# Multi-Objective Scheduling of Many Tasks in Cloud Platforms

Fan Zhang

Kavli Institute for Astrophysics and Space Research

Massachusetts Institute of Technology

Cambridge, MA 02139, USA

Email: f_zhang@mit.edu

Junwei Cao

Research Institute of Information Technology

Tsinghua University

Beijing, China, 100084

Email: jcao@tsinghua.edu.cn

Keqin Li

Department of Computer Science

State University of New York

New Paltz, New York 12561, USA

Email: lik@newpaltz.edu

Samee U. Khan

Department of Electrical and Computer Engineering

North Dakota State University

Fargo, ND 58108-6050, USA

Email: samee.khan@ndsu.edu

## Abstract

The scheduling of a many-task workflow in a distributed computing platform is a well known NP-hard problem. The problem is even more complex and challenging when the virtualized clusters are used to execute a large number of tasks in a cloud computing platform. The difficulty lies in satisfying multiple objectives that may be of conflicting nature. For instance, it is difficult to minimize the makespan of many tasks, while reducing the resource cost and preserving the fault tolerance and/or the quality of service (QoS) at the same time. These conflicting requirements and goals are difficult to optimize due to the unknown runtime conditions, such as the availability of the resources and random workload distributions. Instead of taking a very long time to generate an optimal schedule, we propose a new method to generate suboptimal or sufficiently good schedules for smooth multitask workflows on cloud platforms.

Our new multi-objective scheduling (MOS) scheme is specially tailored for clouds and based on the ordinal optimization (OO) method that was originally developed by the automation community for the design optimization of very complex dynamic systems. We extend the OO scheme to meet the special demands from cloud platforms that apply to virtual clusters of servers from multiple data centers. We prove the sub-optimality through mathematical analysis. The major advantage of our MOS method lies in the significantly reduced scheduling overhead time and yet a close to optimal performance. Extensive experiments were carried out on virtual clusters with 16 to 128 virtual machines. The multitasking workflow is obtained from a real scientific LIGO workload for earth gravitational wave analysis. The experimental results show that our proposed algorithm rapidly and effectively generates a small set of semi-optimal scheduling solutions. On a 128-node virtual cluster, the method results in a thousand times of reduction in the search time for semi-optimal workflow schedules compared with the use of the Monte Carlo and the Blind Pick methods for the same purpose.


**Key Words**: Cloud computing, many-task computing, ordinal optimization, performance evaluation, virtual machines, workflow scheduling.

## 1 INTRODUCTION

Large-scale workflow scheduling demands efficient and simultaneous allocation of heterogeneous CPU, memory, and network bandwidth resources for executing a large number of computational tasks. This resource allocation problem is NP-hard [8], [22]. How to effectively schedule many dependent or independent tasks on distributed sources that could be virtualized clusters of servers in a cloud platform makes the problem even more complex and challenging to solve, with a guaranteed solution quality.

The many-task computing paradigms were treated in [29], [30], [31]. These paradigms pose new challenges to the scalability problem, because they may contain large volumes of datasets and loosely coupled tasks. The optimization requires achieving multiple objectives. For example, it is rather difficult to minimize the scheduling makespan, the total cost, to preserve fault tolerance, and the QoS at the same time. Many researchers have suggested heuristics for the aforesaid problem [39].

The execution of a large-scale workflow, encounters a high degree of randomness in the system and workload conditions [14], [41], such as unpredictable execution times, variable cost factors, and fluctuating workloads that makes the scheduling problem computationally intractable [17]. The lack of information on runtime dynamicity defies the use of deterministic scheduling models, in which the uncertainties are either ignored or simplified with an observed average.

Structural information of the workflow scheduling problem sheds a light on its inner properties and opens the door to many heuristic methods. No free lunch theorems [40] suggest that all of the search algorithms for an optimum of a complex problem perform exactly the same without the prior structural knowledge. We need to dig into the prior knowledge on randomness, or reveal relationship between scheduling policy and performance metrics applied.

The emerging cloud computing paradigm [9], [25], [47] attracts industrial, business, and academic communities. Cloud platforms appeal to handle many loosely coupled tasks simultaneously. Our LIGO [6] benchmark programs are carried out using a virtualized cloud platform with variable number of virtual clusters built with many virtual machines on a fewer physical machines and virtual nodes as shown in Fig. 1 of Section 3. However, due to the fluctuation of many task workloads in realistic and practical cloud platform, resource profiling and simulation stage on thousands of feasible schedules are needed. An optimal schedule on a cloud may take intolerable amount of time to generate. Excessive response time for resource provisioning in a dynamic cloud platform is not acceptable at all.

Motivated by the simulation-based optimization methods in traffic analysis and supply chain management, we extend the *ordinal optimization* (OO) [11], [12] for cloud workflow scheduling. The core of the OO approach is to generate a *rough model* resembling the life of the workflow scheduling problem. The discrepancy between the rough model and the

real model can be resolved with the optimization of the rough model. We do not insist on finding the best policy but a set of suboptimal policies. The evaluation of the rough model results in much lower scheduling overhead by reducing the exhaustive searching time in a much narrowed search space. Our earlier publication [46] have indicated the applicability of using OO in performance improvement for distributed computing system.

The remainder of the paper is organized as follows. Section 2 introduces related work on workflow scheduling and ordinal optimization. Section 3 presents our model for *multi-objective scheduling* (MOS) applications. Section 4 proposes the algorithms for generating semi-optimal schedules to achieve efficient resource provision in clouds. Section 5 presents the LIGO workload [42] to verify the efficiency of our proposed method. Section 6 reports the experimental results using our virtualized cloud platform. Finally, we conclude with some suggestions on future research work.

