Probabilistic Scheduling of Scientific Workflows in Dynamic Cloud Environments

Amelie Chi Zhou, Bingsheng He and Cheng Liu
Nanyang Technological University

Abstract—Recently, we have witnessed scientific workflows from various applications running in the cloud. Due to the pay-as-you-go price scheme, the performance and monetary cost are two important optimization metrics. While there have been previous studies on minimizing the monetary cost for scientific workflows, most of them assume static execution time and static price scheme, and has the QoS notion of satisfying the static deadline. However, cloud environment is *dynamic*, with performance dynamics caused by the interference from concurrent executions and price dynamics like spot prices offered by Amazon EC2. Therefore, we propose the notion of offering probabilistic performance guarantees for individual workflows, which captures both performance and price dynamics. We further develop a probabilistic scheduling framework called *Dyna* to minimize the monetary cost while offering probabilistic deadline guarantees. The framework includes runtime refinement for performance dynamics, and a hybrid instance configuration approach to capture the best of both worlds in price dynamics: on-demand instances offering the reliability guarantee and spot instances usually with a much lower cost. We have developed a simulator with calibrations from real cloud providers. Experimental results with Amazon EC2 settings demonstrate (1) the accuracy of our simulations in capturing the distributions of cost and execution time of running workflows on real cloud environment; (2) the effectiveness of our framework on reducing monetary cost over the existing approaches while offering probabilistic guarantees on deadline requirement.

Keywords—Cloud computing, cloud dynamics, spot prices, monetary cost optimizations, scientific workflows.

1 Introduction

Cloud computing has become a popular computing infrastructure for various applications. One attractive feature of cloud computing is the pay-as-you-go charging scheme, where users only need to pay for the actual consumption of storage and computation hours. This feature unlocks the opportunities of large-scale computation without physically owning a cloud. Recently, we have witnessed many *continuous* workflows from various applications such as sciences and data intensive applications deployed and hosted in the Amazon EC2 or other cloud providers [1], [2]. In those applications, continuous workflows (i.e., jobs) are submitted and executed in the cloud and each workflow is usually associated with a deadline for QoS purposes [3], [4]. In the pay-as-you-go environment, performance and monetary cost are clearly two important optimization factors. However, the cloud environment is dynamic in terms of both performance [5], [6] and prices (e.g., Amazon offers spot prices). In this study, we investigate whether and how we can minimize monetary cost for continuous workflows in such cloud environments while satisfying the QoS requirement of individual workflows.

The problem of minimizing the monetary cost with performance constraints is *not* new. Over the era of grid computing, cost-aware optimization techniques have been extensively studied. Researchers have addressed various problems: minimizing cost given the performance requirements [7], maximizing the performance for given budgets [8] and scheduling optimizations with both cost and performance constraints [9]. When it comes to cloud computing, the pay-as-you-go pricing, virtualization and elasticity features of cloud computing open up various challenges and opportunities [3], [10]. For example, most cloud providers offer instance hour billing model. Partial-hour consumption is always rounded up to one hour. Recently, there have been many studies on monetary cost optimizations with resource allocations and task scheduling according to the features of cloud computing (e.g., [3], [4], [10], [11], [12], [13], [14]). Although the above studies have demonstrated their effectiveness in reducing the monetary cost, all of them assume static task execution time and consider only fixed pricing scheme (only *on-demand* instances in Amazon's terminology). Particularly, they have the following limitations on addressing the dynamics of the current cloud infrastructures.

First, cloud is by design a shared infrastructure. The resources in the cloud, such as the computation, storage and network resources, are shared by many concurrent jobs/tasks. Previous studies [5], [6] have demonstrated significant variances on I/O and network performance. The assumption of static task execution time in the previous studies (e.g., [3], [4], [10], [11], [12], [13], [14]) does not hold in the cloud. Because of performance dynamics, the QoS notion of a static deadline is not desirable to offer performance guarantees. Thus, existing scheduling algorithms need to be revisited and adapted to performance dynamics.

Second, cloud, which has evolved into an economic market [15], has dynamic pricing. Amazon EC2 offers spot instances, whose prices are determined by market demand and supply. Previous studies [3], [7], [16], [17], [4], [18] consider fixed pricing schemes only and their results need revisits in the existence of spot instances. On the other hand, spot instances can be used to reduce monetary cost [19], [20], [21], [22],

[23], [24], because the spot price is usually much lower than the price of on-demand instances of the same type. However, a spot instance may be terminated at any time when the bidding price is lower than the spot price (i.e., out-of-bid events). The usage of spot instances may cause excessive long latency due to failures. Most of the previous studies do not consider deadline constraints of individual workflows.

Those two kinds of cloud dynamics make challenging the problem of minimizing the monetary cost while satisfying QoS requirements of individual workflows. They add two new dimensions for the problem, and dramatically increase the solution space (see Section 3). Moreover, due to those dynamics, the notion of a strict deadline becomes less meaningful. 100% deadline guarantees will lead to the optimizations with worst-case performance/price prediction. Even worse, worst-case predictions can be unknown or unpredictable in some cases (e.g., there are some exceptionally high spot prices in the price history of Amazon EC2).

In order to address performance and price dynamics, we define the notion of probabilistic performance guarantees to represent QoS. The notion means the deadline of individual workflows is satisfied with a certain probability. For example, users may specify the probabilistic deadline guarantee to be 90%, meaning that at least 90% of the workflows can satisfy their own deadlines. Under this notion, we propose a probabilistic framework called *Dyna* to minimize the monetary cost while individual workflows have probabilistic performance guarantees predefined by the user. The framework embraces a series of static and dynamic optimizations for monetary cost optimizations, which are specifically designed for cloud dynamics. We develop probabilistic models to capture the dynamics in I/O and network performance, and spot prices [25], [26]. We further propose a hybrid execution with both spot and on-demand instances, where spot instances are adopted to potentially reduce monetary cost and on-demand instances are used as the last defense to meet deadline constraints.

To allow extensive evaluations of our probabilistic framework without using excessive budget, we have further developed a simulator for continuous workflows in dynamic cloud environments. We calibrate the cloud dynamics from a real cloud provider, particularly for the probabilistic models on I/O and network performance as well as spot prices.

We conduct experiments of running workflows on Amazon EC2 and simulations based on the calibration from Amazon EC2. Our experimental results demonstrate the following two major results.

1) With the calibrations from Amazon EC2, our simulations can accurately capture the distribution of the monetary cost and execution time of running workflows on Amazon EC2. By capturing the cloud dynamics, Dyna offers the optimized monetary cost with probabilistic performance guarantees for users to make scheduling decisions.
2) The hybrid instance configuration approach significantly reduces the monetary cost by 8 – 35% while the runtime refinement reduces the monetary cost by 26 – 34% over other state-of-the-art algorithms when the deadline is guaranteed to be over 90% satisfied.

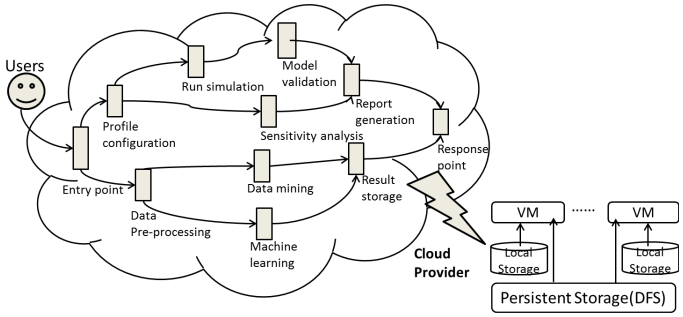


Fig. 1. An example scenario of scientific data analytics workflows.

To the best of our knowledge, our study is the first probabilistic scheduling framework for continuous workflows in the cloud.

Organization. The rest of the paper is organized as follows. We formulate our problem in Section 2 and review the related work in Section 2.3. We present our detailed framework design in Section 3, followed by the experimental results in Section 4. Finally, we conclude this paper in Section 5.

2 Background and Related Work

In this section, we present the application scenario and describe the terminology in our study, followed by the related work.

2.1 Application Scenario

Like many workflow applications in the cloud [3], [1], we consider a typical scenario of offering software as a service in the pay-as-you-go public cloud environment. In this software hosting, application owner deploys a relatively small number of workflow classes with different parameters. We can see many applications have continuous workflows in the cloud [1], [2]. Figure 1 illustrates an example scenario of scientific data analytics workflows. Users (e.g., scientists and officials) can submit their simulation tasks for predictions, or perform sensitivity analysis. Users can also perform data analysis on scientific data with data mining or machine learning techniques. When users submit their jobs, execution deadlines are specified to the jobs for QoS purposes. The application owner buys virtual machine from public cloud providers such as Amazon EC2. Our goal is to provide a probabilistic scheduling framework to application owner, aiming at minimizing the monetary cost.

