Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds

Fan Zhang[1,3], **Junwei Cao**[2,3], **Kai Hwang**[3,4] and **Cheng Wu**[1,3]

[1]National CIMS Engineering and Research Center, Tsinghua University, Beijing 100084, P. R. China
[2]Research Institute of Information Technology, Tsinghua University, Beijing 100084, P. R. China
[3]Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China
[4]University of Southern California, Los Angeles, CA 90089, USA
*e-mail: kaihwang@usc.edu

Abstract— Elastic compute clouds are best represented by the virtual clusters in Amazon EC2 or in IBM RC2. This paper proposes a simulation based approach to scheduling scientific workflows onto elastic clouds. Scheduling multitask workflows in virtual clusters is a NP-hard problem. Excessive simulations in months of time may be needed to produce the optimal schedule using Monte Carlo simulations. To reduce this scheduling overhead is necessary in real-time cloud computing. We present a new workflow scheduling method based on *iterative ordinal optimization* (IOO).

This new method outperforms the Monte Carlo and Blind-Pick methods to yield higher performance against rapid workflow variations. For example, to execute 20,000 tasks on 128 virtual machines for gravitational wave analysis, an ordinal optimized schedule can be generated in a few minutes, which is O(10^3)~O(10^4) faster than using Monte Carlo simulations. The ordinal optimized schedule results in higher throughput with lower memory demand. The cloud experimental results being reported verified our theoretical findings on the relative performance of three workflow scheduling methods studied in this paper.

Keywords*:* Cloud computing; workflow scheduling; ordinal optimization, and virtual clustering

1. INTRODUCTION

There is a growing demand to use Internet clouds or large grids to execute large-scale scientific applications [11]. Scientific workload demand parallel resources from computing infrastructures on demand. One of the best examples is the LIGO (*Laser Interferometer Gravitational-wave Observatory*) [1] experiments for earth science studies. LIGO demands massive data analysis over a workflow with thousands of tasks to be scheduled for parallel execution on a huge number of processors in grids or clouds.

Parallel resources can be provided by computational grid or by elastic clusters in cloud platforms. While grid computing enables wide-area sharing of geographically distributed resources, cloud platform provides virtualized server clusters over large datacenters [2]. Virtual clusters are elastic resources that can scale up or down, dynamically [21].

In general, scheduling of multitask workflow onto distributed computing resources is a NP-hard problem [8]. The main challenge of dynamic workflow scheduling on virtual clusters lies in how to reduce scheduling overhead and handle workload dynamics. An optimal workflow schedule on the cloud may take intolerable amount of simulation time to generate. This is not acceptable if elastic cloud clusters are used in real time applications.

Ho, et al [9] proposed the *ordinal optimization* (OO) method for discrete-event problems with very large solution space. Subsequently, they [10] demonstrated that the OO method is effective to generate a soft or suboptimal solution to most NP-hard problems. In this paper, we extend the OO method iteratively in search of suboptimal schedule to execute scientific workflows on elastic compute clouds.

Our scheduling is based on a new *iterative ordinal optimization* (IOO) approach. During each iteration, the OO is simulated to search for a suboptimal or good-enough schedule. We reduce the search space significantly to result in lower overhead. The inner core of the IOO approach is to generate a rough model resembling the workflow problem. The discrepancy between rough and complete search models is kept rather small in our approach.

In the IOO process, the system absorbs dynamic changes in resources provisioning against the workload variations. The low overhead in IOO-based scheduling appeals to real-time cloud computing applications [3]. The purpose is to generate better schedule from a global perspective over a sequence of workload prediction periods. We present the analytical model of the IOO-based scheduling scheme. Then we demonstrate its effectiveness with extensive LIGO experimental work.

We apply the LIGO workflow in the experiments [4] using hundreds of *virtual machines* (VMs).

Experimental results show that our IOO scheduling achieves higher throughput and reduced memory demands, compared with existing scheduling methods like Monte Carlo [14] and Blind Pick methods[10].

The rest of the paper is organized as follows: Section 2 characterize the workflow scheduling problem on a set of virtual clusters in the compute cloud. Section 3 introduces Monte Carlo and Blind-Pick scheduling methods. Section 4 specifies the process of iterative ordinal optimization. Section 5 provides an overhead analysis of three scheduling methods. The experimental settings and design of LIGO experiments are given in Section 6. Experimental results are reported in Section 7. Finally, we conclude with a review of related work and summarize our technical contributions.

2. WORKFLOW SCHEDULING IN VIRTUAL CLUSTERS IN A CLOUD

A *physical cluster* is built with a fixed number of interconnected servers in a datacenter. Each physical server can be mounted with multiple *virtual machines* (VMs). A *virtual cluster* is formed with multiple VMs, that are logically connected together over several physical clusters. Virtual clusters are dynamically provisioned to users upon demand in *service-level agreement* (SLA) between provider and clients.

When a user job gets done by a virtual cluster, the VM instances are removed from the hosting servers and server resources can be allocated to other users. Figure 1 shows an example cloud platform that has provisioned 4 virtual clusters installed at servers from 3 physical clusters. Each physical cluster is represented by rectangular boxes with different shading. The servers in 3 physical clusters are distinguished by boxes with different shadings.

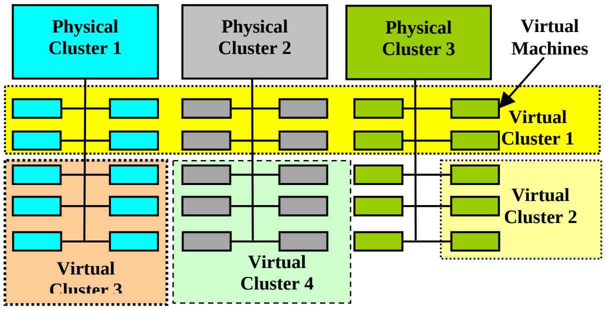


Figure 1: A virtualized cloud platform built with 4 virtual clusters over 3 physical clusters. The VMs installed at various servers are distinguished by different colors.

