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, Senior Member, 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 for generation of 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 time 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% decrease in throughput and 70% increase in the Monte Carlo simulation takes 27.6 hours, 100 times longer time 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% decrease in throughput and 70% increase in memory demand in using the Monte Carlo method. Our LIGO experimental results clearly demonstrate the advantage of using the IOO method. 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, especially 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.

The LIGO raw data was collected at 7.9MB/s from more than 2,000 sampling points [20]. A majority of the raw data are disturbed by noisy sources, such as disturbance of an arrival train that adds a veracity of complication to the data sources [24]. To process the LIGO workload, parallel virtual machines (VMs) are provided as virtual clusters from large-scale data centers [16]. Virtual clusters (VCs) are elastic resources that can dynamically scale up or down.

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 a cloud platform, resource provisioning and stage 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 solving complex problems with a very large solution space. Subsequently, the authors demonstrated that the OO method is effective to generate a soft or suboptimal solution for most of the NP-hard problems. As an example, optimal power flow [25] is an NP-hard 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 multi-objective 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 models 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.

The initial idea of this paper was presented at the CloudCom2011 [39] with some preliminary results. This paper extends significantly from the conference paper with some theoretical proofs supported by an entirely new set of experimental results. 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 multi-stage 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 elaborate on the experimental results and discuss the various aspects pertaining to the IOO performance compared with Monte Carlo and Blind-Pick. Related work are reviewed in Section 5. Finally, we summarize our contributions and discuss future research in Section 6.

2 WORKFLOW SCHEDULING ON CLOUDS

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.

![Figure 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.](image)

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.

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

All of the VCs are distinguished by the index $i$. Let $p_i$ be the expected execution time of a single task within the $i$-th 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 $\delta_i$ 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:
\[ M = \max \{ t_1, t_2, \ldots, t_c \}, \]  

\[ D = \sum_{i}^{c} d_i \times v_i. \]  

where \( c \) virtual clusters are used and \( t_i = p_i \delta/v_i \). 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:

\[ M = \max \{ t_1, t_2, \ldots, t_c \}, \]  

\[ D = \sum_{i}^{c} d_i \times v_i. \]  

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

\[ \theta = [v_1, v_2, \ldots, v_c], \]  

where \( v_i \) is the number of VMs assigned within cluster \( V C_i \). At different time periods, different schedules may be applied. All of the candidate schedules at successive time periods form \( U \). The cardinality of \( U \) can be calculated by the following expression:

\[ u = (v - 1)!/(v! - c!(c - 1)!), \]  

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. 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 \( M(\theta) \) and memory demands \( D(\theta) \), jointly and simultaneously, i.e.,

\[ \min \{ M(\theta) \} \text{ and } \min \{ D(\theta) \} \]  

for all possible schedules \( \theta \) in the search space \( U \).

The formulae for the makespan and memory demand are given in Eq. (1)–Eq. (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 all of the physical machines are populated with as many VMs as assigned by the jobs by other cloud users, then the \( p_i \) must be much lower than usual. The aforementioned scenario will inevitably also lead to a higher value of \( d_i \) due to the 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 \( \theta \), 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 \( \theta \) to obtain the \( M(\theta) \) and \( D(\theta) \) in Eq. (5).

### 2.2 Simulation-based Scheduling Optimization

#### A) The OO Method

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

![Fig. 2. The concept of OO using a set S intersecting with the set G to yield a set G \cap S of \( k \) acceptable (good-enough) schedules.](image)

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 model is applied 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 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 \( a = 98\% \). Note that \( u >> s >> g >> k \).

The probability of obtaining \( k \) or more schedules is given by Eq. (6).

\[ \text{Probability} \{ |G \cap S| \geq k \} \geq a. \]  

In our previous work described in [40], a bi-objective optimization scenario of multitask scheduling on the cloud is resolved based on the OO method, in which both the makespan and memory demands must be optimized within the 2-dimensional optimization space. This is illustrated in Fig. 3.
Each dot within the space corresponds to a working schedule that has been simulated. The dark circles are acceptable good schedules, grey ones are medium, and white circles (schedules) have the lowest performance. The acceptable schedules are scattered sparsely within the space. This may demand a larger set $S$ to cover the schedules, corresponding to a larger number of skylines in a 2-dimensional optimization space. A smaller $S$, may be enough, but may demand heavier scheduling overhead. Therefore, tradeoff does exist between $S$ and the scheduling overhead that a user can tolerate. In this work, during each iteration of the optimization process, the OO is actually applied for the same bi-objective optimization.

**B) Monte Carlo and Blind-Pick**

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.