2.2 Terminology

Instance. An instance is a virtual machine offered by the cloud provider. Each instance belongs to a type of virtual machines with the same provisioned resources such as CPU and RAM. Different types of instances have different capabilities such as CPU speed, I/O speed and network bandwidth. We assign IDs to the instance types in the increasing order of their prices. IDs are consecutive integers, starting from one.

The acquisition has a non-ignorable instance acquisition time. For simplicity, we assume the acquisition time is a constant, lag.

TABLE 1
Statistics on spot prices ($/hour, Dec2011, US East Region) and on-demand prices of Amazon EC2

Instance type	Average	$stdev$	Min	Max	OnDemand
Small	0.063	0.0413	0.036	0.258	0.08
Medium	0.072	0.0001	0.071	0.072	0.16
Large	0.149	0.0220	0.144	0.346	0.32
xLarge	0.288	0.0078	0.288	0.597	0.64

An instance can be on-demand or spot. We adopt the instance definition of Amazon EC2. Amazon adopts the instance hour billing model, whereby partial instance hour usage is rounded to one hour. Both on-demand and spot instances can be terminated when users no longer need them. If it is terminated by users, the users have to pay for any partial hour (rounded up to one hour). For a spot instance, if it is terminated in a out-of-bid event, users do not need to pay for any partial hour of usage.

Table 1 shows some statistics of the price history of four types of spot instances on Amazon in the US East region during December 2011. We also show the price of the on-demand instances for these four types. We have the following observations: a) The spot instances are usually cheaper than on-demand instances. There are some "outlier" points where the maximum price is much higher than the on-demand price (e.g., on the small instance). b) Different types have different variations on the price. Those observations are consistent with the previous studies [26], [25].

Job. A job is expressed as a workflow of tasks with precedence constraints. A job has a deadline. We note that deadlines are "soft deadlines", unlike "hard deadline" as in hard real-time systems.

Task. A task is represented as $<\#instr,\ d_{seqIO},\ d_{rndIO},\ netInput,\ netOutput>$, where $\#instr$ represents the total number of instructions to be executed for the task, d_{seqIO} and d_{rndIO} are the amount of I/O data for sequential and random accesses respectively to local disk, and $netInput$ and $netOutput$ are the amount of input and output data that need to be read in and sent out respectively. We estimate the task execution time as follows, where we extend the execution time estimation with the time period dimension.

$$\begin{aligned} t_{exec}(task, type, t) = \#instr(task)/cpu\_frequency(type) \\ + d_{seqIO}(task)/seqBand(type, t) \\ + d_{rndIO}(task)/rndBand(type, t) \\ + netInput(task)/inBand(type, t) \\ + netOutput(task)/outBand(type, t) \end{aligned} \quad (1)$$

In the estimation, $cpu\_frequency$ describes the CPU capability on instance type $type$; $seqBand$, $rndBand$, $inBand$ and $outBand$ are the average sequential, random I/O and input, output network bandwidth values of instance type $type$ during the time period t. We ignore the latency in the network cost, because the latency is usually a minor issue in data center networks.

Instance configuration. In the hybrid execution of spot and on-demand instances, a task may be executed by multiple instances sequentially. *Instance configuration* of a task is defined as an n-dimension vector: $<$ ($type_1$, $price_1$, $isSpot_1$), ($type_2$, $price_2$, $isSpot_2$), ..., ($type_n$, $price_n$, $isSpot_n$) $>$, meaning the task is potentially executed by n instances belonging to $type_i$ ($1 \leq i \leq n$) and $isSpot_i$ indicates whether the instance is spot or on-demand. If the instance i is a spot instance, $price_i$ is the specified bidding price, and the on-demand price otherwise. In our hybrid instance configuration, on-demand instances can appear at the tail of the vector only. Therefore, $isSpot_i$ must be false when i is equal to n, because the on-demand instance guarantees 100% of success rate on the execution.

2.3 Related Work

For decades, cost-aware optimizations have generate fruitful research for decades in various areas such as grid computing [27], databases [28] and Internet [29]. There are a lot of works related to our study, and we focus on the most relevant ones on cost optimizations and cloud performance dynamics. To the best of our knowledge, this work is the first scheduling framework that leverages both on-demand and spot instances and capture both performance and price dynamics in the cloud.

2.3.1 Cost-aware optimizations

The pay-as-you-go nature of cloud computing attracts many research efforts in dynamic resource provisioning. Dynamic virtual machine provisioning has been determined by control theory [30], [31], machine learning [32] and models [33]. On the other hand, workflow scheduling with deadline and budget constraints (e.g., [7], [16], [17], [4], [18], [34]) has been widely studied. Yu et al. [7] proposed deadline assignment for the tasks within a job and used genetic algorithms to find optimal scheduling plans. Kllapi et al. [4] studied the tradeoff between monetary cost and performance, and modeled the tradeoff as *sky line* operations in databases. Those studies only consider a single workflow with on-demand instances only. Malawski et al. [34] proposed dynamic scheduling strategies for workflow ensembles. While the previous studies [3], [35], [36] proposed auto-scaling techniques, their scaling is based on static execution time of individual tasks. In comparison with the previous work, the unique feature of Dyna is that it targets at offering probabilistic performance guarantees as QoS, instead of static deadlines. Dyna schedules the workflow by explicitly capturing the performance dynamics (particularly for I/O and network performance) in the cloud.

Amazon EC2 spot instances have recently received a lot of interests. Related work can be roughly divided into two categories: modeling spot prices [25], [26] and leveraging spot instances [19], [20], [21], [22], [23], [24].

For modeling spot prices, Yehuda et al. [25] conducted reverse engineering on the spot price and figured out a model consistent with existing price traces. Javadi et al. [26] developed statistical models for different spot instance types. Those models can be adopted to our hybrid execution.

For leveraging spot instances, Chohan et al. [21] proposed a method to utilize the spot instances to speed up the MapReduce tasks, and suggested that fault tolerant mechanisms are essential to run MapReduce jobs on spot instances. Yi et al. [19] introduced some checkpointing mechanisms for reducing cost of spot instances. Further studies [23], [24] used spot

instances with different bidding strategies and incorporating with fault tolerance techniques such as checkpointing, task duplication and migration. Those studies are with spot instance only, without offering any guarantee on meeting the workflow deadline like Dyna does.

2.3.2 Cloud performance dynamics and Tools

A few studies have evaluated the performance of cloud services from different aspects [37], [5], [6]. Our calibration results on Amazon EC2 is consistent with Schad et al.'s work [5]. We focus on calibrating a cloud provider as input to our simulator. Iosup et al. [6] did a cross-platform comparison on four commercial cloud providers (Amazon EC2, GoGrid, ElasticHosts and Mosso) for scientific computing workloads. There have been more in-depth performance studies on specific components (network performance [38], [39] and I/O interference [40], [41]).

There have been some proposals to reduce the performance interference and unpredictability in the cloud, such as network performance [42] and I/O performance [43], [44], [45]. However, by the design of cloud computing, cloud is shared by many concurrent executions. Therefore, the performance dynamics caused by the resource interference is unavoidable.

Some other works have devoted efforts on designing tools for assessing cloud performance. Lenk et al. [46] designed a new performance measuring method that can measure the actual performance of the virtual machines running a specific IaaS service. CloudSleuth [47], CloudHarmony [48], Cloudstone [49] and CloudCMP [50] are four common tools and services that can help customers measure the performance of cloud services. For example, CloudCMP provides comparisons on the performance of computation, storage and network services offered by different cloud providers. Chen et al. [51] have developed WorkflowSim with consideration on system overhead and failures.

In comparison with existing tools, this paper focuses on the statistical analysis on cost and performance for continuous workflows. Our simulator goes beyond the existing tools [46], [48], [47], [49], [51] in two aspects. First, we perform statistical analysis according to cloud dynamics, whereas existing tools are simply based on calibrations. Second, our simulator has realized the scheduling framework for continuous workflows, whereas existing tools adopt straightforward calculations. Moreover, we have evaluated the accuracy of simulations with the comparison of running workflows on Amazon EC2, and showed that the simulation allows users to make wise decisions without excessive monetary costs on running the workflows in real cloud.

3 Framework Design and Implementation

We first formally define the scheduling problem in the dynamic cloud environment. Then we present an overview of the Dyna framework and discuss the details about two key optimizations: runtime refinement and the hybrid instance configuration approach to address the performance and price dynamics respectively. Finally, we present the implementation of the simulator.

