Transformation-Based Monetary Cost Optimizations for Workflows in the Cloud

Amelie Chi Zhou and Bingsheng He

Abstract—Recently, performance and monetary cost optimizations for workflows from various applications in the cloud have become a hot research topic. However, we find that most existing studies adopt ad hoc optimization strategies, which fail to capture the key optimization opportunities for different workloads and cloud offerings (e.g., virtual machines with different prices). This paper proposes *ToF*, a general transformation-based optimization framework for workflows in the cloud. Specifically, ToF formulates six basic workflow transformation operations. An arbitrary performance and cost optimization process can be represented as a transformation plan (i.e., a sequence of basic transformation operations). All transformations form a huge optimization space. We further develop a cost model guided planner to efficiently find the optimized transformation for a predefined goal (e.g., minimizing the monetary cost with a given performance requirement). We develop ToF on real cloud environments including Amazon EC2 and Rackspace. Our experimental results demonstrate the effectiveness of ToF in optimizing the performance and cost in comparison with other existing approaches.

Index Terms—Cloud computing, monetary cost optimizations, workflows

1 Introduction

Cloud computing has become an important computing infrastructure for many applications. Its pay-as-you-go pricing scheme has attracted many application owners to either deploy their applications in the cloud or extend their home clusters to cloud when the demand is too high. In recent years, we have witnessed many scientific applications partially or entirely shifting from traditional computing platforms (e.g., grid) to the cloud [1], [2]. Due to the pay-as-you-go computational behavior, the (monetary) cost has become an important metric for system optimizations [3]. Basically, a workflow management system should balance the cost and performance. Thus, performance and (monetary) cost optimizations have recently become a hot research topic for workflows in the cloud. A lot of scheduling and optimization approaches have been developed (e.g., [4], [5], [6], [7], [8], [9]).

Despite of a lot of research efforts in this area, performance and cost optimizations of workflows in the cloud are still a non-trivial task, because of the following complicated and inter-connected factors. First, users have different requirements on performance and cost. Some existing studies [4], [5] have focused on minimizing the cost while satisfying the performance requirement, some are aiming to optimize performance for a given budget [10] and others are considering the tradeoff between performance and monetary cost [7], [8], [11]. Second, different cloud offerings result in significantly different cost structures for running the workflow. Even from the same cloud provider, there are multiple types of virtual machines (or instances) with different prices and computing capabilities. Third, workflows have very complicated structures and different computation/IO characteristics, as observed in the previous studies [12]. All those factors call for a general and effective approach for performance and cost optimizations.

We review the existing studies and find that most of them are ad hoc in the sense that they fail to capture the optimization opportunities in different user requirements, cloud offerings and workflows. For example, Kllapi et al. [7] consider only a single instance type. However, previous studies [13], [14] have shown that carefully selecting instance types is important for the overall cost. Mao and Humphrey [4] focus on minimizing the cost while satisfying the performance requirement of individual workflows, and simply use a *fixed* sequence of workflow transformation operations. The fixed sequence can be effective for some of the workflows and cloud offerings, but ineffective for others. All those studies potentially lose optimization opportunities for performance and cost.

To address the limitations of current approaches, we believe that an extensible workflow framework is essential for different and even evolving user requirements, cloud offering and workflows. We have identified three design principles. First, the framework should have an extensible design on the workflow optimizations, which adapts to different cloud offerings and workflow structures. Second, the framework should have a general optimization process for different requirements on performance and cost constraints. Last, the framework should be light weight for online decision making.

With these three design principles in mind, we propose *ToF*, a transformation-based optimization framework for optimizing the performance and cost of workflows in the cloud. A workflow is generally modeled as a directed acyclic graph (DAG) of tasks. ToF guides the scheduling of each task in the workflow, including which instance to assign to and when to start execution. When a workflow is

• *The authors are with the School of Computer Engineering, Nanyang Technological University, Singapore.*

Manuscript received 11 July 2013; revised 11 Nov. 2013; accepted 18 Dec. 2013; date of publication 8 Jan. 2014; date of current version 30 Apr. 2014.
Recommended for acceptance by C. Rong.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2013.2297928

TABLE 1
Prices($/hr) and Capabilities of Four Types of On-Demand Instances in Amazon EC2, US East Region

Type	ECU	Mem(GB)	Disk(GB)	Net.	price
m1.small	1	1.7	1×160	Low	0.06
m1.medium	2	3.75	1×410	Moderate	0.12
m1.large	4	7.5	2×420	Moderate	0.24
m1.xlarge	8	15	2×840	High	0.48

TABLE 2
Prices($/hr) and Capabilities of Four Instance Types in Rackspace

RAM(GB)	vCPU	Disk(GB)	Net.(Pub./Int.) (Mbps)	price
0.5	1	20	20/40	0.022
1	1	40	30/60	0.06
2	2	80	60/120	0.12
4	2	160	100/200	0.24

submitted to the cloud, each task is initially assigned to a certain type of instance for execution. Based on the initial instance assignment, we perform transformations on the DAG. We categorize the transformation operations into two kinds, namely *main schemes* and *auxiliary schemes*. The main schemes reduce monetary cost while auxiliary schemes transform workflows into a DAG that is suitable for main schemes to perform cost reduction. Specifically, we have formulated six basic workflow transformation operations (*Merge*, *Demote*, *Split*, *Promote*, *Move* and *Co-scheduling*). The former two are categorized as main schemes and the latter four are auxiliary schemes.

With the transformation operations, we model an arbitrary performance and cost optimization process to be a transformation plan (i.e., a sequence of basic transformation operations). All transformations form a huge optimization space. We further develop a cost model guided planner to help users to efficiently and effectively choose the cost-effective transformation. The cost model estimates the change in both monetary cost and execution time introduced by each transformation operation. Moreover, we develop heuristics (e.g., iteratively choosing the cost-effective main scheme and auxiliary scheme) to reduce the runtime overhead of the optimization process. Note that the initial instance assignment, transformation operations and planner are extensible for different implementations.

We implement the transformation operations and the planner in ToF and evaluate it on Amazon EC2 and Rackspace. We have conducted experiments on the performance of ToF using two real-world workflow structures (Montage [15] and Ligo [16]). We evaluate the individual performance and cost impact of transformation and optimizations in ToF and compare with the state-of-the-art auto-scaling method [4]. ToF reduces the monetary cost by 30 percent in comparison with the Auto-scaling method, with similar optimization overhead. On the other hand, ToF outperforms auto-scaling with 21 percent on the performance given the budget constraint.

This paper makes the following two key contributions. First, we propose a transformation-based workflow optimization system to address the performance and monetary cost optimizations in the cloud. Second, we develop and deploy the workflow optimization system in real cloud environments, and demonstrate its effectiveness and efficiency with extensive experiments. To the best of our knowledge, this work is the first of its kind in developing a general optimization engine for minimizing the monetary cost of running workflows in the cloud.

The rest of this paper is organized as follows. We introduce the background on cloud offerings and application scenarios, and review the related work in Section 2. Section 3 gives a system overview. We present our workflow transformation operations in Section 4, followed by the cost model guided planner in Section 5. We present the experimental results in Section 6 and conclude the paper in Section 7.

2 Preliminary and Related Work

In this section, we first describe cloud computing environments mainly from users' perspective. Next, we present the application scenario and review the related work.

2.1 Cloud Environments

Cloud providers lease computing resources in the form of VMs (or instances). Typically, cloud providers offer multiple types of instances with different capabilities such as CPU speed, RAM size, I/O speed and network bandwidth to satisfy different application demands. Different instance types are charged with different prices. Tables 1 and 2 show the prices and capabilities of four on-demand instance types offered by Amazon EC2 and Rackspace, respectively. Amazon EC2 mainly charges according to the CPU, whereas Rackspace mainly on the RAM size. Both cloud providers adopt the instance hour billing model, whereby partial instance hour usage is rounded up to one hour. Each instance has a non-ignorable instance acquisition time. For simplicity, we assume the acquisition time is a constant, lag. In this paper, we consider a single cloud provider with I instance types, with assigned IDs to the instance types in the increasing order of their prices. IDs are consecutive integers, from one to I.

In addition to the local storage resource on instances, cloud providers also offer persistent storage (such as Amazon S3). When instances are released, the data stored in the local storage will be lost and those on persistent storage are kept. With the persistent storage, users are able to store a snapshot of the current execution and restart the execution later. We call this process "checkpointing".

Diversifying cloud offerings require more general performance and cost optimization processes, instead of specific or ad hoc strategies. Previous studies have studied the impact of different cloud offerings on the same workload: 1) the performance varies significantly among different cloud providers [17], 2) the performance and cost vary significantly with different instance types and numbers used from the same provider [13]. Moreover, cloud itself also evolves. On the one hand, cloud prices, as an economic concept, evolve through the time. Amazon EC2 adjusted the price for several times in the past five years. On the other hand, hardware and software systems evolve, even without users' notice.

