Incorporating Probabilistic Optimizations for Resource Provisioning of Data Processing Workflows

Amelie Chi Zhou, Bingsheng He, Shadi Ibrahim and Reynold Cheng

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 paper, we provide a general solution for workflow performance optimizations considering system variations. Specifically, we model the cloud 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 51% – 70% compared with the state-of-the-art static solutions, and our pruning techniques can greatly reduce the overhead of probabilistic solutions.

Index Terms—Cloud Dynamics, Resource Management, 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% 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 clusters (Section 7.4).

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 the variations of I/O and network performances, can result in major performance variation [9], [7] to large-scale data processing workflows. We have executed four workflows, including two scientific workflows (Montage [1] and CyberShake [10]) and two data analytics queries (Q1 and Q9 of the TPC-H benchmark), on Amazon EC2 for 100 times each (detailed setup in Section 7). The execution times of these workflows illustrate large variances (respectively 24%, 43%, 15% and 21%). 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 (theoretically explained in Section 3).

Why probabilistic method? Existing studies propose various methods such as dynamic scheduling [5], [11] and stochastic

Amelie Chi Zhou is with the College of Computer Science and Software Engineering, Shenzhen University, China.
Bingsheng He is with School of Computing, National University of Singapore.
Shadi Ibrahim is with Inria Rennes - Bretagne Atlantique, France.
Reynold Cheng is with Department of Computer Science, the University of Hong Kong, China.

modeling [12], [13] to address resource provisioning problems considering cloud dynamics. However, they are either relying 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 provisioning of 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 [14] required by many problems with system randomness, such as designing fault-tolerant scheduling techniques for workflows in case of random system failures [15]. Second, it enables the derivation of probabilistic bounds [16] to guarantee the worst-case performance of applications, while the existing static methods only guarantee the average performance.

Why this paper? 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 [17] and efficient probabilistic analysis in hardware design [14]. 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 provisioning 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 pre-processing pruning to reduce the overhead of this important calculation and hence 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 [17] 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 experimented with real-world workflows on Amazon EC2 and with simulations to show the effectiveness of probabilistic optimizations and our pruning techniques. Our experiments demonstrate 51%–70% improvement of probabilistic solutions compared to state-of-the-art static solutions. The pruning techniques of *Prob* bring significant reduction to the overhead of probabilistic solutions (e.g., up to 450x speedup to the Monte Carlo (MC) method). As a result, it takes less than 10 seconds to complete the optimizations for a Montage workflow with more than 10,000 tasks. As recent studies [18] have demonstrated the system variations in clusters, we extend our study with simulations using I/O bandwidth traces collected from a large-scale shared cluster [19]. Preliminary results show that *Prob* can be efficiently applied to improve the resource provisioning effectiveness in clusters.

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. Sections 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

In this section, we present the background and terminology on data processing workflows and cloud dynamics.

2.1 Data Processing Workflows

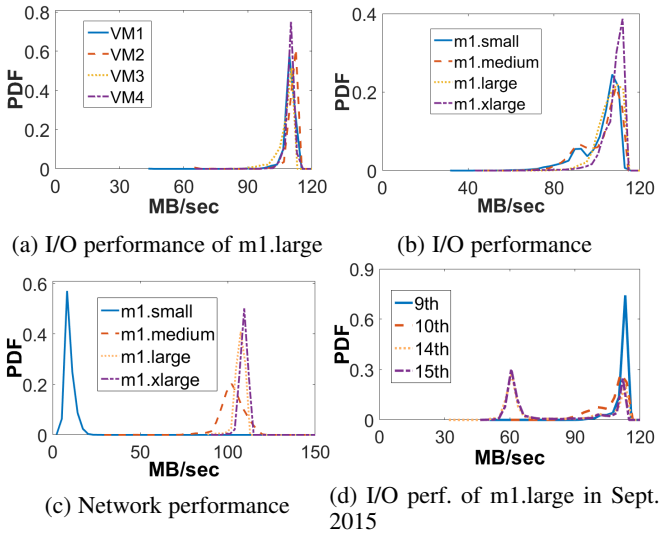
Data processing workflows are usually adopted to represent the execution of complex data-intensive applications. We first introduce several important terms mentioned in this paper.

Workflow: A data processing workflow (a job) can be described as a directed acyclic graph (DAG) [20]. 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 [21] widely used for data-intensive task execution time estimation, which calculates 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 staging input data of the workflow while the exit vertex saves output results.

Resource provisioning: 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 [5], [4], [23].

2.2 Cloud Dynamics Terminology

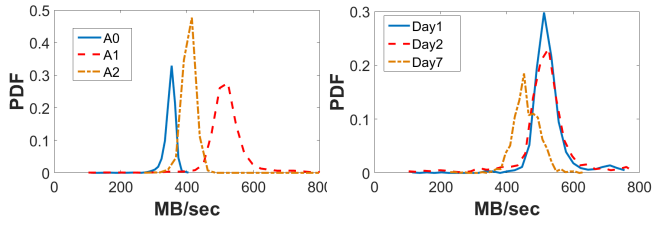
In this paper, we study two kinds of variations, namely the performance variation in I/O and network and system failures variation, which are common in the cloud and decisive to the execution of data processing workflows. In this section, we focus on the I/O and network performance variations (the CPU performance is rather stable in the cloud [7]), and extend the analysis to other



(a) I/O performance of m1.large (b) I/O performance

(c) Network performance (d) I/O perf. of m1.large in Sept. 2015

Fig. 1: The sequential I/O and network performances of different instance types on Amazon EC2.