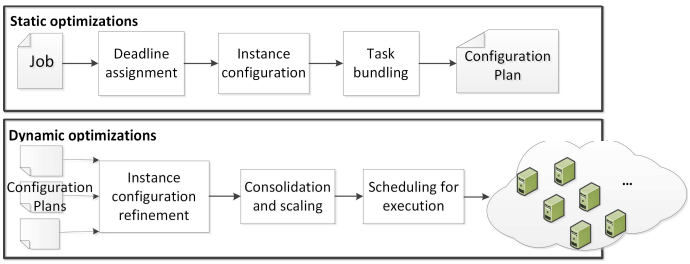


Fig. 2. Static and dynamic optimizations in the Dyna framework

3.1 The scheduling problem in dynamic cloud

As described in Introduction, the workflow scheduling in the cloud must capture two kinds of dynamics: 1) performance dynamics caused by interferences from concurrent executions on the shared resources, 2) price dynamics from spot prices. Due to the fluctuating workloads in the continuous workflow execution and the cloud dynamics, it is very difficult for the application owner to provision the monetary cost accurately before execution. We also need an auto-scaling method to dynamically acquire or release instances with the variance of cloud performance. In particular, we need two plans to solve the problem:

- A *configuration plan* is to statically determine the instance configuration with the consideration of both spot and on-demand instances for every task in the jobs.
- A *runtime scheduling plan* is to acquire/release instances at runtime with the awareness of cloud performance variation to enable consolidation and auto-scaling. Runtime refinement on the instance configuration is also performed.

The configuration plan is obtained offline and the runtime scheduling plan is for adapting provisioning at runtime. Instance configuration is a time consuming process (Section 3.4). It is statically determined in an optimal configuration plan so that the invocations of runtime refinement can reduce. Due to the cloud dynamics, strictly meeting the deadlines means always running the tasks with the worst-case performance. Thus, we propose probabilistic deadline guarantees to capture the cloud dynamics. Based on the probabilistic distributions on monetary cost and execution time, the user can make decisions on scheduling. Therefore, we propose a probabilistic framework to find a configuration plan and a runtime scheduling plan that minimizes the total running cost and satisfies all job deadlines.

3.2 Framework Overview

Dyna is a probabilistic scheduling framework for continuous workflows in the cloud. The main components of Dyna including static and dynamic optimizations are illustrated in Figure 2. The reason that we develop a general framework has two folds. First, there have been some classic job/task scheduling optimizations for monetary cost and performance [7], [8], [9]. A framework allows integrating key optimizations in a holistic manner. Second, different cloud providers may have

different behaviors and offerings in performance and price dynamics. A general framework allows more flexibility in implementing specific algorithms for different cloud providers, without affecting other key optimizations. For example, Dyna supports not only the hybrid instance configurations, but also baseline approaches – on-demand only or spot only.

Static optimizations. When a workflow is submitted to the cloud with a predefined probabilistic deadline guarantee, we analyze each task to find the most cost-effective configuration plan. We perform the static optimization in three steps in order: deadline assignment, instance configuration and task bundling. While those steps have been considered in previous studies [3], [4], [10], [11], their assumptions on static task execution time do not hold on the cloud. Therefore, we specifically re-design them with the goal of achieving the probabilistic deadline guarantees.

We use an existing mechanism to assign the deadline within the workflow, which is introduced and detailed in the previous paper [7]. As we consider the performance dynamics in the cloud, some modifications to the deadline assignment step are necessary for both accuracy and performance guarantee. We set the task execution time according to the probabilistic deadline guarantees. If the deadline guarantee is p%, we estimate the task execution time to be t_0 so that $P(t_{exec} \leq t_0) \geq p\%$, where $P(t_{exec} \leq t_0)$ denotes the probability of the task execution time shorter than t_0. If the deadline guarantee is p%, we estimate the task execution time to be t_0 so that $P(t_{exec} \leq t_0) \geq p\%$, where $P(t_{exec} \leq t_0)$ denotes the probability of the task execution time shorter than t_0. Here, since the relationship between the deadline of each task and the deadline of entire workflow is complicated, we follow a simplified estimation: If every task can successfully finish its execution with the probabilistic deadline guarantee, the deadline requirement for the entire workflow can be met in a probabilistic manner.

The instance configuration step is more complex than that in [3] due to the price dynamics of spot instances. We need to find the cost-effective hybrid instance configuration for each task given the deadline constraints. The solution space is huge. We develop effective greedy algorithms to solve this problem. The details can be found in Section 3.4.

Ideally, we can run multiple tasks sequentially on the same instance in order to utilize the partial hour. Task bundling needs special attentions for the hybrid instance configuration. If spot instances are used in the instance configuration, the total execution time of running the bundled tasks on a spot instance is longer than any one task before bundling. Thus, a higher bidding price for the spot instance may be required and we adjust the bidding price accordingly. The process is similar to the instance configuration algorithm except the instance configuration is fixed in this scenario.

Dynamic optimizations. At runtime, we maintain a pool of spot instances and on-demand instances, organized in lists according to different instance types. Figure 3 shows an example of the pool of current spot and on-demand instances. Three $type_1$ spot instances are now in the pool and we only have two $type_2$ spot instances. The spot and on-demand instances with the same instance type are organized in separated lists.

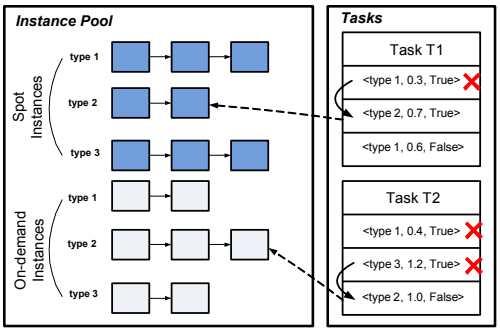


Fig. 3. An example of hybrid execution

When a task's all preceding tasks in the workflow have finished execution, the task becomes ready and can be scheduled to execute on an instance. The instance is selected according to the task's instance configuration which is calculated in the static optimization stage. Figure 3 shows an example with two tasks named as T1 and T2. The instance configuration for T1 is $type_1$, $type_2$ and $type_1$. Initially, this task is assigned to a spot instance with type $type_1$. But the task's execution fails due to out-of-bid event, it will restart on a $type_2$ spot instance. If there is no $type_2$ spot instance available in the instance pool, the task will wait for the setup of a new $type_2$ spot instance which will be bid with the price in the task's hybrid instance configuration. At the meantime, if T2 has failed on both $type_1$ and $type_3$ instances, it will execute on a $type_2$ on-demand instance. Similarly, if there is no $type_2$ on-demand instance in the pool, a new $type_2$ on-demand instance is requested first and added into the instance pool.

The dynamic optimization includes the following steps.

Step 1 – Runtime instance configuration refinement. There are two major scenarios that require runtime instance configuration refinement. Firstly, if a task's preceding tasks finish ahead of their own deadlines (e.g., successfully finish on a spot instance), the task will be given more time to finish. The second scenario is when the actual I/O or network time is far different from the estimations used in generating the static configuration plan. In both scenarios, the difference between the task execution time and the deadline changes, and thus we need to refine the instance configuration. We use the optimal hybrid instance configuration in Section 3.4 for the refinement of the new deadline. To avoid the excessive overhead of hybrid instance configuration, the algorithm is invoked when the difference varies over $threshold\%$ from that in the static optimization (by default $threshold$ = 50).

Step 2 – Consolidation and scaling. We apply the earliest deadline first (EDF) scheduling, with the consideration of instance consolidation and auto-scaling techniques. Consolidating different instances could reduce the new instance acquisition time as well as the instance-hour billing overhead. For each instance in the pool, we record the instance's remaining partial hours. To increase the consolidation opportunity, we can defer the ready task according to its deadline residual.

Due to the difference between spot and on-demand instances, virtual machine consolidation among them needs special care. If the remaining time of the spot (resp. on-demand)

instance is long enough for a ready task's execution, and the task is going to acquire a spot (resp. on-demand) instance of the same type, the task will be assigned to the instance. The consolidation between spot and on-demand instances is possible only for the case whereby spot instance request is consolidated to an on-demand instance, due to the success guarantee.

The auto-scaling method is responsible for instance acquisition/release. For the instances which do not have any task assigned on and are approaching multiples of full instance hours, we release them. When a task with the deadline residual of zero requests an instance and the task is not consolidated to an existing instance, we acquire a new instance from the cloud provider. Now, the instance acquisition/release is dynamically adjusted to the workload fluctuation and at the same time minimizing the budget.

3.3 Addressing Performance Dynamics

Due to the performance dynamics, a static scheduling is usually unfeasible to achieve the minimum monetary cost. Ideally, we can monitor the I/O and network performance at runtime and aggressively refine the instance configuration at runtime. However, due to the huge search space, the hybrid instance configuration refinement is costly (Section 3.4). Thus, we address the performance dynamics in our scheduling framework in two parts: probabilistic static optimization and dynamic optimization. Through probabilistic static optimization, we pick the instance configuration that is most likely to achieve the minimum monetary cost and avoid the excessive runtime overhead of dynamic optimization.