## 2 Related Work and Our Unique Approach

Recently, we have witnessed an escalating interest in the research towards resource allocation in grid workflow scheduling problems. Many classical optimization methods, such as opportunistic load balance, minimum execution time, and minimum completion time are reported in [10], suffrage, min-min, max-min, and auction-based optimization are reported in [4], [26].

Yu *et al.* [43], [44] proposed economy-based methods to handle large-scale grid workflow scheduling under deadline constraints, budget allocation, and QoS. Benoit *et al.* [1] designed resource-aware allocation strategies for divisible loads. Li and Buyya [19] proposed model-driven simulation and grid scheduling strategies. Lu and Zomaya [21] and Subrata *et al.* [36] proposed a hybrid policy and another cooperative game framework. J. Cao *et al.* applied [3] queue-based method to configure multi-server to maximize profit for cloud service providers.

Most of these methods were proposed to address single objective optimization problems. Multiple objectives, if considered, were usually being converted to either a weighted single objective problem or modeled as a constrained single objective problem.

Multi-objective optimization methods were studied by many research groups [7], [28], [18], [34], [38], [43], [45] for grid workflow scheduling. To make a summarization, normally two methods are used. The first one, as introduced before, is by converting all of the objectives into one applying weights to all objectives. The other one is a cone-based method to search for non-dominated solution, such as Pareto optimal front [15]. Concept of layer is defined by introducing Pareto-front in order to compare policy performances [13]. An improved version [37] uses the count that one particular policy dominates others as a measure of the goodness of the policy. Our method extends the Pareto-front

method by employing a new noise level estimation method as introduced in section 4.2.

Recently, Duan *et al.* [8] suggested a low complexity game-theoretic optimization method. Dogan and Özgüner [7] developed a matching and scheduling algorithm for both the execution time and the failure probability that can trade off them to get an optimal selection. Moretti *et al.* [24] suggested all of the pairs to improve usability, performance, and efficiency of a campus grid.

Wieczorek *et al.* [39] analyzed five facets which may have a major impact on the selection of an appropriate scheduling strategy, and proposed taxonomies for multi-objective workflow scheduling. Prodan and Wieczorek [28] proposed a novel dynamic constraint algorithm that outperforms many existing methods, such as LOSS and BDLS to optimize bi-criteria problems. Calheiros *et al*. [2] used a cloud coordinator to scale applications in the elastic cloud platform.

Smith *et al.* [33] proposed robust static resource allocation for distributed computing systems operating under imposed *quality of service* (QoS) constraints. Ozisikyilmaz *et al.* [27] suggested efficient machine learning method for system space exploration. To deal with the complexity caused by the large size of a scale crowd, a hybrid modeling and simulation based method was proposed in [5].

None of the above methods, to the furthest of our knowledge, consider the dynamic and stochastic nature of a cloud workflow scheduling system. However, the predictability of a cloud computing is less likely. To better understand the run-time situation, we propose the MOS, which is a simulation based optimization method systematically built on top of OO, to handle large-scale search space in solving many-task workflow scheduling problem. We took into account of multi-objective evaluation, dynamic and stochastic runtime behavior, limited prior structural information, and resource constraints.

Ever since the introduction of OO in [11], one can search for a small subset of solutions that are sufficiently good and computationally tractable. Along the OO line, many heuristic methods have been proposed in [12] and [35]. The OO quickly narrows down the solution to a subset of “good enough” solutions with manageable overhead. The OO is specifically designed to solve a problem with a large search space. The theoretical extensions and successful applications of OO were fully investigated in [32]. Constrained optimization [20] converts a multi-objective problem into a single-objective constrained optimization problem. Different from this work, we apply OO directly in multi-objective scheduling problems, which simplify the problem by avoiding the above constrained conversion. Selection rules comparison [16] combined with other classical optimization methods such as genetic algorithm, etc. have also been proposed.

In this paper, we modify the OO scheme to meet the special demands from cloud platforms, which we apply to virtual

clusters of servers from multiple data centers.

## 3 Multi-Objective Scheduling

In this section, we introduce our workflow scheduling model. In the latter portion of the section, we will identify the major challenges in realizing the model for efficient applications.

### 3.1 Workflow Scheduling System Model

Consider a workflow scheduling system over $S$ virtual clusters. Each virtual cluster has $m_i$ ($i$ = 1, 2, …, $S$) virtual nodes. We use $W$ workflow managers to control the job processing rates over multiple queues, as shown in Fig. 2. Each workflow manager faces $S$ queues, and each queue corresponds to only one virtual cluster. A *task class* is defined as a set of tasks that have the same task type and can be executed concurrently. There are a total of $K$ task classes.