(a) I/O performance (b) I/O performance of A1

Fig. 2: The sequential I/O performance of different instance types on Windows Azure measured in July 2017.

impacting factors in later sections. Formally, we define the I/O (network) performance 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 solving of resource provisioning problems for workflows in the cloud. Consider a simple 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. Defining the I/O time of the task with a random variable (r.v.) T, we can use the probability distribution (PDF) of I/O bandwidth to calculate the distribution of T as $f_T(t) = f_X(\frac{d}{t})$. 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., $\frac{d}{\sum_x x \cdot f_X(x)}$). With case studies, we will show the effectiveness of the probabilistic model on improving resource provisioning results.

2.3 Features of Cloud Performance Dynamics

We study the spatial and temporal features of cloud performance dynamics in I/O and network. We demonstrate that the probability distributions of performance dynamics are predictable in a short period of time. We conduct the experiments on Amazon EC2 cloud. More details about the measurements can be found in Section 5.

Spatial features. We observe that the performance of different instances of the same type can be modeled with similar/the same probability distributions. Figure 1a shows the distributions of the sequential I/O performances of four m1.large instances of Amazon EC2 running on the same day. The I/O performance of the four instances have very similar distributions. We have performed the measurements on over 100 m1.large instances and found consistent results. Similar observations can be found on other instance types.

Another observation is that, the probability distributions of performances of different types of instances usually have similar patterns with different parameters. Figure 1b and 1c show the sequential I/O and network performance distributions of four instance types of Amazon EC2, respectively. For each type, data are collected from multiple instances over 1 day. The distributions have similar patterns with different mean and variances. For example, the mean network bandwidth is higher on more expensive instance types such as m1.large and m1.xlarge and the performance variance is more severe on cheap instance types such as m1.small and m1.medium.

Temporal features. We find that the probability distributions of the cloud performance are stable within a short time period. Figure 1d shows the sequential I/O performance distributions of m1.large instances measured in four individual days, namely the 9th, 10th, 14th and 15th September, 2015. It can be observed that, the performance distributions in a short period of time are closer to each other in terms of mean and variance. Quantitatively, we adopt the *Bhattacharyya distance* [24] to measure the similarity of two probability distributions. Bhattacharyya distance is commonly used in statistics to measure the similarity of two discrete or continuous probability distributions. The distance metric ranges between 0 and 1.0 and a smaller distance indicates that the two measured distributions are more similar to each other. The distance between the I/O performance distribution of the 9th September and the distribution of the 10th is only 0.03, while the distance between the distribution of the 9th and the 14th is 0.1. This means that the I/O 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). The same observation has also been found on the network performances.

To demonstrate the generality of our observations, we also perform similar experiments on Windows Azure cloud using three general purpose instance types, namely A0, A1 and A2. Figure 2a demonstrates that the spatial feature can also be observed on Windows Azure cloud, where distributions of the sequential I/O performance of different instance types follow similar patterns with different parameters. The A1 instance type has higher performance variation than the other two types. We suspect it is because A1 is the recommended type of Azure and thus has more users. Figure 2b shows that the temporal feature also holds on Azure, where the I/O performance distribution of A1 is more similar to the distribution measured from an adjacent day than the distribution measured after a longer period.

The spatial and temporal predictability of cloud performance dynamics are important to verify that, it is feasible to represent cloud dynamics using their probability distributions and adopt probabilistic methods to optimize the resource provisioning problems.

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 [25] to this problem 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 minimize the workflow execution time while satisfying a certain 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.

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

subject to:

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

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

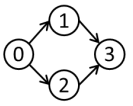
3.2 A Static Solution

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 a traditional static approach for the budget-constrained scheduling problem, and discuss how to incorporate cloud dynamics in the next subsection.

We adopt an existing method [25] which formulates resource provisioning problems as search problems, and adopt generic search or more efficient $A^\star$ 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.

Tid1	Tid2	Time	p
0	1	100	0.6
0	1	130	0.3
0	1	170	0.1
0	2	100	0.6
0	2	130	0.3
0	2	170	0.1
1	3	130	0.7
1	3	140	0.2
1	3	150	0.1
2	3	120	0.5
2	3	145	0.3
2	3	160	0.2



Path 1: 0 → 1 → 3
Path 2: 0 → 2 → 3

- Static method: $T_{path} = \sum_{i \in path} E(T_i)$
  $E[MAX(T_1, T_2)] = 251.5$
- Probabilistic method: $\overrightarrow{T_{Path}} = \sum_{i \in path} \overrightarrow{T_i}$
  $E\left[MAX(\overrightarrow{T_1}, \overrightarrow{T_2})\right] = 257.4$

Fig. 3: An example of probabilistic optimizations.

- State traversal: The search process is like a BFS search. A state s on the l-th level of the search tree transits to another state by incrementing the l-th dimension of s. For example, the initial state $(0, 0, \ldots, 0)$ can transit to the following states: $(1, 0, \ldots, 0)$, $(2, 0, \ldots, 0)$, $\ldots$, and $(K-1, 0, \ldots, 0)$.
- State evaluation: Each found solution is evaluated using Equation 1 for the optimization goal and Equation 2 for the optimization constraint. The static optimizations take the expected I/O and network performance as input to estimate the task execution time. That means, T_{ij} is only a scalar value and the computation of Equation 1 and 2 is light-weight. After evaluating a solution, we compare it with the best found solution and keep the one which has a better optimization goal and satisfies the budget constraint.
- 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).
- Termination: The search process terminates if the entire solution space has been traversed or a pre-defined 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

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

Figure 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.

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 251.5. 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 257.4. The probability of path 2 being the critical path is only 0.48. 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

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

Similarly, we have $E[Z] \geq E[X]$. Thus Lemma1 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.

Probabilistic optimization is costly. A straight-forward 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. 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.

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

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 mainly due to complex workflow structures and costly calculations and comparisons of distributions. To improve the efficiency of *Prob*, we introduce three simple yet effective pruning stages targeting at those affecting factors. 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.