In the static optimization step, the actual execution time of tasks on a submitted workflow is unknown. We develop probabilistic distribution models to describe the performance dynamics for I/O and network. Previous studies [5], [15] show that I/O and network are the major sources of performance dynamics in the cloud due to resource sharing while the CPU performance is rather stable for a given instance type. We define the probability of the I/O and network performance equaling to a certain value x on instance type $type$ to be: $P_{seqBand,type}(seqBand = x)$, $P_{rndBand,type}(rndBand = x)$, $P_{inBand,type}(inBand = x)$, $P_{outBand,type}(outBand = x)$ as the probabilistic distributions for the sequential I/O, random I/O, downloading and uploading network performance from/to persistent storage of instance type $type$, respectively. In our calibrations on Amazon EC2, $P_{rndBand,type}(rndBand = x)$ conforms to normal distributions and the rest conforms to Gamma distributions (Section 4).

Given the I/O and network performance distributions, we develop a probabilistic approach for the static optimizations such as deadline assignment and instance configuration. The deadline assignment uses the estimated task execution time according to the deadline guarantee. For instance configuration, we adopt Monte Carlo method to simulate the execution time of tasks and generate the optimal instance configuration according to the execution time.

In the runtime optimization, we monitor the execution time of the tasks that have been executed and perform runtime instance configuration refinement when the difference between the monitored performance and the performance estimated in the static optimization exceeds $threshold\%$. This is to invoke the configuration refinement only when it can bring a sufficiently large gain on performance/monetary cost.

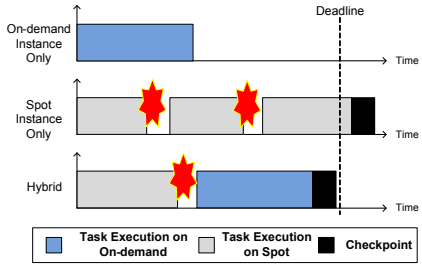


Fig. 4. Hybrid Approach Demonstration

3.4 Addressing Price Dynamics

For individual tasks, we develop a probabilistic static optimization to determine their instance configurations before scheduling (Section 3.3). Conventionally, there are two ways to deal with the instance configuration: with on-demand instances only or with spot instances only. However, these two methods have prominent deficiencies. The on-demand only approach assigns on-demand instances to a task, e.g., the previous study on reducing the monetary cost [3]. However, when spot instances are considered, we can further reduce the cost. If the deadline allows, we can actually run the task with spot instances for advance, as we will see in the hybrid approach. In contrast, the spot-only approach executes the tasks on spot instances. Running a task on spot instances may suffer from the out-of-bid events. The spot-only approach cannot guarantee the execution deadline.

We propose a hybrid instance configuration with the adoption of both on-demand and spot instances for configuring tasks in workflows. Our proposed hybrid approach attempts to use not only spot instances in the early stage of task execution with the expectation of a lower monetary cost, but also one on-demand instance as the last line of defence for the deadline guarantee. That means, tasks may be successfully executed on the spot instances to achieve a lower monetary cost, while the execution deadline is still guaranteed by running the task on an on-demand instance when all executions on prior spot instances fail. Figure 4 compares these approaches working on a single task. If a task fails on a spot instance, its failure does not trigger the re-execution of its precedent tasks. The results of precedent tasks are already materialized to the persistent storage in the cloud (such as Amazon S3).

To find the hybrid instance configuration, we need a procedure of calculating the monetary cost of a given hybrid instance configuration. Consider a hybrid instance configuration of n instances: $<type_i, price_i, isSpot_i> (1 \leq i \leq n)$, where only the last instance is on-demand. The expected cost of the ith instance is $price_i \times time_i$, where $time_i$ is the latest estimation of execution time of the task on the instance type $type_i$. In the sequential execution, the ith (spot) instance succeeds with the probability of $(1 - P(time_i, price_i))$.

Here, we do not take into account the instance hour billing model. That means, we have ignored the rounding monetary cost in the estimation. This is because in the continuous workflow environments, this rounding monetary cost is amortized among many tasks. Enforcing the instance hour billing model could severely limit the optimization space, leading to a suboptimal solution.

We derive the calculation of the expected monetary cost of a given hybrid instance configuration in Algorithm 1. The algorithm starts from the on-demand instance (i.e., the last element in the hybrid instance configuration). The reason that we start with the last instance is to match the order of finding optimization candidates for hybrid instance configuration, as we describe later in this section. For iteration $i < n$, we calculate the expected monetary cost by considering the probability of an out-of-bid event of using spot instance $type_i$.

Existing studies have demonstrated that the spot prices can be predicted using statistics models [26] or reverse engineering [25]. Thus, we assume the spot price distribution is known for all spot instance types. Following the previous studies [26], [25], we define the probabilistic function for a spot instance $P(price, time)$ denoting the probability of an out-of-bid event occurring at the bidding price $price$ after a period of $time$.

Here is an example to demonstrate the calculation of expected monetary cost for a hybrid instance configuration. Suppose we have two instance types A and B. A task needs 100 and 50 minutes to finish execution on instance types A and B, respectively. One example of hybrid instance configuration for the task includes three parts, which are denoted as $<A,\ 0.3,\ True>$, $<B,\ 0.7,\ True>$, $<B,\ 1.1,\ False>$ respectively. According to Algorithm 1, the cost of running task on on-demand instance (i.e., the last element in hybrid configuration) is calculated first and the variable C has the value of 0.917 (i.e., $1.1 \times (50/60)$). We assume that the probability is 0.8 when using type-B spot instance for 50 minutes with the bidding price $\$0.7$. The expected cost of running the task in the last two instance configuration can be calculated as $0.917 \times (1.0 - 0.8) + 0.7 \times (50/60) \times 0.8$ (i.e., $\$0.650$). If the probability of running a task for 100 minutes in a type-A spot instance is 0.6 when the bidding price is 0.4, the expected monetary cost for the given hybrid instance configuration is calculated as $\$0.560$.

Algorithm 1 Expected monetary cost calculation, $\mathcal{C}$, for a hybrid instance configuration of n instances

Require: Hybrid instance configuration $<type_i,\ price_i,\ isSpot_i>$ $(1 \le i \le n)$
1: $\mathcal{C} = 0$;
2: Denote $time_i$ to be the estimated execution time on the ith instance with type $type_i$ according to the time vector;
3: **for** $i = n;\ i >= 1;\ i = i - 1$ **do**
4: $time_i = Task.time(type_i)$;
5: **if** $!isSpot_i$ **then**
6: $\mathcal{C} += time_i * price_i$;
7: **else**
8: $p = P(price_i,\ time_i)$ for instance type $type_i$;
9: $\mathcal{C} = \mathcal{C} * (1 - p) + time_i * price_i * p$;
10: **return** $\mathcal{C}$;

Given the expected monetary cost calculation, we now develop the algorithm of finding the hybrid instance configuration for a task. However, the optimization space is still large, even for a single task. We need to consider not only the instance types in each configuration, but also the bidding price for each spot instance. Note, a higher bidding price increases the monetary cost but the risk of an out-of-bid event is lower. Thus, we develop a greedy algorithm for finding the hybrid instance configuration.

We formulate the problem as a *bounded search* problem. We start the search with the on-demand instance first, and gradually search the potential candidate by considering spot instances in a hybrid instance configuration candidate. The reason for starting with on-demand instance is that it is more determined than starting with the spot instance. Thus, we start with all the *partial* configurations with an on-demand instance. We maintain a priority queue for all the partial configurations according to their expected monetary cost. At each time of considering an additional spot instance, we pick the most promising partial configuration (i.e., the one with the smallest expected monetary cost in the queue), and then add an instance satisfying two constraints: a) adding the instance, execution still meets the deadline; b) it maximizes the monetary cost reduction among all the possible instance types. If such an instance type is not available, the potential configuration becomes *complete*. Otherwise, we add the partial configuration back to the priority queue. During the process, we maintain the cheapest complete configuration. When the priority queue becomes empty, the cheapest complete configuration we have maintained is the optimal hybrid instance configuration.

There are several issues that are worth some discussion. First, the overhead of this optimization process can be high. To allow early termination, we offer a parameter named $MaxCandidate$ to avoid too long execution time to find the optimal instance configuration. If the number of partial configuration generations is larger than $MaxCandidate$, we stop the optimization process. Second, the accuracy of the probabilistic function for out-of-bid event affects the effect of our optimization, but does not affect the correctness of our approach by meeting the deadline requirement. Lastly, the output of a task is stored into the persistent storage (e.g., Amazon S3). There are a number of advantages for this simple scheme. First, when a task on a spot instance succeeds finishing executions and outputting the results to the persistent storage, it does not need to be re-executed even if the spot instance is terminated. Second, if intermediate data are directly transferred from parent tasks to child tasks, the instance running the parent task has to be on until all the data transfers are finished. That causes the monetary waste. Lastly, on Amazon EC2, we have observed that the network bandwidth between instances and Amazon S3 is higher than that between instances.