Each virtual cluster can be formed with either physical machines or VMs hosted by multiple physical clusters. The virtual clusters boundaries are shown by 4 dot/dash-line boxes. The provisioning of VMs to a virtual cluster can be dynamically done upon user demand. The queuing model of the workflow task dispatching is shown in Fig. 2.

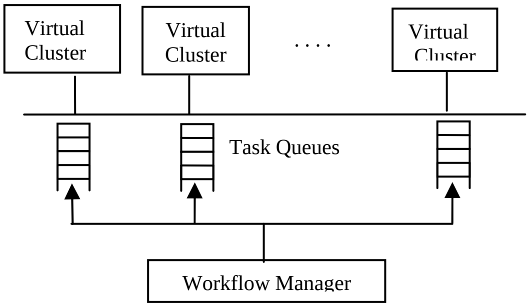


Figure 2. The workflow scheduling model dispatches multiple tasks to virtual clusters for parallel execution in a cloud platform

In the above scheduling model, we define *task class* as a set of computing jobs of the same type and they can be executed, concurrently in VMs in the same virtual cluster. For simplicity in analysis, we assume that all VMs in the same virtual cluster take equal amount of time to execute their assigned tasks. In other word, the task execution time in a VM is the basic time unit in performance analysis. A summary of basic notations used in this paper are listed in Table 1.

Table 1. Notations and Basic Definitions

Term	Basic Definition
U	Candidate set of all u possible schedules
S	Selection set of s schedules to simulate
G	Acceptable set of g good-enough schedules
k	Number of overlapped schedules between G and S
N	The number of simulations per schedule performed in Monte Carlo or Blind-Pick scheduling methods
n	The number of IOO simulations per schedule
θ	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

A. Workflow Scheduling Model

Let p_i be the expected execution time of a single VM in the i-th cluster. Let v_i be the number of VMs in the cluster. We have $\beta_i = v_i/p_i$ as the task processing rate in a cluster. Let δ_i be the number of tasks in queue i. Then we have the execution time $t_i = \delta_i/\beta_i = p_i\delta_i/v_i$ by the i-th cluster. All virtual clusters are distinguished by the index i. We define the *makespan* of all n tasks in a scientific workflow by:

$$M = \text{Max}\ \{t_1, t_2, \ldots, t_c\} \qquad \textbf{(1)}$$

where c virtual clusters are used and $t_i = p_i\ \delta_i / v_i$. This makespan is the total execution time between the start and finish of all tasks in a multitask workflow. We denote d_i as the memory used by one VM in the VC$_i$. Then, the total *memory demand* by all VMs is:

$$D = \sum_{i=1}^{C} d_i \times v_i \qquad \textbf{(2)}$$

A resource-reservation *schedule* specifies the sets of VMs provisioned at successive time slots, called periods. For example, the j^{th} schedule $\boldsymbol{\theta_j}$ is represented by a set of VMs allocated in c clusters in a *schedule space U*. This schedule is thus represented by a c-dimensional vector:

$$\boldsymbol{\theta_j} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_c}] \qquad \textbf{(3)}$$

where $\boldsymbol{v_i}$ is the number of VMs assigned in cluster i. At different time periods, different schedules may be applied. All candidate schedules at successive time periods form a schedule space U. The *cardinality* of U is estimated by the following expression:

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

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

For example, if we use 20 VMs in 7 clusters for 7 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, this number is way too high. It will lead to excessive simulation overhead time. Thus, we need to reduce the schedule search space significantly.

The following objective function is used to search for the suboptimal schedule for the workflow scheduling. In general, we need to conduct an exhaustive search to minimize a pair of objective functions on all makespans and memory demands by all possible schedules:

$$\left\{ \min_{\theta_j \in U} M\left(\theta_j\right), \min_{\theta_j \in U} D\left(\theta_j\right) \right\} \qquad \textbf{(5)}$$

against all possible schedules $\boldsymbol{\theta_j}$ in the search space U.

In simulating each schedule $\boldsymbol{\theta_j}$, we need to generate the value of $\boldsymbol{p_i}$ and $\boldsymbol{d_i}$ before we can calculate the makespan M and memory demand D in Eq.(1) and (2). We use the average over all simulations on $\boldsymbol{\theta_j}$ to get the minimum $M(\boldsymbol{\theta_j})$ and $D(\boldsymbol{\theta_j})$ in Eq.(5). The random variables $\boldsymbol{p_i}$ and $\boldsymbol{d_i}$, could be also estimated offline and using a rough estimation.

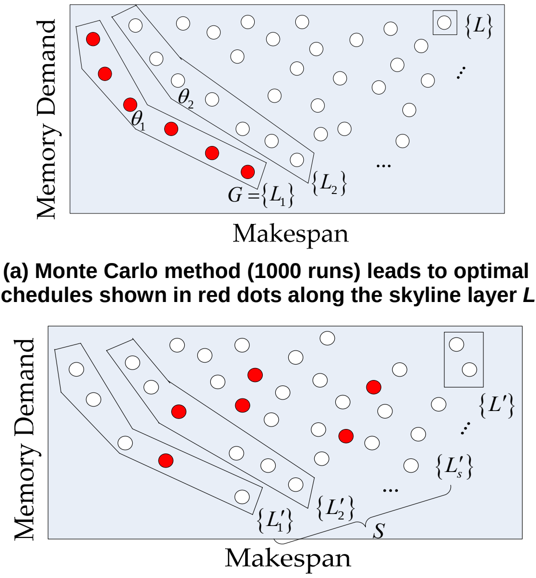
3. Monte Carlo and Blind Pick Scheduling Methods

In this section, Monte Carlo Method and Blind-Pick method are introduced for workflow scheduling. In Monte Carlo simulation, we simulate N = 1,000 runs for each value $\boldsymbol{p_i}$ and $\boldsymbol{d_i}$. to assess the expected makespan and memory demand. It is desired to reduce the simulation runs to n = 10 runs as a *rough model* to make scheduling decisions.