4.1 Pre-Processing

Calculating the execution time of a workflow is a common operation in many resource provisioning problems of workflows. Due to the 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 Figure 4, we use the structure of a real-world scientific workflow named Montage [26] 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 Figure 4, we initially have 48 paths in S. We introduce the following two techniques to reduce the size of S.

Critical Path Pruning. This technique eliminates the paths that are impossible to be the critical path from S. For example, in Figure 4, as the execution time of path P_2 has to be shorter

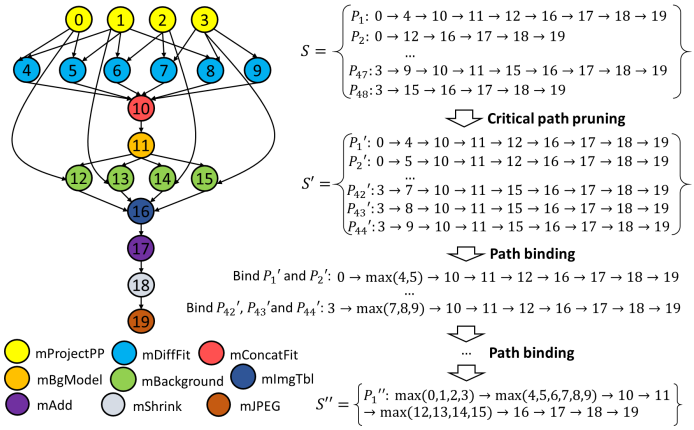


Fig. 4: An example of pre-processing optimizations using Montage workflow.

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 $O(|S|^2 \times V)$.

Path Binding. Given S' in Figure 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 the workflow execution time distribution is reduced to 8 distribution additions and 11 distribution 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% 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 i-th nodes ($i = 0, \ldots, L-1$) are the same, they are adopted as the i-th node of the binded path. Otherwise, we bind the i-th nodes of the paths as one node, the execution time of which is the maximum of the i-th 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 time and the results can be reused for different resource provisioning problems of the same workflow structure.

4.2 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. To apply *ADD* and *MAX* on n ($n > 2$) distributions, we perform the operators for $n-1$ times. For example, the execution time distribution of path P_2 in Figure 4 can be calculated as

$$PDF_{P_2} = ADD(ADD(ADD(ADD(ADD(PDF_0, PDF_{12}), PDF_{16}), PDF_{17}), PDF_{18}), PDF_{19}) \quad (8)$$

where PDF_i denotes the execution time distribution of task i on the instance type assigned by the current evaluated solution.

The calculation of *ADD* and *MAX* operations are introduced in Equations 4 and 5, respectively. As discussed in Section 3, a straight-forward implementation for *ADD* and *MAX* is to use the sampling-based MC method. However, this implementation can have a large overhead 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* [27] and *task clustering* [28]. 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.

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 Figure 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.

With the naive MC implementation, the overhead of the *ADD* operation is $O(3M)$, which includes the time for generating execution time distributions of the two tasks and the time for calculating the sum distribution. After applying task bundling, the overhead of *ADD* is reduced to $O(M)$.

Optimizing *MAX* with task clustering. In many data processing workflows, the data volume to be processed is huge. The execution is usually in a data parallel manner. Data are divided into a large number of partitions with equal sizes and are processed in parallel with many tasks. We denote those parallel tasks as *equivalent tasks*. 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 Figure 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 MAX-operation 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 $O(1)$.

4.3 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.

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 $X > Y$. Then we have $g(X,Z) > g(Y,Z)$ for any r.v. Z independent from X and Y, where $g(\cdot)$ is either ADD or MAX.*

Proof. When $g(\cdot)$ is *ADD*, as $P(X > Y) > 0.5$, we have $P(X + Z > Y + Z) > 0.5$. When $g(\cdot)$ is *MAX*, we have

$$\begin{aligned} & P(max\{X,Z\} > max\{Y,Z\}) \\ =& P(max\{X,Z\} > Y) + P(max\{X,Z\} > Z) + \\ & P(max\{X,Z\} < min\{Y,Z\}) - 1 \\ >& 0.5 + 1 + 0 - 1 = 0.5. \end{aligned}$$

Thus Lemm 2 is proven. □

Consider a concrete example using the workflow in Figure 4. Consider comparing two solutions s and s', and s differs from s' on the configurations of task 10 and 11. With the partial evaluation, we consider s' results in shorter workflow execution time than s if

$$P(\mathit{ADD}(\mathit{PDF}^s_{10}, \mathit{PDF}^s_{11}) > \mathit{ADD}(\mathit{PDF}^{s'}_{10}, \mathit{PDF}^{s'}_{11})) > 0.5 \quad (9)$$

where PDF^s_i and $\mathit{PDF}^{s'}_i$ are the execution time distributions of task i on instance type indicated in s and s', respectively. In this way, we reduce the overhead of comparing the two solutions from 38 probabilistic operations (either *ADD* or *MAX*) and one distribution comparison to two *ADD* and one distribution comparison.

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.

Distribution comparison pruning. After solution evaluations (either partial or full evaluations), we repeatedly use Equation 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 [17] 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 Equations 10 and 11, 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 Equation 6.

$$\text{If } X.l \leq Y.r \leq X.r, P(X > Y) \geq 1 - F_X(Y.r) \quad (10)$$

$$\text{If } X.l \leq Y.l \leq X.r, P(X > Y) \leq 1 - F_X(Y.l) \quad (11)$$

5 Implementation Details

Figure 5 shows how Prob can be used by existing data processing systems (e.g., Hadoop). 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. Second, *Prob* offers a runtime library which exposes the probabilistic operations (e.g., *ADD* and *MAX*)

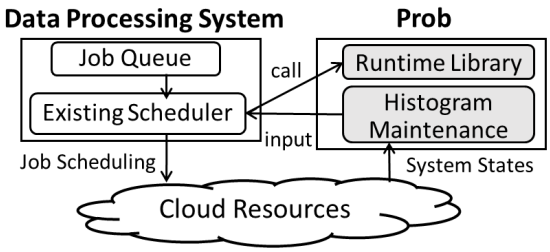


Fig. 5: System implementation.

and pruning 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.