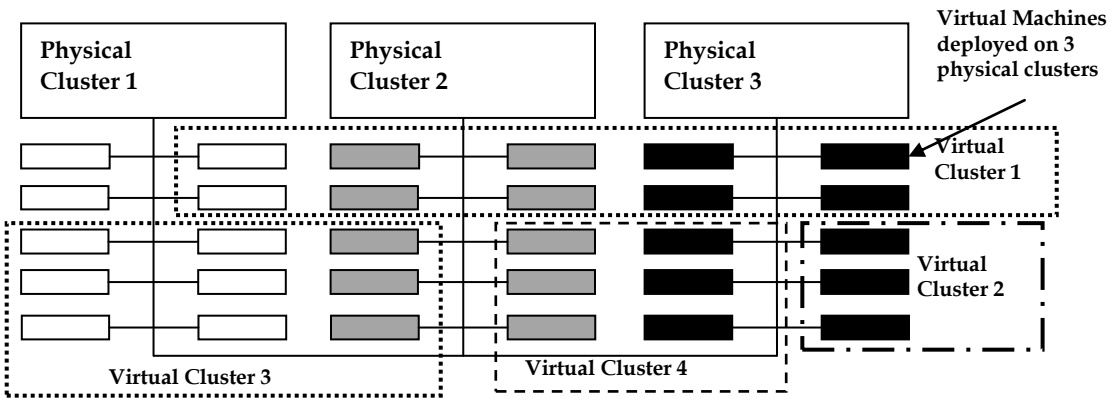



Figure 1. A cloud platform built with four virtual clusters over three physical clusters. Each physical cluster consists of a number of interconnected servers, represented by the rectangular boxes with three different shadings for the three physical clusters shown. The virtual machines (VMs) are implemented on the servers (physical machines). Each virtual cluster can be formed with either physical machines or VMs hosted by multiple physical clusters. The virtual clusters boundaries are shown by four dot/dash-line boxes. The provisioning of VMs to a virtual cluster can be dynamically done upon user demands.

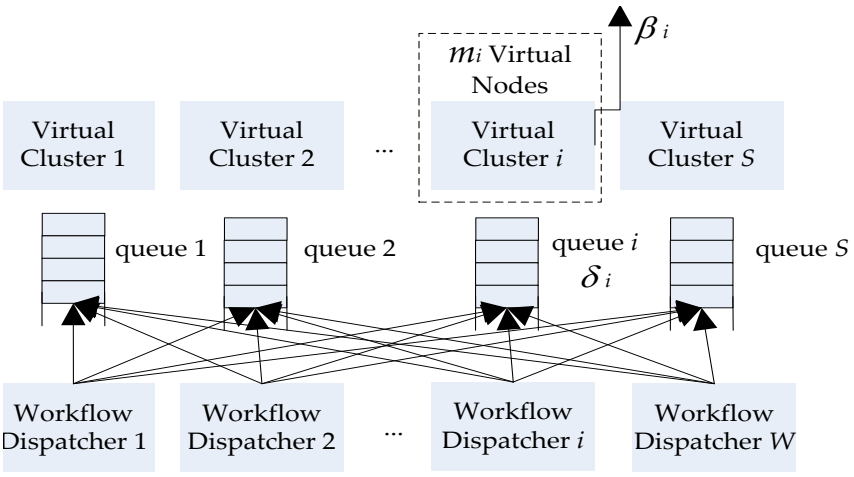



Figure 2. A queuing model of the VM resource allocation system for a virtualized cloud platform. Multiple workflow dispatchers are employed to distribute tasks to various queues. Each virtual cluster uses a dedicated queue to receive the incoming tasks from various workflows. The number of VMs (or virtual nodes) in each virtual cluster is denoted by $m_i$. The service rate is denoted by $\delta_i$ for queue *i*.

To benefit readers, we summarize the basic notations and their meanings below. The subscript *i* denotes virtual cluster *i*. The superscript *k* denotes the task class *k*.

**Table 1. Notations Used in Our Workflow System**

| Notation | Definition and Description |
|---|---|
| $\delta_i^{(k)}$ | Number of tasks in class *k* |
| $p_i^{(k)}$ | Expected execution time of tasks in class *k* |
| $\theta_i^{(k)}$ | Virtual nodes allocated to execute task class *k*. |
| $\beta_i^{(k)} = \theta_i^{(k)} / p_i^{(k)}$ | Job processing rate of task class *k* |
| $t_i^{(k)} = \delta_i^{(k)} / \beta_i^{(k)}$ | Remaining execution time of task class *k*. |
| $t^k$ | $\max\left\{t_1^{(k)}, t_2^{(k)}, ..., t_s^{(k)}\right\}$, remaining execution time of task class *k*. |
| $c_i^{(k)}$ | Cost of using one resource site for task class *k* |
| $C_i^{(k)} = c_i^{(k)} \theta_i^{(k)}$ | Total cost of task class *k* |

For simplicity, we describe a bi-objective model for minimizing the task execution time and resource operational cost. The first metric $J_1$ is the minimization of the sum of all execution times $t^k$. The minimization of the total cost $J_2$ is our

second optimization metric.

These two objective functions and the constraints are defined in the below mentioned equation to formulate our scheduling model. We need to choose a set of virtual node allocation policies $\{\theta_i^{(k)}\}$ for each task class *k* at virtual cluster *i*. The purpose is to minimize the total execution time ($J_1$) and the total systems cost ($J_2$), i.e.,

$$\left\{\min_{\theta_i^{(k)}}\left(J_1=\sum_{k=1}^{K}t^k, J_2=\sum_{i=1}^{S}\sum_{k=1}^{K}C_i^{(k)}\right)\right\}$$

$$=\left\{\min_{\theta_i^{(k)}}\begin{pmatrix} J_1=\sum_{k=1}^{K}\max\left(\left(\delta_i^{(k)}*p_i^{(k)}\right)\Big/\theta_i^{(k)}\right) & (i=1,2,\ldots S) \\ J_2=\sum_{i=1}^{S}\sum_{k=1}^{K}c_i^{(k)}\theta_i^{(k)} & \end{pmatrix}\right\}, \quad (1)$$

subject to $\sum_{k=1}^{K}\theta_i^{(k)}=m_i$. In general, we need to define *N* objective functions if there are *N* performance metrics to optimize.