3.5 Simulator Implementation

Due to performance and price dynamics, it is an nontrivial task to compare the performance and monetary cost of different scheduling algorithms. In order to make informative decisions on continuous workflows, we need to compare the probabilistic

distributions of cost and performance. This is in contrast with the existing static approach.

Comparing probabilistic distributions can be costly in the real cloud environment. Because there is no closed-form models for estimating both performance and price dynamics in the scheduling algorithm, the basic approach – Monte Carlo is used, which requires many runs of executions. Suppose a job has the depth of ten and the width of five and each task in the job takes half an hour for execution, and the Monte Carlo simulation runs $1,000$ (or $1k$) times. The cost for obtaining the monetary cost and performance distribution of this single job on the Small instance on Amazon EC2 is $1,000 \times (0.5h \times \$0.08/h \times 10 \times 5)$ (around $2000). The cost is even higher by considering multiple types of instances. We have already analyzed the solution space in Section 3.4.

In this paper, we propose to develop a simulator with calibrations on real clouds. The calibration is to allow our simulation sufficiently close to the real cloud environments without causing excessive monetary costs. Since the calibration cost occurs in initialization only, the simulation of different workflows and different scheduling algorithms does not increase the cost. We will evaluate the accuracy of our simulator in capturing the distributions of monetary cost and execution time of running workflows on real cloud environments in Section 4.2.

The simulator takes continuous workflows in the form of DAGs and performance and price distributions calibrated from the cloud as input. The simulator runs Monte Carlo simulations to generate the distributions of monetary cost and execution time for different scheduling algorithms. With those distributions, users can make more informative decisions prior to running the continuous workflows on the cloud. The example decisions include comparing different scheduling algorithms, and selecting the most cost-efficient cloud provider.

We have implemented Dyna as well as a number of comparisons in the simulator. We consider two baseline algorithms.

- **Static.** This baseline approach is the same as the previous study in [3] which only adopts on-demand instances. The task execution time is fixed to be the expected task execution time.
- **Spot-only.** This baseline approach adopts spot instances only [19]. Similar to Static, the task execution time is fixed to be the expected task execution time.

Additionally, we also implement the following variants to assess the impact of individual optimizations: hybrid instance configuration and runtime configuration refinement.

- **Dyna-NS** is Dyna without using spot instances. Dyna-NS has the static optimizations in estimating the task execution time according to the probabilistic deadline guarantees.
- **Dyna-NPD** is Dyna without runtime configuration refinement. The task execution time is fixed to be the expected task execution time. This is to evaluate the impact of capturing performance dynamics.

4 Evaluation

In this section, we present the simulation results of the proposed approach with the calibration on Amazon EC2.

4.1 Experimental Setup

We present the experimental setup for real cloud environments (i.e., Amazon EC2) and for simulations.

4.1.1 Setup for Cloud Environments

We have two sets of experiments on real cloud environments: firstly running workflows for evaluating the accuracy of simulation, and secondly cloud calibrations for simulation input.

Workflow executions. We have implemented our scheduling framework in Amazon EC2. Particularly, we run workflows with different scheduling algorithms in the cloud and measure their execution time and monetary cost. We have adopted DAGMan [52] to manage task dependencies and added Condor [53] to the Amazon Machine Image (AMI) to manage task execution and instance acquisition. The CPU computation, I/O and network is conducted with our homegrown C/C++ routines, subject to the number of instructions, the amount of I/O and network in the task. The experimental setup on workflow is the same as those in simulations (described in Section 4.1.2).

Calibration. We measure the performance of CPU, I/O and network for different instance types. We find that CPU performance is rather stable, which is consistent with the previous studies [5]. Thus, we focus on the calibration for I/O and network performance.

We measure both sequential and random I/O performance for local disks. The sequential I/O reads performance is measured with *hdparm*. The random I/O performance is measured by generating random I/O reads of 512 bytes each. Reads and writes have similar performance results, and we do not distinguish them in this study.

We measure the uploading and downloading bandwidth between different types of instances and Amazon S3. The bandwidth is measured from uploading and downloading a file to/from S3. The file size is set to $8MB$. We also measured the network bandwidth between two instances using Iperf [54]. We find that the network bandwidth among different instances is generally lower than that between instances of the same type and S3.

We mainly consider four frequently used instance types namely Small, Medium, Large and xLarge (i.e., extra large). The hourly costs of the on-demand instance for these four types are shown in Table 1. We acquire the instances from the US East region. Those four instances have also been used in the previous studies [19]. In particular, we repeat the performance measurement on each kind of instance for $1k$ times. When an instance has been turned on for a full hour, it will be turned off and a new instance of the same type will be created to continue the measurement. The measurement results are used to model the probabilistic distributions of I/O and network performance.

As for the instance acquisition time (lag), our experiments show that each on-demand instance acquisition costs 2 minutes and spot instance acquisition costs 7 minutes on average. This is consistent with the existing studies by Mao et al. [55].

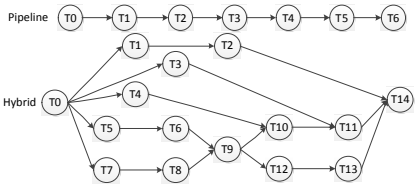


Fig. 5. Workflow structures: pipeline and hybrid

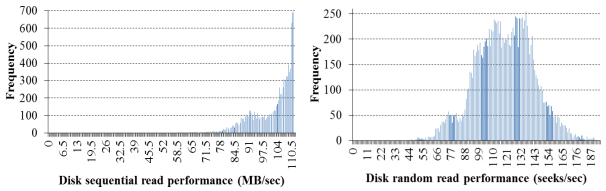


Fig. 6. Histograms of sequential and random I/O performance on Medium instances

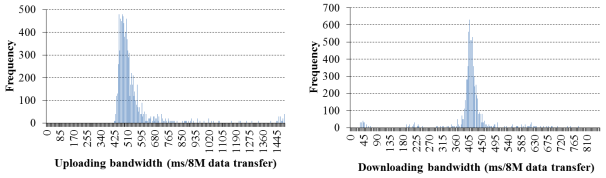


Fig. 7. Histograms of uploading and downloading bandwidth between Medium instances and S3 storage

4.1.2 Setups for Simulation

We conduct our simulation studies on a local server with an Intel Xeon X5675 CPU and 24G memory. Our scheduling algorithm is efficient, which can handle about 270 tasks per second on a single CPU core. We use the performance and the price settings and the history spot prices from Amazon EC2. We use the historical spot prices in December 2011. The simulator starts from a random time point in the spot price history.

The number of Monte Carlo runs on performance and price dynamics was set to $1k$ by default. We experimentally evaluate the impact of this parameter in Section 4.5.

There have been some papers on characterizing scientific workflows [56], [57]. In this paper, we consider two common structures in those scientific workloads in the experiments: *pipeline* and *hybrid*. As shown in Figure 5, pipeline is a simple workflow structure and hybrid is a more complex structure. Those structures have also been used in the previous studies [7], [3], [4].

We consider Amdahl's law for the execution time of a task on different instance types. There are four basic types of tasks considered in the experiments. The CPU execution time of a task on the Small instance type is set to 120 $mins$, 90 $mins$, 60 $mins$ and 30 $mins$ separately for the four types of tasks. The CPU execution time on other instance types is set according to the proportion P (denoted as the parallel rate) of a program that can be made fully parallel as defined in Amdahl's law. By default, $P = 95\%$. We set the $\#instr$ parameter according to CPU frequency so that their CPU execution time on different instance types is as our setting. When generating a specific task, we further set each task's d_{seqIO}, d_{rndIO}, $netInput$ and $netOutput$ so that the average execution time on Small instances for CPU, I/O and network are 80%, 10% and 10% of the total execution time respectively for compute-intensive (CI) applications and 30%, 60% and 10% respectively for IO-intensive (II) applications by default. The random I/O takes 20% of the total I/O time.

The deadline is an important factor for the candidate space of determining the hybrid instance configuration. There are two deadline settings with particular interests: D_{min} and D_{max}, the average execution time of all the tasks in the critical path of the workflow all on the Large and Small instances, respectively. By default, we set the deadline to be $\frac{D_{min}+D_{max}}{2}$.

We assume the job arrival conforms to a Poisson distribution. The parameter λ in the Poisson distribution affects the chance for virtual machine reuse. We also assume λ is sufficiently large so that at least two workflow instances are executed in the cloud at any time for a high utilization of the instance pool.

As for metrics, we study the average monetary cost and elapsed time for a workflow. Given the probabilistic deadline hit rate requirement (i.e., the expected ratio for the workflows satisfying the deadline), we consider monetary cost as the main matric for comparing different scheduling algorithms that can achieve the requirement.