5.1 Maintenance of Probability Distributions

As optimization inputs, 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 and failures. *Prob* collects those states from the cloud and maintains the data in an online and lightweight manner.

System states are calibrated and updated periodically. Instance-related states are measured from the cloud platform by opportunistically taking the idle time of running instances. As the CPU performance is rather stable in the cloud, we mainly focus on calibrating I/O and network performances. We measure the sequential I/O reads performance (reads and writes have similar performances) of local disks with hdparm and the network bandwidth between two instances with Iperf [29]. Job-related states can be obtained either by a piggyback measurement of the execution time during workflow executions, or a regular "heartbeat" measurement if there is no chance for piggyback measurements. The benefit is that, we can obtain some history data as a by-product of workflow executions without additional cost.

A naive sampling method for the MC implementation can be extremely inefficient to achieve a high accuracy. To make the distribution calculations more efficient, we maintain system state calibrations in the form of histograms and adopt a nonuniform sampling method to reduce the sampling size while preserving the desired computational accuracy. 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. We descretize the distributions of system state values with nonuniform intervals, where more samples are taken at variable values with higher probability. Assume that the sample size for a r.v. X is N, then for $\forall x_i$ ($i \in 0 \ldots, N-2$), we have $P(x_{i+1}) - P(x_i) = \frac{1}{N}$.

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. To this method, one important parameter is the window size. Different kinds of cloud dynamics may have different suitable window sizes. For each dynamics factor, we calculate the Bhattacharyya distance [24] between the real and predicted distributions for different window sizes using historical data. We select the window size which generates the shortest Bhattacharyya distance for the window-based distribution prediction. According to our sensitivity studies, we set this parameter to one day by default for cloud I/O and network performance distributions.

5.2 Runtime Library

We develop a runtime library to expose the pruning techniques of *Prob* for data processing systems to easily incorporate probabilistic optimizations into their existing scheduling methods. There are three types of APIs in the library, as summarized in Table 1.

Pre-processing APIs. We provide the *PreProcessing()* API, which implements the path enumeration, critical path pruning and path binding optimizations. It takes a workflow, a source node and a target node as input and returns a set of paths from the source node to the target node as output. The output paths are candidates of the critical path. This API can be called before the solution search process.

Workflow-related APIs. We provide four APIs related to the workflow-specific optimizations. The *TaskBundling()* and *TaskClustering()* APIs implement task bundling and task clustering optimizations, respectively. Each API takes a path as input, and return the optimized path with bundled/clustered nodes as output. The *Max()* and *Add()* APIs take two probability distributions as input and return their maximum and sum distributions as output, respectively. These APIs are useful for implementing the solution evaluation logic with performance objectives, e.g., calculating the workflow execution time distribution.

Distribution-related APIs. We provide the *PartialEval()* API, which takes two found solutions as input and returns their partial evaluation results as output. The outputs can be used to compare the relative optimality of the two solutions. The comparison can be performed with the *Compare()* API, which takes two probability distributions as input and return their comparison result as output. Users can utilize the APIs to implement the solution searching logic, e.g., comparing the evaluations of found solutions to find a good one.

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 [25] 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 preprocessing, 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.

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

Algorithm 1 Static (left) and probabilistic (right) search algorithms for budget-constrained scheduling problem on workflow W.

```
1: Preserve the best solution CurBest;
2: P = PreProcessing(W, entry, exit);
3: ...                                   ▷ State traversal
4: Given a solution s, assign instance configurations in s to each task in W;
5: Initialize workflow execution time as time = 0;
6: for each path in P do
7:     TaskBundling(TaskClustering(path));
8:     Initialize the length of path as t = 0;
9:     for each task tk on path do
10:
11:
12:        t = t + t_tk;
13:    if t ≥ time then
14:        time = t;
15:
16: ...          ▷ Evaluate the expected cost
17: if cost ≤ B then
18:    if time < CurBest.time then
19:        CurBest = s;
20: ...          ▷ Go back to state traversal
```

```
Preserve the best solution CurBest;
P = PreProcessing(W, entry, exit);
...                                      ▷ State traversal
Given a solution s, assign instance configurations in s to each task in W;
Define workflow execution time distribution Time;
for each path in P do
    TaskBundling(TaskClustering(path));
    Define the length distribution of path as T;
    for each task tk on path do
        if tk is the first task in path then
            T = T_tk;
        else T = Add(T, T_tk);
    if path is the first path in P then
        Time = T;
    else Time = Max(Time, T)
...                          ▷ Evaluate the expected cost
if cost ≤ B then
    if Compare(CurBest.Time, Time) ≥ 0.5 then
        CurBest = s;
...                          ▷ Go back to state traversal
```

6 More Cloud Dynamics

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], [30], 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.

$$\min E[\sum\nolimits_{i \in CP} T_i \times (1 + \frac{C_k}{V_i} + \frac{C_r \cdot V_i}{2F_i})] \tag{12}$$

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 13, 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., $Y_i = T_i/F_i$).

$$x_i^* = \sqrt{\frac{E(T_i) \cdot E(Y_i)}{2C_k}} \tag{13}$$

The probabilistic method. With *Prob*, both T_i and Y_i in Equation 13 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 sub-optimal fault tolerance solutions.

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 machine with 24GB DRAM and a 6-core Intel Xeon CPU. Workflows are executed on Amazon EC2 or a cloud simulator.

7.1 Experimental Setup

Data processing workflows. We consider data processing workflows from scientific applications and data analytics queries implemented based on Hadoop, denoted as scientific and MR workflows, respectively. Figure 6 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 Montage source code. The input data are the 2MASS J-band images covering 8-degree by 8-degree areas retrieved from the Montage archive [31]. Since CyberShake is not open-sourced, we construct synthetic workflows with 1000 tasks each using workflow generator [26].