2.2 Application Scenarios

In this study, we consider the application scenario of users purchasing instances from a public cloud provider to execute their workflows. Due to the pay-as-you-go nature of cloud computing, users need to pay to the public cloud provider. In this scenario, users (e.g., scientists and officials) submit their workflows to our workflow management system, ToF, with predefined goals on the monetary cost and performance. We assume users are independent with each other, and there are no dependencies among users.

Users usually have different requirements on performance and monetary cost [7]. One may want to minimize the monetary cost of workflows while satisfying the predefined deadlines. Our system assumes the users specify the deadline for the entire workflow. This assumption has been adopted in many existing workflow systems [4], [5], [8]. We do not require users to specify the deadline for each task. The deadline constraint of each task is automatically determined by ToF with an existing deadline assignment algorithm. We use deadlines to represent the QoS requirements. Note, the deadline is a soft deadline, not a hard deadline. The other may want to minimize the execution time while satisfying the predefined budget. More advanced constraints like skyline are also possible, and we refer readers to the previous paper [7] for more details. In order to support flexible settings on the performance and cost constraints, we need a general workflow optimization approach. Therefore, the goal of this paper is to develop a general and effective optimization framework for workflows in the cloud, which is able to address different cloud offerings and different user requirements.

2.3 Cost Optimization for Workflows

Performance and cost optimizations are a classic research topic for decades. Many scheduling and provisioning algorithms have been proposed leveraging market-based techniques [11], rule-based techniques [8] and models [6] for cost, performance and other optimization objectives. Different applications require different performance and cost optimizations. Many relevant performance and cost optimizations can be found in databases [18], Internet [19], distributed systems [20] and so on. Performance and cost optimizations for workflows have been well studied on grid [21], cloud [11] and heterogenous computing environments [22]. Specifically, we review the most relevant workflow optimizations in the grid and in the cloud.

The workflow scheduling problem has been widely studied in the grid environment [23]. Usually, execution time is considered as the most important optimization criterion. There have been many studies focusing on minimizing makespan [24], [25]. Under the grid market concept, the cost minimization problem has been studied in a good number of studies (e.g., [21], [26], [27], [28], [29]). Some of them performed dual-objective optimizations [26].

Cloud, as a market, is different from grid [30], [31] in that the total ownership cost of running a workflow is considered a much more important optimization criterion. Compared with grid, cloud has unique features: billing hour pricing, multiple instance types, elasticity, non-ignorable instance acquisition time and so on. Thus, the techniques invented in the grid environment need to be revisited, and new algorithms and mechanisms should be developed for performance and cost optimizations in the cloud.

Performance and cost optimization techniques have been developed for a single cloud provider and multiple cloud providers. There have been much more research studies on a single cloud than on multiple clouds.

On a single cloud, the research can be generally divided into three categories. The first category is auto-scaling (e.g., [4], [5]). The studies in this category are usually based on heuristics, applying a fixed sequence of transformations on the workflow. Mao and Humphrey [4] performed the following operations one by one: merge ("task bundling" in their paper), deadline assignment and allocating cost-efficient machines, scaling and consolidation. This fixed sequence of workflow transformations is ad hoc in the sense that there are many possible sequences of transformations. The author have not justify or prove the applicability of this fixed sequence. As demonstrated in our experiments, workflows can have very different transformations for minimizing the monetary cost. The second category is dynamic provisioning with prediction models (e.g., [6], [32], [33]). Building prediction models is orthogonal to this study. The last category is workflow scheduling [7], [8] with performance and cost constraints. Kllapi et al. consider the tradeoff between completion time and money cost for data processing flows [7]. Malawski et al. [8] proposed task scheduling and provisioning methods for grouped scientific workflows called ensembles, considering both budget and deadline constraints. This paper belongs to this category, but goes beyond the existing studies in two major aspects. First, we formally formulate the performance and cost optimization process with workflow transformation models. Second, we develop a cost model guided optimization framework to optimally utilize the transformation operations.

Considering multiple cloud providers, Fard et al. [11] introduced a pricing model and a truthful mechanism for scheduling workflows given performance and cost optimization objectives. Lucas Simarro et al. [34] considered the problem of a cloud broker in the context of multiple cloud providers with dynamic prices. We refer readers to a recent survey [35] for inter-cloud optimizations. This paper focuses on a single cloud provider, which is the common case in practice.

Recently, researchers have also paid attention to energy consumption of workflows. Ben Othman et al. [36] considered the optimization goal of energy consumption and performance. Zhu et al. [12] developed power-aware consolidation to reduce the energy consumption while keeping the performance constraint. Consolidation is similar to our "co-scheduling" transformation. Since energy consumption behaviors are significantly different from monetary cost behaviors, energy-aware optimizations may not be applicable to cost-aware optimizations, and vice versa. It is our future work to extend our framework to reduce the energy consumption.

3 Overview of ToF

To support workflow management in the cloud, we have adopted DAGMan [37] to manage task dependencies and

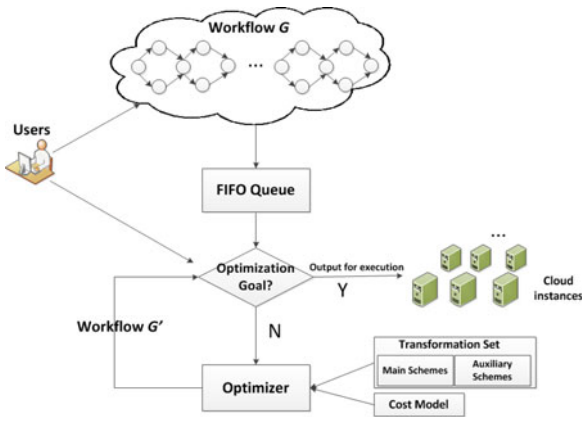


Fig. 1. System overview of ToF.

added Condor [38] to manage task execution and instance acquisition on real cloud providers. We consider Amazon EC2 and Rackspace in this study.

ToF has two major components for performance and cost optimizations: transformation model and planner. The transformation model defines the set of transformation operations for a workflow, and the planner performs the transformation on the workflow according to the cost model. Fig. 1 shows the application model of ToF.

The planner is ran periodically. The period is denoted as the *plan period*. The workflows submitted during the epoch are temporarily stored in a queue, and then the planner performs optimization for all the workflows in the queue at the beginning of a plan period. This batch processing has two major benefits: first, maximizing the instance sharing and reuse so that the cost is reduced; second, reducing the planner overhead.

In each plan period, after all the workflows in the queue have been optimized, they are submitted to the cloud with their execution plan for execution. The execution plan is a set of instance requests. Each request includes the instance type, the starting and ending time of the requested instance and the task(s) scheduled to the instance.

To enable instance reuse at runtime, we maintain a pool of running instances, organized in lists according to different instance types. During runtime, instances in the pool which reach hourly running time and are currently idle will be turned off. An instance request is processed at the instance starting time by first looking into the instance pool for an idle instance of the requested type. If such an instance is found, it will be selected for workflow execution. Otherwise, a new instance of the requested type is acquired from the cloud provider. Thus, the instances started during workflows execution can be properly reused and their utilizations are improved. Additionally, if we can reuse the instances, the instance acquisition time is eliminated.

We present the details of the transformation and optimization components in the next two sections.

4 Workflow Transformation

In this section, we present the details on the transformation model. The details on planner is presented in Section 5.

4.1 Transformation-Based Optimizations

We define the following terminologies for transformation-based optimizations.

Workflow. Generally, a workflow structure can be represented by a DAG $G = (V, E)$, where V is the set of vertices representing the tasks in the workflow and E is the set of edges indicating the data dependencies between tasks.

Initial instance assignment. In each workflow, a task is initially assigned to an instance type for execution. According to the assigned instance type, each task execution is represented by the earliest start time, the latest end time and an instance. Our algorithm does not assume any specific initial instance assignment. We present a number of heuristic-based methods for initial instance assignment in Section 5. Due to the data dependencies among the tasks, the assigned instances themselves also form a DAG. We call this DAG *instance assignment graph* (denoted as $\mathcal{G}$), which has the same structure as G.

Transformation operation. We define a transformation operation to be a structural change in the instance assignment DAG, denoted as $o(\mathcal{G}) \rightarrow \mathcal{G}'$, where $\mathcal{G}$ and $\mathcal{G}'$ are the instance assignment DAG before and after the transformation, respectively.

We further define the basic (or *atomic*) transformation operation to be non-divisible. The six transformation operations that we will define later are all atomic transformation operations. To make our presentation clearer, our notation of an atomic transformation is simply on the most relevant instances (i.e., vertices of the instance DAG). Note, other vertices in the instance DAG need to be adjusted accordingly after the transformation. For example, we represent promoting an instance from type i to type j to be $\mathcal{P}(V_i(t_0, t_1)) \rightarrow V_j(t_2, t_3)$, where $i < j$, assuming the task runs on V_i (an instance of type i) from t_0 to t_1 before promotion; the task runs on V_j (an instance of type j) from t_2 to t_3 after promotion. The formulation of other transformation operations can be founded in Table 3.

