Adaptive Workflow Scheduling on Cloud Computing Platforms with Iterative Ordinal Optimization

Fan Zhang, *Senior Member, IEEE*, Junwei Cao, *Senior Member, IEEE*, Kai Hwang, *Fellow, IEEE*, Keqin Li, *Fellow, IEEE*, and Samee U. Khan, *Senior Member, IEEE*

Abstract—The scheduling of multitask jobs on clouds is an NP-hard problem. The problem becomes even worse when complex workflows are executed on elastic clouds, such as Amazon EC2 or IBM RC2. The main difficulty lies in the large search space and high overhead of generating optimal schedules, especially for real-time applications with dynamic workloads. In this work, a new *iterative ordinal optimization* (IOO) method is proposed. The *ordinal optimization* method is applied in each iteration to achieve sub-optimal schedules. IOO aims at generating more efficient schedules from a global perspective over a long period. We prove through overhead analysis the advantages in time and space efficiency in using the IOO method. The IOO method is designed to adapt to system dynamism to yield suboptimal performance. In cloud experiments on IBM RC2 cloud, we execute 20,000 tasks in LIGO (*Laser Interferometer Gravitational-wave Observatory*) verification workflow on 128 virtual machines. The IOO schedule is generated in less than 1,000 seconds, while using the Monte Carlo simulation takes 27.6 hours, 100 times longer to yield an optimal schedule. The IOO-optimized schedule results in a throughput of 1,100 tasks/sec with 7 GB memory demand, compared with 60 percent decrease in throughput and 70 percent increase in memory demand in using the Monte Carlo method. Our LIGO experimental results clearly demonstrate the advantage of using the IOO-based workflow scheduling over the traditional blind-pick, ordinal optimization, or Monte Carlo methods. These numerical results are also validated by the theoretical complexity and overhead analysis provided.

Index Terms—Autonomic provisioning, big data, cloud computing, iterative ordinal optimization, and workflow scheduling

1 Introduction

Scientific workflows demand massive resources from various computing infrastructures to process massive amount of big data. Automatic provisioning of such big data applications on the cloud platform is challenging since current resource management and scheduling approaches may not be able to scale well, especial under highly dynamic conditions.

For example, the Laser Interferometer Gravitational-wave Observatory (LIGO) experiments digest terabytes of data per day [1]. The LIGO workload demands data-intensive analysis over workflow pipelines with millions of tasks to be scheduled on a computational grid or a cloud [8]. A typical LIGO workload shows the volume, velocity, and variety characteristics of big data.

To process the LIGO workload, parallel virtual machines (VMs) are provided as virtual clusters (VCs) from large-scale data centers [16]. Virtual clusters are elastic resources that can dynamically scale in or out.

The workload usually fluctuates, leading to period- or stage-based scheduling necessary. During each stage, a schedule is applied for the current workload, and the schedule may or may not change depending on the similarity of the workloads of consecutive periods. Therefore, iterative scheduling, which applies optimal schedules in a multi-stage manner becomes significant.

In general, scheduling multitask workflows on any distributed computing resources (including clouds) is an NP-hard problem [38]. The main challenge of dynamic workflow scheduling on virtual clusters lies in how to reduce the scheduling overhead to adapt to the workload dynamics with heavy fluctuations. In other words, the iterative scheduling also needs to be provided in real time.

In a real cloud platform, however, resource profiling and stage-based simulation on thousands or millions of feasible schedules are often performed, if an optimal solution is demanded. An optimal workflow schedule on a cloud may take weeks to generate [16].

Ho et al. [14] proposed the *ordinal optimization* (OO) method for dealing with complex problems with a very large schedule space. Subsequently, the authors demonstrated that the OO method is effective to generate a soft or suboptimal schedule for most of the NP-hard problems. As an example, optimal power flow [25] is an NP-hard

- *F. Zhang is with the Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. E-mail: f_zhang@mit.edu.*
- *J. Cao is with the Research Institute of Information Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: jcao@mail.tsinghua.edu.cn, jcao@tsinghua.edu.cn.*
- *K. Hwang is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089. E-mail: kaihwang@usc.edu.*
- *Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*
- *S.U. Khan is with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050. E-mail: samee.khan@ndsu.edu.*

Manuscript received 28 Feb. 2014; revised 27 June 2014; accepted 16 July 2014. Date of publication 20 Aug. 2014; date of current version 10 June 2015. Recommended for acceptance by R. Ranjan, L. Wang, A. Zomaya, D. Georgakopoulos, G. Wang, and X.-H. Sun.
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.2014.2350490

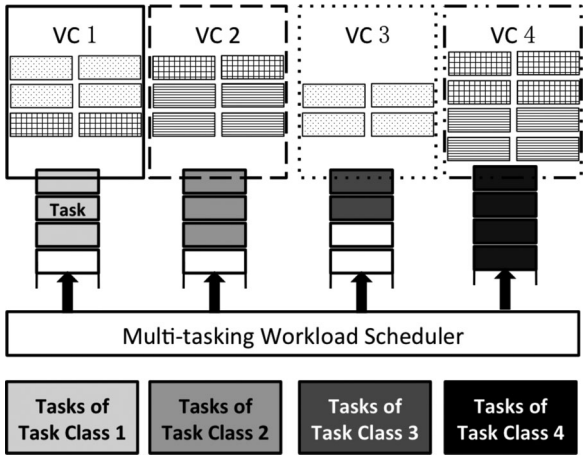


Fig. 1. The multitasking workload scheduler dispatches multiple tasks to VCs for parallel execution in a cloud platform. Each VC is responsible for one task class.

TABLE 1
Basic Notations and Definitions

Notation	Definition
U	Candidate set of all u possible schedules
S	Selection set of s schedules to simulate
G	Acceptance set of g good-enough schedules
N	Number of simulation runs per schedule candidate by Monte Carlo or Blind-Pick scheduling methods
n	The number of OO simulations per schedule
l	The number of layers needed to select for OO
$\boldsymbol{\theta}$	A working schedule in the schedule space U
P	Average task execution time on a single VM
D	Average task memory demand on a single VM
h	Time to simulate a schedule by Monte Carlo method
M	Makespan to execute all tasks in a workflow
T	Total workflow throughput in a cloud platform
D	Total memory demand in using virtual clusters
H	Overhead time of a particular scheduling method

problem that was handled by the OO method with some success.

The low overhead in the OO-based scheduling is attractive in real-time cloud computing applications [15], [30]. In our previous work [40], the OO is also applied to the multiobjective scheduling (MOS) of many tasks in cloud platforms. The inner core of the approach is to generate a rough model resembling the workflow problem. The discrepancy between the rough and precise search model is minimized. We reduce the search space significantly to lower the scheduling overhead.

In this paper, a new *iterative ordinal optimization* (IOO) algorithm is proposed. The IOO applies the OO method iteratively, in search of adaptive schedules to execute scientific workflows on elastic cloud compute nodes with dynamic workloads. During each iteration, the OO is applied to search for a suboptimal or good-enough schedule with very low overhead. From a global point of view, IOO can process more successive iterations fast enough to absorb the dynamism of the workload variations. A synopsis of our contributions of this work is summarized below.

- We present an analytical model of an autonomic resource provisioning scheme for multitasking big-data scientific application on a cloud platform. A follow-up novel simulation based approach is introduced to tailor for the need of tackling such a scheduling problem.
- We systematically extend the OO method to a multistage scheduling scenario. Benefiting from the low overhead and efficiency of OO, the IOO is able to apply the OO in an iterative fashion so that the IOO has much better adaptability to the dynamic workload. During each period of scheduling, the OO can only achieve sub-optimal schedules; the purpose of the IOO is to generate better schedules from a global perspective over a sequence of workload periods.
- Thereafter, we demonstrate the effectiveness of the proposed IOO approach with an extensive benchmarking with the LIGO experimental data. We apply the LIGO workflow [6] using hundreds of VMs. Both theoretical and experimental results show that the IOO scheduling method achieves higher throughput with lower memory demand, compared to the other two simulation-based approaches, Monte Carlo [28] and Blind-Pick [14].

The rest of the paper is organized as follows. Section 2 characterizes the workflow scheduling problem on the VCs in a cloud with existing optimization methods introduced. In Section 3, we provide details on the proposed IOO method with theoretical overhead analysis. In Section 4, the experimental settings and design of LIGO experiments are provided. We also elaborative on the experimental results and discuss the various aspects pertaining to the IOO performance compared with Monte Carlo and Blind-Pick. Related works are reviewed in Section 5. Finally, we summarize our contributions and discuss future research in Section 6.

2 Workflow Scheduling on Clouds

In this section, we first introduce a unique workflow-scheduling model for our cloud platform as a start. A simulation-based optimization method, which distinguishes itself from most of its existing literature, is followed.

2.1 Workflow Scheduling Model

The provisioning of VMs to a virtual cluster is dynamically performed upon user demand. For clarity, an example job dispatching queuing model for mapping subdivided workflow tasks is given in Fig. 1. In this scheduling model, we define a task class as a set of computing jobs of the same type, which can be executed concurrently on VMs within the same virtual cluster.

For the sake of simplicity during the analysis, we assume that all of the VMs within the same cluster take equal amount of time to execute the assigned tasks. In other words, the task execution time in a VM is the basic time unit in the performance analysis. For the easy of the reader, a summary of the most frequently used notations and definitions in this paper are listed in Table 1.