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

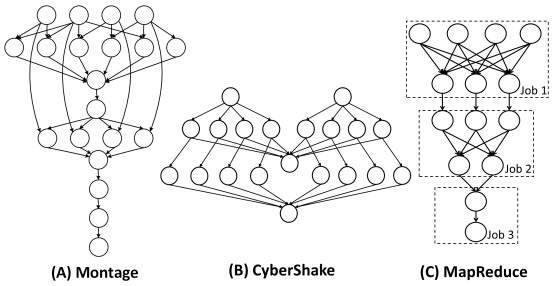
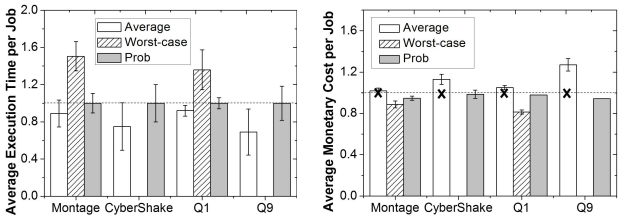
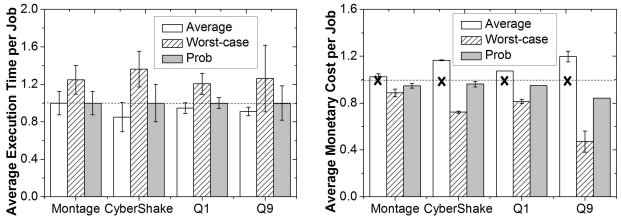


Fig. 6: Structures of Montage, CyberShake and MapReduce workflows.



(a) Tight budget



(b) Loose budget

Fig. 7: Normalized results of budget-constrained scheduling under tight and loose budget. Cross marks stand for budget violations.

is usually composed of several MR jobs. Q1 is composed of two MR jobs and Q9 is composed of seven MR jobs.

Implementation. We conduct our experiments on both real clouds and simulator to study the workflow optimization problems in a controlled and in-depth manner. On Amazon EC2, we utilize an existing workflow management system named Pegasus [20] to execute scientific workflows, and deploy Hadoop and Hive to run the TPC-H queries. We adopt an existing cloud simulator [25] designed based on CloudSim [32] to simulate the dynamic cloud environment. We run the budget-constraint scheduling evaluations on Amazon EC2 and the fault tolerance evaluations with trace-driven simulations. As CyberShake is not open-sourced, we use simulations for all CyberShake evaluations.

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 use four types of Amazon EC2 instances with different prices and computational capabilities, including m1.small, m1.medium, m1.large and m1.xlarge. We compare *Prob* with two static algorithms named *Average* and *Worst-case*, which optimize the performance of workflows using the expected and the 99-th percentile of cloud dynamics

TABLE 2: Hours/costs of different instance types used during one CyberShake execution under loose budget.

	m1.small	m1.medium	m1.large	m1.xlarge
Average	0	0	161h/$28.4	16h/$5.7
Worst-case	507h/$22.3	0	0	0
Prob	0	0	169h/$29.6	0

distributions, respectively. The static algorithms adopt the search method [25] 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 instances and on m1.xlarge instances, respectively. Given a budget, we run the compared algorithms for 100 times and compare the obtained average monetary cost and execution time. The results in Section 1 are obtained from this setting with a loose budget.

Fault tolerance. We evaluate the average makespan of executing the four workflows on a virtual cluster of 50 m1.xlarge instances using simulator. 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, and vary the checkpoint overhead from 2, 10 to 50 seconds to study the behavior of *Prob* under different system performances. We study the failure events of tasks with different lengths in Google Trace [6] and correlate the MTBF of tasks with their lengths using polynomial regression. 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. We assume that no failure happens during the checkpointing.

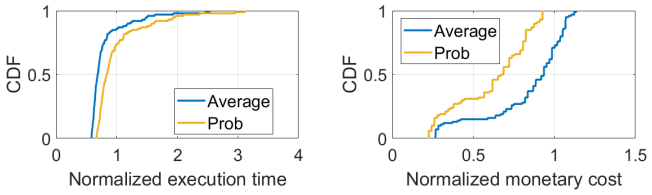
We compare *Prob* with the static Young's formula [6] to study the impact of cloud dynamics in performance and system failures. To further breakdown the impact of the two dynamics, we compare *Prob* with ProbNP and ProbNF, which are *Prob* without considering cloud dynamics in performance and failures,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

7.2 Optimization Effectiveness

For both optimization problems, compared to state-of-the-art static methods which ignore cloud dynamics, *Prob* is able to improve the optimization effectiveness while satisfying constraints.

Budget-constrained Scheduling. Figure 7a and 7b show the average execution time and monetary cost optimization results of the compared algorithms when the budget is set to tight and loose, respectively. The execution time results are normalized to that 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. We have the following observations.

Compared to Worst-case, *Prob* obtains better performance results while satisfying the budget constraint under all settings. *Prob* reduces the expected execution time of Montage, CyberShake, Q1 and Q9 by 51%, N/A, 36% and N/A under the tight budget, and by 24%, 37%, 19% and 26% under the loose budget. Worst-case cannot find a feasible solution for CyberShake and Q9 under the tight budget because these two workflows are more data- and network-intensive. As Worst-case tends to over-estimate the average execution time of workflows, it always chooses cheaper instance types than *Prob* to guarantee the budget, as shown in



(a) Execution time (b) Monetary cost

Fig. 8: Cumulative distributions of normalized execution time and monetary cost results of Cybershake under loose budget.