We use Monte Carlo simulation to generate optimal schedules under the **heavy** scheduling overhead. Figure 3 shows a bi-objective optimization scenario, by which both the makespan and memory demand need to be minimized in the 2-dimensional optimization space. Each dot in 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.3(a). These optimal choices are marked in red dots. Mathematically, there is no schedule in the 2-D space, which is better or less than those red schedules along the skyline in terms of makespan and memory demand. In other words, with a fixed memory demand, all skyline makespans are lower than those above the skyline.

Similarly, with a fixed makespan, all memory demands along the skyline are lower than those above the skyline. Researchers working in the automation community, call the skyline schedules a *pareto front*, which correspond to the acceptable or good-enough schedule set G. If $\{L_1\}$ is removed, $\{L_2\}$ can be achieved in the same way. By processing all schedules in such a way, the searching space can be divided into a series of pareto fronts denoted by $\{L_1\}$,$\{L_2\}$,…, $\{L\}$.



(a) Monte Carlo method (1000 runs) leads to optimal schedules shown in red dots along the skyline layer L_1

(b) Rough model evaluation (10 runs) of all the schedules leads to optimal schedules scatter sparsely in the space

Figure 3. Monte Carlo and rough model evaluation of a bi-objective optimization problem. Red dots in (a) are good-enough schedules. In (b), the good-enough schedules are scattered due to rough model evaluation. We could still find one good-enough schedules in the skyline layer.

It is difficult to determine *S* to cover all schedules in *G* given a rough model evaluation. *Optimization modes* and *noise levels* introduced in section 4 are two important factors. With noises introduced, the good-enough schedules in set *G* are spread in multiple layers of the skyline under rough evaluation.

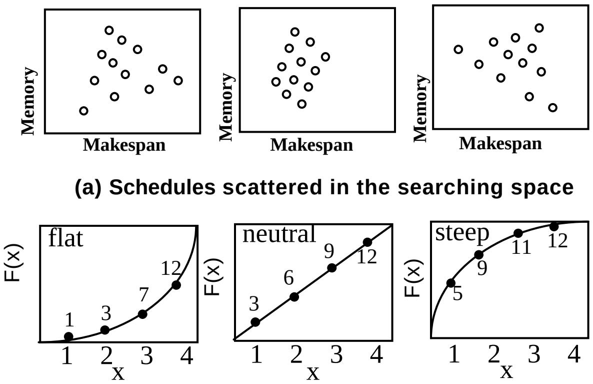
The rough model used Fig.3(b) results in lower scheduling overhead. The good-enough schedules are scattered sparsely in the space. This may demand a large selection set *S* to cover all these schedules. A smaller *S* is enough, but may demand heavier scheduling overhead. Thus, tradeoff does exist between *s* and the scheduling overhead that can be tolerated.

In the next section, we show that applications with a steep optimization mode and small noise, our method can be quite efficient.

4. ITERATIVE ODINAL OPTIMIZATION FOR WORKFLOW SCHEDULING

Optimization mode describes how schedules are scattered in the searching space as shown in Fig. 4 in three modes. In Fig.4(a), there are 12 schedules scattered in 4 skyline layers. In Fig.4(b), corresponding optimization modes are shown. The *x* identifies the layer index, and *F(x)* denotes how many schedules are in the first *x* skyline layers.

If schedules are scattered in the *steep mode* as shown in the rightmost figures in Figs. 4(a) and 4(b), it would be very much **easy** to find out the suboptimal schedules for the optimization problem. This is because most schedules are converged to the zero point, one search could get 5 good schedules. On the other hand, only one good schedule is available in the *flat mode*. The *neutral mode* corresponds to a uniform distribution in all skyline layers. We expect steep mode to ensure our optimization much more efficient.



(a) Schedules scattered in the searching space

(b) Corresponding optimization modes

Figure 4. Three optimization modes for three different schedules, by which 12 schedules are scattered to form 4 skyline layers. In (b), the corresponding optimization mode for (a) is shown. The x identifies the layer index, and F(x) denotes the number of schedules in x front layers.

The workflow scheduling for LIGO gravitational wave data analysis pipelines satisfies a steep mode shown in section 7. This leads to tradeoffs between scheduling overhead and performance levels desired.

Mathematically, the *noise level* of the performance pair $\{M(\theta_j), D(\theta_j)\}$ is determined by the maximum standard deviation from makespan and memory demand. Large number of simulation runs *n* leads to small noise while at the cost of high simulation overhead. This is as if we were flipping a coin, more runs (large *n*, small noise) can lead to the probability of each side closer to 0.5, but results to more simulation time used.