All of the VCs are distinguished by the index i. Let p_i be the expected execution time of a single task within the ith virtual cluster, VC_i. Let v_i be the number of VMs in VC_i. We

have $\beta_i = v_i/p_i$ as the task processing rate of cluster VC_i. Let δ_i be the number of tasks of the corresponding queue.

In light of the above model, we obtain the execution time of a task as $t_i = \delta_i/\beta_i = p_i\delta_i/v_i$. We define the *makespan* of all n tasks in a scientific workflow by:

$$M = \max\{t_1, t_2, \ldots, t_c\}, \tag{1}$$

where c virtual clusters are used. The makespan is the total execution time between the start and finish of all tasks within a multitask workflow. We denote d_i as the memory used by one of the VMs within a cluster. Based on the above, the total *memory demand* by all VMs is calculated by:

$$D = \sum_{i=1}^{c} d_i \times v_i. \tag{2}$$

A resource-reservation schedule specifies the sets of VMs provisioned at successive time slots, called periods. For example, the jth schedule θ_j is represented by a set of VMs allocated in c clusters in a schedule space U. Therefore, such a schedule can be represented by a c-dimensional vector:

$$\theta_j = [v_1, v_2, \ldots, v_c]. \tag{3}$$

At different time periods, different schedules may be applied. All of the candidate schedules at one time period form U. The cardinality of U can be calculated by the following expression:

$$u = (v-1)!/[(v-c)!(c-1)!], \tag{4}$$

where v is the total number of VMs used in c clusters. The parameter u counts the number of ways to partition a set of v VMs into c nonempty clusters.

For example, if we use $v = 20$ VMs in $c = 7$ clusters for seven task classes, then we need to assess $u = 27{,}132$ possible schedules to search for the best schedule to allocate the VMs. In a special case where $c = 2$ gives $u = v - 1$. It can be seen that only the task count that has been assigned to each VC matters. Therefore, we would make a simplified assumption that all the tasks within one VC have the same execution time and memory demand as if they were a set of homogeneous and embarrassingly parallel tasks.

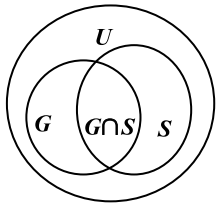
Using simulation to determine the best schedule, such a number is deemed too high, leading to excessive simulation overhead time. Therefore, we must significantly reduce the schedule search space.

The following objective function is used to search for the suboptimal schedules for the workflow scheduling. In general, we must conduct an exhaustive search to minimize a pair of objective functions on all possible makespan $M(\theta_j)$ and memory demands $D(\theta_j)$, jointly and simultaneously, i.e.,

$$\text{Min}\{M(\theta_j)\} \text{ and } \text{Min}\{D(\theta_j)\} \tag{5}$$

for all possible schedules θ_j in the search space U.

Eq. (5) attempts to optimize both the makespan and the demanded memory simultaneously. Such an optimization attempt, however, is not always possible due to a potential conflict resource demand. Therefore, we optimize the objectives in a skyline manner, or mathematically defined as pareto-front, as shown in Fig. 5 in the next section. Instead of finding one θ to optimize two objectives, we relax the target down to a set of *satisfactory* schedules, that is, no other schedule in the search space beats any satisfactory schedule in both the two objectives.



U: The candidate set of all possible u schedules in exhaustive search
G: The acceptance set of g schedules
S: Good-enough schedule set of s schedules
Note that $u >> s >> g >> k = |G \cap S|$

Fig. 2. The concept of OO using a set S intersecting with the set G to yield a set G ∩ S of k acceptable (good-enough) schedules.

The formulae for the makespan and memory demand are given in Eqs. (1)-(5). The time/space complexity defies the traditional heuristic approach. In a resource-sharing cloud platform, the values of p_i and d_i cannot be easily determined before runtime. For example, if some of the VCs were under provisioned with VM resource, these VCs would contend physical compute resources, such as CPU and memory. Unavoidably this leads to a very large variation of task execution time and significant extra memory usage due to memory ballooning effect. All these result in p_i and d_i very unpredictable. A good example illustrating this scenario is a large number of Map tasks competing for resources in a small-size VM cluster in a Hadoop MapReduce job, where the resource demand is unpredictable due to such resource contention.

If the VMs are allocated in different geographical regions, where the workload of each region is highly diversified, then the problem becomes worse. Therefore, in simulating each schedule θ_j, we must also profile the resource usage of the VMs, generate p_i, and d_i before we calculate M and D. We use the average over all of the simulations runs on θ_j to obtain the $M(\theta_j)$ and $D(\theta_j)$ in Eq. (5).

Problem clarification. Based on the workflow-scheduling model defined above, we need to calculate a time-series of intermediate schedules θ_j in order to optimize the makespan and memory demand over a given time-series workloads. Therefore, the input is such kind of workloads, or number of tasks arrives for each VC at arbitrary time points. The output is the time points when new schedules are applied and the schedules themselves.

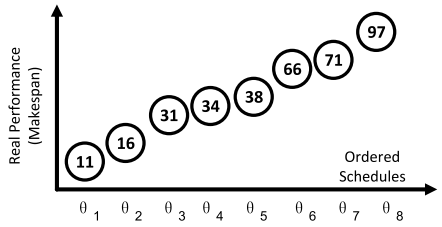
2.2 Simulation-Based Scheduling Optimization

2.2.1 The OO Method: Single Objective Scenario

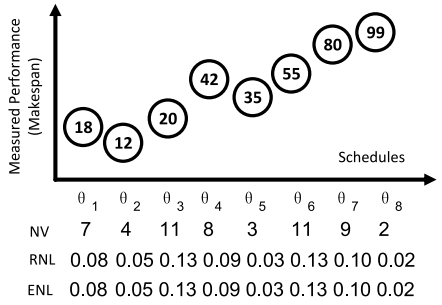
The basic concept of the OO is illustrated in Fig. 2.

Let U be a candidate set of all $u = |U|$ possible schedules. The set U is used in the exhaustive search of the best schedule. It is noteworthy to mention that the Monte Carlo applies to U, very slowly. In practice, we must define an acceptance set G of $g = |G|$ schedules. The schedules in G are acceptable or good-enough choices that are the top g schedules in U. In the OO, a rough and computationally fast model is applied to U. Then, a promising but smaller schedule set S is derived.

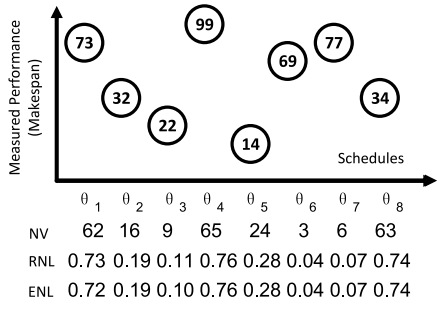
One can test only a reduced selection set S of s promising or good-enough schedules. The OO method slowly searches from S to generate at least k good-enough schedules in G. The success rate of such a search is set at $\alpha = 98\%$. Note that $u \gg s \gg g \gg k = |G \cap S|$. For example, if in Eq. (4), the value of u is equal to 27,132, then we will have $s = 190$,



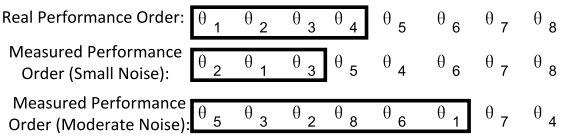
(a) Real performance of the eight schedules by using a large number (*N, e.g. 1000*) of repeated simulation runs



(b) Small Noise: Measured performance of the eight schedules by using a moderate number (*n', e.g. 100*) of repeated simulation runs



(c) Large Noise: Measured performance of the eight schedules by using a small number (*n'', e.g. 10*) of repeated simulation runs



(d) Comparison of the schedule order of the three cases

Fig. 3. A demonstration of how OO works in a single-objective scheduling problem. At least three schedules are required from the good-enough schedule set G = {θ_1, θ_2, θ_3, θ_4}. OO reduces the computing overhead from having to assess eight schedules to three or six schedules contingent upon noise level.

$g = 10$, and $k = 1$ for a single best schedule. These simulation parameters could be defined in the simulated optimization process, based on the requirements of a particular workflow application. In general, we must satisfy the following condition in the OO process:

$$\text{Probability}\{|G \cap S| \geq k\} \geq \alpha. \tag{6}$$

In Fig. 3, a simple single-objective example is provided to illustrate the main idea behind the OO method. Consider a candidate set $U = \{\theta_1, \theta_2, \ldots, \theta_8\}$ of eight schedules. Suppose that the given optimization problem is to find the minimum value among $M(\theta)$. Using the Monte Carlo method [28] to simulate $M(\theta)$ is a very time consuming process due to the exhaustive search in N simulations per schedule. Using a rough model with an approximated objective function $M'(\theta)$, one can reduce the simulation to n times, where $n \ll N$. The rough model can be seen as the Monte Carlo simulation plus a random noise defined by:

$$M'(\theta) = M(\theta) + \text{Noice\_Value}. \tag{7}$$

In Figs. 3a 3c), the value inside each circle represents the performance of that schedule. In Fig. 3a, the values are the *Real Performance* (*RP*), or $M(\theta)$, of applying those schedules, which are achieved by large number of repeated Monte Carlo simulation runs. In Figs. 3b and 3c), the values, or what we call as *Measured Performance* (*MP*), or $M'(\theta)$, are approximately assessed by the fast approaches. The approach used in Fig. 3c is faster than that used in Fig. 3b due to the reduced simulation runs for each schedule. The approach can be simply interpreted as estimating the area of an irregular shape inside a square. The more sampling points we toss, the better estimation we achieve, however, we have to tolerate higher computing overhead.