TABLE 3: Differences between *Prob* and Average on the expected execution time calculated from entry task to each level of Q9.

level	0	1	2	3	4	5	6	7	8-13
diff. (%)	4.8	7.4	9.4	10.5	13.7	15.9	18.3	21.5	27.6

Table 2. The improvement of *Prob* over Worst-case is larger when the budget is tight. This is because cheap instances are chosen when the budget is tight and performance variations of workflows are more severe on cheap instances.

Compared to Average, *Prob* is able to guarantee the budget constraint under all settings. The Average method tends to underestimate the expected execution time of workflows. As a result, the monetary cost estimated by Average for each found solution is lower than the real cost. Differences between the cost estimated by *Prob* and Average during solution search for Montage, CyberShake, Q1 and Q9 are up to 2%, 14%, 12% and 20%, respectively, under the tight budget. Under-estimation of the cost leads to infeasible solutions due to budget violations, as illustrated by the cross marks in Figure 7a and 7b. To further understand the impact of cloud dynamics to Average, we show the cumulative distributions of the monetary cost and execution time results of Cybershake under the loose budget. As shown in Figure 8b, among the 100 times of executions, only around 70% of results obtained by Average satisfy the budget constraint (i.e., normalized cost ≤ 1) while *Prob* can 100% guarantee the budget constraint.

We observe that Average performs especially poor on Q9, due to several reasons. First, due to the all-to-all structure of MR workflows, there are a large number of *MAX* operations when calculating the workflow execution time, which causes large errors with the static methods. Second, Q9 is composed of seven MR jobs, which is more complicated than Q1. The errors obtained on each level of Q9 accumulate and thus make Average seriously violating the budget constraint. Table 3 shows the differences between *Prob* and Average on the expected execution time calculated from the entry to each level of the Q9 workflow structure.

The above observations demonstrate that, considering cloud performance dynamics gives more accurate estimations of workflow performance in the cloud, so that budget constraints can be satisfied and better optimization results for workflows can be generated.

Fault tolerance. Figure 10 shows the normalized average workflow execution time results obtained by the compared algorithms. All results are normalized to those of Young's formula [6]. *Prob* obtains the best result under all settings.

Compared to Young's formula, *Prob* reduces the average execution time of Montage, CyberShake, Q1 and Q9 by 11%, 11%, 47% and 33% when the checkpoint overhead is 2 seconds, by 26%, 19%, 63%, 47% when the checkpoint overhead is 10 seconds and by 74%, 24%, 72%, 74% when the checkpoint overhead is 50 seconds, respectively. From the distribution's point of view, Figure 11 shows an example of the optimization results of Montage when checkpoint overhead is 10 seconds. The figure demonstrates that *Prob* can guarantee better optimization result than its static counterpart at most of the time (over 94% in the example). The improvement obtained by *Prob* for CyberShake is smaller than the other workflows. This is mainly because the execution time of CyberShake is dominated by one long-running task (the input data loading task).

TABLE 4: Checkpoint interval optimized by Prob for Q1 tasks. All values are normalized to those of Young's formula.

ckpt overhead	level-1	level-2	level-3	level-4
2s	0.36	0.36	0.36	0.36
10s	0.42	0.35	0.35	0.35
50s	0.49	0.27	0.33	0.27

TABLE 5: Optimization overhead (sec) of *Prob* and the average speedup over MC for the two workflow optimization problems.

	Montage	CyberShake	Q1	Q9	speedup
Bgt-constrained	2.5	1.7	6.1	7.0	450x
Fault tolerance	1.6	1.2	1.2	3.1	50x

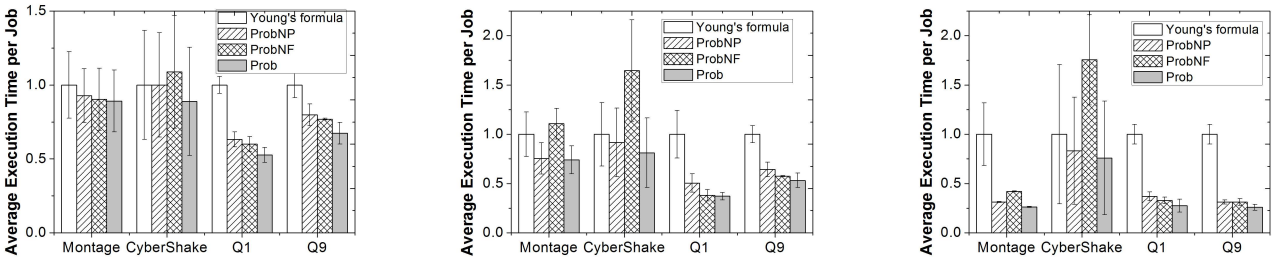
We look into the checkpoint interval optimized by *Prob* and Young's formula. Table 4 shows the checkpoint interval optimized by *Prob* for Q1 tasks on the four levels of workflow structure. All values are normalized to those of Young's formula. *Prob* generates much shorter checkpoint intervals compared to Young's formula and can better optimize the workflow execution time in the dynamic cloud. During the simulations with checkpoint interval optimized by Young's formula, 25% of the total instance time are wasted on rollback while the rollback loss in *Prob* is only 17%. When checkpoint overhead increases, the relative checkpoint interval optimized by *Prob* over that of Young's formula deceases. This explains why *Prob* reduces more execution time under high checkpoint overhead in Figure 10.

Comparing the results of ProbNP and ProbNF, we find that system failure dynamics and cloud performance dynamics are both affecting the performance of workflows executing in the cloud. Considering only one of the two dynamic factors is not enough, and can sometimes perform worse than the static method. For example, due to the large performance variance of CyberShake and the nonlinear relationship between task execution time and MTBF, ProbNF can lead to much worse optimization results than Young's formula. By incorporating both of the two dynamics, *Prob* has a better performance than both ProbNP and ProbNF.

7.3 Comparison with MC

Compared with naive sampling-based MC (Section 3.3), *Prob* is able to obtain good optimization results with a low optimization overhead for both optimization problems. In the following, we set the sampling size of MC to 10 thousands.

