observations.

First, system dynamics have become the norm rather than the exception in large-scale systems such as clouds and shared clusters. Compared to existing static algorithms which obtain poor or even infeasible solutions, our probabilistic approach can greatly improve the optimization effectiveness for resource provisioning problems of workflows considering system dynamics.

Second, there can be more than one dynamic factors causing system variations. To obtain good results, we have to consider all dynamic factors when optimizing resource provisioning problems. *Prob* provides an easy to implement approach for such complicated problems and our experiments have shown good optimization results when facing both system performance variations and failure variations.

Third, the probabilistic approach involves time-consuming distribution-related computations. Experiments have shown that the pruning techniques of *Prob* can greatly reduce the overhead of probabilistic optimizations.

8 Related Work

We introduce related work of this paper from the following directions, including workflow optimization problems in the cloud, dynamic cloud environment and existing optimization approaches for improving the efficiency of probabilistic methods.

8.1 Workflow Optimizations in the Cloud

Performance and cost optimizations for data-centric workflows in the cloud have been studied by a number of existing work [3], [4], [5], [35], [36], [37]. Mao *et al.* [3] proposed an auto-scaling method which maximize the performance of workflows in the cloud under budget constraints. Abrishami *et al.* [35] proposed methods based on partial critical paths to minimize the workflow execution cost while meeting a deadline. Malawski *et al.* [5] propose to optimize for workflow ensembles under both budget and deadline constraints. Kllapi *et al.* [4] proposed a generic optimizer for both constrained and skyline optimization problems of dataflows in the cloud. Some studies focuse on workflow optimization problems in larger scale, such as multi-clouds and geo-distributed clouds [38], [39], [40], [41]. Diaz-Montes *et al.* [38] proposed a framework to support different scheduling policies for data-intensive workflows in multi-clouds. Pu *et al.* [39] designed a system to optimize the latency of big data analytics in geo-distributed data centers. Kloudas *et al.* [40] considered how to reduce the cross-DC data transfer for wide area big data analytics. The above mentioned studies show that performance is indeed an important optimization goal for various workflow optimization problems. However, most of these studies have not considered the impact of cloud dynamics to the optimization effectiveness.

Reliability is another important concern for large-scale data analytics in the cloud and checkpoint/restart is a widely used mechanism to achieve fault-tolerance for such applications. Sheng Di *et al.* [6] proposed a new formula to calculate the optimal number of checkpoints for the large-scale cloud environment and Subhendu *et al.* [42] introduced live migration on this basis to further reduce the time waste caused by failures. However, these studies considered only part of the cloud dynamic factors (i.e., failure dynamics) and may lead to suboptimal optimization results as demonstrated by our experiments.

8.2 Dynamic Cloud Environment

Many existing work have studied the performance dynamics in the cloud. Previous studies have demonstrated significant variances on the cloud performance [7], [43]. The distribution model (histogram) has also been adopted [7] to measure and analyze the cloud performance variances. Another work [43] proposed to exploit I/O re-routing to reduce process competition on I/O resources and hence reduce I/O performance variations. However, as shown by their work, performance variability can be reduced but not fully mitigated. In this paper, we focus on utilizing the cloud performance distributions to improve the optimization effectiveness for data processing workflows in the cloud.

Recently, there are some studies proposing various methods such as dynamic scheduling and stochastic modeling to address the resource provisioning problem considering cloud dynamics (or uncertainties) [44]. Dynamic scheduling has been used to make adaptive decisions for resource allocation in the cloud considering performance variations [5], [12]. Compared to offline optimization methods, they can make better performance estimations at runtime using either simple history-based estimation [5] or more sophisticated model-based estimation [12]. However, as we have observed, cloud performances can vary abruptly and it is hard to have accurate estimation on their *exact* values. A number of studies have adopted the stochastic programming model for the problem. The optimization variables are represented as random variables and their probability distributions can be estimated from historical data [13], [14]. Those probability distributions are used in stochastic analysis to make resource allocation decisions while providing quantile-based QoS guarantees [14].

This kind of methods require remodeling the resource provisioning problems using stochastic models, which can sometimes be quite complicated.