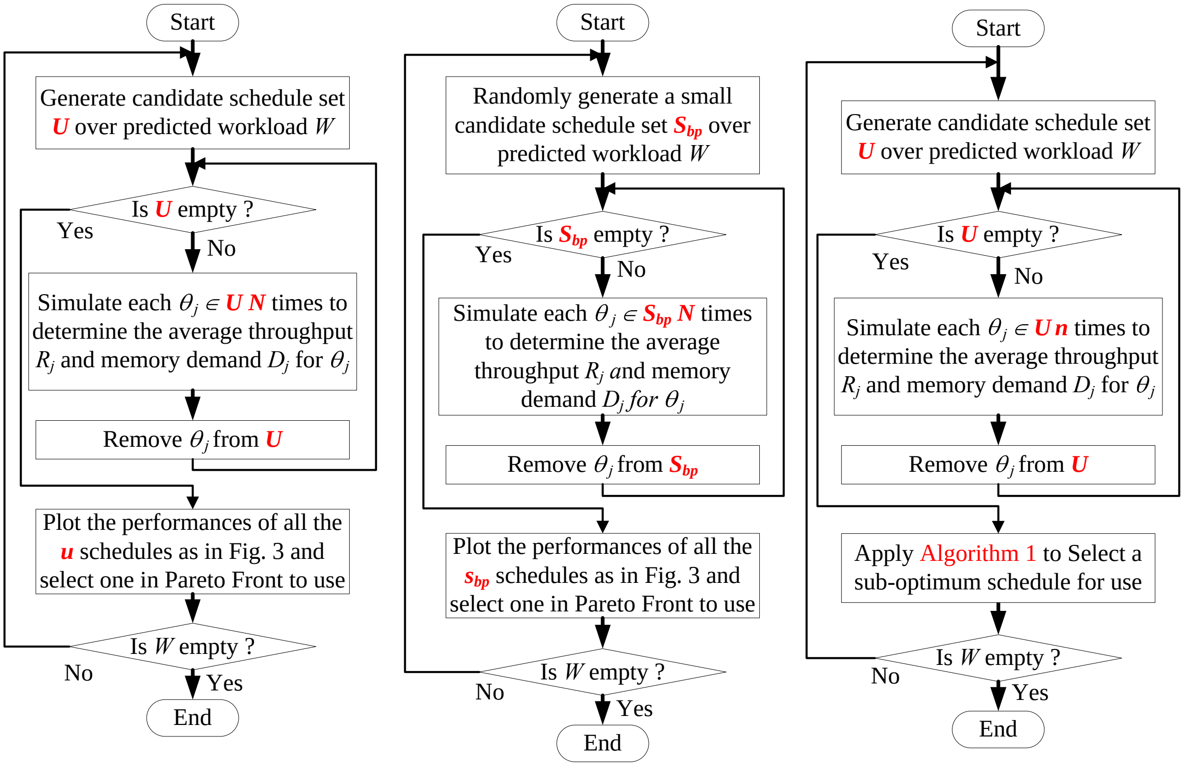
Given the above analysis, the size *s* of the schedule selection set *S* has been determined in [22] as a function of the optimization mode and the noise level tolerated by numerous regression analysis on top of different optimization modes and noise levels. The detailed expression for computing *s* can be found in [22] and it will not be repeated here.

As a comparison, the *U* space is very large, say $u = 27, 132$ in our experiment, while the selection space applies to $s = 190$ schedules in IOO. Per each schedule θ_j simulation, the Monte Carlo and Blind-pick Methods need to simulation N =1,000 runs to compute the average throughput R_j and memory demand $D_{j.}$ To implement the IOO-based simulation, only $n = 10$ runs of simulation is performed to generate the performance pair $\{M(\theta_j), D(\theta_j)\}$, and $N = 1{,}000$ runs followed on a rather smaller reduced set *S*.

Algorithm 1. Ordinal Optimization for Optimizing the workflow schedule

```
Input:  Simulated performances of all the u schedule
        {(M(θ_1), D(θ_1)), …, (M(θ_u), D(θ_u))}, n=10 runs each
Output:  A suboptimal workflow schedule to use
Procedure:
1. Assess the optimization mode (steep mode)
2. Calculate the noise level (NL)
3. Calculate the selection set S based on 1 and 2
4. Simulate the schedules at s front skyline layers, {θ_1,
   θ_2 ,…, θ_s}, N = 1000 runs per each schedule
5. Plot the above schedules in as in Fig. 3, select one
   schedule in Pateto front for use
```

Algorithm 1 specifies the ordinal optimization for selecting a suboptimal workflow schedule using the IOO method. In Step 1, it calculates the optimization mode to find out how the throughput and memory demand are scattered in the space, as shown in Fig. 4 and Fig. 9 in section 6(B). In Step 2, we calculate the *noise level* (*NL*). In Step 3, we determine the value of *s* based on [22] and selects the schedules of the first *s* skylines as *S*. In Step 4, we perform *n* Monte Carlo simulations for each schedule in *S*. Finally, we apply Algorithm 1 to narrow down to the final choice. It should be noted that that $N >> n$ in general. In our LIGO experiments, we have applied $N = 1{,}000$ and n =10.


Start
Generate candidate schedule set U over predicted workload W
Is U empty ?
Yes
No
Simulate each $\theta_j \in U$ N times to determine the average throughput R_j and memory demand D_j for θ_j
Remove θ_j from U
Plot the performances of all the u schedules as in Fig. 3 and select one in Pareto Front to use
Is W empty ?
No
Yes
End
Start
Randomly generate a small candidate schedule set S_{bp} over predicted workload W
Is S_{bp} empty ?
Yes
No
Simulate each $\theta_j \in S_{bp}$ N times to determine the average throughput R_j and memory demand D_j for θ_j
Remove θ_j from S_{bp}
Plot the performances of all the s_{bp} schedules as in Fig. 3 and select one in Pareto Front to use
Is W empty ?
No
Yes
End
Start
Generate candidate schedule set U over predicted workload W
Is U empty ?
Yes
No
Simulate each $\theta_j \in U$ n times to determine the average throughput R_j and memory demand D_j for θ_j
Remove θ_j from U
Apply Algorithm 1 to Select a sub-optimum schedule for use
Is W empty ?
No
Yes
End

(a) Monte Carlo Method based on exhaustive search

(b) Blind-Pick Method using a reduced selection set S_{bp}

(c) The IOO Method based on iterative ordinal optimization

Figure 5 Three workflow scheduling methods for simulated LIGO experiments on an elastic cloud platform with 128 VMs installed at the IBM Beijing Research Center.

Iterative ordinal optimization (IOO) is specified to generate suboptimal schedules in virtual clusters with dynamic workload. Scheduling solutions are generated in an iterative way. During each iteration, suboptimal or good-enough schedules are obtained. The IOO method adapt to system dynamism in terms of dynamic workload and VM provisioning in virtual clusters.