Effectiveness. We calculate the average ratio of the optimized results between *Prob* and MC. If the ratio is closer to 1.0, it means that *Prob* achieves a closer optimization result to MC. For the two optimization problems, the ratios are 1.05 and 1.01, respectively, which demonstrates the effectiveness of *Prob*. This is mainly due to that all optimizations in *Prob* can theoretically



(a) ckpt overhead=2s (b) ckpt overhead=10s (c) ckpt overhead=50s

Fig. 10: Normalized results for the fault tolerance problem.

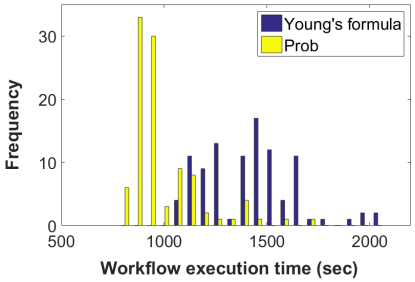
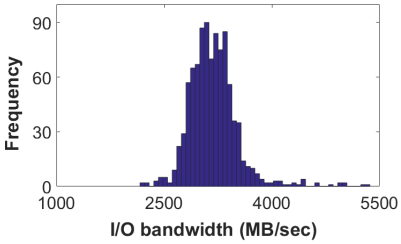
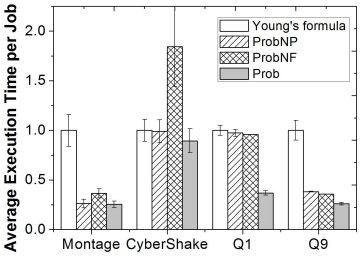


Fig. 11: Execution time distribution of Montage. Checkpoint overhead is 10 seconds.



(a) I/O performance variation



(b) Normalized results

Fig. 12: Simulation for fault tolerance problem in the cluster.

guarantee that the input probability distributions are not changed after optimizations.

Efficiency. Table 5 summarizes the optimization overhead of *Prob* and the average speedup over MC. 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 workflow-specific optimizations can reduce the path distribution evaluation overhead by 20% on average. With our partial solution evaluation pruning, we reduce solution searching overhead by up to 78% for budget-constrained scheduling problem.

7.4 Initial Study on Data-Intensive Clusters

So far, our study has focused on cloud environments. Similar to the cloud, data-intensive clusters are known to exhibit performance variations [8], [18] from different system aspects. For example, different from the cloud, data-intensive clusters are equipped with separate storage severs to handle I/O requests from all applications. Thus, the I/O performance can be largely dynamic due to cross-application interferences on the parallel file system level. Hereafter, we present our initial results on extending *Prob* for resource provisioning in the cluster environment, with special focus on the I/O performance variation in the cluster.

We simulate the fault tolerance problem with I/O bandwidth traces obtained from the Grid'5000 cluster [19], which is a large-scale platform shared by over 500 users. The I/O trace is collected by measuring the I/O bandwidth given to one application when simultaneously running multiple applications on the same cluster (refer to [18] for more details on the experimental setting). Figure 12a shows the obtained histogram of the I/O bandwidth. We again use the Google Trace to generate failures and set the checkpoint overhead to 50 seconds due to the large checkpoint overhead of parallel file systems [33].

Figure 12b shows the optimized average execution time results. Similar to the cloud evaluation results, *Prob* performs the best among all compared algorithms, which demonstrates the effectiveness of *Prob* on the shared cluster environment. The improvement of *Prob* over Young's formula is smaller than that in the cloud. This is mainly because the I/O performance in the shared cluster is relatively more stable than the cloud I/O performance (e.g., Figure 1b). It is our future work to study the performance of *Prob* on clusters in more details.

8 Related Work

We categorize the related work into the following categories: workflow optimization problems in the cloud, dynamic cloud environment and existing optimization approaches for improving the efficiency of probabilistic methods.

Workflow optimizations in the cloud. Performance and cost optimizations for data-centric workflows in the cloud have been studied by a number of existing works [3], [4], [5]. Mao et al. [3] proposed an auto-scaling method which maximize the

performance of workflows in the cloud under budget constraints. 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. However, most of the above mentioned studies have not considered the impact of cloud dynamics to the optimization effectiveness.

Dynamic cloud environment. Many existing works have studied the performance dynamics in the cloud. Previous works have demonstrated significant variances on the cloud performance [7], [34]. The distribution model (histogram) is also adopted in the previous work [7] to measure and analyze the cloud performance variances. 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) [35].

Dynamic scheduling methods have been proposed to make adaptive decisions for resource allocation in the cloud considering performance variations [5], [11]. 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 [11]. 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 [12], [13]. Those probability distributions are used in stochastic analysis to make resource allocation decisions while providing quantile-based QoS guarantees [13]. This kind of methods require remodeling the resource provisioning problems using stochastic models, which can sometimes be quite complicated. Probabilistic approaches are recently adopted [36], [25] for resource provisioning at offline time while still mitigating the impact of cloud dynamics to resource provisioning results. However, those approaches have a large overhead due to the costly calculations on probability distributions. Also, those solutions are designed for a specific goal while *Prob* can be adopted by different resource provisioning methods to incorporate cloud dynamics.

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 [37], aggregations and nearest neighbor [38] on probabilistic databases [39]. In business workflows, the probabilistic model and stochastic models have been adopted to describe the QoS, such as availability and reliability, of Web services [13]. 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 51% – 70% compared with the state-of-the-art static solutions, and our pruning techniques can greatly reduce the overhead of probabilistic solutions. As future work, we plan to study the effectiveness of *Prob* on more use cases and different systems.

References

[1] NASA/IPAC, "Montage workflow," http://montage.ipac.caltech.edu/docs/download2.html, 2005.

[2] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *SIGMOD '10*, pp. 1013–1020.