Probabilistic approaches are recently adopted [45], [46] for resource provisioning at offline time while still mitigating the impact of cloud dynamics to resource provisioning results. For example, UP [47] is a probabilistic method which studies the uncertainty propogation in data processing systems to achieve bounded accuracy when processing uncertain data. However, their method assumes given probability distribution while *Prob* does not have any assumption on the distribution of cloud dynamics. 3Sigma [48] is another distribution-based method proposed for cluster scheduling considering runtime uncertainty. Different from *Prob*, their work does not consider the uncertainty propagation in complicated job structures such as workflows. Dyna is a probabilistic optimization framework [46] which optimizes the monetary cost for scientific workflows in the cloud using spot instances. However, their work have a large overhead due to the costly calculations on probability distributions. The pruning techniques proposed in *Prob* can greatly reduce the optimization overhead and make the utilization of probabilistic distributions for workflow optimizations more practical. This paper extends our previously published work [22] by adding a new resource provisioning problem of workflows, which requires handling more complicated and correlated cloud dynamics factors. With this extension, we show that taming system dynamics on resource optimization for data processing workflows is an important yet challenging problem. *Prob* provides a general, effective and efficient solution to the problem.

8.3 Efficient Probabilistic Methods

The probabilistic distribution model has been adopted in different research domains to improve the optimization results in dynamic environments. In database field, efficiently evaluating probabilistic queries over imprecise data is a hot research topic. Many pruning mechanisms have been proposed to improve the efficiencies of answering probabilistic queries such as top-k [49], aggregations and nearest neighbor [50] on probabilistic databases [51]. In business workflows, the probabilistic model and stochastic models have been adopted to describe the QoS, such as availability and reliability, of Web services [14]. Different from those studies, our probabilistic optimization techniques are specially designed for the performance optimizations of data processing workflows.

9 Conclusion

In this paper, we propose a probabilistic optimization method named *Prob* for the workflow performance optimizations considering system variations. Specifically, we discuss the resource provisioning problem of workflows in the cloud as an example. We model the *cloud dynamics* as time-dependent random variables and take their *probability distributions* as optimization input to the resource provisioning problems of workflows. Thus, the probabilistic optimizations can have a better view of system performance and generate more accurate optimization results. The main challenge of *Prob* is the large computation overhead due to complex workflow structures and the large number of probability calculations. To address this challenge, we propose three pruning techniques to simplify workflow structure and reduce the probability evaluation overhead. We implement our techniques in a runtime library, which allows users to incorporate efficient probabilistic optimization into their existing resource provisioning methods. Experiments on two common resource provisioning problems show that probabilistic solutions can improve the performance by up to 65 percent compared to the state-of-the-art static algorithms, and our pruning techniques can greatly reduce the overhead of probabilistic solutions.

Acknowledgments

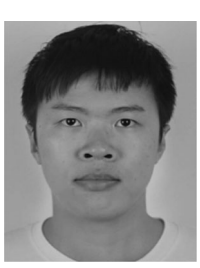
This work was supported in part by the National Natural Science Foundation of China under Grant 61802260, in part by the Shenzhen Science and Technology Foundation under Grant JCYJ20180305125737520, and in part by the Natural Science Foundation of SZU under Grant 000370. The work of Bingsheng He was supported in part by a collaborative grant from Microsoft Research Asia. The work of Shadi Ibrahim work was supported by ANR KerStream Project under Grant ANR-16-CE25-0014-01. The work of Reynold Cheng was supported in part by the Research Grants Council of HK through RGC Projects HKU under Grants 17229116, 106150091, and 17205115, the University of HK under Grants 104004572, 102009508, and 104004129, and in part by the Innovation&Technology Commission of HK through ITF Project MRP/029/18.

References

[1] Jacob *et al.*, "Montage: An astronomical image mosaicking toolkit," *Astrophys. Source Code Library*, record ascl:1010.036, 2010.
[2] A. Thusoo *et al.*, "Data warehousing and analytics infrastructure at facebook," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 1013–1020.
[3] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 67–78.
[4] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 289–300.
[5] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IAAS Clouds," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.
[6] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2013, pp. 64:1–64:12.
[7] J. Schad, J. Dittrich, and J.-A. Quian é-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *Proc. VLDB Endowment*, vol. 3, pp. 460–471, 2010.
[8] B. Acun, P. Miller, and L. V. Kale, "Variation among processors under turbo boost in HPC systems," in *Proc. Int. Conf. Supercomput.*, 2016, pp. 6:1–6:12.
[9] Z. Ou, H. Zhuang, J. K. Nurminen, A. Yl ä-Jääski, and P. Hui, "Exploiting hardware heterogeneity within the same instance type of amazon EC2," in *Proc. 4th USENIX Conf. Hot Top. Cloud Comput.*, 2012.
[10] A. C. Zhou, B. He, X. Cheng, and C. T. Lau, "A declarative optimization engine for resource provisioning of scientific workflows in IAAS clouds," in *Proc. 24th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2015, pp. 223–234.
[11] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with BORG," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.
[12] M. R. Hoseinyfarahabady, H. R. D. Samani, L. M. Leslie, Y. C. Lee, and A. Y. Zomaya, "Handling uncertainty: Pareto-efficient BOT scheduling on hybrid clouds," in *Proc. 42nd Int. Conf. Parallel Process.*, 2013, pp. 419–428.