Noise Value (*NV*) is calculated by subtracting the value of the real and measured performances. Because we only care about the significance, the value of noise is always chosen positive. For example, the noise value equaling to seven in Fig. 3b is calculated by subtracting 18 and 11. There are two noise levels, or what is called *Real Noise Level* (*RNL*) and *Estimated Noise Level* (*ENL*). They are calculated by the following expressions separately:

$$RNL(\theta_i) = RNL(\theta_i)/(Max(M(U)) - Min(M(U)) \tag{8a}$$

$$ENL(\theta_i) = RNL(\theta_i)/(Max(M'(U)) - Min(M'(U)) \tag{8b}$$

$$NL(U) = Max(RNL(U)) \approx Max(ENL(U)). \tag{8c}$$

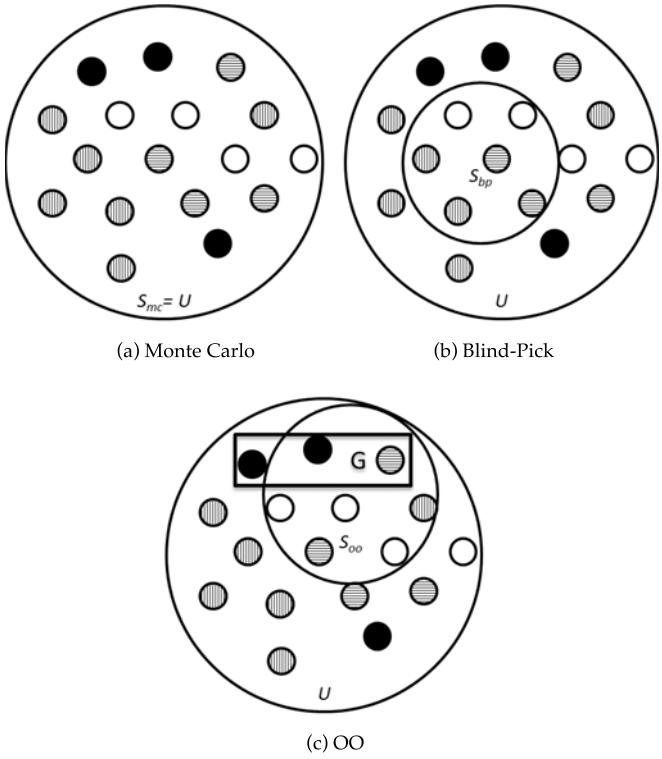
$Max(M(U)) = Max\{M(\theta_i), \theta_i \in U\}$. Similar definition applies to $Min(M(U))$, $Max(M'(U))$, $Min(M'(U))$, $NL(U)$, $RNL(U)$, $ENL(U)$. Because calculating the real performance $M(U)$ is time consuming, $M'(U)$ is used instead to estimate the noise level. The noise level is derived as the maximum among all the schedules as shown in Equation (8c). Therefore, the noise levels are 0.13 and 0.76 for Figs. 3b and 3c cases, respectively.

In Fig. 3d, we order the schedules based on the cases from Figs. 3b and 3c). Suppose that g is equal to 4 and k is equal to 3. In the small noise case, we need to get s to be equal to 3. In the moderate noise case, the value should be 6 as shown in the black rectangular box.

The affectiveness of OO, as we can see from the case study, is in the reduction of the evaluation set from U, which has eight schedules, to a set of three schedules S in the small noise level case. The value becomes 6 in the moderate noise case. Eventually, it reduces the computing overhead and promises to deliver good schedules. The OO method avoids the precise but slow search among millions of schedules to be simulated in real-life applications.

2.2.2 Monte Carlo, Blind-Pick and OO Method

Monte Carlo and Blind-Pick methods are used to compare with the OO method. In Fig. 4, Search spaces for the three known methods are illustrated.



(a) Monte Carlo (b) Blind-Pick

(c) OO

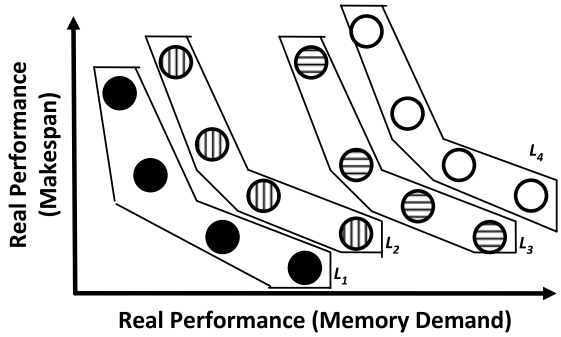
Fig. 4. The variation in the search space in three workflow scheduling methods.

Fig. 4a shows that the Monte Carlo method simulates each schedule in schedule set U for a long time (N runs for each). The selection set S_{mc} equals to the schedule set U. It can select the entire good-enough schedules, denoted by the *dark circles* in the set G, at the cost of intolerable runtime. However, the Blind-Pick method selects a random set S_{bp}. There is no guarantee that the good-enough schedules would be in its selection set. On the other hand, the OO takes a "sneak peek" at all of the schedules (n runs for each schedule in U), selects the most promising set S_{oo}, and applies the longer simulation (N runs for each schedule) to the schedules of S_{oo}. This results in good-enough schedules with significantly reduced overhead.

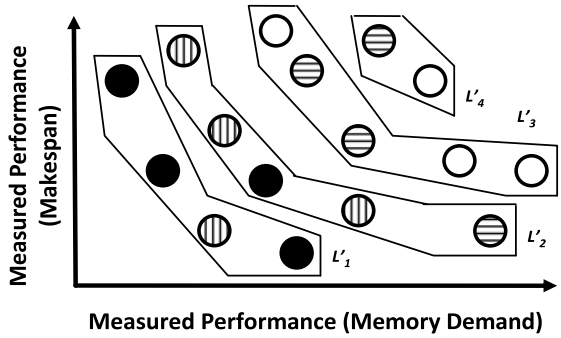
2.2.3 The OO Method: Bi-Objective Scenario

The OO method can also be applied to high dimensional space optimization problem. Fig. 5 illustrates a bi-objective optimization scenario, in which both the makespan and memory demands must be optimized within the two-dimensional optimization space. Each dot within the space corresponds to a working schedule that has been simulated.

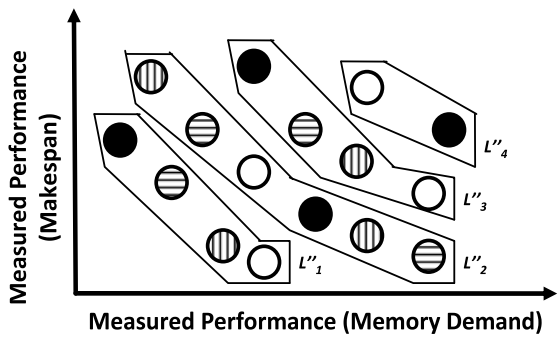
Through exhaustive simulation, the Monte Carlo method produces a set of "optimal" schedules along the skyline layer in Fig. 5a. These optimal choices are marked by the dark dots. Mathematically, there is no schedule within the 2-D space that is better or less than those dark schedules along the skyline, in terms of M and/or D. In other words, with a fixed memory demand, all of the skyline M are lower than those above the skyline. Similarly, with a fixed M, all D along the skyline are lower than those above the skyline. The above mentioned phenomenon of skyline schedules is known as a Pareto front that corresponds to the acceptance



(a) Monte Carlo method leads to optimal schedules in shading dots along the bottom skyline layer L1.



(b) Rough model of moderate noise. Suppose g = 4, k = 3, u = 16, then we have l = 1 which has s = 4 schedules to evaluate N times for each.



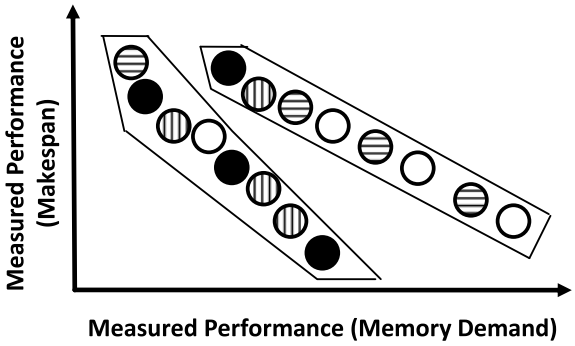
(c) Rough model of large noise. Suppose g = 4, k = 3, u = 16, then we have l = 3 which has s = 14 schedules to evaluate N times for each.

Fig. 5. Part (a) shows the exhaustive Monte Carlo method with optimal schedule produced. Part (b) applies a rough model to solve a bi-objective optimization problem with acceptable (suboptimal) schedules scattered over the entire search space. However, we still get two optimal schedules at the bottom skyline layer.

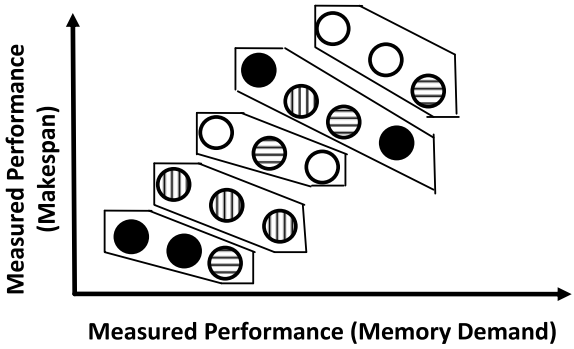
set G. If $\{L_1\}$ is removed, then $\{L_2\}$ can be achieved in the same way. By processing all of the schedules in such a way, the searching space can be divided into a series of Pareto fronts denoted by $\{L_1\}, \{L_2\}, \ldots, \{L\}$.