[3] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *IPDPS '13*, pp. 67–78.

[4] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in *SIGMOD '11*, 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 *SC '12*.

[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 *SC '13*, pp. 64:1–64:12.

[7] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *VLDB*, 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 *ICS '16*, pp. 6:1–6:12.

[9] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting hardware heterogeneity within the same instance type of amazon ec2," in *HotCloud'12*, 2012.

[10] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[11] 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 *ICPP '13*, pp. 419–428.

[12] O. Adam, Y. C. Lee, and A. Zomaya, "Stochastic resource provisioning for containerized multi-tier web services in clouds," *IEEE TPDS*, vol. PP, 2016.

[13] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A stochastic programming approach for qos-aware service composition," in *CCGRID '08*, pp. 226–233.

[14] F. J. Cazorla and et. al, "Proartis: Probabilistically analyzable real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, pp. 94:1–94:26, 2013.

[15] D. Poola, K. Ramamohanarao, and R. Buyya, "Fault-tolerant workflow scheduling using spot instances on clouds," in *ICCS 2014*, pp. 523–533.

[16] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *ICS '06*, pp. 324–334.

[17] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia, "Efficient join processing over uncertain data," in *CIKM '06*, pp. 738–747.

[18] 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 *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Chicago, United States, May 2016, conference. [Online]. Available: https://hal.inria.fr/hal-01270630

[19] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[20] E. Deelman and et. al, "Pegasus, a workflow management system for science automation," *FGCS*, vol. 46, pp. 17–35, 2015.

[21] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *E-SCIENCE '05*, pp. 140–147.

[22] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[23] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12, 2012, pp. 781–786.

[24] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99–109, 1943.

[25] 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 *HPDC '15*, pp. 223–234.

[26] Pegasus, "Workflow generator," https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator, 2014.

[27] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *SC '11*, 2011, pp. 49:1–49:12.

[28] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta, "Workflow task clustering for best effort systems with pegasus," in *MG '08*, 2008, pp. 9:1–9:8.

[29] Iperf, *https://github.com/esnet/iperf*, accessed on July 2017.

[30] S. Di, D. Kondo, and C.-L. Wang, "Optimization of composite cloud service processing with virtual machines," *IEEE TOC*, pp. 1755–1768, 2015.

[31] NASA/IPAC, "Montage archive," http://hachi.ipac.caltech.edu:8080/montage/, 2005.

[32] R. N. Calheiros and et. al, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Softw. Pract. Exper.*, 2011.

[33] S. Di, L. Bautista-Gomez, and F. Cappello, "Optimization of a multilevel checkpoint model with uncertain execution scales," in *SC '14*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 907–918. [Online]. Available: https://doi.org/10.1109/SC.2014.79

[34] 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 *SoCC '12*, pp. 20:1–20:14.

[35] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E.-G. Talbi, "Towards understanding uncertainty in Cloud computing resource provisioning," in *ICCS 15*, pp. 1772–1781.

[36] M. L. D. Vedova, D. Tessera, and M. C. Calzarossa, "Probabilistic provisioning and scheduling in uncertain cloud environments," in *ISCC '16*, pp. 797–803.

[37] L. Mo, R. Cheng, X. Li, D. W. Cheung, and X. S. Yang, "Cleaning uncertain data for top-k queries," in *ICDE '13*, pp. 134–145.

[38] J. Niedermayer, A. Züfle, T. Emrich, M. Renz, N. Mamoulis, L. Chen, and H.-P. Kriegel, "Probabilistic nearest neighbor queries on uncertain moving object trajectories," *VLDB*, vol. 7, pp. 205–216, 2013.

[39] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.

Amelie Chi Zhou received her Bachelor (2005-2009) and Master (2009-2011) degree from Beihang University, and her PhD degree (2011-2015) in computer science from Nanyang Technological University, Singapore. She is currently an Assistant Professor in College of Computer Science and Software Engineering, Shenzhen University, China. Prior to that, she was a Postdoc fellow in the Inria Rennes research center, France. Her research interests include cloud computing, big data processing and distributed systems.

Bingsheng He received the bachelor degree in computer science from Shanghai Jiao Tong University (1999-2003), and the PhD degree in computer science in Hong Kong University of Science and Technology (2003-2008). He is an Associate Professor in School of Computing, National University of Singapore. His research interests are high performance computing, distributed and parallel systems, and database systems. Since 2010, he has (co-)chaired a number of international conferences and workshops, including IEEE CloudCom 2014/2015 and HardBD2016. He has served in editor board of international journals, including IEEE Transactions on Cloud Computing (IEEE TCC) and IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS).

Shadi Ibrahim is a permanent Inria Research Scientist. He obtained his Ph.D. in Computer Science from Huazhong University of Science and Technology in Wuhan of China in 2011. From November 2011 to September 2013, he was a postdoc researcher within the KerData research team working on scalable Big Data managements on clouds. His research interests are in cloud computing, big data management, virtualization technology, and file and storage systems. He has published several research papers in recognized Big Data and cloud computing research conferences and journals including TPDS, FGCS, PPNA, SC, IPDPS, Mascots, CCGrid, ICPP, SCC, Cluster, and Cloudcom.

Reynold Cheng received the PhD degree from the Department of Computer Science, Purdue University, in 2005. He is an associate professor in the Department of Computer Science, University of Hong Kong. He received an Outstanding Young Researcher Award in 2011-2012 from HKU. He has served as a PC member and reviewer for international conferences (e.g., SIGMOD, VLDB, ICDE, and KDD) and journals (e.g., the IEEE Transactions on Knowledge and Data Engineering, the ACM Transactions on Database Systems, the Very Large Data Base Journal, and the Information Systems). He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering, and was on the EIC selection committee of the IEEE Transactions on Knowledge and Data Engineering. He is a member of the IEEE.