### 3.2 Randomness and Simulation-based Optimization

Let Θ be the scheduling policy space of all of the possible solutions, i.e., $\Theta=\{\theta_i^{(k)} \mid i=1,2,\ldots,S;\ k=1,2,\ldots,K\}$. Let $\xi$ be a random variable to cover the randomness associated with resource uncertainty. In our problem, they are characterized by two parameters, i.e., $t_i^{(k)}$ and $c_i^{(k)}$, defined in Eq. (1). The following objective function is used to search for suboptimal policies for workflow scheduling. We attempt to minimize among the expected values as shown in Eq. (2):

$$\min_{\theta_i^{(k)}\in\Theta}\left\{J_l\left(\theta_i^{(k)}\right)\right\}$$

$$\equiv\min_{\theta_i^{(k)}\in\Theta}\left\{E_{t_i^{(k)}}\left[E_{c_i^{(k)}}\left[J_l\left(\theta_i^{(k)};c_i^{(k)},t_i^{(k)};T\right)\right]\right]\right\}$$

$$\approx\frac{1}{n}\sum\nolimits_{j=1}^{n}J_l\left(\theta_i^{(k)};\xi_j;T\right), \quad l=1,2. \quad (2)$$

Mathematically, we formulate the performance of the model as $J_l\left(\theta_i^{(k)};c_i^{(k)},t_i^{(k)};T\right), l=1,2$ that is a trajectory or sample path as the experiment evolves by time *T*. Then, we take the expectation with respect to the distribution of all the randomness, $t_i^{(k)}$ and $c_i^{(k)}$. To simplify the representation, we use $\xi_j$ to denote the randomness in $j^{\text{th}}$ replication of experiment. At last, the arithmetic mean of the *N* experiment is taken to get the *true performance* or *ideal performance* as we illustrate later, for policy $\theta_i^{(k)}$. Usually, we use a large number *n* in real experiments in order to compensate for the existing of large randomness.

### 3.3 Four Technical Challenges

To apply the above model, we must face four major technical challenges as briefed below.

(1) Limited knowledge of the randomness - The runtime conditions of the random variables ($t_i^{(k)}$, $c_i^{(k)}$) in real time are intractable. Profiling is the only solution to get their real time values for scheduling purpose. However, the collecting of CPU and memory information should be applied to all the scheduling policies in the search space.

(2) Very large search space - The number of feasible policies (search space size) in the above resource allocation problem is $|\Theta| = S * H(K,\theta_i - K) = S(\theta_i - 1)!/((\theta_i - K)!(K-1)!)$. This parameter $H(K,\theta_i - K)$ counts the number of ways to partition a set of $\theta_i$ VMs into $K$ nonempty clusters. Then $|\Theta|$ gives the total number of partition ways over all the $S$ clusters. This number can become extremely large when a new task class, namely $K + 1$, or a new site, namely $S + 1$, becomes available.

(3) Too many random variables to handle - There are $2*K*S$ random variables in this scheduling model to handle.

(4) Multiple objectives evaluation - In this workflow scenario, we have two objectives to optimize, which is much more difficult than having only one objective. We resort to a cone-based method (Pareto Optimal Front) [15] to handle such a problem, which is extendable to more objectives. The Pareto Optimal Front usually contains a set of policies. The details of this concept and the related solutions are introduced in Section 4.2.

## 4 VECTORIZED ORDINAL OPTIMIZATION

The OO method applies only to single objective optimization. The vector ordinal optimization (VOO) [15] method optimizes over multiple objective functions. In this section, we first specify the OO algorithm. Thereafter, we describe the MOS algorithm based on VOO as an extension of the OO algorithm.

### 4.1 Ordinal Optimization (OO) Method

The tenet in OO method is the *order* versus *value* observation. More specifically, exploring the best (order) policy is much easier than finding out the execution time and cost of that policy (value).

Instead of finding the optimal policy $\theta^*$ in the whole search space, the OO method searches for a small set $S$, which contains $k$ good enough policies. The success probability of such a search is set at $\alpha$ (e.g., 98%). The good enough policies are the top $g$ ($g \geq k$) in the search space $\Theta$. The numbers $k$ and $g$ are preset by the users. They follow the condition in Eq. (3):

$$P\left[\left|G \cap S\right| \geq k\right] \geq \alpha . \quad (3)$$

Formally, we specify the OO method in Algorithm 1. The *ideal performance,* denoted by $J\left(\theta_i^{(k)}\right)$, is obtained by averaging an $N$ times repeated Monte Carlo simulation for all the random variables. This $N$ is a large number, such as

1000 times in our case. The *measured performance* or *observed performance*, denoted by $\hat{J}\left(\theta_i^{(k)}\right)$, is obtained by averaging a less times repeated Monte Carlo simulation, say $n$ ($n$ << $N$) times, for all the random variables. That is why it is called *rough model*. Then, we formulate the discrepancy between the two models in Eq. (4):

$$\hat{J}\left(\theta_i^{(k)}\right) = J\left(\theta_i^{(k)}\right) + noise\ . \tag{4}$$

Instead of using the time consuming $N$-repeated simulations, we only have to use the rough model, which simulates $n$ times. The top $s$ best performance polices under the rough model is selected. The OO method guarantees that the $s$ policies include at least $k$ good enough policies with a high probability. The probability is set as $\alpha$ in Eq. (3). Finally, the selected $s$ policies should be evaluated under the $N$-repeated simulations in order to get one for scheduling use. We can see that the OO method narrows down the search space to $s$, instead of search among the entire space $\Theta$ that may have millions of policies to go through.

The number $s$ is determined by the following regression function $f$ with an approximated value:

$$s = f(\alpha, g, k, \omega, noise\ level) = e^{Z_0}(k)^{\rho}(g)^{\beta} + \eta \tag{5}$$

The values $\omega$ and *noise level* can be estimated based on the rough model simulation runs. The values such as $\alpha$, $k$, g are defined before experiment runs in Eq. (3). Value $\eta$ can be looked up in OO regression table. Value $e$ is a mathematical constant which equals to 2.71828.