As shown in Figs. 5b and 5c with noises introduced, the good-enough schedules G are now spread in multiple layers of the skyline. With a small noise, two layers of the schedules cover the set G while a moderate noise requires four lays. Based on the discussion, the OO method determines the cardinality of layer $l = |L|$ to cover the required number, denoted by k, of the schedules in set G. Section 4 introduces using a rough model in evaluating the schedules along multiple layers of the skyline.

The rough model used in Fig. 5c results in much lower scheduling overhead. The acceptable schedules are scattered sparsely within the space. This may demand a larger



(a) Two layers. Suppose g = 4, k = 2, u = 16, then we have l = 1 which has s = 8 schedules to evaluate N times for each.



(b) Five layers. Suppose g = 4, k = 2, u = 16, then we have l = 1 which has s = 3 schedules to evaluate N times for each.

Fig. 6. The schedules are distributed differently in the measured performance plane. To derive at least two schedules, a search has to access eight schedules in (a) compared to three schedules in (b).

set S to cover the good-enough schedules. A smaller S is enough, but may demand heavier scheduling overhead, such as depicted in Fig. 5b. Therefore, tradeoff does exist between S and the scheduling overhead that a user can tolerate. In Monte Carlo, we simulate $N = 1{,}000$ times for each value of p_i and d_i to yield the expected M and D. Using our proposed IOO method, we significantly reduce the number of simulation runs. In fact, we have $n = 10$ runs using the rough model described in Fig. 5b.

Other than the noise level that determines the selection set S, the distribution of the schedules in the measured performance plane also matters. An example is given in Fig. 6. Suppose it is required to find at least two shading schedules, at least one layer is required in both cases. In Fig. 6a, we need to assess eight schedules compared to three schedules in Fig. 6b. Such a schedule distribution is technically called the Ordered Performance Curve (OPC).

The OPC combined with the noise level, can be used to calculate the number of layers to compose the set S, which contains the required two shading schedules. The detailed expression for composing the set S can be found in [41] and due to the constricted space, it will not be repeated here. A well-established OPC with a satisfactory noise level leads to the advantage of the proposed IOO approach in the LIGO gravitational wave data analysis.

3 Iterative Ordinal Optimization

The proposed IOO method is specified below to generate the optimal workflow schedules in the VCs with dynamic workload. The scheduling solutions are generated in an iterative fashion. During each of the iteration, suboptimal or good-enough schedules are obtained via OO-based bi-objective scheduling. The IOO method adapts to the system dynamism pertaining to the fluctuations in the workload and resource provisioning in VCs.

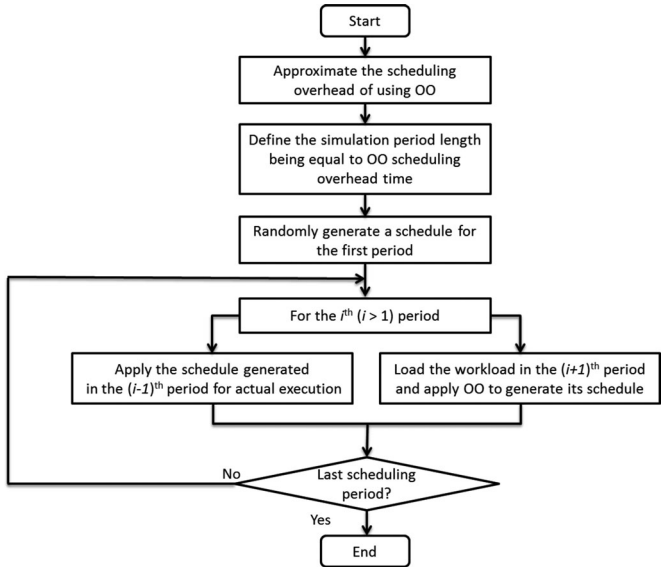


Fig. 7. Flow chart of the new iterative ordinal optimization (IOO) for suboptimal schedule generation.

3.1 The IOO Flowchart

The procedure of applying IOO in real-time multi-period multi-tasking workflow scheduling is reported in Fig. 7.

Since IOO is essentially based on OO for each scheduling period, we use the word OO in the flowchart. The input of IOO is an known workload with its time length being W_L. The outputs of IOO are a set of time-series schedules, each schedule being in a form of Equation (3). The time interval between two such schedules is the execution time needed to use OO to generate a schedule. We also call this time interval as the overhead of applying OO method. Suppose the overhead is H, then the total number of schedules can be calculated as W_L/H.

Therefore, IOO estimates the OO overhead in each of its scheduling period as a start. Then IOO partitions the whole scheduling period W_L into such equal-length sub-periods, with each sub-period being the length of H. For each period, e.g. the ith period in the flowchart, IOO primarily works on two steps in parallel as follows.

1) IOO applies the schedule generated from its period (i-1)th to execute the workload at the current period.
2) IOO takes the workload of its next period (i+1)th, and performs the simulation process over all the possible schedules, to generate an optimized schedule by using the OO method.

Notice that the above steps (1) and (2) are executed in parallel, instead of all the time-series schedules are generated in advance because we need to tailor the solution to real-time scheduling purpose. For example, we know the new workload generated at the *10*th schedule period at the beginning, but we don't know the runtime workload unfinished from the 9th schedule period, which needs to be rolled

over to the 10th period. Therefore, we have to wait until the 9th period in order to get fair workload estimation. However, if we assume all the workload in a schedule period can be finished, then we can calculate all the time-series schedules in advance and apply them one by one in real scheduling scenario.

In Fig. 7, we illustrate the IOO process being divided into two layers of simulation. During each iteration, IOO applies the schedule generated during the last iteration for execution of the workload. Meanwhile, the OO method is applied at the inner loop for fast generation of sub-optimal schedules for the next iteration.

Monte Carlo and Blind-Pick are also processed in such an iterative way. Therefore, the overhead becomes either Monte Carlo or Blind-pick method in the second and third step in Fig. 7 above, and the schedule method is also accordingly replaced.

3.2 Complexity Analysis

We use two metrics in our performance analysis. The first metric is the throughput *T* defined as

$$T = N_t/(t_1 - t_0), \tag{9}$$

where N_t is the total number of tasks completed within the time interval $[t_0, t_1]$. The second metric is *scheduling overhead* H that is defined as the simulation time used to schedule VM resources for mapping a scientific workflow to a virtualized cloud platform. The scheduling overhead *H* is composed of two components, namely:

$$H = H_1 + H_2, \tag{10}$$

where H_1 is the time to select the set *U* or *S* of candidate schedules to simulate its performance (*n*-time evaluation phase), and H_2 is the time to apply the precise but slow simulation to evaluate all of the selected schedules (*N*-time evaluation phase).

We assess below the overhead to simulate any schedule θ_j in using the Monte Carlo method. Based on the flow chart in Fig. 7, the Monte Carlo process contains the following five steps:

1) Generate the values of d_i and p_i with time h_1.
2) Simulate θ_j using (d_i, p_i) with time h_2.
3) Compute the throughput and memory demands with time h_3.
4) Go to Step (1) for *N* or *n* loops with time h_4.
5) Calculate the average throughput and memory demand for all *N* simulations with time h_5.

In Table 1, we defined *h* as the time to perform one slow simulation on a given schedule. The variable *N* is the number of repeated simulation performed per schedule and *n* is the reduced simulation number used in the IOO method. The parameters are $u = |U|$, $g = |G|$, and $s_{bp} = |S_{bp}|$ for the Blind-Pick method, and $s = |S|$ for the IOO method, respectively. The parameter $k = |G \cap S|$. In the following theorem, we analyze the time complexities of the three workflow scheduling methods considered in this paper.

Theorem 1. *The overhead time* H *to simulate all of the selected schedules within the three workflow scheduling schemes can be represented as*

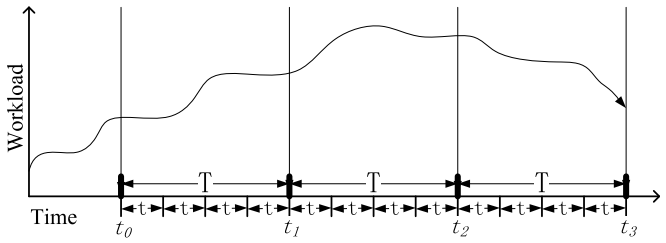


Fig. 8. IOO adaptability to dynamic workload variations.

$$H_{mc} = Nuh, \tag{11a}$$

$$H_{bp} = Nkuh/g, \tag{11b}$$

$$H_{ioo} = nuh + Nhs, \tag{11c}$$

where all of the input parameters are defined in Table 1.

Proof. We have $H_{mc} = u[N(h_1 + h_2 + h_3 + h_4) + h_5]$. The variables h_1, h_3, h_4, and h_5 terms are negligible, when compared with the much greater magnitude of h_2. In Eq. (11a), we simply denote $h = h_2$. We have $H_1 = 0$ for the Blind-Pick method. By using the results described in [17], we have the probability distribution for the intersection set $P\big[|G \cap S_{bp}| = k\big] = C_g^k C_{u-g}^{s_{bp}-i} / C_u^{s_{bp}}$. Therefore, we have $Exp[|G \cap S_{bp}|] = gs_{bp}/u$. To simulate $s_{bp} = ku/g$ schedules, it takes $H_{bp} = H_2 = Nkuh/g$ time. For the IOO method, we have $H_1 = unh$, and $H_2 = Nhs$, leading to the expression for H_{ioo}. □

The following theorem proves the speedup of the proposed IOO method, compared with the other two known methods.