Since each instance is associated with one or more tasks running on it, we use the expressions of "transformation on an instance" and "transformation on a task" alternatively in the rest of this paper.

Optimization sequence. We define the *transformation set* to be the set of atomic transformation operations defined in the system. Given a transformation set S, we define an optimization sequence (or a transformation plan) to be a sequence of atomic transformation operations applied to the instance DAG. Particularly, an optimization sequence consists of n atomic operations: $o_1, o_2, \ldots, o_n$ $(n \geq 1)$. o_i $(1 \leq i \leq n)$ belongs to one atomic operation defined in S.

4.2 Transformation Set

In current ToF, we have developed six basic transformation operations, namely *Merge*, *Split*, *Promote*, *Demote*, *Move* and *Co-scheduling*. These basic transformations are simple and lightweight. Moreover, they can capture the current cloud features considered in this paper. They are the most common operations and widely applicable to workflow structures. For example, the operations of all the comparison algorithms used in the experiments can be represented using those transformations. However, we do not claim

TABLE 3
Details of the Six Transformation Operations

Name	Category	Description	Formulation
Merge	Main	Merge multiple tasks to the same instance to fully utilize partial hour.	$\mathcal{M}(V_i(t_0,t_1),V_i(t_2,t_3)) \rightarrow V_i(t_0,t_3)$
Demote	Main	Assign a task to a cheaper instance type.	$\mathcal{D}(V_i(t_0,t_1)) \rightarrow V_j(t_2,t_3)$, where $i > j$
Move	Auxiliary	Delay a task to execute later.	$\mathcal{V}(V_i(t_0,t_1)) \rightarrow V_i(t_2,t_3)$, where $t_3 = t_2 + (t_1 - t_0)$
Promote	Auxiliary	Assign a task to a better instance type.	$\mathcal{P}(V_i(t_0,t_1)) \rightarrow V_j(t_2,t_3)$, where $i < j$
Split	Auxiliary	Stop a running task at some checkpoint and restart it later.	$\mathcal{S}(V_i(t_0,t_1)) \rightarrow V_{i1}(t_0,t_2),V_{i2}(t_3,t_4)$
Co-scheduling	Auxiliary	Assign two or more tasks to the same instance for execution.	$\mathcal{C}(V_i(t_0,t_1),V_i(t_2,t_3)) \rightarrow V_i(min(t_0,t_2),max(t_1,t_3))$

TFEET- The formulation $V_i(t_0,t_1)$ stands for an instance of type i and the task on this instance starts at t_0 while ends at t_1.

they form a complete set. Users can extend more transformation operations into the transformation set. Adding a transformation operation requires the modifications including adding the cost model and transformation implementation on the instance DAG.

Based on their capabilities in reducing monetary cost, we categorize the transformation operations into two kinds, namely *main* schemes and *auxiliary* schemes. A main scheme can reduce the monetary cost while an auxiliary scheme simply transforms the workflows so that the transformed workflow is suitable for main schemes to reduce cost. By definition, Merge and Demote are main schemes, and the other four operations are auxiliary schemes.

Table 3 summarizes the definition and categorization for the six operations. In the remainder of this section, we describe their implementation details.

Some examples of transformation are illustrated in Fig. 2. We illustrate the transformation operations with an *instance-time* chart, where the *x*-axis represents time and *y*-axis represents the instance. An instance-time chart is similar to Gantt chart, with the box width as the execution time and with dependencies between boxes. The height of the boxes stand for prices of instances. During the transformation, we maintain the structural dependency among tasks even after transformations. Special cares are given to Merge and Split. We present the details in the next section.

4.2.1 Main Schemes

Merge ($\mathcal{M}$). The Merge operation performs on two vertices when they are assigned to the same type of instances and one vertex is assigned to start (shortly) after the completion of the other one. Through merge, the two tasks are assigned to the same instance, and the two instance nodes in the instance DAG are merged to form a super-node, which maintains the hierarchical relationship among the merged nodes and also the structural dependency with other nodes in the DAG. This super-node can be treated the same as the other instance nodes for further transformations. This operation increases the usage of partial hours of instances. Take Fig. 2, for example. Task A_1 and A_2 are assigned to the same instance type but different instances. In our implementation, after the Merge operation, the two tasks are running on the

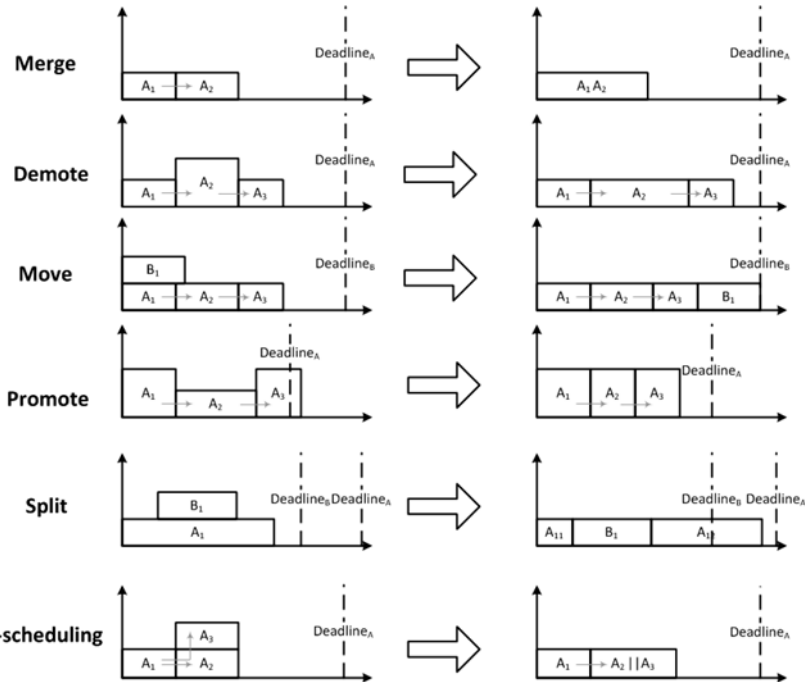


Fig. 2. Use cases of the six transformation operations shown in the instance-time chart.

same instance. There is no need to adjust other vertices in the instance assignment DAG.

Demote ($\mathcal{D}$). Demote operation performs on a single vertex by demoting the execution to a cheaper instance. That can reduce the cost at the risk of a longer execution time. In our implementation, we not only need to change the demoted vertex, but also need to delay the starting time of all vertices that have dependencies on the demoted vertex. In Fig. 2, task A_2 is demoted from an instance of type i to a cheaper instance of type j which is the same instance type as task A_1 and A_3. Demotion may have chained effects with other transformation operations. In this example, besides the cost saved by demotion, merging task A_1, A_2 and A_3 can further save the cost.

4.2.2 Auxiliary Schemes

Move ($\mathcal{V}$). The Move operation moves a task afterward to the end of another task in order to make chances for main schemes to further reduce cost. Note, the instance DAG uses the earliest start time and thus there is no "move forward" transformation. In our implementation, we not only need to change the moved vertex, but also need to delay the starting time of all vertices that are dependent on the moved vertex.

A key decision for the Move operation is where to move the task. Generally we have two cases: move a task to another task so that both tasks are assigned to the same instance type, or to a task with a different instance type. In the former case, we expect a Merge operation after the Move operation. In the latter case, a Demote and a Merge operation need to be performed to reduce cost. Our cost model is used to guide this decision (Section 5). Fig. 2 shows an example of the first case, where task B_1 of job B is moved to the end of task A_3. After the Move operation, we can merge A_3 and B_1.

Promote ($\mathcal{P}$). The Promote operation is a dual operation for the Demote operation. It promotes a task to a better instance at the benefit of reducing the execution time. The implementation is the same as the Demote operation.

There are mainly two incentives to perform the Promote operation. The first is to meet deadline requirement as shown in Fig. 2. Second, although promotion itself is not cost-effective, it creates chances for main schemes such as the Merge operation to perform. For example, in Fig. 2, we promote task A_2 from instance type i to a better instance type j. After the Promote operation, A_1, A_2 and A_3 are all assigned to the same instance type and thus can be merged to fully utilize instance partial hours.

Split ($\mathcal{S}$). The Split operation splits a task into two, which is equivalent to suspending a running task so that we can make room for a more urgent task which is assigned to the same instance type (with a Merge operation). With the checkpointing technique, the suspended task can restart after the completion of the urgent task. The Split operation causes the checkpointing overhead. The Split operation splits a node in the instance DAG into two sub-nodes, at the time point when the urgent task starts. We maintain their structural dependency with other nodes in the DAG. The sub-nodes can be treated the same as the other instance nodes for further transformations.