In Fig. 3, we give a simple but concrete example to explain the OO method. Suppose the optimization function is linear $\min_{\theta\in\Theta} J(\theta) = \theta$, $\Theta = \{0,1,...,9\}$. (This function is unknown beforehand.) Therefore, we use a rough model $\hat{J}(\theta)$ where $\hat{J}(\theta) = J(\theta) + noise$. With this noise, the order has changed as shown in Fig. 3. For example, the best policy in $J(\theta)$ becomes rank 2 in $\hat{J}(\theta)$, and the second best policy in $J(\theta)$ becomes rank 4 in $\hat{J}(\theta)$. However, this variation is not too large. Selecting the top two policies in the rough model $\hat{J}(\theta)$ has one good-enough policy in $J(\theta)$.

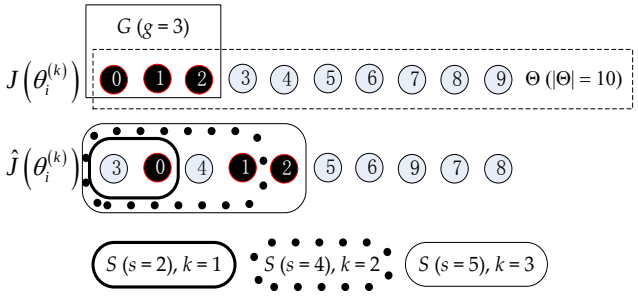



Figure 3. An example to illustrate how ordinal optimization works. The search space consists of 10 scheduling polices or schedules in ascending order. Good-enough set $G$ is shown by the left (best) 3 (set $g$ = 3) policies. We have to select two policies ($s$ = 2) to get at least one good-enough policy ($k$ = 1) in this example. Selecting four policies ($s$ = 4) would include two ($k$ = 2) good-enough policies.

In Algorithm 1, we show the steps of applying OO method. Suppose there is only one optimization objective. From lines 1 to 9, we use a rough model (10 repeated runs) to get the $s$ candidate policies. Then we apply the true model ($N$=1000) simulation runs on the $s$ candidate policies to find one for use. We select the 10 repeated runs intentionally, which accounts for 1% of the true model runs. In Corollary 1 below, we show this increases the sample mean variation by one order of magnitude. This variation is within the tolerance scope of our benchmark application.

**Corollary 1.** Suppose each simulation sample has Gaussian noise (0, $\sigma^2$). The variance of $n$ samples mean is $\sigma^2/n$. Furthermore, if the noise is i.i.d., then following the Central Limit Theorem, when $n$ is large the distribution of the sample mean converges to Gaussian with variance $\sigma^2/n$. If $\sigma$=1, in order to make sqrt{$\sigma^2/n$}=0.1 (reduce by one order of magnitude), we have $\sigma^2/n$ =0.01. Therefore $n$=1000 (two orders of magnitude larger than $n$=10).

**Algorithm 1. Ordinal Optimization for Workflow Scheduling**

**Input:**

$\Theta = \{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\}$: scheduling policy space

$g$: cardinality of $G$

$k$: alignment level

$\alpha$: alignment probability

**Output:** The measured best one

**Procedure:**

1. **for** all i = 1 : |Θ|
2. Simulate each policy $\theta_i$ *ten* times
3. Output time of $\theta_i$ as $t(\theta_i)$
4. **endfor**
5. Order the policies by a performance metric in ascending order $\{\theta_{[1]}, \theta_{[2]}, \ldots, \theta_{[|\Theta|]}\}$
6. Calculate $\omega$ and *noise level*
7. Look up the coefficient table to get ($Z_0$, $\rho$, $\beta$, $\eta$)
8. Generate the selection set $S$ using Eq. (5), $S = \{\theta_{[1]}, \theta_{[2]}, \ldots, \theta_{[s]}\}$
9. Simulate each policy in the selection set $N$ times
10. Apply the best policy from the simulation results in step 9

In Theorem 1, we give a lower bound of the probability $\alpha$, which is called alignment probability. In practice, this probability is set by users who apply the ordinal optimization based method. The larger this probability value is, the

larger the selection set $S$ should be. This is because the chance that a large selection set $S$ contains at least $k$ good-enough policies is larger than a small set $S$. Suppose $S$ equals to the whole candidate set $\Theta$, then the alignment probability $\alpha$ can be as much as 100%.

**Theorem 1** (Lower bound of the alignment probability of single objective problem): Suppose the size of selection set is $s$, and the size of good enough set is $g$. The lower bound of alignment probability is:

$$P\left[\left|G\cap S\right|\geq k\right] \geq \sum_{i=k}^{\min(g,s)}\binom{g}{i}\binom{|\Theta|-g}{s-i}\Big/\binom{|\Theta|}{s} \tag{6}$$

**Proof**: If the size of the selection set is $s$, then there are totally $\binom{|\Theta|}{s}$ ways of generating such a selection set. Suppose there are $i$ policies being selected from the good enough set. Therefore, there are $\binom{g}{i}$ ways for such a selection inside the good-enough set $G$. Meanwhile, there are $\binom{|\Theta|-g}{s-i}$ ways for the policies that are selected but outside the good-enough set $G$. Then, $\binom{g}{i}\binom{|\Theta|-g}{s-i}$ denotes the number of selections that there are $i$ good enough policies being selected in $G$. Thus, the probability of $P\left[\left|G\cap S\right|=k\right]$ is given by $\binom{g}{i}\binom{|\Theta|-g}{s-i}\Big/\binom{|\Theta|}{s}$. The value $k$ is constrained by $k\leq\min(g,s)$. Summarizing all the possible values of $i$ gives Eq. (6).