Theorem 2: *Using simulation to generate the suboptimal schedules, the IOO scheduling method is*

$$R_1 = H_{mc}/H_{ioo} = 1/(n/N + s/u), \tag{12a}$$

$$R_2 = H_{bp}/H_{ioo} = (k/g)/(n/N + s/u), \tag{12b}$$

times faster than the Monte Carlo method and the Blind-Pick method, respectively.

Proof. Consider the speedup ratio $R_1 = H_{mc}/H_{ioo} = Nuh/(nuh + Nhs) = 1/(n/N + s/u)$ and the speedup ratio $R_2 = H_{bp}/H_{ioo} = (Nkuh/g)/(nuh + Nhs) = (k/g)/(n/N + s/u)$. R_1 and R_2 are thus proven. □

The following corollary gives the speedup of our proposed IOO method when $u > s$.

Corollary 1. *When $u > s$, the second term in both of the denominators of R_1 and R_2 tends to become zero, and we have the speedup values $R_1 \approx N/n$ and $R_2 \approx Nk/ng$.*

Because a user must define the magnitude of g and k under the constraint $u > s$, in our simulation experiments, we had $u = 27{,}132$, $N = 1{,}000$, $s = 190$, $n = 10$, $g = 10$, and $k = 1$ for the LIGO workload. Therefore, the overhead reduction ratios of the IOO method are $R_1 \approx 100$ and $R_2 \approx 10$ times over the Monte Carlo and Blind-Pick, respectively. We will verify these analytical results with the simulation results to be reported in Sections 4.3.1.

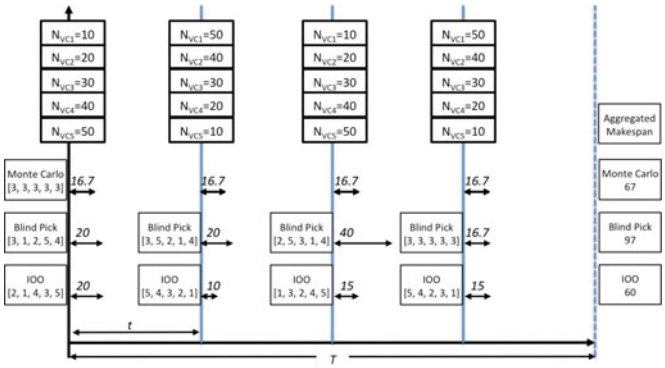


Fig. 9. An example showing the timing advantage of IOO.

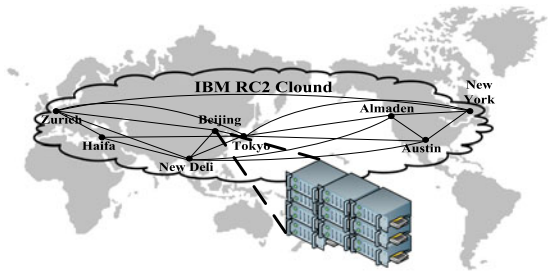


Fig. 10. Research compute cloud (RC2) over eight IBM Research and Development Centers.

3.3 IOO Advantages

We illustrate the rationale behind the IOO method in Fig. 8. Let T be the scheduling overhead using the Monte Carlo method, and t be that of the IOO method. For example, at time t_0, Monte Carlo is used for simulation. It is not until t_1 that the Monte Carlo can generate its optimal schedule. While the solution is optimized at t_1, it is not possible to generate such an optimized schedule between t_1 and t_2. As for the proposed IOO method, at time t_1, the predicted workload is used to generate a suboptimal schedule at time $t_1 + t$, and then at $t_1 + 2t, \ldots$, and so on.

This process is continued at each of the period to capture the variation in the workload in a much finer granularity to improve the performance. The IOO is carried out dynamically to iteratively upgrade the performance. During each iteration, the workflow scheduling follows the OO method to reduce the overhead and generate a good-enough solution. From a global point of view, the successive iterations are processed fast enough to absorb the dynamism of the system, and the overall performance is improved.

A further simplified example is given in Fig. 9 to show how IOO can be advantageous when being applied to such a multi-period scheduling problem.

Suppose a scheduling platform use 15 VMs to process five different task classes. Four batches of workloads arrive during the scheduling period of the Monte Carlo method. For example, $N_{VC1} = 10$ at the start point means there are 10 independent tasks of task class one arrive at VC1. We can see that the workload is very fluctuating due to the big variance between two consecutive workloads.

Monte Carlo, Blind-Pick and the IOO methods use their selected schedule. Take the first batch of workload as an example, Blind-Pick chooses a schedule [3, 1, 2, 5, 4] that indicates that the VC1 has three VMs, VC2 has one VM, etc. Monte Carlo applies a same schedule over all of the four periods.

For simplicity, we assume that each of the VM processes each task at a fixed rate that equals to one task per second. The performance metric used here is the aggregated makespan of the all the scheduling periods. For each period, the period-makespan is calculated by the time to finish the slowest task class. For example, the makespan of using Blind-Pick method at the first batch of workload is calculated by $\text{Max}\{10/3, 20/1, 30/2, 40/5, 50/4\} = 20$.

Aggregating the four scheduling periods, the total makespan of using the Blind-Pick method comes to 97 seconds. Similarly, the aggregated makespan of using the Monte Carlo and the IOO methods are 67 and 60 seconds, respectively. The IOO shows its advantage in this case.

As we can deduce from the workload characteristics, the best schedule for all the four periods should be [1, 2, 3, 4, 5], [5, 4, 3, 2, 1], [1, 2, 3, 4, 5], [5, 4, 3, 2, 1] respectively. The IOO probably cannot achieve all of the best schedules due to the reduced simulation time t. However, the IOO still selects the schedules very similar to the sequence and leads to an overall best performance.

4 LIGO Workflow Experiments

In this section, we design the LIGO experiments used to test the effectiveness of the proposed IOO method for scientific workflow. First, we introduce the experimental settings. Thereafter, we examine the LIGO task classes and task parallelism.

4.1 Experimental Settings

The cloud simulations/experiments were carried out using 10 servers of IBM RC2 Cloud at the IBM China Development Laboratory, Beijing (see Fig. 10) similar to the Amazon Web Service [3].

Each sever is equipped with Intel Xeon MP 7150N processor and 24 GB memory. The virtualized physical servers in the IBM Beijing Center are specified in Table 2. We installed up to fourteen VMs per physical server. A physical server runs with the OpenSuSE11/OS. All of the LIGO tasks were written in Java. With ten servers, we experimented with 128 VM instances. To test the scalability of the various scheduling techniques, we vary the VC configuration from sixteen to 32, 64, and 128 VMs.

4.2 LIGO Verification Workflow

The LIGO project was designed for the detection of gravitational waves on earth surface. This is a large-scale scientific

TABLE 2
Virtualized Physical Cluster

Cluster Size	10 servers per physical cluster
Node Architecture	IBM X3950 Server built with 16-core Xeon MP 7150N, 24 GB memory running with the OpenSuSE 11 OS
VM Architecture	CPU: 1 Core with 1 GB memory running with OS: OpenSuSE 11
VMs/server	14 VMs in each server

TABLE 3
Task Classes in a LIGO Workflow

Task Class	Functional Characteristics	Parallel Tasks
Class-1	Operations after tinplating	3,576
Class-2	Restraints of interferometers	2,755
Class-3	Integrity contingency	5,114
Class-4	Inevitability of contingency	1,026
Class-5	Service reachability	4,962
Class-6	Service Terminatability	792
Class-7	Variable garbage collection	226

experiment as predicted by Einstein's general theory of relativity a century ago. The LIGO workflow demands an identification of the potential faults before the actual program execution. Our analysis was performed over different verification logic demands. Each verification logic or task class contains many subtasks over massive data sets.

The workload involved is divided into seven task classes, as listed in Table 3. It embodies analysis of three sensitive detectors (L1, H1, and H2) on the earth surface. Sufficient cloud resources (VMs) are provisioned to satisfy the LIGO workflows. The seven independent task classes can be executed in parallel on their own VCs.

Task Class 1 executes a template bank (*TmpltBank*) in two major steps (*Inspiral* and *TrigBank*). The Class 2 matches the expected wave of H2 (*Inspiral_H2*) until both data in H1 and H2 pass two contingency tests. The Class 3 minimizes the noise signal ratio by testing the data collected. This task has the highest degree of parallel (DOP). Task Class 4 continues the process of matching the expected waves to create template banks.

The Class 5 ensures that all of the services are reachable. This task also has a high DOP. Class 6 ensures that all of the services are fermentable with limited DOP. Finally, the Class 7 collects the garbage of used intermediate variables that has the lowest DOP. We want to find a range of solutions to use θ_j to minimize both M and D defined in Table 1. In our LIGO experiments, there are seven task classes running on 20 VMs. There are 27,132 schedules in total to be evaluated.

4.3 Comparative Results and Discussions

In this section, we report the simulation/experimental results. First, we show that the measured scheduling overhead of applying the IOO approach. Thereafter, we report the measured makespan and memory demands in the LIGO workload experiments.