In the example of Split in Fig. 2, due to the deadline constraint, the Move operation is not applicable. We can split task A_1 at the start time of B_1 and let B_1 execute first. Then we restart A_1 at the end time of B_1. The start time of the instance containing A_{11} is the same as the instance containing A_1 and its end time is the same as the start time of task B_1. The start time of the instance containing A_{12} is the same as the end time of task B_1 and its end time equals to the sum of its start time and the execution time of A_{12}. If the checkpointing and restarting overhead is small, the benefit of the Merge operation is compensated.

Co-scheduling ($\mathcal{C}$). Some instance types (e.g., m1.xlarge has eight virtual processing units on Amazon) can afford multiple tasks running at the same time. The Co-scheduling operation is to run multiple tasks which have similar start and end time as well as similar leftover time before deadlines on the same instance. The number of tasks is set according to the instance capability and performance degradation. We use the existing model [12] to estimate the performance degradation.

Fig. 2 shows a case of co-scheduling where task A_2 and A_3 have the same start and end time and because they belong to the same job and have similar leftover time before deadline. A Co-scheduling operation can be applied to A_2 and A_3. We update the end time of the co-scheduled tasks according to the performance degradation estimation and the start time of other tasks which are dependent on A_2 and A_3 accordingly.

5 ToF Planner

Having formulated the transformation set, we introduce our design and implementation of the cost-model based planner in ToF. ToF is a cost model guided greedy approach to search for the optimal solution and is thus an approximation approach. The planner guides ToF to find the optimized transformation sequence for workflows. The selection of each transformation operation in the transformation sequence is guided by a cost model, which estimates the monetary cost and execution time changes introduced by individually applying each transformation operation in the transformation set.

5.1 Planner

There are a number of technical challenges in designing and implementing the planner. First, the transformation operations are composable. The order of applying transformation operations also matters for performance and cost optimizations. The searching space for an optimal transformation sequence is huge. Second, the optimization is an online process and should be lightweight. We should find a good balance between the quality of the transformation sequence and the runtime overhead of the planner. Due to the huge space, a thorough exploration of the optimization space is impractical. Third, the planner should be able to handle different tradeoffs on the monetary cost and performance goals.

To address these challenges, we have the following designs on the planner. First, the planner is ran periodically, as introduced in Section 3.

Second, the planner has two heuristics to reduce the searching space. First of all, it uses main schemes and

TABLE 4
All the Rules Used on Monetary Cost Optimization

Rule	Condition	Action
1	Merge can reduce cost	Perform Merge
2	Demote can reduce cost while satisfies deadline constraint	Perform Demote
3	Move can reduce cost while satisfies deadline constraint	Perform Move
4	Promote can reduce cost	Perform Promote
5	Split can reduce cost while satisfies deadline constraint	Perform Split
6	Co-scheduling can reduce cost while satisfies deadline constraint	Perform Co-scheduling

auxiliary schemes alternatively during the optimization process. Second of all, the planner uses the cost model to prune the "unpromising" transformations.

Third, the planner is rule-based. The rule is defined to consist of two components: condition and action, where the condition is usually defined based on the cost and performance optimization goal (e.g., the estimated monetary cost can be reduced by 20 percent) and the action consists of transformations on the workflow. As an example, we show all the rules used on monetary cost optimizations in our experiments in Table 4. The set of rules in ToF affects the effectiveness of optimizations. Users can define their own set of rules used in ToF, to reflect their goals on the tradeoff between monetary cost and performance. When applying a transformation operation such as Move, Demote and Split, we pretend to apply the transformation, and check whether the transformed workflow violates the given earliest start and latest end time constraints in order to satisfy the deadline requirement. If the constraints are violated, we will not perform the operation. That means, this study focuses on the goal of minimizing the monetary cost while satisfying the deadline. We briefly evaluate the goal of minimizing the execution time given a budget.

Algorithm 1 illustrates the overall optimization process of ToF in one plan period, which has implemented all the three above-mentioned designs. Initially, each task in the workflow is assigned with an instance type determined by an instance assignment heuristic. We present and discuss three initial instance assignment heuristics later in this section. The rules are based on Table 4. It iteratively selects the cost-effective transformation. In one iteration, we first estimate the (potential) cost reduction of each operation which satisfies the deadline constraint. The cost model is described in Section 5.2. Second, we select and perform the operation with the most cost reduction. All selected transformations form the optimization sequence.

Fig. 3 shows an example of applying Algorithm 1 to a simple structured workflow with three tasks. The deadline of the workflow is 120 minutes and the execution time of Tasks 0, 1 and 2 on the assigned instance types are 30, 30 and 40 minutes respectively. In the first iteration, we first check the operations in main schemes and find that no one can reduce cost. We then check the operations in auxiliary schemes and select the Move operation to perform as it can introduce the most cost reduction. In the next iteration, Merge from the main schemes is selected and performed, after which no operation can further reduce the cost of the workflow. After applying the Move and Merge operations, the charging hours of executing this workflow is reduced from three to two.

Algorithm 1 The optimization process of ToF for workflows in one plan period.

```
1: Queue all coming workflows in a queue Q;
2: for each workflow w in Q do
3:     Determine the initial assigned instance type for each task in w;
4: repeat
5:     for each o_m in main schemes (i.e., Merge and Demote) do
6:         Pretend to apply o_m and check whether the earliest start or
           latest end time constraint of any task in w is violated after
           applying o_m;
7:         if No time constraint is violated then
8:             Estimate the cost reduced by performing o_m using the
               cost model;
9:     Select and perform the operation in main schemes which has
       the largest cost reduction;
10:    for each o_a in auxiliary schemes (i.e., Move, Promote, Split
       and Co-scheduling) do
11:        Pretend to apply o_a and check whether the earliest start or
           latest end time constraint of any task in w is violated after
           applying o_a;
12:        if No time constraint is violated then
13:            Estimate the cost reduced by performing o_a using the
               cost model;
14:    Select and perform the operation in auxiliary schemes which
       has the largest cost reduction;
15: until No operation has a cost reduction;
16: return Optimized instance assignment graph for each workflow.
```

Plan period is an important tuning parameter in the planner. If the period is long, more workflows are buffered in the queue. To make the optimization plan, more combinations of tasks need to be checked for transformations and the optimization space becomes much larger. When the period parameter is short, the optimization space gets smaller and the chance for operating transformations to reduce cost decreases. We study the impact of this parameter in Section 6.

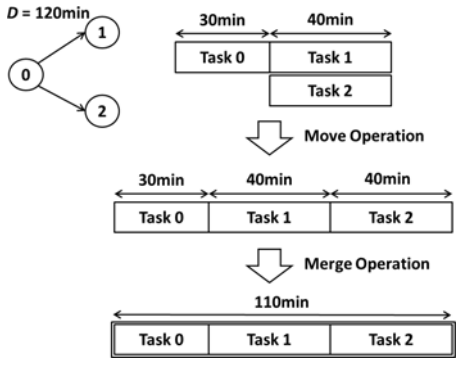


Fig. 3. Example of applying transformation operations on a three node structured workflow.

Initial instance assignment. It considers multiple heuristics. We experimentally evaluate these heuristics, and pick the one with the best result. In this paper, we present three initialization heuristics for initial instance assignment, namely *Best-fit*, *Worst-fit* and *Most-efficient*.

The Best-fit heuristic assigns each task with the most expensive instance type. This is to maximize performance but at the cost of a high monetary cost. Ideally, it should satisfy the deadline. Otherwise, we raise an error to the user.

The Worst-fit heuristic first assigns each task with the cheapest instance type to minimize the cost. Then, we apply the GAIN approach [39] to repeatedly re-assign tasks to a better instance type. GAIN is a greedy approach which picks the task with the largest benefit in execution time until the deadline requirement is met.

The Most-efficient heuristic configures each task according to deadline assignment. It first performs deadline assignment using an existing approach [40] and then assigns each task to the most cost-efficient instance type. This approach is also used in the previous study [4].

Complexity analysis. To compute the time complexity of our proposed algorithm, suppose Algorithm 1 has received N workflows each with M tasks. Assume there are on average k iterations in the optimization. The algorithmic complexity of Algorithm 1 is $O(N \cdot M^2 + kM \cdot N)$. In our evaluation, we find k is 97 and 43 for a Ligo and a Montage workflow, respectively.

Our planner is memory efficient. In our experiment, the memory overhead of the planner is 210 KB for a Ligo workflow and 190 KB for a Montage workflow.

5.2 Cost and Time Estimation

We develop simple yet effective cost models to estimate the cost and the time changes for applying one transformation operation on the instance DAG. Since an auxiliary scheme does not directly reduce the cost, we estimate the potential cost saving of the main schemes after applying the auxiliary scheme. As for the time estimation, the changes of execution time need to be propagated to all the tasks with dependencies on the vertices affected by the transformation operation. This paper estimates the worst case for the change of execution time, since worst-case analysis usually can have simplified estimation process.