Figure 5 above shows the flow charts of the three scheduling process for the Monte Carlo, Blind-pick and IOO methods. Note that the large search space U is applied in Monte Carlo and IOO method at the outer loop on all possible schedules. IOO then use a much reduced set S as shown in algorithm 1. The Blind-Pick Method applies the randomly reduced selection set S_{bp}.

Let T be the time period of scheduling using the Monte Carlo method, and t be that for IOO. For example, at time t_0, Monte Carlo is used for simulation. It is not until t_1 can Monte Carlo generate its optimal schedule. While the solution is optimized at the time t_1, It is not possible to generate such an optimized schedule between t_1 and t_2. As for IOO 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, …., similarly.

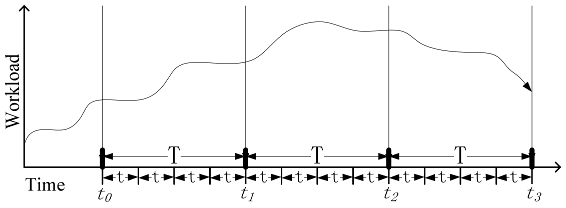


Figure 6 IOO adaption to dynamic workload.

This process is continued at each period to capture the variation of the workload in order to improve the performance. The IOO is carried out dynamically to upgrade the performance, iteratively. In each iteration, the workflow scheduling follows a steep mode to reduce the overhead and generate a good enough solution. From a global point of view, the successive iterations are processed fast enough to adapt to the dynamic workload of the system.

5. Design of LIGO Worflow Experiments

In this section, we present the design of the LIGO experiments to test the effectiveness of the IOO scheduling method for scientific workflow in cloud platform. First we introduce the experimental settings, Then we analyze the

LIGO task classes that can be explored by using multiple VMs in the cloud.

A. Experimental setting :

The cloud experiments are carried out using 10 servers of IBM RC2 Cloud at IBM China Development Laboratory, Beijing (Fig.7). Each sever is equipped with Intel Xeon MP 7150N processor, 24 GB memory. The virtualized physical servers in IBM Beijing Center are specified in Table 2.

We install up to 14 *virtual machines* (VM) per each physical server. The physical server runs with the OpenSuSE11/OS. All LIGO tasks are written in Java. With 10 servers, we could experiment up to a virtual cluster of 128 VM instances. To test the scalable performance, we vary the virtual cluster configuration from 16 to 32, 64 and 128 VMs.

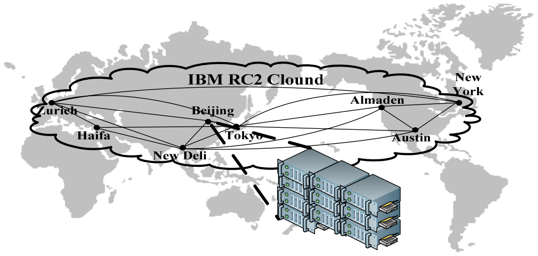


Figure 7. Research compute cloud (RC2) over 8 IBM R/D Centers, where our experiments were conducted at the IBM Beijing Center.

Table 2 Virtualized Physical Cluster

Cluster Size	10 servers per physical cluster
Node Architecture	IBM X3950 with 16-core Xeon MP 7150N, 24 GB memory with the openSuSE 11
VM Architecture	CPU: 1 vCPU deployed in 1 pCPU with 1 GB memory running with OpenSuSE 11
VMs/Hypervisor	14 VMs in each server/Xen 3.0.3

B. Multitask Analysis of LIGO Workflow

The LIGO (*Laser Interferometer Gravitational-wave Observatory*) is designed for direct detection of earth's gravitational waves, This is a large-scale scientific experiment as predicted by Einstein's General Theory of Relativity a century ago. We analyze the LIGO workload to exploit the parallelism in using the virtual clusters in a elastic compute cloud. The computations involved are divided into seven task classes in Table 3. It embodies three sensitive detectors (L1, H1, H2) on earth surface.

Gravitational-wave data analysis is carried out in a workflow pipelined manner, since multiple tasks have to be executed at geographically dispersed data sources concurrently. The verification of a LIGO workflow is essential to identify potential faults before the actual program execution. Each verification contains many subtasks over massive data sets. We use many virtual machines to explore the DoP in these task classes. Sufficient cloud resources (VMs) are provisioned to satisfy the demand of LIGO workflows. The seven independent task classes can be executed in parallel [19].

Table 3. Task Classes in a LIGO Workflow

Task Class	Functional Characteristics	Number of 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

We want to find a range of solutions to use θ_j to minimize both makespan *M* and memory demand *D*. On our LIGO workflow experiments, there are 7 task classes and 20 virtual machines. There are 27,132 schedules in total to be evaluated. Feasible allocation schedule is θ_j. Then for each task class, the number of VMs allocated for its execution is identified as [1, 20]. The steep optimization mode guarantees that the IOO can be effectively applied in LIGO workload.

6. EXPERIMENTAL PERFORMANCE RESULTS

In this section, we report the experimental results, First we show snap shot of the simulated schedule distribution of our experiment. Then we report the scheduling overhead, makespan and memory demands in LIGO experiments.

A. Bi-Objective Optimized Scheduling

In Fig. 8, we map the schedule performance pair {*M, D*} into a 2-dimensional space. Each schedule is simulated 1000 runs to assess its makespan *M* and memory demand *D.* Each schedule is represented by a black dot. The small circled dots (in blue color) form the *G* set of good-enough schedules. The suboptimal schedules generated by IOO method are identified by large red circles. Some accepted suboptimal schedules overlap with the blue schedules at the skyline. This graph illustrates clearly the effectiveness of the IOO scheme to make fast scheduling decision in a large-scale cloud, while still delivering a set of good-enough schedules.