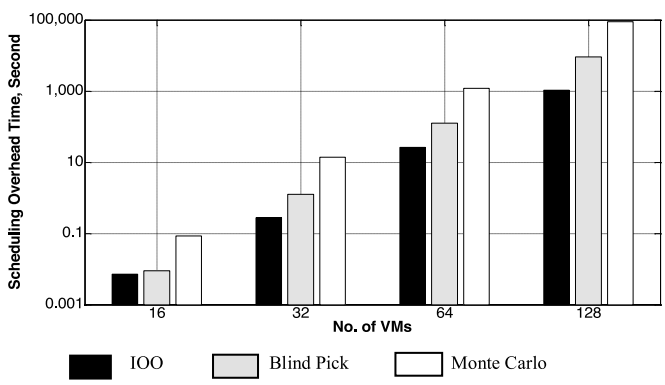


Fig. 11. Simulation overhead times of three workflow scheduling methods under 10 percent good-enough schedules.

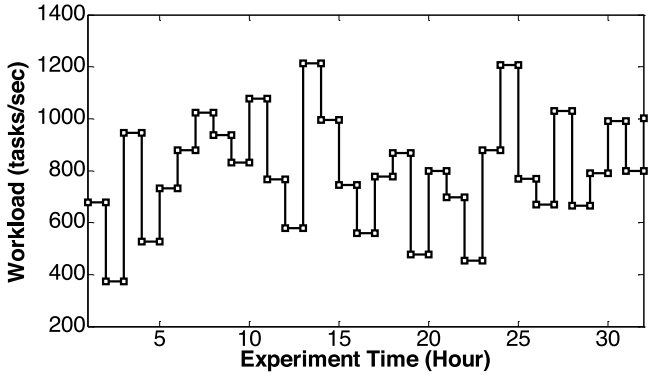


Fig. 12. Illustration of the dynamic workload in a single iteration.

4.3.1 Scheduling Overhead

The Monte Carlo simulations exhaust the entire scheduling space. For each schedule, one must implement all of the task classes on all VCs. The time used for this exhaustive search causes a great amount of scheduling overhead. The proposed IOO method requires the least amount scheduling overhead compared to the rest of the techniques, as demonstrated in Fig. 11. The schedule is generated by averaging over a small set of schedules.

Our proposed IOO method avoids the exhaustive search experienced by using the Monte Carlo method. We search in a much smaller schedule space. Good-enough schedules are found in a few iterations of each OO process. As shown in Fig. 11, the IOO performance for the workflow scheduling is evaluated by comparing the overhead times among three methods for variable number of VMs used. This comparison is made under the assumption of $k/g = 10\%$ good-enough schedules to apply the IOO method.

As the cluster size varies from sixteen to 128 VMs, we observe that the IOO method has a scheduling overhead of 1,000 sec, compared to 100,000 sec, with the Monte Carlo method. The scheduling overhead of Blind-Pick is slightly lower than the Monte Carlo method, but still much higher than the IOO method. As the number of VMs increases, the trade-off space also increases. It is noteworthy to mention that these results stay within the theoretical bounds set in Theorem 1.

4.3.2 Throughput and Memory Demand

To further investigate experiment details, a truncated illustration of dynamic workload is included in Fig. 12 and the corresponding experimental results are included in Fig. 13.

The number of tasks and virtual machines are default. The simulation time of Monte Carlo is 32 hours that is taken as the period time. Only two periods of data are shown in Fig. 13. Dynamically changing of workload and corresponding scheduling results are illustrated. In Fig. 13, throughput and memory demand of all the methods are calculated for each stage. The proposed IOO method does not have better performance in every iteration.

While at the beginning of each iteration the Monte Carlo can generate a best schedule, within each iteration, it is obvious that the Monte Carlo shows the worst adaptability.

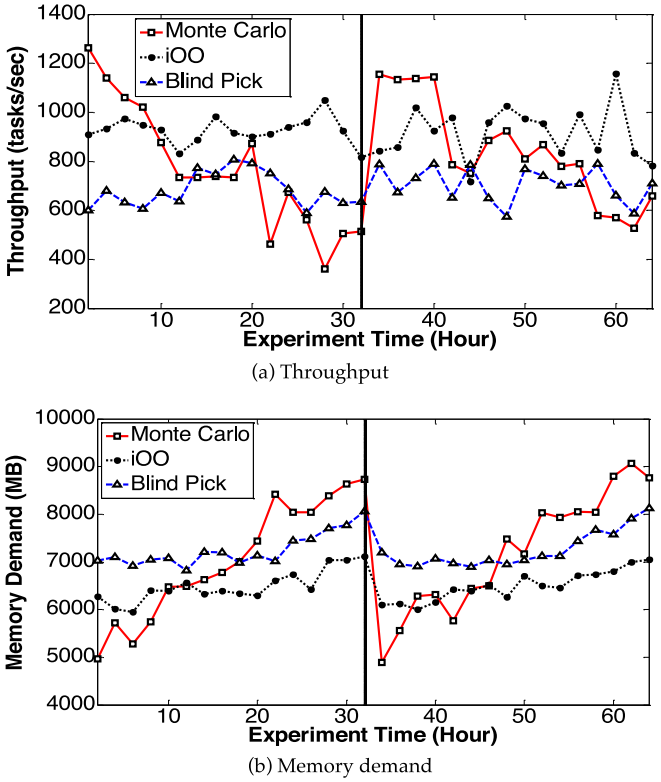


Fig. 13. Throughput and memory results running the LIGO verification workflow on the RC2 cloud platform.

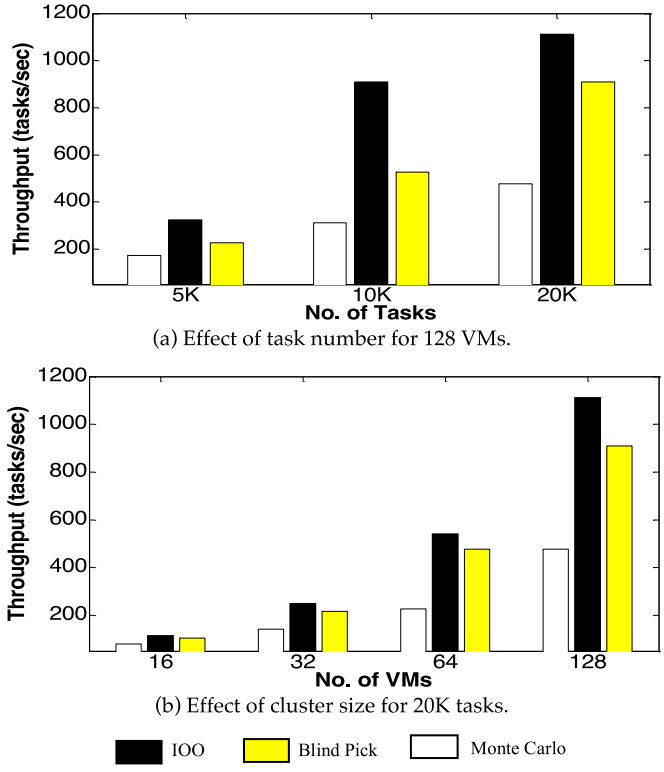


Fig. 14. Relative performance of three workflow scheduling methods plotted against task and cluster sizes.

This is due to the fact that it cannot generate new schedules fast enough to adapt to new workload. On the other hand, the proposed IOO method maintains the exact same performance level during all stages of an iteration. The IOO method's performance gain can be better illustrated from a global perspective, detailed below.

We present in Fig. 14, the average throughput and memory demands during the whole experiment period. The experiments were carried out in eight simulation periods. Each period lasts a time length of h of a single Monte Carlo simulation. During each period, the OO method is applied iteratively, as the IOO scheduling time is much shorter. The default workload contains 20,000 LIGO tasks and 128 VMs.

In Fig. 14a, we demonstrate the effect of the increase in the number of tasks in the LIGO application on the system performance. The proposed IOO method demonstrates approximately three times higher throughput than the Monte Carlo method with the variation in the number of tasks. The proposed IOO method offers 20 to 40 percent times higher throughput than that of Blind-Pick as the task number varies.

Fig. 14b shows the effects of the VC size on the throughput performance of the three scheduling methods. We observed a 2.2 to 3.5 times throughput gain by the IOO over Monte Carlo as the VC size increases from sixteen to 128 VMs. The IOO method exhibited approximately 15 to 30 percent throughput gain over Blind-Pick as the VC size increases.

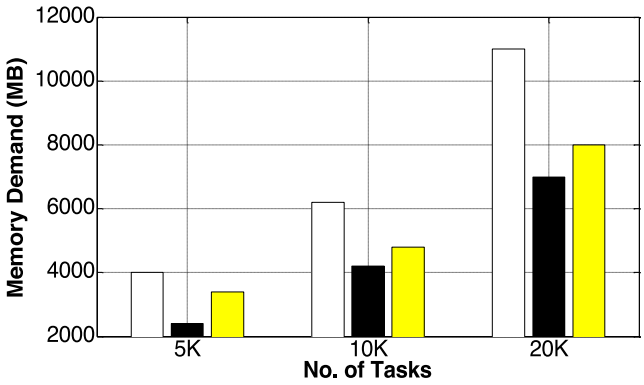
Fig. 15 shows the relative memory demands of the three workflow scheduling methods. The results are plotted as a function of the task number and cluster size. From the results we can observe that the Monte Carlo has the highest memory demand, the IOO method requires the least memory. The Blind-Pick method sits in the middle. In Fig. 15a, the IOO method saved about 45 percent memory from that demanded by the Monte Carlo method. The Blind-Pick method required about 20 percent higher memory than the IOO method as the task number increases.