4.2 Cloud Dynamics

We present the performance dynamics observed on Amazon EC2. The price dynamics have been presented in Table 1 of Section 2.

Figure 6 and Figure 7 show the measurements of I/O and network performance with Amazon S3 for Medium instances. We have observed similar results on other instance types. We make the following observations.

First, both I/O and network performances can be modeled with normal or Gamma distributions. We verify the distributions with null hypothesis, and find that (1) the sequential I/O performance and uploading and downloading network bandwidth from/to S3 of the four instance types follow Gamma distribution; (2) the random I/O performance distributions on the four instance types follow normal distribution. The parameters of these distributions are presented in Tables 2 and 3.

Second, the I/O and network performance of the same instance type varies significantly, especially for Small and Medium instances. This can be observed from the θ parameter of Gamma distributions or the σ parameter of normal distributions in Tables 2 and 3. Additionally, random I/O performance varies more significantly than sequential I/O performance on the same instance type. The coefficient of variance of sequential and random I/O performance on Medium are 9% and 18.7%, respectively.

Third, the performance between different instance types also differ greatly from each other. This can be observed from the $k \cdot \theta$ parameter (the expected value) of Gamma distributions or the μ parameter of normal distributions in Tables 2 and 3.

TABLE 2
Parameters of I/O performance distributions

Instance type	Sequential I/O (Gamma)	Random I/O (Normal)
Small	$k = 129.3, \theta = 0.79$	$\mu = 150.3, \sigma = 50.0$
Medium	$k = 127.1, \theta = 0.80$	$\mu = 120.3, \sigma = 22.5$
Large	$k = 376.6, \theta = 0.28$	$\mu = 172.9, \sigma = 34.8$
xLarge	$k = 408.1, \theta = 0.26$	$\mu = 1034.0, \sigma = 146.4$

TABLE 3
Gamma distribution parameters on bandwidth between an instance and S3

Instance type	Uploading bandwidth	Downloading bandwidth
Small	$k = 233.2, \theta = 4.3$	$k = 31.9, \theta = 27.6$
Medium	$k = 12.4, \theta = 43.8$	$k = 22.7, \theta = 18.1$
Large	$k = 26.4, \theta = 18.5$	$k = 1.6, \theta = 20.2$
xLarge	$k = 21.3, \theta = 23.4$	$k = 5.4, \theta = 3.6$

4.3 Accuracy of Simulations

Before presenting the detailed simulation results, we evaluate the accuracy of our simulator by comparing the cumulative distribution of monetary cost and execution time of running workflows between simulations and real cloud environments. Note, because of performance dynamics, we study the distribution of multiple runs, instead of the results obtained from individual runs.

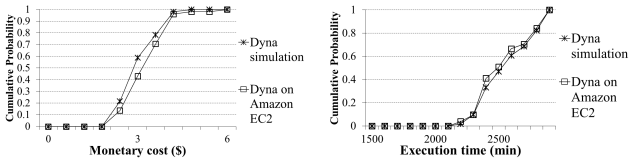
Figure 8 shows cumulative distributions of monetary cost and execution time obtained from simulation and real measurements from Amazon EC2. The workflows are II in hybrid structures under default settings. The number of runs on Amazon EC2 is 50 due to the budget constraint. The real-world experiments cost about $600. The result shows that our simulation is very close to the real cloud environment on capturing the distribution of the monetary cost and execution time. We have observed similar results for workloads with different structures and deadline settings. Thus, we can rely on our simulator to extensively evaluate the effectiveness of different scheduling algorithms in the remainder of this section.

4.4 Overall Comparison

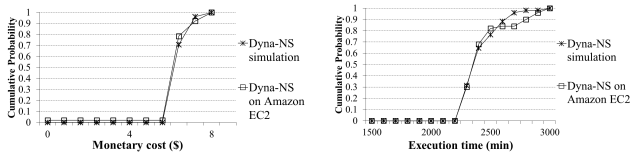
In this sub-section, we present the overall comparison results of Dyna with other scheduling methods with simulation.

We perform the experiments with different deadline guarantee requirements, ranging from 50% to 100%. This simulates the scenario that a user specifies different performance requirements on the workflow. Other parameters in the experiments use their default settings.

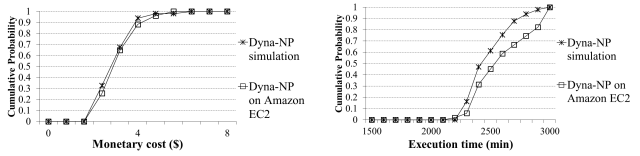
Figure 9 and 10 show the average monetary cost and average execution time of the static algorithm and three dynamic algorithms on pipeline and hybrid structured CI workflows. Since Dyna and Dyna-NS capture the performance dynamics by estimating the task execution time according to the probabilistic deadline guarantees, Dyna and Dyna-NS can have different results for the given deadline guarantee requirements. In contrast, the other algorithms have a single result on monetary cost and execution time. Note, we do not show the results for the Spot-only approach. The reason is, although the Spot-only approach may generate scheduling plans with smaller



(a) Monetary cost obtained by Dyna (b) Execution time obtained by Dyna

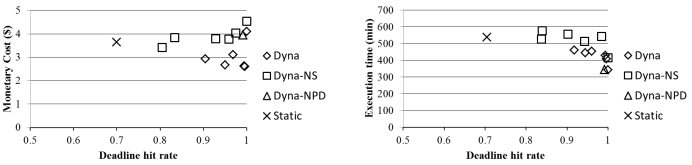


(c) Monetary cost obtained by Dyna-NS (d) Execution time obtained by Dyna-NS



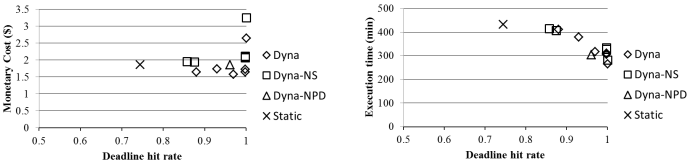
(e) Monetary cost obtained by Dyna-NPD (f) Execution time obtained by Dyna-NPD

Fig. 8. Cumulative distribution of monetary cost and execution time results obtained from simulation and real experiments on Amazon EC2 (II workflows)



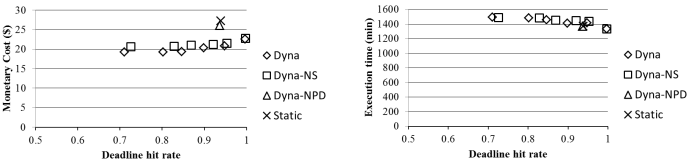
(a) Average monetary cost (b) Average execution time

Fig. 9. The average monetary cost and average execution time of compared algorithms on hybrid CI workflows under different required deadline hit rates



(a) Average monetary cost (b) Average execution time

Fig. 10. The average monetary cost and average execution time of compared algorithms on pipeline CI workflows under different required deadline hit rates



(a) Average monetary cost (b) Average execution time

Fig. 11. The average monetary cost and average execution time of compared algorithms on hybrid II workflows under different required deadline hit rates

monetary costs than Dyna, it can hardly achieve the deadline guarantees (usually with a probability smaller than 10%).

Overall, Dyna offers the optimized monetary cost and elapsed time for a given deadline guarantee requirement. This is in contrast with the static approaches. Moreover, with similar deadline hit rates, Dyna achieves a lower monetary cost as well as a shorter execution time. Particularly, we make the following observations.

First, the proposed optimization techniques in Dyna significantly reduce the monetary cost. With the hybrid instance configuration approach, Dyna gains great monetary reduction over the other compared algorithms. In both application types, the average monetary cost of Dyna is much less than Dyna-NS and Static, which do not adopt spot instances during scheduling. This demonstrates the effectiveness of the hybrid instance configuration approach on reducing monetary cost. The monetary cost reduction is $9-35\%$ and $7-29\%$ over Dyna-NS and Static, respectively. The runtime optimization technique is also contributing to reducing monetary cost. Dyna achieves a much smaller average monetary cost than Dyna-NPD, with the reduction $7-34\%$. The deadline refinement assigns the residual time of parent tasks to their child tasks. The child tasks have longer time to finish execution and thus are able to choose cheaper instances. Our detailed studies on hybrid CI workflows show that almost 33% of the instances used in Dyna are Small instances while Dyna-NPD uses almost no Small instances.