![Fig. 4. The variation in the search space in three workflow scheduling methods.](image)

Fig. 4(a) 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 all of the superior performing 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 the selection set. On the other hand, the OO takes a “sneak peek” at all of the schedules, selects the most promising set $S_{oo}$, and apply the longer simulation to the schedules of $S_{oo}$. This results in superior performing schedules with significantly reduced overhead.

### 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.

#### 3.1 The IOO Flowchart

In Fig. 5, we illustrate the IOO process is 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 a similar way with much more iteration times due to the use of a much enlarged search space.

![Fig. 5. Flow chart of the new iterative ordinal optimization (IOO) for suboptimal schedule generation.](image)

During each period, IOO applies the schedule generated during the last period for actual execution of the workload. Meanwhile, the OO method is applied for fast generation of sub-optimal schedules for the next period. It is noteworthy to mention that the Monte Carlo and Blind-Pick are also processed in a similar way; however, the time period is much longer due to the scheduling overhead of the methods are higher.

#### 3.2 Complexity Analysis

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

$$T = \frac{N_t}{(t_1 - t_0)}, \quad (7)$$

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:

$$H = H_1 + H_2, \quad (8)$$

where $H_1$ is the time to select the set $U$ or $S$ of candidate schedules to simulate its performance, and $H_2$ is the time to apply the precise but slow simulation to evaluate all of the selected schedules.

We assess below the overhead to simulate any schedule $\theta$ in using the Monte Carlo method. Based on the flow chart in Fig. 5, the Monte Carlo process contains the following five steps:

1. Generate the values of $d_i$ and $p_i$ with time $h_i$. 

- Approximate the scheduling overhead of using OO
- Define the simulation period length being equal to OO scheduling overhead time
- Randomly generate a schedule for the first period
- For the $P^{th}$ ($P > 1$) period
- Apply the schedule generated in the ($P-1$)th period for actual execution
- Load the workload in the ($P+1$)th period and apply OO to generate its schedule

- Last scheduling period?
- Yes
- End
(2) Simulate \( \theta \) using \((d, p)\) 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:

\[
H_{mc} = Nuh, \tag{9a}
\]
\[
H_{bp} = Nkuh/g, \tag{9b}
\]
\[
H_{ioo} = nhu + Nhs, \tag{9c}
\]

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

**Proof.** We have \( H_{mc} = u[N(h_1+ h_2+ h_3)+h_5]. \) The variables \( h_1, h_2, h_3, h_4 \), and \( h_5 \) terms are negligible, when compared with the much greater magnitude of \( h \). 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 \( P[G \cap S_{bp}] = k \leq C_s C_{bp}^{n} / C^n \).

Therefore, we have \( \text{Exp}[|G \cap S_{bp}|] = g s_{bp} / u \). To simulate \( s_{bp} = k u h / g \) schedules, it takes \( H_{bp} = H_2 = N k u h / g \) time. For the IOO method, we have \( H_1 = uni, \) and \( H_2 = N h s, \) leading to the expression for \( H_{ioo} \). \( \square \)

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

\[
R_1 = H_{mc} / H_{ioo} = 1/ (u/N + s/u), \tag{10a}
\]
\[
R_2 = H_{bp} / H_{ioo} = (k/g)/(u/N + s/u), \tag{10b}
\]
times faster than the Monte Carlo method and the Blind-Pick method, respectively.

**Proof.** Consider the speedup ratio \( R_1 = H_{mc} / H_{ioo} = N u h / (n u h + N h s) = 1/ (u/N + s/u), \) and the speedup ratio \( R_2 = H_{bp} / H_{ioo} = (N k u h / g)/(n u h + N h s) = (k/g)/(u/N + s/u). \) \( R_1 \) and \( R_2 \) are thus proven. \( \square \)

The following corollary gives the speedup of our proposed IOO method when \( u \gg s \).

**Corollary 1:** When \( u \gg s \), 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 \( u \gg s \), 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 4.3A.

### 3.3 IOO Advantages

We illustrate the rationale behind the IOO method in Fig. 6. 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 \( t_2 \), 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 \( t + 2t, \ldots, \) and so on.

![Fig. 6. IOO adaptability to dynamic workload variations.](image)

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. 7 to show how IOO can be advantageous when being applied to such a multi-period scheduling problem.

![Fig. 7. An example showing the timing advantage of IOO.](image)
Suppose a scheduling platform uses 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 3 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 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 \( \max\{10/3, 20/1, 30/2, 40/5, 50/4\} = 20 \).

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 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. However, the IOO still selects the schedules very similar to the sequence to lead to the 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 ten servers of IBM RC2 Cloud at the IBM China Development Laboratory, Beijing (see Fig. 8) similar to the Amazon Web Service [3].