In Fig. 15b for 20,000 tasks, the memory demands were 11.5 GB, 8 GB, and 7 GB on 128 VMs for the Monte Carlo, Blind-Pick, and IOO methods, respectively. The reported results indicate that the IOO outperforms the two other competing methods by providing a low-overhead scheduling solution. The reduced overhead leads to less simulation and profiling time that plays a key role in offering shorter and finer granularity schedule periods in real runs. These cater to the need for scheduling fluctuating workflow.

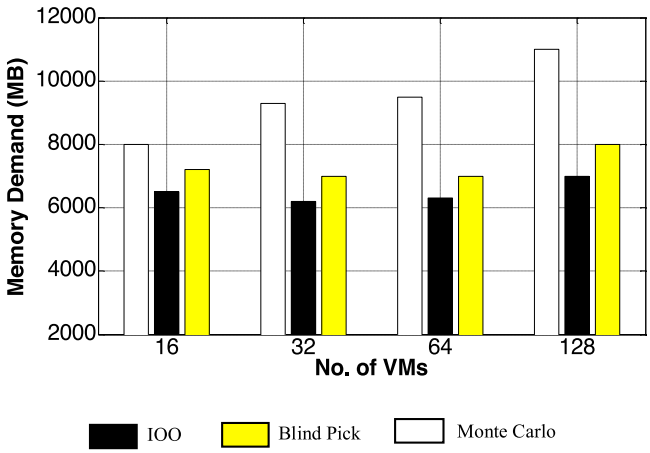
5 Related Work

In this section, we review related work on cloud resource provisioning and task scheduling, compared to our proposed IOO approach to solving the same problem. Scheduling large-scale scientific tasks on supercomputers or grid systems has been studied by many researchers in the past. In particular, we see a growing interest in [2], [10], [11], [12], [13], [18], [22], [23], [26], [29], [30],[31], [35], [37], [38]. There is an escalating interest on resource allocation for scientific workflows on computing clouds [9], [15], [27], [32].

Many classical optimization methods, such as opportunistic load balance, minimum execution time, and minimum completion time, are described in [11]. Several heuristics, such as sufferage, min-min, max-min, and auction have been proposed in the past [31].



(a) Effect of task number for 128 VMs.



(b) Effect of cluster size for 20,000 tasks.

Fig. 15. Memory demands of three workflow scheduling methods plotted over variable tasks and cluster sizes.

Yu and Buyya [38] proposed economy-based methods to handle large-scale grid workflow scheduling under deadline constraints, budget allocation, and QoS. Benoit et al. [5] designed resource-aware allocation strategies for divisible loads. Li and Buyya [23] proposed model-driven simulation of grid scheduling strategies. Lu and Zomaya [26] proposed a hybrid scheduling method for job scheduling in heterogeneous computational grids.

These scheduling approaches in computational grids [12] serve different purpose than the methods proposed in the autonomic big-data cloud platform. The cloud scheduling methods, such as IOO, partition and allocate computing resource in an elastic manner to improve the throughput of multitasking workloads.

In 1992, Ho et al. [14] proposed the OO method for discrete-event dynamic systems. In their work, they demonstrated that the OO method is effective to generate a soft or suboptimal solution to most NP-hard problems. The OO technique has been applied in advanced automation and industrial manufacturing [18], [19], [33].

Wieczorek et al. [36] analyzed the five facets that may have a major impact on the selection of an appropriate scheduling strategy. They proposed taxonomies to classify multi-objective workflow scheduling schemes. Prodan and Wieczorek [29] proposed a dynamic algorithm, which outperforms the SOLOS and BDLS methods to optimize bi-criteria problems.

TABLE 4
Summary of Three Workflow Scheduling Methods

Method	Strength and Advantages	Weakness and Limitations
Monte Carlo	High accuracy to obtain optimal schedule. Monte Carlo results in high system throughput with reduced memory demand for a fixed and long scheduling period	High simulation overhead due to exhaustive search for optimality, The method does not adapt to fast variation in workload. The performance will degrade with extended scheduling periods.
Blind-Pick	With moderate overhead, this method applies to a reduced search space and can adapt to the fast variation in workload to some extent.	Moderate accuracy due to lower overhead. With a poor selection set, the performance will degrade to that of Monte Carlo.
IOO	With low overhead, the IOO can adapt to the fast variation in workload to obtain suboptimal schedules that run with high multitasking throughput and reduced memory demand	The suboptimal schedules generated at each period may not be as optimal as that generated by Monte Carlo. Under a high noise level, the IOO-generated schedule may degrade.

In the past, Cao et al. [7], [39] have studied the LIGO problems on the grid environments. Duan et al. [12] suggested a game-theoretic optimization method. Dogan and Özgüner [11] developed a matching and scheduling algorithm. Smith et al. [31] have proposed robust static resource allocation for distributed computing systems operating under imposed QoS constraints. None of these methods investigated the profiles and runtime system performance. Our proposed IOO method fills the gap.

Runtime uncertainty is handled in Batista's work [4]. Our work inherits the idea of OO, which reduces scheduling overhead by narrowing down the search space. Along the OO line, many other heuristic methods have been proposed [21], [34], [41]. These methods quickly reduce the subset of "good enough" solutions with manageable overhead.

The OO is specifically designed to solve large problems in automated manufacturing, communication, power systems, and distributed computing systems [39]. The simulation based optimization tenet in our IOO is directly inspired by these real-life applications.

Different the selection rules for the OO are compared in [17] to discuss the relative performance. The consesus is that no selection rule is absolutely better than the others in all applications. We apply the OO method in an iterative way to dynamically optimize cloud scheduling or provisioning scenario to meet special requirements of scientific workflows, such as LIGO.

6 Conclusions

This paper offers a first attempt to an iterative application of the OO method for fast dynamic multitask workload scheduling in a cloud computing platform. The major advantage of the IOO method resides in its adaptability to a scenario with fluctuating workloads. The IOO method is compared with Monte Carlo and Blind-Pick. The conclusions of our findings are summarized in Table 4.

The major contributions of this paper are summarized below in four technical aspects.

1) The IOO method worked very well on a cloud platform under dynamically changing workload. We reduced the search space from a very large space of 27,132 candidate schedules to a much smaller search space of 190 schedules. The low-overhead IOO method adapted to the workload variations. This method captured the workload dynamics to make fast scheduling decisions.
2) Compared with what we had reported in the CloudCom 2011 conference paper [39], we performed a thorough theoretical time/space complexity analysis for the three comparative scheduling methods. We also quantitatively proved the adaptability of our IOO method. Simulation/experimental results also echoed the theoretical claims.
3) Large-scale LIGO gravitational wave data analysis pipelines are used to effectively test the new IOO method. The verification workflow of LIGO data analysis offered a typical multitask application that required real-time dynamic task scheduling support on cloud platforms.
4 We provided an efficient and effective profiling and simulation method for multitask workload scheduling in a virtualized cloud platform. The cloud service environments contained many uncertainty factors that were dealt with appropriately by the proposed IOO method. Our IOO method applied well in EC2-like cloud services to increase the throughput and in S3-like services to reduce the memory demands.

Acknowledgments

This work was supported in part by Ministry of Science and Technology of China under National 973 Basic Research Program (grants No. 2013CB228206 and No. 2011CB302505), National Natural Science Foundation of China (grant No. 61472200 and No. 61233016) and National Science Foundation under grant CCF-1016966. The work was also partially supported by an IBM Fellowship for Fan Zhang, and by the Intellectual Ventures endowment to Tsinghua University. This work was carried out when the author was with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China.