[13] O. Adam, Y. C. Lee, and A. Y. Zomaya, "Stochastic resource provisioning for containerized multi-tier web services in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2060–2073, Jul. 2017.
[14] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A stochastic programming approach for QOS-aware service composition," in *Proc. 8th IEEE Int. Symp. Cluster Comput. Grid*, 2008, pp. 226–233.
[15] F. J. Cazorla *et al.*, "PROARTIS: Probabilistically analyzable real-time systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, pp. 94:1–94:26, 2013.
[16] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault-tolerant workflow scheduling using spot instances on clouds," in *Proc. Int. Conf. Comput. Sci.*, 2014, pp. 523–533.
[17] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *Proc. 20th Annu. Int. Conf. Supercomput.*, 2006, pp. 324–334.
[18] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia, "Efficient join processing over uncertain data," in *Proc. 15th ACM Int. Conf. Inf. Knowl. Manage.*, 2006, pp. 738–747.
[19] E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Gener. Comput. Syst.*, vol. 46, pp. 17–35, 2015.
[20] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *Proc. 1st Int. Conf. e-Sci. Grid Comput.*, 2005, pp. 140–147.
[21] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 781–786.
[22] A. C. Zhou, Y. Xiao, B. He, S. Ibrahim, and R. Cheng, "Incorporating probabilistic optimizations for resource provisioning of data processing workflows," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.
[23] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 1–13.
[24] E. Deelman *et al.*, "Pegasus: A workflow management system for science automation," *Future Gener. Comput. Syst.*, vol. 46, pp. 17–35, 2015.
[25] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, pp. 49:1–49:12.
[26] G. Singh *et al.*, "Workflow task clustering for best effort systems with Pegasus," in *Proc. 15th ACM Mardi Gras Conf. Lightweight Mash-Ups Lambda Grids Understanding Spectrum Distrib. Comput. Requirements, Appl. Tools, Infrastructures, Interoperability, A Incremental Adoption Key Capabilities*, 2008, pp. 9:1–9:8.
[27] S. Di, D. Kondo, and C.-L. Wang, "Optimization of composite cloud service processing with virtual machines," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1755–1768, Jun. 2015.
[28] G. B. Berriman, J. C. Good, A. C. Laity, and M. Kong, "The montage image mosaic service: Custom image mosaics on-demand," in *Proc. Astronomical Data Anal. Softw. Syst. 17th ASP Conf. Ser.*, 2008, pp. 83–86. [Online]. Available: https://resolver.caltech.edu/CaltechAUTHORS:20100512-105547114
[29] L. Wang *et al.*, "BigDataBench: A big data benchmark suite from internet services," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Architecture*, 2014, pp. 488–499.
[30] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose , and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
[31] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the root causes of cross-application I/O interference in HPC storage systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 750–759.
[32] D. Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in Proc. *Int. Conf. Cloud Comput. Serv. Sci.*, 2013, vol. 367, pp. 3–20.
[33] S. Di, L. Bautista-Gomez, and F. Cappello, "Optimization of a multilevel checkpoint model with uncertain execution scales," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2014, pp. 907–918.
[34] L. Ramakrishnan and D. Gannon, "A survey of distributed workflow characteristics and resource requirements," Sch. Informat. Comput. Eng., Indiana Univ., *Tech. Rep. TR671*, 2008.
[35] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.* vol. 29, pp. 158–169, 2013.
[36] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on grids," in *Proc. 8th IEEE/ACM Int. Conf. Grid Comput.*, 2007, pp. 10–17.
[37] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proc. 3rd Int. Symp. Parallel Architectures, Algorithms Programm.*, 2010, pp. 89–96.
[38] J. Diaz-Montes, M. Diaz-Granados, M. Zou, S. Tao, and M. Parashar, "Supporting data-intensive workflows in software-defined federated multi-clouds," *Trans. Cloud Comput.*, vol. 6, no. 1, pp. 250–263, Jan.–Mar. 2018.
[39] Q. Pu *et al.*, "Low latency Geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.
[40] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," *Proc. VLDB Endowment*, vol. 9, pp. 72–83, 2015.
[41] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 323–336.
[42] S. Behera, L. Wan, F. Mueller, M. Wolf, and S. Klasky, "Orchestrating fault prediction with live migration and checkpointing," in *Proc. 29th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2020, pp. 167–171.
[43] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: Exploiting performance heterogeneity in public clouds," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 20:1–20:14.
[44] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E.-G. Talbi, "Towards understanding uncertainty in cloud computing resource provisioning," in *Proc. Int. Conf. Comput. Sci.*, 2015, pp. 1772–1781.
[45] M. L. D. Vedova, D. Tessera, and M. C. Calzarossa, "Probabilistic provisioning and scheduling in uncertain cloud environments," in *Proc. Int. Symp. Comput. Commun.*, 2016, pp. 797–803.
[46] A. C. Zhou, B. He, and C. Liu, "Monetary cost optimizations for hosting workflow-as-a-service in IAAS clouds," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 34–48, Jan.–Mar. 2016.
[47] I. Manousakis, Í. Goiri, R. Bianchini, S. Rigo, and T. D. Nguyen, "Uncertainty propagation in data processing systems," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 95–106.
[48] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger, "3SIGMA: Distribution-based cluster scheduling for runtime uncertainty," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–17.
[49] L. Mo, R. Cheng, X. Li, D. W. Cheung, and X. S. Yang, "Cleaning uncertain data for top-k queries," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 134–145.
[50] J. Niedermayer *et al.*, "Probabilistic nearest neighbor queries on uncertain moving object trajectories," *Proc. VLDB Endowment*, vol. 7, pp. 205–216, 2013.
[51] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *Int. J. Very Large Data Bases*, vol. 16, no. 4, pp. 523–544, 2007.