In the following estimations, we use the following notations. The time estimation and price units are in hours. The hourly price of a type-i instance is P_i. Since partial hours are rounded, a task A with execution time t_A on the instance costs $\lceil t_A \rceil \times P_i$.

We present our cost and time estimation for each transformation operation as follows.

Merge operation. Assume tasks A and B are each running on a type-i instance and they can be merged to the same type-i instance. The execution time of task A and B are t_A and t_B respectively and the time from the end of task A to the start of task B is t_{gap}. Then the cost saved by merging tasks A and B can be estimated as below:

$$(\lceil t_A \rceil + \lceil t_B \rceil - \lceil (t_A + t_B + t_{gap}) \rceil) \times P_i. \quad (1)$$

Since the Merge operation does not change the total execution time of a job, we estimate the change of time to be zero.

Move operation. Assume task A can be moved to the end of task B to merge with B on the same type-i instance. The execution time of task A and B are t_A and t_B respectively. Although the Move operation does not reduce cost directly, the main schemes following it can save cost. Using the Merge operation as an example, the cost saved by moving task A to the end of task B can be estimated as below:

$$(\lceil t_A \rceil + \lceil t_B \rceil - \lceil (t_A + t_B) \rceil) \times P_i. \quad (2)$$

Note that, after moving task A to the end of task B, the start and end time of all tasks that are dependent on A will change accordingly.

Estimating the time changed by applying the Move operation is more complex. Assume task A is moved t_m afterwards to be merged with task B. If A is on the critical path of the workflow, the execution time of the workflow is delayed for t_m. If task A is not on the critical path, the delay time is less than t_m. However, the overhead of deciding whether a task is on the critical path is $\mathcal{O}(V + E)$. To reduce time complexity, we simply estimate the execution time is increased by t_m. We adopt the same idea for the time estimation of the rest operations.

Promote and Demote operation. Since the Promote operation is a dual operation of Demote, we present the cost and time estimation process using Demote. Assume we can demote a task from a type-i instance to a type-j instance, where $i > j$. The execution time of the task on the type-i instance is t_i and the execution time of the task on an instance of type j (denoted as t_j) can be calculated by its parallel rate. In this study, we estimate the task execution time according to Amdahl's law. The description can be found in Section 6. The cost change introduced by the Demote operation is approximately:

$$\lceil t_i \rceil \times P_i - \lceil t_j \rceil \times P_j. \quad (3)$$

Approximately, we estimate the execution time is changed by $t_i - t_j$.

Split operation. Assume a task A starts on a type-i instance at time t_1 and ends at time t_2. The Split operation can split task A at time t_s and resume the rest of the execution at time t_r. The cost saved by the Split operation is estimated as:

$$(\lceil (t_2 - t_1) \rceil - \lceil (t_s - t_1) \rceil - \lceil (t_2 - t_s) \rceil) \times P_i. \quad (4)$$

The total execution time is estimated to be delayed by $t_r - t_s$.

Co-scheduling operation. Assume tasks A and B are each running on a type-i instance and their execution time are t_A and t_B respectively. After Co-scheduling operation, tasks A and B are running on the same type-i instance. Suppose the performance degradation rate is r_d ($r_d \geq 1$). That means, the task execution time is increased by a factor of r_d. The cost saved by the Co-scheduling operation is estimated as:

$$(\lceil t_A \rceil + \lceil t_B \rceil - \lceil \max(t_A \times r_d, t_B \times_d) \rceil) \times P_i. \quad (5)$$

Approximately, the execution time of the job that task A belongs to is increased by $max(r_d \times t_A, r_d \times t_B) - t_A$ and the execution time of the job that task B belongs to is increased by $max(r_d \times t_A, r_d \times t_B) - t_B$.

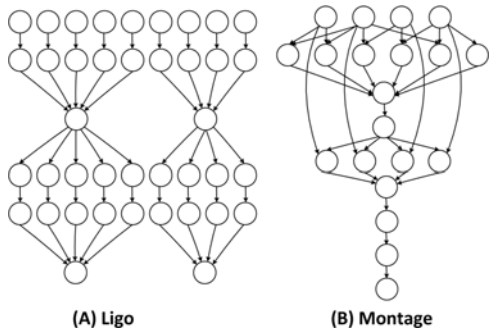


Fig. 4. Workflow structures of Ligo and Montage.

6 Evaluation

In this section, we present the evaluation results of our proposed approach.

6.1 Experimental Setup

We conduct our evaluations on real cloud platforms, including Amazon EC2 and Rackspace. The experiments were performed in December 2012 and the total cost for the experiments is approximately $6,000. We mainly consider the on-demand instance types, as shown in Tables 1 and 2.

Workload. We have used synthetic workloads based on two real-world applications, namely Montage and Ligo. Montage is an astronomical application widely used as a Grid and parallel computing benchmark. Ligo (Laser Interferometer Gravitational Wave Observatory) is an application used to detect gravitational-wave. The structures of the two workflows are shown in Fig. 4. The two applications have different workflow structures and parallelism. Ligo has more parallelism and Montage has more complicated structure. We generate tasks of different types from a set of predefined task types. The execution time of four different types of tasks on the m1.small instance type are set as 30, 90, 150 and 210 minutes. For each task type, we estimate its execution time on other instance types according to Amdahl's law. Assume the execution time of a task on an instance of type i is t_i. Assume the CPU capability of the type-j instance is k times as powerful as the type-i instance, we have the task execution time on the instance of type j to be $t_j = Pr \times \frac{t_i}{k} + (1 - Pr) \times t_i$, where Pr is the parallel rate defined in Amdahl's law. The I/O and network time of workloads are included in the sequential part of the total execution time. In Section 6.3.2, we vary Pr to study the effectiveness of ToF on workloads with different I/O and network characteristics. The performance degradation rate for Co-scheduling operation is around 1.25 in our study. We define D_{max} as the execution time of a workflow when its tasks are all assigned to the cheapest instance type while D_{min} as the execution time of a workflow when all tasks are assigned to the most expensive instance type.

To assess our framework under the context of different workflow types, we study the workload with continuous submissions of one workflow type (either Montage or Ligo), as well as a mixed workload of the two applications. In the mixed workload, the workflow type is randomly selected between Montage and Ligo. We assume the workflow submission rate follows Poisson distribution. By default, the arrival rate of workflows is 0.1 per minute. For each workload, we submit around 100 jobs which is sufficiently large for measuring the stable performance.

Comparisons. To evaluate the effectiveness of the proposed techniques in ToF, we have implemented the following algorithms for comparisons:

- *Baseline.* The Baseline approach simply uses the initial instance assignment in ToF, without any transformations and optimizations.
- *Auto-scaling [4].* We adopt the Auto-scaling approach [4] as the state-of-the-art comparison.
- *Greedy.* The Greedy approach is similar to ToF. The difference is that, ToF uses an planner to guide the transformation while the Greedy approach randomly selects a transformation operation until the termination condition is reached.

Theoretically, all the comparison approaches are inferior to ToF. Specifically, Baseline only adopts the initialization technique used in ToF, without the latter transformation-based optimizations. Auto-scaling is basically a special case of ToF, where the bundling, consolidation and parallelism reduction operations adopted in Auto-scaling are equivalent to the Merge, Demote/Promote and Move operations in ToF. However, Auto-scaling does not have the cost model as guidance whereas ToF does. While Greedy has the same set of transformation rules, it uses a random approach to determine which transformation to use. As our cost model is sufficiently accurate, Greedy is inferior to ToF in terms of both optimization overhead and the effectiveness of optimization.

All the metrics are normalized to those of Baseline. To demonstrate the effectiveness of our framework, we mainly compare the optimized total monetary cost and average execution time of workloads.

We perform sensitivity studies on the parameters of ToF: initialization heuristic (IH), plan period (P), deadline (D) and parallel rate (Pr). We also study the performance of ToF under different arrival rates of workflows. The default settings for those parameters are as follows: IH is *Most-efficient*, P is 15 minutes, D is $0.5 \times (D_{max} + D_{min})$ and Pr is 0.6. Each evaluation is repeated for 10 times and the average result is reported.

In the following evaluations, we first verify the accuracy of the cost estimation model. Next, we present the optimization results of ToF on minimizing the monetary cost while satisfying the performance requirements. Finally, we present the evaluation of ToF on optimizing the workflow execution time given budget requirements in Section 6.4.

6.2 Evaluations on Cost Estimation

In this section, we evaluate the accuracy of our cost estimation model.