B. Simulated Scheduling Overhead

Traditional method uses the Monte Carlo [14] simulation to exhaust the entire schedule space. For each schedule, one must implement all task classes on all virtual clusters. The time used for this exhaustive search causes a great amount of scheduling overhead. The IOO method require the least scheduling overhead as demonstrated in Fig.10. The schedules are generated by testing an estimated search time by averaging over a small set of schedules. The tradeoff of our IOO method is that it avoids the exhaustive search in using Monte Carlo method. Good-enough schedules can be found in a few iterations of the IOO process.

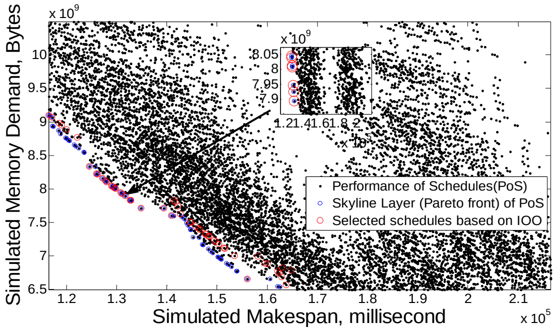
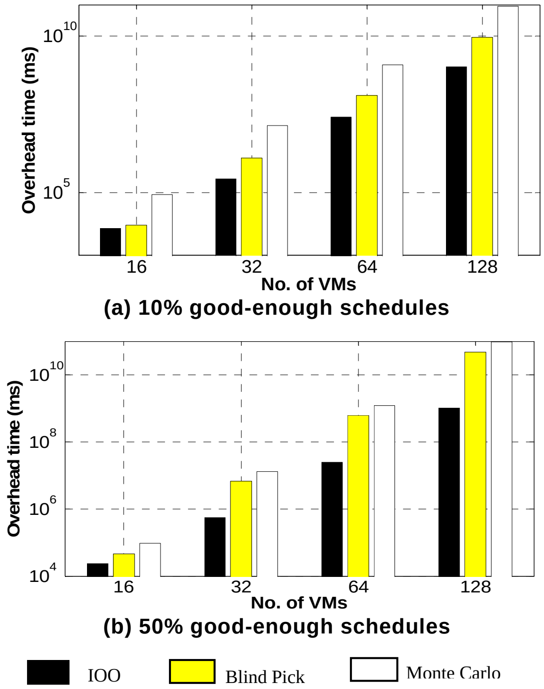


Figure 8. The scheduling results of a bi-objective scheduling over the LIGO workload. There are 27,132 schedules (dots) shown in the 2-D performance space.



(a) 10% good-enough schedules

(b) 50% good-enough schedules

Figure 10 Simulation overhead time plotted against the cluster size under two user choices (10% and 50%) of good-enough schedules.

As shown in Fig.10, IOO performance for workflow scheduling is evaluated by overhead time comparison among three methods over 16, 32, 64, 128 virtual machines. This comparison is made under two acquisition of good enough schedules (10% versus 50%). If a higher percentage of good-enough schedules is demanded, the scheduling overhead will increase.

As the cluster size scales from 16 to 128, we observe that IOO method has an scheduling overhead reduction of to tens or hundreds times than using 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 tradeoff space also increases. These experimental results are upper bounded by the theoretical prediction given in Section 5.

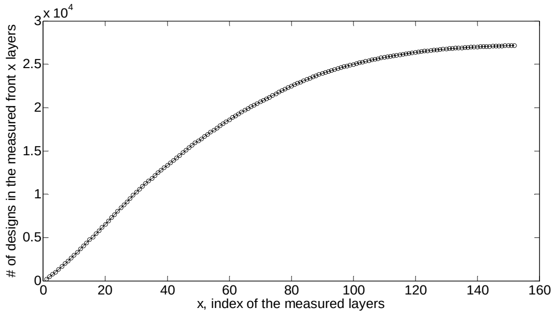


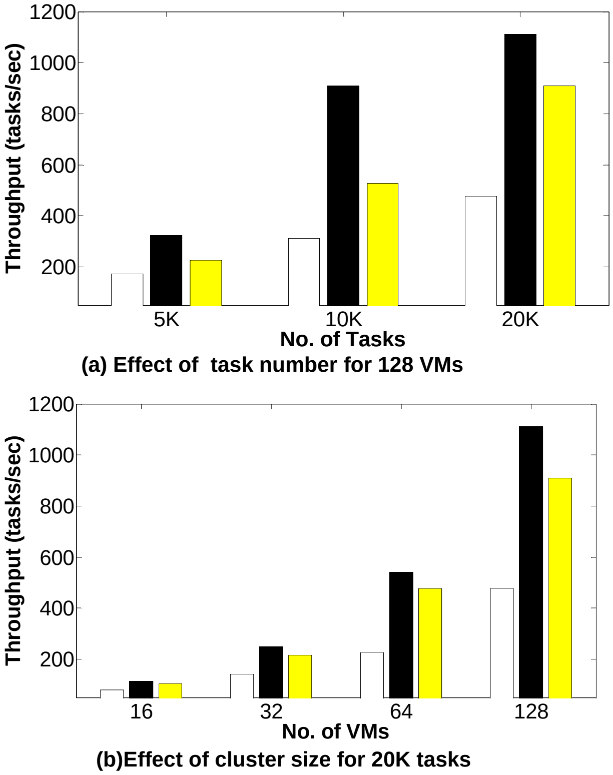
Figure 9. The optimization mode of the performance of *n* simulation runs in IOO. It is a steep type corresponding to the good-enough schedule

C. Throughput Performance and Memory Demand

In Figs 11 and 12, the performance metrics are average throughput and memory demand during the whole experiment period. The experiments are carried out in 8 simulation periods. Each period lasts the time *h* of a single Monte Carlo simulation. During each period, the OO is repeated iteratively, since the IOO scheduling time is much shorter. By default, we consider 20,000 LIGO tasks and 128 virtual machines in the cloud.