Second, Dyna is always able to hit the deadlines with higher probabilities on achieving the deadline guarantees under the same monetary cost. This demonstrates the effectiveness of the hybrid instance configuration technique on guaranteeing deadlines. With similar deadline hit rate, the average execution time of Dyna is close to or even shorter than the other compared algorithms. Interestingly, Dyna has an even shorter execution time than Dyna-NS, where Dyna-NS adopts on-demand instances only. This is because with the hybrid instance configuration, Dyna may choose a spot instance of better performance but with similar monetary cost to an on-demand instance. Our detailed studies show around 57% of the tasks in the pipeline workflows were assigned with better types of spot instances in Dyna than Dyna-NS. Thus, this result demonstrates that our hybrid instance configuration algorithm is able to generate cheaper and faster scheduling plans than on-demand instances only.

Third, the cost reduction of Dyna over the other algorithms is more significant on hybrid workflows than that on pipeline workflows. This is because in hybrid workflow, the tasks that are not in the critical path have higher probability of using spot instances for a smaller monetary cost. Our detailed experiments show that the ratio of using spot instances in hybrid CI workflows is 1.6 times higher than that in pipeline CI workflows.

After studying the comparison with CI workflows, we study the II workflows. Figure 11 shows the comparison for hybrid structured II workflows. Since II workflows have more performance dynamics, the runtime refinement optimization reduces the monetary cost of Dyna, with the reduction to be 18 - 30% over Static and 13 - 26% over Dyna-NPD. On the other hand,

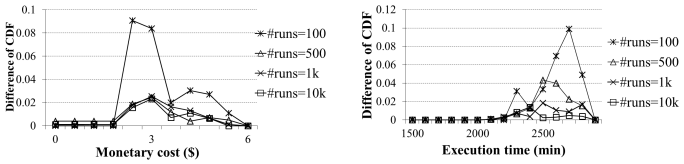


Fig. 12. Absolute difference of the cumulative distributions of execution time and monetary cost when number of Monte Carlo runs varies from 100 to $10k$ from that when the number of Monte Carlo runs is $100k$.

the expensive instance type mainly has more powerful CPUs than the cheap instance type. However, the I/O performance advantage of the expensive instance over the cheap instance is much smaller. The cost-efficiency benefits of choosing better instance types are mainly from the computation part. Thus the monetary cost reduction brought by the hybrid instance configuration approach in II workflows is not as significant as in CI applications, ranging from 1 - 8% over Dyna-NS. As for execution time, Dyna is almost the best for different deadline requirements.

4.5 Sensitivity analysis

We have conducted sensitivity analysis on different workflows. Since we observed similar results across workflows, we focus on hybrid II workflows in the following.

Monte Carlo runs. Figure 12 shows the absolute difference on the CDF of the monetary cost and execution time when the number of Monte Carlo runs ($\#runs$) is 100, 500, $1,000$ (or $1k$) and $10k$ compared with $\#runs = 100k$ (the largest number of runs that we have tested). The results shown are obtained from II workflows with more extensive performance dynamics. We have observed smaller differences in CI workflows. The difference decreases as we increase the number of Monte Carlo runs. Specifically, when $\#runs$ is larger than $1k$, the difference is consistently smaller than 2% in most. Thus, $1k$ runs are already sufficiently accurate for the simulation. On the other hand, the average simulation time of one workflow for $\#runs = 1k$ is around two minutes, which is only around 10% of $\#runs = 10k$. Thus, we choose $\#runs = 1k$ as a good balance between efficiency and accuracy of Monte Carlo simulations.

Performance dynamics. We study the impact of performance dynamics. In particular, we fix the I/O time at three times of the network time while varying the CPU time to be 80%, 60%, 40% and 20% of the total execution time. We consider first two cases as CI applications, denoted as CI-1 and CI-2; the later two as II applications, denoted as II-1 and II-2. Figure 13 compares the CDF of the monetary cost of Dyna and Dyna-NPD. As the performance dynamics become more extensive, Dyna always costs less than Dyna-NPD in all the four applications and the cost saved by Dyna from Dyna-NPD increases, from 17% to 37% on average. This observation shows the advantage of our runtime optimization techniques upon different amount of performance dynamics.

Deadline. Figure 14 shows the monetary cost of Dyna for meeting the varying deadlines with the probability of 90%. As

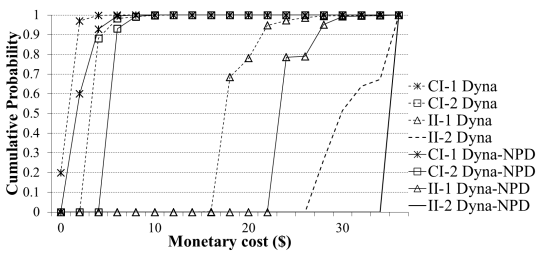


Fig. 13. Cumulative distributions of the monetary cost results obtained by Dyna and Dyna-NPD

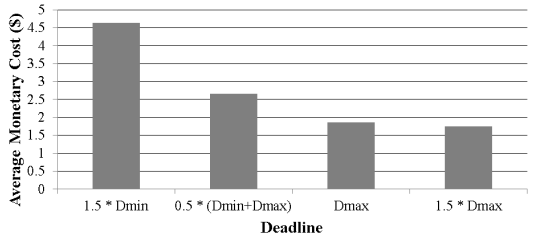
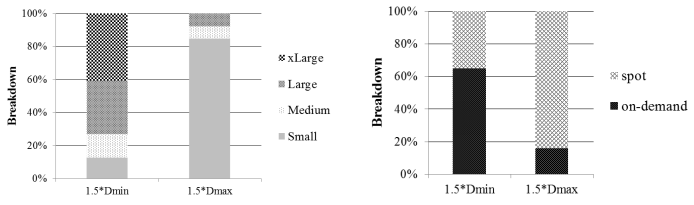


Fig. 14. Average monetary cost needed for meeting the varying deadlines by Dyna

the deadline gets loose, the monetary cost is decreased since more cheaper instances (either on-demand or spot instances) are selected for execution. We further break down the number of different types of on-demand and spot instances when the deadlines are $1.5 \times D_{min}$ and $1.5 \times D_{max}$ as shown in Figure 15. When the deadline is loose ($1.5 \times D_{max}$), Dyna mostly uses Small and Medium instances and over 80% of the instances with spot instance.

Task parallelism. We study the parallel rate parameter P in Amdahl's law. Figure 16 shows the monetary cost obtained by Dyna as parameter P varies. As P increases, the monetary cost is decreasing. This is because as P increases, it is more cost-efficient to choose high performance instances for execution since there are more parallel part in the workload that can benefit from the higher performance instances.

Runtime refinement. We study the impact of the threshold of runtime instance configuration refinement. Figure 17 shows the average monetary cost and execution time of Dyna on the hybrid II workflow when the $threshold\%$ value changes from 10% to 80%. The average monetary cost becomes stable after $threshold\% = 50\%$ and the average execution time gets stable



(a) Breakdown according to instance types (b) Breakdown according to on-demand/spot

Fig. 15. Breakdown of the instances adopted by Dyna when the deadlines are $1.5 \times D_{min}$ and $1.5 \times D_{max}$

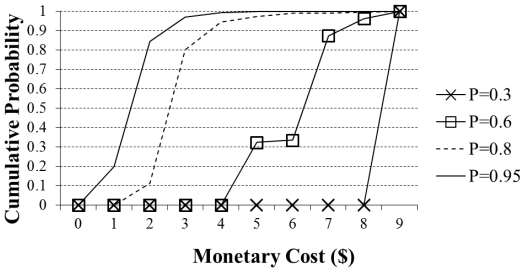


Fig. 16. Cumulative distribution of monetary cost obtained by Dyna when varying P from 0.3 to 0.95

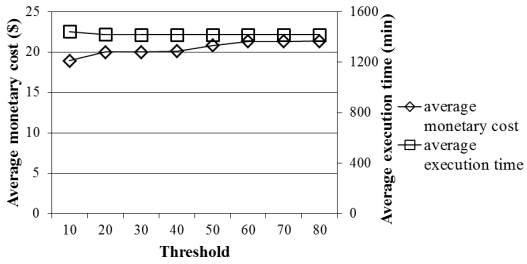


Fig. 17. Average monetary cost and execution time of Dyna on hybrid II workflows when varying $threshold$ from 10 to 80

after $threshold\% = 10\%$. Considering the overhead of runtime refinement, we set $threshold\% = 50\%$.

4.6 Summary and Discussions

In summary, the results clearly demonstrate the benefit of our probabilistic framework on guiding users to make decisions on monetary cost and performance choices in the dynamic cloud. Dyna is the most monetary cost efficient scheduling algorithm among the comparisons.

First, the I/O and network performance dynamics bring high performance unpredictability to the cloud. Compared with the conventional approaches, our framework offers two advantages. First, instead of offering a single point of performance and cost as in conventional approaches, our framework gives a probabilistic distribution on performance and cost and enables users to make wiser decision. The monetary cost varies significantly, showing the disadvantage of the conventional static approaches. Second, our deadline refinement in the framework is able to minimize the monetary cost while satisfying the probabilistic deadline guarantees. This is in contrast with the previous work with the assumption on static execution time.