### 4.2 Multi-Objective Scheduling (MOS)

The multiple objective extension of OO leads to the vectorized ordinal optimization (VOO) method [15]. Suppose the optimization problem is defined in Eq. (7) below:

$$\min_{\theta_i^{(k)}\in\Theta} J\left(\theta_i^{(k)}\right)=\min_{\theta_i^{(k)}\in\Theta}\left(J_1\left(\theta_i^{(k)}\right),J_2\left(\theta_i^{(k)}\right),\cdots J_m\left(\theta_i^{(k)}\right)\right)^{\tau} \tag{7}$$

$$J_l\left(\theta_i^{(k)}\right)=E_{\xi}\left[L\left(\theta_i^{(k)},\xi\right)\right],\ \ l=1,2,\ldots,m$$

In general, there are $m$ optimization metrics. $\tau$ is the inverse of a vector. First, we introduce the key concepts, e.g., layers, Pareto front, etc., in multi-objective programming. Then, we continue with the vectorized extension. The VOO differs from the OO in six technical aspects as briefly characterized below.

(1) Dominance $(\succ)$: We say $\theta_x$ dominates $\theta_y$ ($\theta_x\succ\theta_y$) in Eq. (7), if $\forall\, l\in[1,\ldots,m]$, $J_l(\theta_x)\leq J_l(\theta_y)$, and $\exists\, l\in[1,\ldots,m]$, $J_l(\theta_x)<J_l(\theta_y)$.

(2) Pareto front: In Fig. 4(a), each white dot policy is dominated by at least one of the red dot policies, thus the red dot policies form the non-dominated layer, which is called the Pareto front $\{\ell_1\}$.

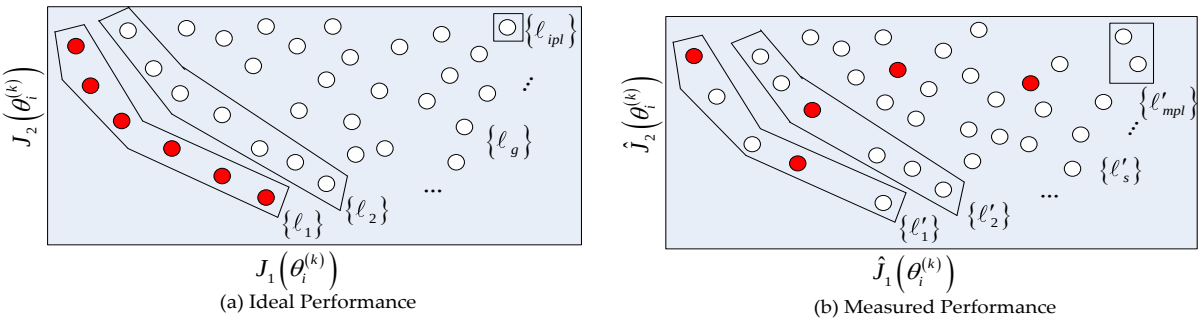



Figure 4. Illustration of layers and prateo fronts in both *ideal performance* and *measured performance* of a dual-objective optimization problem. Red dots in (a) are policies in Pareto front. We set $g$ = 1 (all the policies in Pareto front are good-enough solutions), at least 1 in the first layer ($s$ = 1) of the measured performance should be selected to align 2 good policies ($k$ = 2).

(3) Good enough set: The front $g$ layers $\{\{\ell_1\},...,\{\ell_g\}\}$ of the ideal performance are defined as the good enough set, denoted by $G$, as shown in Fig. 4(a).

(4) Selected set: The front $s$ layers $\{\{\ell'_1\},...,\{\ell'_s\}\}$ of the measured performance are defined as the selected set, denoted by $S$, as shown in Fig. 4(b).

(5) Ω type: It is also called *vector ordered performance curve* in VOO-based optimization. This concept is used to describe how the policies generated by the rough model are scattered in the search space as shown in Fig. 5. If the policies scattered in steep mode (the third Figure in both Fig. 5(a) and Fig. 5(b)), it would be easier to locate the good enough policies for a minimization problem. This is because most of them are located in the front $g$ layers. For example, if we set g = 2, the number of good enough polices in steep type is 9 compared with 3 in flat type. In this example, we can see that Ω type is also an important factor that size of selection set $s$ depends on.

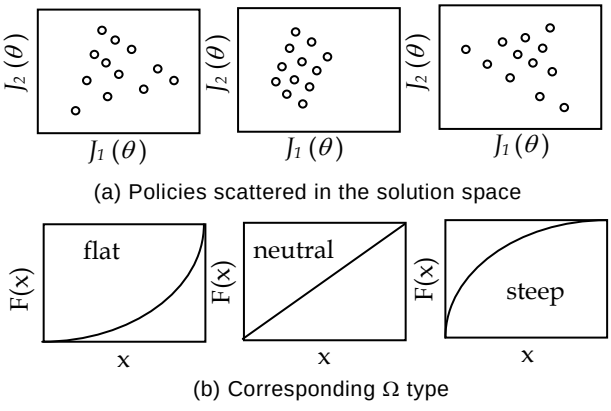



Figure 5. In (a), three kinds of Ω types are shown, by which 12 policies are scattered to generate 4 layers. In (b), the corresponding Ω

types for (a) are shown. The *x* identifies the layer index, and F(x) denotes how many policies are in the front *x* layers.