Experiments are carried out by comparison of 3 methods over scalable number of tasks and virtual machines. In Fig.11(a), we demonstrate the effect of task number in the LIGO application. The IOO method demonstrates about 3 times higher throughput than the Monte Carlo method as the task number varies. The IOO method offers 20 to 40% times faster throughput than that of the Blind-Pick method as the task number varies. Figure 11(b) shows the effects of the virtual cluster size on the throughput performance of the three scheduling methods. We observed a 2.2 to 3.5 times throughput gain of the IOO method over the Monte Carlo method as the cluster increases from 16 to 128 VMs. The IOO method has about 15% to 30% throughput gain over the Blind-Pick method as the cluster size increases.

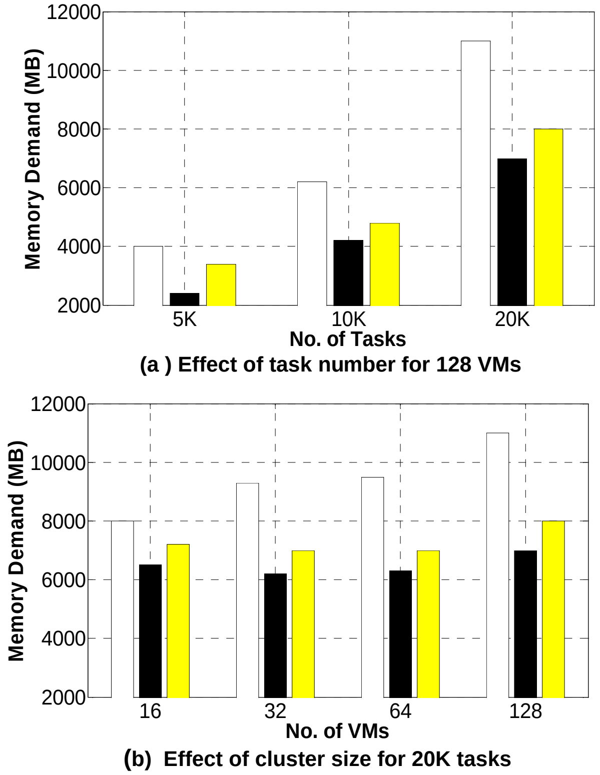
Figure 12 shows the relative memory demands of 3 workflow scheduling methods. The results are plotted as a function of the task number and cluster size. The Monte Carlo method has the highest memort demand. The IOO method requires the least memoy. The Blind-ick method sits in the middle. The IOO method saves about 45% memory from that demanded by the Monte Carlo method. Blind-Pick method requires 80% ~ 20% higher memory than IOO method as the task number increases. For 20,000 subdivided tasks, the memory demands are 11.5 GB, 8 GB, and 7 GB on 128 VMs for the Monte carlo, Blind-Pick and IOO methods, respectively.



(a) Effect of task number for 128 VMs

(b)Effect of cluster size for 20K tasks

Figure 11 Relative throughput of 3 workflow scheduling methods against the` task number and cluster size



(a) Effect of task number for 128 VMs

(b) Effect of cluster size for 20K tasks

Figure 12 Relative memory demands of 3 workflow scheduling methods plotted against task number and cluster size, separately

7. Related Work and Conclusions

In the past, most workflow scheduling work was done for computational grids or on large-scale heterogeneous systems. Our work is the first attempt to schedule cloud workflows on virtual clusters. In the final section, we review some related work on cloud resource provisioning, task scheduling and ordinal optimization. We summarize our research findings and discuss the future extensions.

A. Related Work

In the past, scheduling large-scale scientific tasks to grid or heterogeneous systems has been studied by many researchers [6], [12], [13], [15], [16]. We see an escalating interest on resource allocation for scientific workflows on the Internet clouds[11]. Many classical optimization methods, such as opportunistic load balance, minimum execution time, and minimum completion time, are described in [8].

Benoit, et al [3] designed resource-aware allocation strategies for divisible loads. Li and Buyya [12] proposed model-driven simulation of grid scheduling strategies. Zomaya, et al [13], [16] proposed a hybrid scheduling method for scheduling in large systems.

In 1992, Ho, et al [9] proposed the *ordinal optimization* (OO) method for discrete-event dynamic systems. Along the OO line, many heuristic methods have been proposed [17], [20], [21]. Subsequently, Ho, et al [10] demonstrated that the OO method is effective to generate a soft or suboptimal solution to most NP-hard problems. Subsequently, the OO technique has been applied in advanced automation and industrial manufacturing [17], [20], [22].

Wieczorek, et al [18] analyzed five facets which may have a major impact on the selection of an appropriate scheduling strategy. They proposed a taxonomy to classify multi-objective workflow scheduling schemes. Prodan and Wieczorek [15] proposed a dynamic algorithm, which outperforms the LOSS3 and BDLS methods to optimize bi-criteria problems.

In the past, Cao, et al [5], [20] have studied the LIGO problems in grid environments. Duan, et al [7] suggested a game-theoretic optimization method. Dogan, et al [6] developed a matching and scheduling algorithm.

B. Concluding Remarks

This paper offers the first attempt to extend the ordinal optimization for fast dynamic workflow scheduling on virtual clusters in a cloud computing platform. The major advantage of IOO method lies in significantly lowered overhead in producing suboptimal schedules. The method appeals especially to a scenario of time varying workload. Major technical contributions of this work is the IOO method appeals to work well on elastic cloud under dynamic workload. Then we use large-scale LIGO gravitational wave data analysis pipelines to test the new IOO approach effectively. Finally, We provide the first model on workflow scheduling in a cloud platform. The cloud environments contain many uncertainty factors. Our IOO approach applies well in EC2-like cloud services to upgrade the throughput and in S3-like services to reduce the memory demand.