![Fig. 8. Research compute cloud (RC2) over eight IBM Research and Development Centers.](image)

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.

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

#### 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 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.

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

Task Class 1 executes a template bank (TmpltBank) in two major steps (Inspiral and TriBank). 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 $\theta_j$ 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.

#### A) 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. 9. The schedule is generated by averaging over a small set of schedules.

![Fig. 9. Simulation overhead times of three workflow scheduling methods under 10% good-enough schedules.](image)

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. 9, 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 increase, the trade-off space also increases. It is noteworthy to mention that these results stay within the theoretical bounds set in Theorem 1.

#### B) Throughput and Memory Demand

To further investigate experiment details, a truncated illustration of dynamic workload is included in Fig. 10 and the corresponding experimental results are included in Fig. 11. The numbers 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. 11. Dynamically changing of workload and corresponding scheduling results are illustrated. In Fig. 11, throughput and memory demand of all the methods are calculated for each stage. The proposed IOO method may not have better performance all of the time during each of the iteration.

![Fig. 10. Illustration of the dynamic workload in a single iteration.](image)

![Fig. 11. Throughput and memory results running the LIGO verification workflow on the RC2 cloud platform.](image)
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. 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. 12, 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.

![Effect of task number for 128 VMs.](image)

![Effect of cluster size for 20K tasks.](image)

Fig. 12. Relative performance of three workflow scheduling methods plotted against task and cluster sizes.

In Fig. 12(a), 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% times higher throughput than that of Blind-Pick as the task number varies.

Fig. 12(b) 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% throughput gain over Blind-Pick as the VC size increases.

![Effect of task number for 128 VMs.](image)

![Effect of cluster size for 20,000 tasks.](image)

Fig. 13. Memory demands of three workflow scheduling methods plotted over variable tasks and cluster sizes.

In Fig. 13(b) 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, 10–13, 18, 22–23, 26, 29 ~ 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].


These scheduling approaches in computational grids [12] serve different purpose than the methods proposed in the autonomous 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.

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, Siegel, and Maciejewski [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 tenant 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 consensus 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 offered the 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 was compared with Monte Carlo and Blind-Pick. The conclusions of our findings are summarized in Table 4.

<table>
<thead>
<tr>
<th>TABLE 4. SUMMARY OF THREE WORKFLOW SCHEDULING METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
</tr>
<tr>
<td>Mont Carlo</td>
</tr>
<tr>
<td>Blind -Pick</td>
</tr>
<tr>
<td>IOO</td>
</tr>
</tbody>
</table>

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 the National Nature Science Foundation of China under grant No. 61233016, by the Ministry of Science and Technology of China under National 973 Basic Research Grants No. 2011CB302505 and No. 2013CB228206. The work was also partially supported by an IBM Fellowship for Fan Zhang, and by the Intellectual Ventures endowment to Tsinghua University.

REFERENCES

[20] A. Lazzarini, “Data from the LIGO I Science Run”, http://www.ligo.caltech.edu/docs/P/P010002-00/P010002-00.pdf
AUTHORS’ BIOGRAPHIES

Fan Zhang (SM'13) 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. He received his Ph.D. in Department of Control Science and Engineering, Tsinghua University in Jan. 2012.

From 2011 to 2013 he was a research scientist at Cloud Computing Laboratory, Carnegie Mellon University. An IEEE Senior Member, he received an 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 Ph.D. 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 his Ph.D. in computer science from the University of Warwick, Coventry, UK, in 2001. He received his bachelor and master degrees in control theories and engineering in 1998 and 1996, respectively, both from Tsinghua University, Beijing, China. He is currently 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 over 150 papers and cited by international scholars for over 6,000 times. He has authored or edited 6 books. His research is focused on distributed computing technologies and applications. Prof. Cao is a Senior Member of the IEEE Computer Society and a Member of the ACM and CCF.

Kai Hwang (F’86) 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. He received the Ph.D. from University of California, Berkeley in 1972. He has published 8 books and 230 papers, which have been cited over 12,000 times with a citation h-index of 49. 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.

An IEEE Fellow, Hwang received 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 Trans. on Cloud Computing and International Journal of Big Data Intelligence. He also co-chair the Second International Conf. 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 and 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. Dr. Li has over 245 research publications and has received several Best Paper Awards for his research work. He is currently on the editorial boards of IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers.

Samee U. Khan (SM’12) is an assistant professor at the North Dakota State University. He received his Ph.D. from the University of Texas at Arlington in 2007. Dr. Khan's 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 over 225 publications with two receiving best paper awards. He is a Fellow of the IET and a Fellow of the BCS.