(6) Noise level: Noise level (*NL*) is used to define the similarity between the measured performance (rough model) and the ideal performance (real model). Mathematically, the noise level of the $l^{\text{th}}$ performance metric $J_l\left(\theta_i^{(k)}\right)$ is calculated as follows:

$$NL\left(J_l\left(\theta_i^{(k)}\right)\right)=\max_l\left\{\sigma\left(J_l\left(\theta_i^{(k)}\right)\right)\right\}\approx\max_l\left\{\sigma\left(\hat{J}_l\left(\theta_i^{(k)}\right)\right)\right\} \tag{8}$$

In Eq. (8), we first calculate the maximum performance standard deviation ($\sigma$) of all the policies, and then get the largest one along all the performance metrics (*l*). *NL* indicates the impact of the noise on performance using the rough model. However, we cannot get all $J_l\left(\theta_i^{(k)}\right)$ through *N* times experiments in MOS method. Instead, we approximate its value using the measured performance value $\hat{J}_l\left(\theta_i^{(k)}\right)$.

In OO method, three levels of noise are classified. If $0 < NL \le 0.5$, it is a small noise level problem. If $0.5 < NL \le 1$, it is a neutral noise level problem. If $1 < NL \le 2.5$, it is a large noise level problem.

**Theorem 2** (Lower bound of the alignment probability of multi-objective problem): Given the multiple objective optimization problem defined in Eq. (6), suppose the size of the $j^{\text{th}}$ layer $\left\{\ell_j\right\}$ is denoted by $\left|\ell_j\right|$, $j = 1,2,\ldots,ipl$, and the size of $\left\{\ell'_j\right\}$ is $\left|\ell'_j\right|$, $j = 1,2,\ldots,mpl$, the alignment probability is:

$$\begin{aligned}&P\left[\left|G\cap S\right|\ge k\right]\\&\ge\sum_{i=k}^{\min\left(\sum_{j=1}^{g}\left|\ell_j\right|,\sum_{j=1}^{s}\left|\ell'_j\right|\right)}\binom{\sum_{j=1}^{g}\left|\ell_j\right|}{i}\binom{\left|\Theta\right|-\sum_{j=1}^{g}\left|\ell_j\right|}{\sum_{j=1}^{s}\left|\ell'_j\right|-i}\Bigg/\binom{\left|\Theta\right|}{\sum_{j=1}^{s}\left|\ell'_j\right|}\end{aligned} \tag{9}$$

Values *ipl* and *mpl* denote the total number of the ideal performance layers and measured performance layers.

**Proof:** The size of the new good enough set is $\sum_{j=1}^{g}\left|\ell_j\right|$ and the size of the new selection set is $\sum_{j=1}^{s}\left|\ell'_j\right|$. We replace the *g* and *s* in Theorem 1 with $\sum_{j=1}^{g}\left|\ell_j\right|$ and $\sum_{j=1}^{s}\left|\ell'_j\right|$, then conclusion of Eq. (9) is proven.

MOS guarantees that if we select the front *s* observed layers, $S=\left\{\left\{\ell'_1\right\},\left\{\ell'_2\right\},\ldots,\left\{\ell'_s\right\}\right\}$, we can get at least *k* good enough policies in $G=\left\{\left\{\ell_1\right\},\ldots,\left\{\ell_g\right\}\right\}$ with a probability not less than $\alpha$, namely $P\left[\left|G\cap S\right|\ge k\right]\ge\alpha$. The number *k*, *g* and $\alpha$ are preset by users. $k\le\min\left(\sum_{j=1}^{g}\left|\ell_j\right|,\sum_{j=1}^{s}\left|\ell'_j\right|\right)$.

The size of the selection set *s* is also determined by Eq. (5). In Chapter IV of [12], the authors did regressed analysis to derive the coefficient table based on 10,000 policies with 100 layers in total. The analytical results should be revised

accordingly since our solution space $|\Theta|$ and measured performance layers *mpl* are different.

Based on the number of measure performance layers, we adjust the values of $g'$ and $k'$ as follows:

$$g' = \max\left\{1,\ \left\lfloor (100/mpl)\times g \right\rfloor\right\} \tag{10.a}$$

$$k' = \max\left\{1,\ (10{,}000/|\Theta|)\times k\right\} \tag{10.b}$$

Then, we look up the table given in Ho's book [12] to obtain the coefficients ($Z_0$, $\rho$, $\beta$, $\eta$) of a regression function to calculate $s'$ as follows:

$$s'(k', g') = e^{Z_0}(k')^{\rho}(g')^{\beta} + \eta \tag{11}$$

Finally, the $s$ is calculate by

$$s = \left\lceil (mpl/100)\times s' \right\rceil \tag{12}$$

All of the above procedures are summarized in Algorithm 2, which specifies the steps of the multi-objective scheduling of many task workflows, systematically based on vectorized ordinal optimization.

**Algorithm 2. MOS of Workflow Scheduling**

**Input:**

$\Theta = \{\theta_1, \theta_2, \ldots, \theta_{|\Theta|}\}$: policy space

*g*: number of good enough set

*k*: alignment level

$\alpha$: alignment probability

*n*: experiment time

**Output:** A set of good-enough scheduling policies for allocating resources

**Procedure:**

1. **for** all $i = 1 : |\Theta|$
2. Test each policy $\theta_i$ *ten* times
3. Output time and cost of $\theta_i$ as $\{\hat{J}_1(\theta_i), \hat{J}_2(\theta_i)\}$
4. **endfor**
5. Plot $\{\hat{J}_1(\theta_i), \hat{J}_2(\theta_i)\}$ ($i \in [1 : |\Theta|]$) in x-y axis
6. Plot the measured performance as Fig. 4(b)
7. Calculate the $\Omega$ and the *noise level*
8. Adjust *g* and *k* to get *g'* and *k'* using Eq. (10.a) and Eq. (10.b)
9. Look up the coefficient table to get ($Z_0$, $\rho$, $\beta$, $\eta$)
10. Calculate the initial *s'* using Eq. (11)
11. Adjust *s'* to get the related *s* using Eq. (12)
12. Test the policies at front *s* layers $\{\theta_{[1]}, \theta_{[2]}, \ldots, \theta_{[s]}\}$ to select the policies we need

# 5 LIGO Workflow Analysis

We first introduce our LIGO application background and many-task workload characterizations. Thereafter we design the details of the implementation steps to further describe the procedure of MOS.