Amelie Chi Zhou received the PhD degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2016. She is currently an assistant professor with Shenzhen University, China. Before joining Shenzhen University, she was a postdoc fellow with Inria-Bretagne Research Center, France. Her research interests include cloud computing, high performance computing, big data processing, and resource management.



Welin Xue received the bachelor's degree in 2019 from Shenzhen University, where he is currently working toward the master's degree with the School of Computer Science and Software Engineering. His research interest includes parallel and distributed systems, big data processing, and resource management.

Yao Xiao received the bachelor's degree from the School of Automation, Huazhong University of Science and Technology, in 2012. He is currently working toward the PhD degree with the School of Computer Science and Software Engineering, Shenzhen University, China. His research interests include cloud computing, big data processing, and resource management.

Bingsheng He received the PhD degree from the Hong Kong University of Science & Technology in 2008. He is currently an associate professor with the Department of Computer Science, National University of Singapore (NUS). Before joining NUS in May 2016, he was a researcher with System Research Group, Microsoft Research Asia from 2008 to 2010, and a faculty member with Nanyang Technological University, Singapore. His research interests include big data management, parallel and distributed systems, and cloud computing.

Shadi Ibrahim (Member, IEEE) received his PhD degree in Computer Science from Huazhong University of Science and Technology, China in 2011. He is a permanent Inria Research Scientist. From Nov. 2011 to Sept. 2013, he was a postdoc researcher with KerData team working on scalable Big Data managements on clouds. His research interests are in cloud and Fog/Edge computing, scalable Big data management, Data-Intensive computing, HPC, virtualization technology, and file and storage systems. He has published research papers in recognized Big Data and cloud computing research conferences and journals including TPDS, FGCS, PPNA, SC, IPDPS, ICDCS Mascots, CCGrid, ICPP, SCC, Cluster, and Cloudcom. He received the IEEE TCSC Award for Excellence in Scalable Computing (Middle Career Researcher) in 2020. He is an associate editor for the IEEE Internet Computing magazine. He is a Senior member of the ACM.

Reynold Cheng (Member, IEEE) received the PhD degree from the Department of Computer Science, Purdue University, in 2005. He is currently an associate professor with the Department of Computer Science, University of Hong Kong (HKU). He is an associate editor for the *IEEE Transactions on Knowledge and Data*, and was on the editor-in-chief selection committee of the *IEEE Transactions on Knowledge and Data*. He was a PC member and a reviewer for international conferences, including SIGMOD, VLDB, ICDE, KDD and journals, including *IEEE Transactions on Knowledge and Data*, *ACM Transactions on Database Systems*, *International Journal on Very Large Data Bases,* and *Information Systems*. He was the recipient of Outstanding Young Researcher Award in 2011–2012 from HKU.