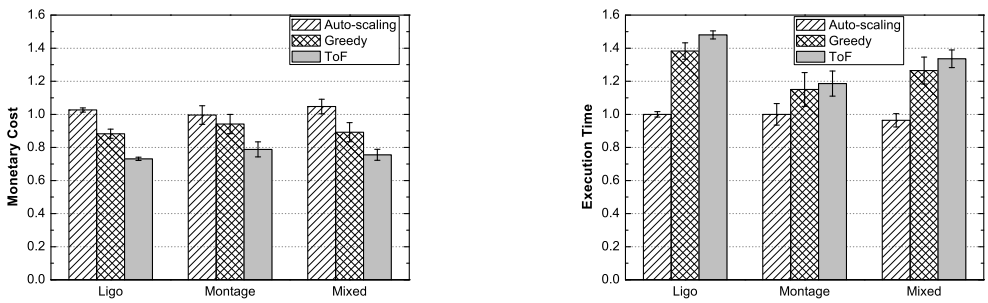
Tables 5 and 6 show the real and estimated monetary cost saving of specific operations on Ligo and Montage workflows in one plan period. The estimated monetary cost saving is returned by our cost estimation model and the real cost saving is calculated by differentiating the total monetary cost before and after performing an operation. The estimated cost is almost the same as the real cost on both workflows except for some outliers. The outliers exist mainly because our cost estimation model can only look-

TABLE 5
The Real and Estimated Monetary Cost Saved by Transformation Operations in One Plan Period on Ligo Workflow

	Iteration Step										
	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Step 10	Step 11
Transformation Operations	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{D} \rightarrow \mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{D} \rightarrow \mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{D} \rightarrow \mathcal{V} \rightarrow \mathcal{M}$
Real Cost	0.06	0.06	0.06	0.12	0.12	0.06	0.06	0.12	0.24	0.4	0.12
Estimate Cost	0.06	0.06	0.12	0.12	0.12	0.06	0.12	0.12	0.24	0.4	0.12

TABLE 6
The Real and Estimated Monetary Cost Saved by Transformation Operations in One Plan Period on Montage Workflow

	Iteration Step								
	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9
Transformation Operations	$\mathcal{P} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{P} \rightarrow \mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$	$\mathcal{V} \rightarrow \mathcal{M}$
Real Cost	0.12	0.06	0.06	0.12	0.12	0.06	0.18	0.06	0.06
Estimate Cost	0.12	0.12	0.12	0.12	0.06	0.06	0.18	0.06	0.06



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

Fig. 5. Cost optimization results on different workflow structures under the pricing scheme of Amazon EC2.

ahead one step. The further cost change caused by other tasks dependent on the operated task is ignored in estimation. The optimization overhead of ToF is 0.4 and 0.2 seconds for one Montage job and one Ligo job, respectively.

6.3 Results on Cost Optimizations

In this section, we first present the overall comparison results of ToF with other optimization methods on minimizing monetary cost and then conduct sensitivity studies of ToF on different parameters. The optimization goal is to minimize the monetary cost given a deadline to each workflow.

6.3.1 Overall Results

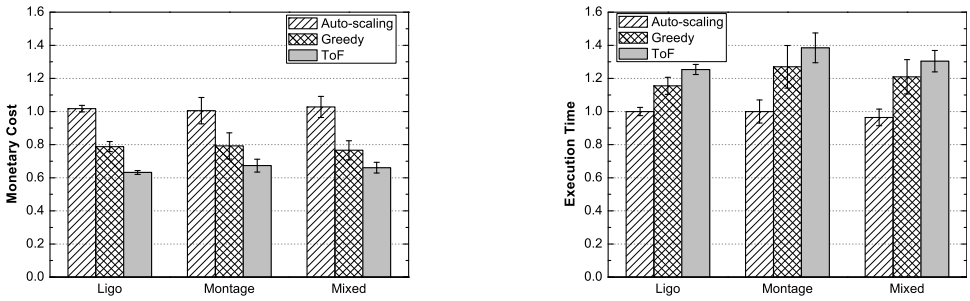
Fig. 5a shows the overall monetary cost results of ToF, Auto-scaling and Greedy methods on the Montage, Ligo and mixed workload using the pricing scheme of Amazon EC2. The standard errors of the overall monetary cost of ToF, Auto-scaling and Greedy are 0.01-0.05, 0.01-0.06 and 0.03-0.06 respectively on the tested workloads. On average, ToF obtains the smallest monetary cost in all three workloads, meaning the designed transformation set is suitable for different structures of workflows. ToF saves more monetary cost than Baseline, Auto-scaling and Greedy by 21-27, 21-30 and 15-17 percent, respectively. Auto-scaling has similar monetary cost result as Baseline. This means Auto-scaling has missed a great number of optimization opportunities that can be discovered by our transformation operations. Fig. 5b shows the average execution time results. The standard errors of the average execution time of ToF, Auto-scaling and Greedy are 0.02-0.08, 0.02-0.07 and 0.05-0.1 respectively on the tested workloads. Although the average execution time of ToF is longer than the other algorithms, it guarantees the deadline requirement in all cases.

To better understand the cost saving of ToF over the other compared algorithms, we analyze the number of instances started and their average utilizations during workflow execution. Table 7 shows the results for Ligo. All the methods do not choose any m1.xlarge, simply because the deadline is sufficiently loose for using cheaper instances. The total number of instances started in ToF is smaller than Baseline and Auto-scaling. This is because the transformation operations such as Move and Merge have made the execution plan with high utilizations. Compared to Greedy, ToF manages to make execution plans with more cheaper instance types. The utilizations of the instances in ToF are also higher because of selecting the proper transformation guided by the cost model.

We have also performed the overall experiment under the pricing scheme of Rackspace. Fig. 6a shows the overall monetary cost results and Fig. 6b shows the average execution time results on Rackspace. The standard errors of the overall monetary cost of ToF, Auto-scaling and Greedy are 0.01-0.04, 0.02-0.06 and 0.03-0.06 respectively on all tested

TABLE 7
Total Number of Instances Started and Their Average Utilizations during the Execution of Ligo Workflows under the Pricing Scheme of Amazon EC2

Algorithm	Baseline		Auto-scaling		Greedy		ToF	
Instance Type	# instances	Average Uti.	# instances	Average Uti.	# instances	Average Uti.	# instances	Average Uti.
m1.small	447	93%	448	91%	293	95%	432	97%
m1.medium	141	94%	132	93%	119	95%	62	94%
m1.large	96	95%	91	94%	81	94%	69	94%
m1.xlarge	0	–	0	–	0	–	0	–



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

Fig. 6. Cost optimization results on different workflow structures under the pricing scheme of Rackspace.

workloads. The standard errors of the average execution time of ToF, Auto-scaling and Greedy are 0.03-0.09, 0.02-0.07 and 0.05-0.13 respectively on the tested workloads. We have observed similar findings as the results of Amazon EC2, and thus focus on the evaluations on Amazon EC2 in the remainder of this section.

6.3.2 Sensitivity Studies

We have conducted sensitivity studies on different parameters. Since we have observed similar results across workloads, we focus on Ligo workload in the sensitivity studies. In each study, we vary one parameter at a time and keep other parameters in their default settings. We only present the monetary cost optimization results in the following studies to learn the sensitivity trends of ToF over the parameters.

Initial instance assignment heuristics. Fig. 7 shows the monetary cost results under the three different initialization heuristics, namely Best-fit, Worst-fit and Most-efficient. Note, Auto-scaling uses the Most-efficient heuristic only. ToF achieves a smaller cost than Baseline, Auto-scaling and Greedy in all cases, which shows the effectiveness of our transformation operations and the cost-based rule. ToF obtains the most significant cost reduction over Baseline under the Best-fit heuristic. This is because all tasks are assigned to the best instance type after the Best-fit initialization, which provides a significant space for cost optimizations using Demote.

Plan period. We vary the plan period parameter from 5 to 25 minutes and present the resulting monetary cost results in Fig. 8. The monetary cost obtained by ToF decreases when plan period varies from 5 to 20, but increases when the plan period increases to 25. Although a longer plan period offers a larger optimization space, it tightens the deadline of workflows since the workflows have to wait longer in the queue in order to be scheduled. A suitable choice of the plan period should balance these two issues. In our study, we choose the plan period of 15 minutes.

Deadline. Deadline is an important factor to determine the initial configurations of tasks and further affects the optimization results. We evaluate the compared algorithms under deadline requirement varying from $1.5 \times D_{min}$, $0.5\times (D_{min} + D_{max})$ to D_{max}. Fig. 9 shows the resulting monetary cost results. ToF obtains the lowest monetary cost among the compared algorithms under all deadlines. As the deadline becomes loose, more tasks can be assigned to cheaper

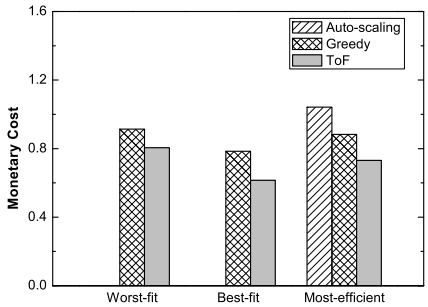


Fig. 7. Overall monetary cost results of sensitivity studies on the initial instance assignment heuristics.