The LIGO scientific collaboration research group at Tsinghua University will establish a new cloud infrastructure for enabling real-time gravitational-wave burst data analysis using virtualization technology. Improved workflow scheduling and performance optimization lead to faster execution of data analysis pipelines. This opens up a new research front for scientific cloud computing in addition to business applications installed at most public clouds.

Acknowledgments: This work is supported by Ministry of Science and Technology of China under National 973 Basic Research Grants No. 2011CB302805 and No. 2011CB302505, National 863 high-tech Program Grant No. 2011AA040501, and National Science Foundation of China (grant No. 60803017). This work is also funded by Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation. Fan Zhang was supported by a PhD Fellowship from IBM. Hwang wants to thank the support of Intellectual Ventures for his Chair Professorship at Tsinghua University.

REFERENCES :

[1] A. Abramovici, W. E. Althouse, et. al., “LIGO: The Laser Interferometer Gravitational-Wave Observatory”, *Science,* Vol. 256, No. 5055, pp. 325 – 333, 1992.

[2] P. Barham, and B. Dragovic, “Xen and the Art of Virtualization”, *Proc. of 19th ACM symp. on Operating Systems Principles,* Bolton Landing, NY, pp. 164-177, 2003.

[3] A. Benoit, L. Marchal, J. Pineau, Y. Robert, F. Vivien. “Resource-aware Allocation Strategies for Divisible Loads on Large-scale Systems”. *Proc. of IEEE Int’l Parallel and Distributed Processing Symp.*(IPDPS’09), Rome, Italy. 2009.

[4] 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 *Workflows for eScience: Scientific Workflows for Grids,* Springer Verlag, pp. 39-59, 2007.

[5] J. Cao, S. A. Jarvis, S. Saini and G. R. Nudd, “GridFlow: Workflow Management for Grid Computing”, *Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid,* Tokyo, Japan, 198-205, 2003.

[6] A. Dogan, and F. Özgüner. “Biobjective Scheduling Algorithms for Execution Time–Reliability Trade-off in Heterogeneous Computing Systems”. T*he Computer Journal,* vol. 48, no.3, pp.300-314, 2005.

[7] R. Duan, R. Prodan, and T. Fahringer, “Performance and Cost Optimization for Multiple Large-scale Grid Workflow Applications”, *Proc. of IEEE/ACM Int’l Conf. on SuperComputing* (SC’07), Reno, 2007.

[8] R. Freund, et al, “Scheduling Resources in Multi-user, Heterogeneous, Computing Environments with SmartNet”, *Proc. of the 7th Heterogenous Computing Workshop* (HCW’98), Washington, DC, 1998.

[9] Y. C. Ho, R. Sreenivas, and P. Vaklili, “Ordinal Optimization of Discrete Event Dynamic Systems”, *Journal of Discrete Event Dynamic Systems,* Vol. 2, No. 2, pp.61-88, 1992.

[10] Y. C. Ho, Q. C. Zhao, and Q. S. Jia. Ordinal *Optimization, Soft Optimization for Hard problems.* Springer, 2007.

[11] K.. Hwang, G. Fox, and J. Dongarra, *Distributed and Cloud Computing Systems:, Grids,Clouds, and The Future Internet,* Morgan Kauffmann, 2011

[12] H. Li and R. Buyya, “Model-driven Simulation of Grid Scheduling Strategies”, *Proc. of 3rd IEEE Int’l Conf. on e-Science and Grid Computing,* 2007.

[13] K. Lu, and A. Y. Zomaya, “A Hybrid Schedule for Job Scheduling and Load Balancing in Heterogeneous Computational Grids,” *IEEE Int’l Parallel & Distributed Processing Symp.,* July 5–8, pp. 121–128, Austria.

[14] N. Metropolis and S. Ulam, “The Monte Carlo Method”, *Journal of the American Statistical Association*, 44 (247), pp.335–341, 1949.

[15] R. Prodan and M. Wieczorek, “Bi-criteria Scheduling of Scientific Grid Workflows”. *IEEE Trans. on Automation Science and Engineering*, 2009.

[16] R. Subrata, A. Y. Zomaya, and B. Landfeldt, “A Cooperative Game Framework for QoS Guided Job Allocation Schemes in Grids,” *IEEE Trans. on Computers,* Vol. 57, No. 10, pp. 1413–1422, 2008.

[17] S. Teng, L. H. Lee, and E. P.Chew, “Multi-objective Ordinal Optimization for Simulation Optimization Problems”, *Automatica,* pp.1884-1895, 2007.

[18] M. Wieczorek, R. Prodan, and A. Hoheisel, “Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem” *CoreGRID,* TR6-0106, 2007

[19] K. Xu, J. Cao, L. Liu, and C. Wu. “Performance Optimization of Temporal Reasoning for Grid Workflows Using Relaxed Region Analysis”. *Proc. of the 2nd IEEE Int. Conf. on Advanced Information Networking and Applications Workshops*, Okinawa, Japan, pp.187-194, 2008.

[20] F. Zhang, J. Cao, L. Liu, and C. Wu, “Fast Autotuning Configurations of Parameters in Distributed Computing Systems Using Ordinal Optimization”, *Proc. 38th Int. Conf. on Parallel Processing Workshops,* Vienna, 190-197, 2009.

[21] F. Zhang, J. Cao, X. Song, H. Cai, and C. Wu, “AMREF: An Adaptive MapReduce Framework for Real Time Applications”, *Proc. of 9th Int. Conf. on Grid and Cloud Computing* (GCC’10), China, 2010.

[22] Q. C. Zhao, Y. C. Ho, and Q. S. Jia. Vector Ordinal Optimization”, *Journal of Optimization Theory and Applications,* Vol. 125, No. 2, pp. 259-274, May 2005.