References

[1] A. Abramovici, W. E. Althouse, W. E. Althouse, R. W. P. Drever, Y. Gürsel, S. Kawamura, F. J. Raab, D. Shoemaker, L. Sievers, R. E. Spero, K. S. Thorne, R. E. Vogt, R. Weiss, S. E. Whitcomb, and M. E. Zucker, "LIGO: The laser interferometer gravitational-wave observatory," *Science*, vol. 256, no. 5055, pp. 325–333, 1992.
[2] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. Parallel Distrib. Syst.* vol. 23, no. 8, pp. 1400–1414, Aug. 2012.
[3] Amazon EC2 and S3, Elastic compute cloud (ec2) and simple scalable storage (s3) [Online]. Available: http://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud
[4] D. M. Batista and N. L. S. da Fonseca, "Scheduling grid tasks in face of uncertain communication demands," *IEEE Trans. Netw. Serv. Manage.*, vol. 8, no. 2, pp. 92–103, Jun. 2011.
[5] A. Benoit, L. Marchal, J. Pineau, Y. Robert, and F. Vivien, "Resource-aware allocation strategies for divisible loads on large-scale systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Rome, 2009, pp. 1–9.
[6] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis," in *Proc. Workflows eScience.: Scientific. Workflows Grids*, 2007, pp. 39–59.
[7] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: Workflow management for grid computing," in *Proc. 3rd IEEE/ACM Int. Symp. Clust. Comput. Grid*, Tokyo, Japan, 2003, pp. 198–205.
[8] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, "GriPhyN and LIGO, Building a virtual data grid for gravitational wave scientists," in *Proc. 11th IEEE Int. Symp. High Perform. Distrib. Comput.*, 2002, pp. 225–234.
[9] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in *Proc. ACM/IEEE Conf. Supercomput.*, Austin, TX, USA, 2008, pp. 1–12.
[10] P. Delias, A. Doulamis, N. Doulamis, and N. Matsatsinis, "Optimizing resource conflicts in workflow management systems", *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 3, pp. 417–432, Mar. 2011.
[11] A. Dogan and F. Özgüner, "Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, 2005.
[12] R. Duan, R. Prodan, and T. Fahringer, "Performance and cost optimization for multiple large-scale grid workflow applications," in *Proc. IEEE/ACM Int. Conf. SuperComput.*, 2007, p. 12.
[13] L. Golubchik and J. Lui, "Bounding of performance measures for threshold-baded systems: Theory and application to dynamic resource management in video-on-demand servers," *IEEE Trans. Comput.*, vol. 51, no. 4, pp. 353–372, Apr. 2002.
[14] Y. C. Ho, Q. C. Zhao, and Q. S. Jia, *Ordinal Optimization, Soft Optimization for Hard problems*. New York, NY, USA: Springer, 2007.
[15] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *Proc. IEEE Int. Conf. eScience*, Dec. 2008, pp. 640–645
[16] K. Hwang, G. Fox, and J. Dongarra, *Distributed and Cloud Computing Systems: From Parallel Processing to The Internet of Things*. San Mateo, CA USA: Morgan Kauffmann, 2011
[17] Q. S. Jia, Y. C. Ho, and Q. C. Zhao, "Comparison of selection rules for ordinal optimization," *Math. Comput. Modell.*, vol. 43, no. 9, pp. 1150–1171, 2006.
[18] J. Kolodziej and S. U. Khan, "Multi-level Hierarchical genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment," *Inf. Sci.*, vol. 214, pp. 1–19, 2012.
[19] J. Kolodziej, S. U. Khan, L. Wang, M. Kisiel-Dorohinicki, S. A. Madani, E. Niewiadomska-Szynkiewicz, A. Y. Zomaya, and C.-Z. Xu, "Security, energy, and performance-aware resource allocation mechanisms for computational grids," *Future Generation Comput. Syst.*, vol. 31, pp. 77–92, 2014.
[20] A. Lazzarini. Data from the LIGO I Science Run [Online]. Available: http://www.ligo.caltech.edu/docs/P/P010002-00(P010002-00.pdf, 2003.
[21] D. Li, L. H. Lee, and Y. C. Ho, "Constrained ordinal optimization," *Inf. Sci.*, vol. 148, nos. 1–4, pp. 201–220, 2002.
[22] H. Li, "Realistic workload modeling and its performance impacts in large-scale escience grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 480–493, Apr. 2010.
[23] H. Li and R. Buyya, "Model-driven simulation of grid scheduling strategies," in *Proc. 3rd IEEE Int. Conf. e-Science Grid Comput.*, 2007, pp. 287–294.
[24] LIGO and Virgo Collaborations, "Application of a hough search for continuous gravitational waves on data from the 5th ligo science run," General Relativity and Quantum Cosmology, arXiv:1311.2409v2., 2013.
[25] S. Y. Lin, Y. C. Ho, and C. H. Lin, "An ordinal optimization theory-based algorithm for solving the optimal power flow problem with discrete control variables," *IEEE Trans. Power Syst.* vol. 19, no. 1, pp. 276–286, Feb. 2004
[26] K. Lu and A. Y. Zomaya, "A hybrid schedule for job scheduling and load balancing in heterogeneous computational grids," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Jul. 2007, pp. 121–128.
[27] S. U. R. Malik, S. U. Khan, and S. K. Srinivasan, "Modeling and analysis of state-of-the-art vm-based cloud management platforms," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 50–63, Jan.–Jun. 2013.

[28] N. Metropolis and S. Ulam, "The monte carlo method," *J. Amer. Stat. Assoc.*, vol. 44, no. 247, pp. 335–341, 1949.
[29] R. Prodan and M. Wieczorek, "Bi-criteria scheduling of scientific grid workflows," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 364–376, 2010.
[30] X. Qin, W. Wang, and P. Mishra, "TCEC: Temperature and energy-constrained scheduling in real-time multitasking systems," *IEEE Trans. Comput.-Aid. Design Integr. Circuits Syst.*, vol. 31, no. 8, pp. 1159–1168, Aug. 2012
[31] J. Smith, H. J. Siegel, and A. A. Maciejewski, "A stochastic model for robust resource allocation in heterogeneous parallel and distributed computing systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Miami, FL, USA, 2008, pp. 1–5.
[32] T. S. Somasundaram and K. Govindarajan, "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud," *Future Generation Comput. Syst.*. vol. 74, no. 3, pp. 2152–2165, 2014.
[33] R. Subrata, A. Y. Zomaya, and B. Landfeldt, "A cooperative game framework for qos guided job allocation schemes in grids," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1413–1422, Oct. 2008.
[34] S. Teng, L. H. Lee, and E. P. Chew, "Multi-objective ordinal optimization for simulation optimization problems," *Automatica*, vol. 43, pp. 1884–1895, 2007.
[35] N. Tziritas, C.-Z. Xu, T. Loukopoulos, S. U. Khan, and Z. Yu, "Application-aware workload consolidation to minimize both energy consumption and network load in cloud environments," in *Proc. 42nd IEEE Int. Conf. Parallel Process.*, Lyon, France, 2013, pp. 449–457.
[36] M. Wieczorek, R. Prodan, and A. Hoheisel, "Taxonomies of the multi-criteria grid workflow scheduling problem" in *Grid Middleware and Services*. New York, NY, USA: Springer, 2007, pp. 237–264.
[37] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence windows," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 7, pp. 925–938, Jul. 2010.
[38] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Sci. Program.*, vol. 14, pp. 217–230, 2006.
[39] F. Zhang, J. Cao, K. Hwang, and C. Wu, "Ordinal optimized scheduling of scientific workflows in elastic compute clouds," in *Proc. 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Athens, Greece, Nov. 29-Dec. 1 2011, pp. 9–17.
[40] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Comput. Syst.*, vol. 37, pp. 309–330, 2014.
[41] Q. C. Zhao, Y. C. Ho, and Q. S. Jia, "Vector ordinal optimization," *J. Optimization Theory Appl.*, vol. 125, no. 2, pp. 259–274, May 2005.

Fan Zhang (SM'13) received the PhD degree in the Department of Control Science and Engineering, Tsinghua University in Jan. 2012. He is currently a postdoctoral associate with the Kavli Institute for Astrophysics and Space Research at Massachusetts Institute of Technology. He is also a sponsored researcher in Tsinghua University, Beijing, China. He has been appointed as a visiting associate professor in the Shenzhen Institute of advanced technology, Chinese Academy of Science since Jan 2014. From 2011 to 2013, he was a research scientist at Cloud Computing Laboratory, Carnegie Mellon University. He is a senior member of the IEEE. He received a Honorarium Research Funding Award from the University of Chicago and Argonne National Laboratory (2013), a Meritorious Service Award (2013) from *IEEE Transactions on Service Computing*, two IBM PhD Fellowship Awards (2010 and 2011). His research interests include big-data scientific computing applications, simulation-based optimization approaches, cloud computing, and novel programming models for streaming data applications on elastic cloud platforms.

Junwei Cao (SM'05) received the bachelor's and master's degrees in control theories and engineering in 1998 and 1996, respectively, both from Tsinghua University, Beijing, China. He received the PhD degree in computer science from the University of Warwick, Coventry, United Kingdom, in 2001. He is currently a professor and deputy director of Research Institute of Information Technology, Tsinghua University, Beijing, China. He is also director of Open Platform and Technology Division, Tsinghua National Laboratory for Information Science and Technology. Prior to joining Tsinghua University in 2006, he was a research scientist at MIT LIGO Laboratory and NEC Laboratories Europe for about 5 years. He has published more than 150 papers and cited by international scholars for over 6,000 times. He has authored or edited six books. His research is focused on distributed computing technologies and applications. He is a senior member of the IEEE Computer Society and a member of the ACM and CCF.

Kai Hwang (F'86) received the PhD degree from the University of California, Berkeley in 1972. He is a professor of electrical engineering and computer science, University of Southern California. He is also an EMC-endowed visiting chair professor at Tsinghua University, China. His has published eight books and 230 papers, which have been cited over 13,000 times with a citation h-index of 50. His latest book *Distributed and Cloud Computing* (with G. Fox and J. Dongarra) was published by Kaufmann in 2011 which has been translated to Chinese in 2013. He ia a fellow of the IEEE. He received the CFC Outstanding Achievement Award in 2004, the Founders Award from IEEE IPDPS-2011 and a Lifetime Achievement Award from IEEE Cloudcom-2012. He has served as the editor-in-chief of the *Journal of Parallel and Distributed Computing* for 28 years and delivered 35 keynote speeches in major IEEE/ACM Conferences. Presently, he serves on the editorial boards of *IEEE Transactions on Cloud Computing* and *International Journal of Big Data Intelligence*. He also co-chair the *Second ASE International Conference on Big Data Science, Social Computing, and Cybersecurity* held at Stanford University in May 27-29, 2014.

Keqin Li (SM'96) is a distinguished professor of computer science at the State University of New York. He is an Intellectual-Ventures endowed visiting chair professor at Tsinghua University, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has more than 300 research publications and has received several Best Paper Awards for his research work. He is currently on the editorial boards of *IEEE Transactions on Computers* and *IEEE Transactions on Cloud Computing*. He is a fellow of the IEEE.

Samee U. Khan (SM'12) received the PhD degree from the University of Texas at Arlington in 2007. He is an associate professor at the North Dakota State University. His research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in more than 225 publications with two receiving best paper awards. He is a fellow of the IET and a fellow of the BCS.