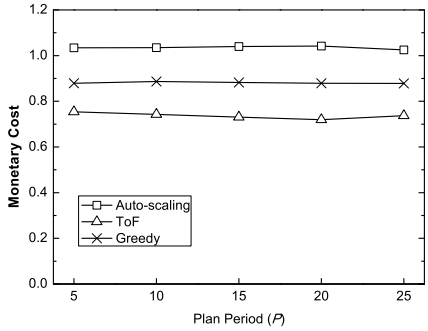


Fig. 8. Overall monetary cost results of sensitivity studies on the plan period parameter.

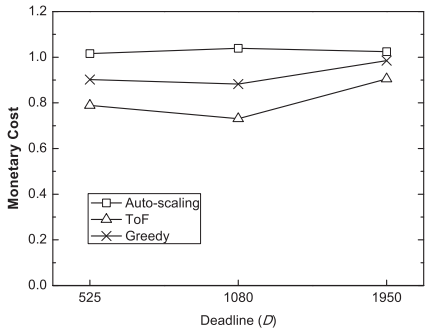


Fig. 9. Overall monetary cost results of sensitivity studies on deadline.

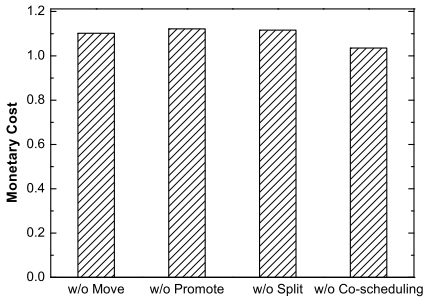


Fig. 11. Overall monetary cost results of sensitivity studies on the effectiveness of each single auxiliary operation.

instance types. The cost saving of ToF over Baseline and Auto-scaling is higher when deadline is 1,080 minutes. This is because when deadline is tight, e.g., 525 minutes, many tasks have to be assigned to expensive instance types in order to meet deadline requirement. In this case, the operations such as Demote are not applicable. Similarly, when deadline is loose, e.g., 1,950 minutes, many tasks can be assigned to cheap instances and there is no need to perform operations such as Demote. When deadline is 1,950 minutes, only the m1.small type of instances are used and all compared algorithms are almost the same.

Parallel rate (Pr). We also evaluate the performance of the compared algorithms when the parallel rate parameter varies from 0.4, 0.6, 0.8 to 0.95. The evaluated results are shown in Fig. 10. Different parallel rate values stand for different application types. For example, parallel rate of 0.8 and 0.95 could stand for compute-intensive applications while in applications with parallel rate of 0.4 and 0.6, network or I/O time are dominating the total execution time. As the parallel rate increases, the monetary cost of the compared algorithms decreases. This is because, with a higher parallel rate, tasks can benefit more from instances with higher capability. Under different parallel rates, ToF always obtains the lowest monetary cost among all compared algorithms. This demonstrates the effectiveness of our design to a wide areas of applications. ToF saves more cost when parallel rate is higher. This is because the compute-intensive applications can benefit more from transformations like Demote.

Effectiveness of auxiliary schemes. We also evaluate the effectiveness of each single operation in auxiliary schemes on cost optimization. We evaluate ToF without the Move operation (denoted as w/o Move), ToF without the Promote operation (denoted as w/o Promote), ToF without the Split operation (denoted as w/o Split) and ToF without the Co-scheduling operation (denoted as w/o Co-scheduling) in each experiment. Fig. 11 shows the optimized monetary cost results. All results are normalized to the ToF result with all auxiliary schemes. The monetary cost results are all above one, meaning that ToF performs the best for monetary cost optimization when using all the auxiliary schemes. The monetary cost obtained by w/o Co-scheduling is smaller than the others and the number of Co-scheduling operations adopted during ToF execution is also smaller than the other operations. This is because the tasks in Ligo are of different types, the chance of two tasks having similar start and end time as well as similar leftover time before deadlines is low.

Arrival rate. We evaluate the effectiveness of ToF when the arrival rate of workflows varies from 0.1, 0.2, 0.4, 0.6 to 0.8. Fig. 12 shows the optimized total monetary cost results. ToF has the lowest cost among the compared algorithms under all arrival rates. The cost saving of ToF slightly increases, from 27 to 32 percent over Baseline, when the arrival rate gets larger. This is because with a larger arrival rate, there are more workflows queuing up within one plan period, which creates more chances to perform the transformation operations.

6.4 Results on Performance Optimizations

In this section, we evaluate ToF with the goal of minimizing workflow execution time given a budget on each workflow. The extension is simple. First, in the initialization step, the three initialization heuristics are extended to satisfy budget requirement. The Best-fit and Most-efficient heuristics adopt the GAIN [39] approach to make sure of budget while

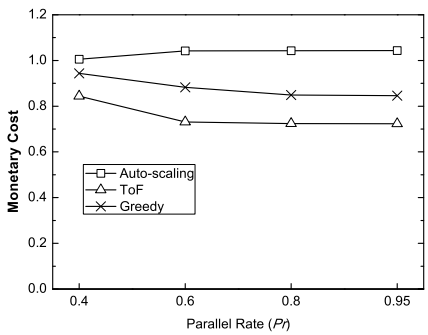


Fig. 10. Overall monetary cost results of sensitivity studies on the parallel rate parameter.

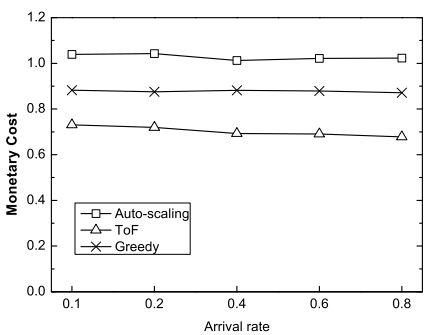
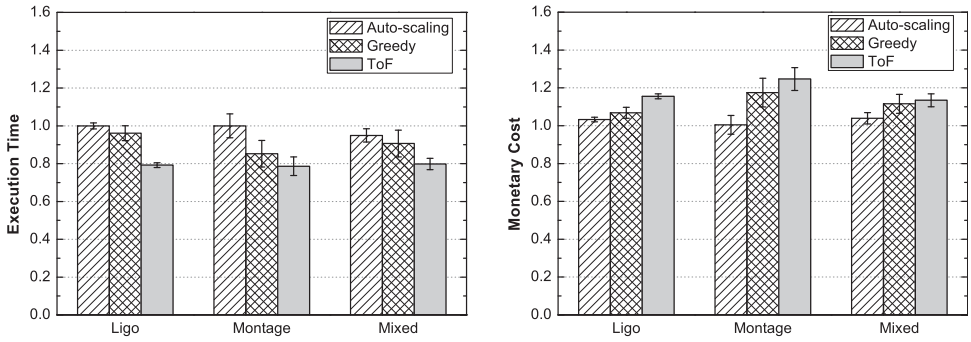


Fig. 12. Overall monetary cost results of sensitivity studies on the arrival rate of workflows.



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

Fig. 13. Execution time optimization results of the three compared algorithms on different workflow structures.

the Worst-fit heuristic simply assigns each task to the cheapest instance type. The planner guides the optimization process with a time-based rule. Since Auto-scaling was not proposed to optimize execution time, we also modified it in the following way: starting from a tight deadline, each task is configured with the most cost-efficient instance type according to deadline assignment. According to the current task configuration, if the total cost is larger than budget, we loose the deadline a bit and re-run the task configuration until the total cost is less than budget. We resume the rest of the auto-scaling optimizations from this preprocessed state.

Fig. 13a shows the optimized average execution time and Fig. 13b shows the corresponding total cost obtained by the compared algorithms on three different workflows. All parameters are set as default. Promote operation is now the only main scheme and the other operations are categorized as auxiliary schemes. The budget is set as twice the total cost of executing the workflow when all of its tasks are assigned to the cheapest instance type. The standard errors of the average execution time of ToF, Auto-scaling and Greedy are 0.01-0.05, 0.02-0.06 and 0.04-0.07 respectively on the tested workloads. The standard errors of the total cost of ToF, Auto-scaling and Greedy are 0.01-0.06, 0.01-0.05 and 0.03-0.08 respectively. On all the three workloads, ToF obtains the best average execution time in all workflow structures. Specifically, it reduces the average execution time over Baseline, Auto-scaling and Greedy by 20-21, 16-21 and 8-18 percent, respectively. This result demonstrates the effectiveness of the extended ToF on optimizing execution time and hence validates the extensibility of our framework.