Second, the spot instance is a cloud offering derived from economics. Cloud providers use spot prices to adjust the demand/supply of computing resources. As individual application owners, this is a valuable opportunity to reduce the monetary cost. Our study demonstrates how spot instances can be exploited without deadline violations. Our experiments demonstrate the cost reduction of $8 - 35\%$ over on-demand only methods.

5 Conclusion

Cloud is a dynamic environment with performance and price dynamics, which make the assumption of static task execution time and the QoS definition of static deadlines undesirable. In this paper, we propose the notion of probabilistic performance guarantees as QoS in dynamic cloud environments. We develop a probabilistic framework named Dyna for scheduling continuous workflows with the goal of minimizing the monetary cost while satisfying their probabilistic deadline guarantees. The framework embraces a series of static and dynamic optimizations, and allows new optimizations for monetary cost. We further develop runtime instance configuration refinement for performance dynamics and hybrid instance configuration of spot and on-demand instances for price dynamics. Our experimental results demonstrate that Dyna achieves much lower monetary cost than the state-of-the-art approaches, and also notably reduces the execution time. Moreover, instead of offering a single prediction of the current approaches, Dyna gives the probabilistic distributions on monetary cost and performance to help users to make more informative decisions upon cloud dynamics.

References

[1] Amazon Case Studies, "http://aws.amazon.com/solutions/case-studies/."

[2] Windows Azure Case Studies, "http://www.microsoft.com/azure/casestudies.mspx?HPCC."

[3] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. of 2011 Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, 2011, pp. 49:1–49:12.

[4] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis, "Schedule optimization for data processing flows on the cloud," in *Proc. of the 2011 ACM SIGMOD Int'l Conference on Management of data (SIGMOD '11)*, 2011, pp. 289–300.

[5] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[6] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[7] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *1st Int'l Conference on e-Science and Grid Computing*, 2005.

[8] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *the CoreGRID Workshop on Integrated research in Grid Computing*, 2005.

[9] R. Duan, R. Prodan, and T. Fahringer, "Performance and cost optimization for multiple large-scale grid workflow applications," in *Proc. of the 2007 ACM/IEEE conference on Supercomputing (SC'07)*, 2007, pp. 12:1–12:12.

[10] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for iaas clouds," in *Future Generation Computer Systems, Elsevier*, 2013, pp. 158–169.

[11] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," in *Future Generation Computer Systems, Elsevier*, 2011, pp. 1011–1026.

[12] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proceedings of the International Conference on Computer Communications*, ser. INFOCOM '12, 2012.

[13] F. Zhang, J. Cao, K. Hwang, and C. Wu, "Ordinal optimized scheduling of scientific workflows in elastic compute clouds," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11, 2011.

[14] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12, 2012, pp. 22:1–22:11.

[15] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proc. of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, 2010, pp. 6–6.

[16] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade-off management for scheduling parallel applications on utility grids," *Future Gener. Comput. Syst.*, vol. 26, October 2010.

[17] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *Integrated Research in Grid Computing*, 2007.

[18] A. M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Proc. of the IEEE 2nd Int'l Conference on Cloud Computing Technology and Science (CloudCom '10)*, 2010, pp. 351–359.

[19] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on amazon cloud spot instances," *IEEE Transactions on Services Computing*, 2011.

[20] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Proc. of the 2010 IEEE Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*, 2010, pp. 257–266.

[21] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proc. of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, 2010, pp. 7–7.

[22] M. Mattess, C. Vecchiola, and R. Buyya, "Managing peak loads by leasing cloud infrastructure services from a spot market," in *Proc. of the IEEE 12th Int'l Conference on High Performance Computing and Communications (HPCC '10)*, 2010, pp. 180–188.

[23] M. Mazzucco and M. Dumas, "Achieving performance and availability guarantees with spot instances," in *Proc. of the 2011 IEEE Int'l Conference on High Performance Computing and Communications (HPCC '11)*, 2011, pp. 296–303.

[24] W. Voorsluys and R. Buyya, "Reliable provisioning of spot instances for compute-intensive applications," pp. 542–549, 2012.

[25] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," in *Proc. of the 2011 IEEE Third Int'l Conference on Cloud Computing Technology and Science (CloudCom '11)*, 2011, pp. 304–311.

[26] B. Javadi, R. Thulasiram, and R. Buyya, "Statistical modeling of spot instance prices in public cloud environments," in *IEEE/ACM Int'l Conference on Utility and Cloud Computing*, 2011.

[27] C. Ernemann, V. Hamscher, and R. Yahyapour, "Economic scheduling in grid computing," in *Revised Papers from the 8th Int'l Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, 2002, pp. 128–152.

[28] J. Gray and G. Graefe, "The five-minute rule ten years later, and other computer storage rules of thumb," *SIGMOD Rec.*, vol. 26, no. 4, pp. 63–68, Dec. 1997.

[29] M. Adler, J. yi Cai, J. K. Shapiro, and D. F. Towsley, "Estimation of congestion price using probabilistic packet marking," in *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, 2003, pp. 2068–2078.

[30] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proc. of the 19th ACM Int'l Symposium on High Performance Distributed Computing (HPDC '10)*, 2010, pp. 304–307.

[31] B. Bouterse and H. Perros, "Scheduling cloud capacity for time-varying customer demand," in *CloudNet*, 2012.

[32] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proc. of the 4th Int'l Conference on Autonomic Computing (ICAC '07)*, 2007.

[33] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. of the 31st Int'l Conference on Dsitributed Computing Systems (ICDCS '11)*, 2011.

[34] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proc. of 2012 Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*, 2012.

[35] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. of the IEEE 4th Int'l Conference on Cloud Computing (CLOUD '11)*, 2011, pp. 500–507.

[36] J. Yang, J. Qiu, and Y. Li, "A profile-based approach to just-in-time scalability for cloud applications," in *Proc. of the 2009 IEEE Int'l Conference on Cloud Computing (CLOUD '09)*, 2009, pp. 9–16.

[37] M. S. Rehman and M. F. Sakr, "Initial findings for provisioning variation in cloud computing," in *Proc. of the IEEE 2nd Int'l Conference on Cloud Computing Technology and Science (CloudCom '10)*, 2010, pp. 473–479.
[38] S. Suakanto, S. H. Supangkat, Suhardi, and R. Saragih, "Performance measurement of cloud computing services," *CoRR*, vol. abs/1205.1622, 2012.
[39] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Proc. of the 29th conference on Information communications (INFOCOM'10)*, 2010, pp. 1163–1171.
[40] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Proc. of the 2010 IEEE 3rd Int'l Conference on Cloud Computing (CLOUD '10)*, 2010, pp. 51–58.
[41] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, Y. Cao, and L. Liu, "Who is your neighbor: Net i/o performance interference in virtualized clouds," *IEEE Transactions on Services Computing*, vol. 99, 2012.
[42] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. of the ACM SIGCOMM 2011 conference*, ser. SIGCOMM '11, 2011, pp. 242–253.
[43] M. Hovestadt, O. Kao, A. Kliem, and D. Warneke, "Evaluating adaptive compression to mitigate the effects of shared I/O in clouds," in *Proc. of the 2011 IEEE Int'l Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '11)*, May 2011, pp. 1042–1051.
[44] X. Lin, Y. Mao, F. Li, and R. Ricci, "Towards fair sharing of block storage in a multi-tenant cloud," in *Proc. of the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, Jun. 2012.
[45] R. C. Chiang and H. H. Huang, "Tracon: interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. of 2011 Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, 2011, pp. 47:1–47:12.
[46] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? performance benchmarking for infrastructure-as-a-service offerings," in *Proc. of the 2011 IEEE 4th Int'l Conference on Cloud Computing (CLOUD '11)*, 2011, pp. 484–491.
[47] CloudSleuth, https://www.cloudsleuth.net/web/guest/home/.
[48] CloudHarmony, http://www.cloudharmony.com.
[49] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, O. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of Cloud Computing and Its Applications*, 2008.
[50] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proc. of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*, 2010, pp. 1–14.
[51] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *The 8th IEEE Int'l Conference on eScience 2012 (eScience 2012)*, 2012.
[52] CondorTeam, "DAGMan", "http://cs.wisc.edu/condor/dagman."
[53] M. L. et al., "Condor: A hunter of idle workstations," in *8th International Conference on Distributed Computing Systems*, 1988, pp. 104–111.
[54] "Iperf," http://iperf.sourceforge.net.
[55] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proc. of the 2012 IEEE 5th Int'l Conference on Cloud Computing (CLOUD '12)*, 2012, pp. 423–430.
[56] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," in *Future Generation Computer Systems, Elsevier*, 2013, pp. 682–692.
[57] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, and K. Vahi, "Characterization of scientific workflows," in *Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS'08)*, 2008.