7 Conclusion

Performance and monetary cost optimizations for running workflows from different applications in the cloud have become a hot and important research topic. However, most existing studies fail to offer general optimizations to capture optimization opportunities in different user requirements, cloud offerings and workflows. To bridge this gap, we propose a workflow transformation-based optimization framework namely ToF. We formulate the performance and cost optimizations of workflows in the cloud as transformation and optimization. The two components are designed to be extensible for user requirements on performance and cost, cloud offerings and workflows. Particularly, we formulate six basic transformation operations. We further develop a cost model guided planner to efficiently and effectively find the suitable transformation sequence for the given performance and cost goal. We evaluate our framework using real-world scientific workflow applications and compare with other state-of-the-art scheduling algorithms. Results show our framework outperforms the state-of-the-art Auto-scaling algorithm by 30 percent for monetary cost optimization, and by 21 percent for the execution time optimization. Moreover, the planner is lightweight for online optimization in the cloud environments. As for future work, we consider ToF on multiple clouds. Still, there are many practical and challenging issues for current multi-cloud environments [41]. Those issues include relatively limited cross-cloud network bandwidth and lacking of cloud standards among cloud providers.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. They acknowledge the support from the Singapore National Research Foundation under its Environmental & Water Technologies Strategic Research Programme and administered by the Environment & Water Industry Programme Office (EWI) of the PUB, under project 1002-IRIS-09. The author Amelie Chi Zhou is also with Nanyang Environment and Water Research Institute (NEWRI). The authors thank Dr. Chua Hock Chye Lloyd for his input on the early version of this paper.

References

[1] Amazon Case Studies, http://aws.amazon.com/solutions/case-studies/, 2014.
[2] Windows Azure Case Studies, "http//www.microsoft.com/azure/casestudies.mspx." 2014.
[3] S. Ibrahim, B. He, and H. Jin, "Towards Pay-As-You-Consume Cloud Computing," *Proc. IEEE Int'l Conf. Services Computing (SCC)*, pp. 370-377, 2011.
[4] M. Mao and M. Humphrey, "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 49:1-49:12, 2011.
[5] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-Scaling with Deadline and Budget Constraints," *Proc. IEEE/ACM 11th Int'l Conf. Grid Computing (GRID)*, pp. 41-48, 2010.
[6] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost Optimized Provisioning of Elastic Resources for Application Workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011-1026, 2011.

[7] H. Kllapi, E. Sitaridi, M.M. Tsangaris, and Y. Ioannidis, "Schedule Optimization for Data Processing Flows on the Cloud," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2011.
[8] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 22:1-22:11, 2012.
[9] Y. Gong, B. He, and J. Zhong, "Network Performance Aware MPI Collective Communication Operations in the Cloud," *IEEE Trans. Parallel and Distributed Syst.*, vol. 99, 2013, doi: 10.1109/TPDS.2013.96.
[10] M. Mao and M. Humphrey, "Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows," *Proc. IEEE 27th Int'l Symp. Parallel and Distributed Processing (IPDPS)*, 2013.
[11] H.M. Fard, R. Prodan, and T. Fahringer, "A Truthful Dynamic Workflow Scheduling Mechanism for Commercial Multicloud Environments," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1203-1212, June 2013.
[12] Q. Zhu, J. Zhu, and G. Agrawal, "Power-Aware Consolidation of Scientific Workflows in Virtualized Environments," *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1-12, 2010.
[13] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed Systems Meet Economics: Pricing in the Cloud," *Proc. Second USENIX Conf. Hot Topics in Cloud Computing (HotCloud)*, pp. 6-6, 2010.
[14] H. Herodotou and S. Babu, "Profiling, What-If Analysis, and Cost-Based Optimization of Mapreduce Programs," *Proc. VLDB Endowment*, vol. 4, no. 11, pp. 1111-1122, 2011.
[15] J.C. Jacob, D.S. Katz, G.B. Berriman, J.C. Good, A.C. Laity, E. Deelman, C. Kesselman, G. Singh, M. Su, T.A. Prince, and R. Williams, "Montage: A Grid Portal and Software Toolkit for Science, Grade Astronomical Image Mosaicking," *Int'l J. Computational Science and Eng.*, vol. 4, no. 2, pp. 73-87, July 2009.
[16] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, and S. Koranda, "Griphyn and Ligo, Building a Virtual Data Grid for Gravitational Wave Scientists," *Proc. 11th IEEE Int'l Symp. High Performance Distributed Computing (HPDC)*, 2002.
[17] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, "Experiences Using Cloud Computing for a Scientific Workflow Application," *Proc. Second Int'l Workshop Scientific Cloud Computing (ScienceCloud*, pp. 15-24, 2011.
[18] G. Graefe and J. Gray, "The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb," Technical Report MSR-TR-97-33, Microsoft Research, 1997.
[19] R.T.B. Ma, D.-M. Chiu, J.C.S. Lui, V. Misra, and D. Rubenstein, "Internet Economics: The Use of Shapley Value for ISP Settlement," *Proc. ACM CoNEXT Conf.*, 2007.
[20] J. Gray, "Distributed Computing Economics," Technical Report tr-2003-24, Microsoft, 2003.
[21] S. Abrishami, M. Naghibzadeh, and D.H.J. Epema, "Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400-1414, Aug. 2012.
[22] H.M. Fard, R. Prodan, J.J.D. Barrionuevo, and T. Fahringer, "A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments," *Proc. 12th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGRID)*, pp. 300-309, 2012.
[23] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," technical report, 2005.
[24] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," *Proc. 18th Int'l Parallel and Distributed Symp. (IPDPS)*, 2004.
[25] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming*, vol. 13, no. 3, pp. 219-237, 2005.
[26] Y.C. Lee, R. Subrata, and A.Y. Zomaya, "On the Performance of a Dual-Objective Optimization Model for Workflow Applications on Grid Platforms," *IEEE Trans. Parallel and Distributed Systems*, vol. 20, no. 9, pp. 1273-1284, Sept. 2009.
[27] J. Yu and Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms," *Scientific Programming*, vol. 14, nos. 3/4, pp. 217-230, 2006.
[28] G. Mateescu and G. Mateescu, "Quality of Service on the Grid Via Metascheduling with Resource Co-Scheduling and Co-Reservation," *Int'l J. High Performance Computing Applications*, vol. 17, pp. 209-218, 2003.
[29] K. Yoshimoto, P. Kovatch, and P. Andrews, "Co-Scheduling with User-Settable Reservations," *Proc. 11th Int'l Conf. Job Scheduling Strategies for Parallel Processing (JSSPP)*, pp. 146-156, 2005.
[30] R. Buyya, C.S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering It Services As Computing Utilities," *Proc. 10th IEEE Int'l Conf. High Performance Computing and Comm. (HPCC)*, pp. 5-13, 2008.
[31] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. Grd Computing Environments Workshop (GCE)*, pp. 1-10, 2008.
[32] M.D. de Assuncao, A. di Costanzo, and R. Buyya, "Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters," *Proc. 18th ACM Int'l Symp. High Performance Distributed Computing (HPDC)*, pp. 141-150, 2009.
[33] H.C. Lim, S. Babu, J.S. Chase, and S.S. Parekh, "Automated Control in Cloud Computing: Challenges and Opportunities," *Proc. First Workshop Automated Control for Datacenters and Clouds (ACDC)*, pp. 13-18, 2009.
[34] J. Lucas Simarro, R. Moreno Vozmediano, R. Montero, and I. Llorente, "Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments," *Proc. Int'l Conf. High Performance Computing and Simulation (HPCS)*, pp. 1-7, 2011.
[35] N. Grozev and R. Buyya, "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, vol. 44, pp. 369-390, Mar. 2014.
[36] A. Ben Othman, J. Nicod, L. Philippe, and V. Rehn Sonigo, "Optimal Energy Consumption and Throughput for Workflow Applications on Distributed Architectures," *Proc. IEEE 21st Int'l Workshop Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 115-120, 2012.
[37] DAGMan: Directed Acyclic Graph Manager, http://cs.wisc.edu/condor/dagman, 2014.
[38] M.L. et al., "Condor: A Hunter of Idle Workstations," *Proc. Eigth Int'l Conf. Distributed Computing Systems (ICDCS)*, 1988.
[39] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M.D. Dikaiakos, "Scheduling Workflows with Budget Constraints," *Proc. Integrated Research in Grid Computing: CoreGRID Integration Workshop*, 2007.
[40] J. Yu, R. Buyya, and C.K. Tham, "Cost-Based Scheduling of Scientific Workflow Application on Utility Grids," *Proc. First Int'l Conf. e-Science and Grid Computing (E-SCIENCE)*, pp. 140-147, 2005.
[41] X. Tang, C. Chen, and B. He, "Green-Aware Workload Scheduling in Geographically Distributed Data Centers," *Proc. IEEE Fourth Int'l Conf. Cloud Computing Technology and Science (CLOUDCOM)*, pp. 82-89, 2012.

Amelie Chi Zhou received the bachelor's degree in 2009 and the master's degree in 2011, both from Beihang University. She is currently working toward the PhD degree at the School of Computer Engineering of Nanyang Technological University, Singapore. Her research interests include cloud computing and database systems.

Bingsheng He received the bachelor's 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 assistant professor in the Division of Networks and Distributed Systems, School of Computer Engineering of Nanyang Technological University, Singapore. His research interests include high-performance computing, cloud computing, and database systems.