## 5.1 Backgrounds and Workloads Characterization

Gravitational waves are produced by the movement of energy in mass of dense material in the earth. The LIGO (*Laser Interferometer Gravitational wave Observatory*) [6] embodies three most sensitive detectors (L1, H1, H2) in the world. The detection of the Gravitational Wave is very meaningfully for physical scientists to explore the origin of the Universe. Fig. 6 shows a typical detection workflow.

Real execution of the detection workflow requires a pre-verification phase. This phase can be utilized to understand potential run-time failures before the real data-analysis runs, which is normally very costly. The verification process contains several parallel and independent logics. Each verification logic is characterized by a class of tasks, which requires many resource sites (CPU and memory resources) as also being shown in Fig. 7. Different task requires different but unknown compute resources beforehand. Thus, a profiling or simulation is applied to understand the task properties for the follow-up scheduling stage.

In this benchmark application, we allocate a fix amount of virtual nodes for the seven verification logics. There are thousands of loosely coupled tasks in each verification logic. Thus, this is a typical many-task computing application.

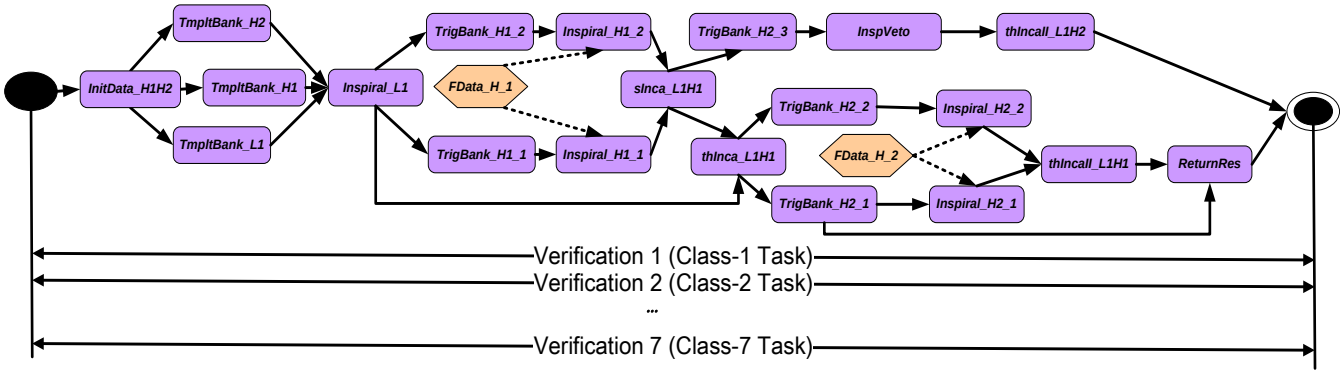



Figure 6. An example of resource allocation in many-task workflow for the LIGO workload, where the tasks are the 7 classes of verification programs running in parallel through the whole workflow.

The LIGO workloads are parallel programs to verify the following 7 task classes as illustrated in Fig. 6. The seven verification logics or task classes are executed in parallel. Below, we present a synopsis of the workload characteristics and the detailed descriptions can be found in [42].

Class-1 task is used to guarantee that once a temple bank (*TmpltBank*) has been created, two steps (*Inspiral* and

*TrigBank*) should immediately follow up.

Class-2 task ruled that process of matching with the expected wave of *H2* (*Inspiral_H2*) should be suspended, until both data in *H1* and *H2* pass the contingency tests (*Inca_L1H1, thInca_L1H1*).

Class-3 task denotes the data collected by three interferometers have to pass all contingency tests (*sInca*, *thInca* and *thIncall*) to minimize the noise signal ratio.

Class-4 task checks contingency tests (*thInca_L1H1*) should follow the process of matching with expected waves is done or template banks are created.

Class-5 and Class-6 task ensure all the services can be reached and properly terminated.

Class-7 task is used to check if all the garbage or temporary variables generated in runtime can be collected**.**

As aforesaid mentioned, runtime condition of the above seven task class are unknown a prior. The expected execution time of each task in Table 1 is profiled beforehand. Stochastic distribution of execution time corresponding to each task class is therefore generated. In the simulation phase, the execution time of each task is sampled from its own distribution region. Random execution time for each task class as it is, multiple samples should be collected to offer more simulation runs. More details are introduced in section 3.2.

**Table 2. Seven Task Classes Tested in Virtual Clusters**

| Task Class | Functional Characteristics | # of Parallel Tasks | # of Subtasks |
|---|---|---|---|
| Class-1 | Operations after templating | 3,576 | 947 |
| Class-2 | Restraints of interferometers | 2,755 | 6,961 |
| Class-3 | Integrity contingency | 5,114 | 185 |
| Class-4 | Inevitability of contingency | 1,026 | 1,225 |
| Class-5 | Service reachability | 4,962 | 1,225 |
| Class-6 | Service Terminatability | 792 | 1,315 |
| Class-7 | Variable garbage collection | 226 | 4,645 |

### 5.2 Implementation Considerations

We want to find a range of solutions to use $\theta_i^{(k)}$ for parallel workflows to minimize both execution time $J_1$ and total cost $J_2$. The 12 steps in Algorithm 2 of the MOS method over the LIGO datasets are described below.

Step 1 (Find measured performance of $\theta_i^{(k)}$): If $\delta_i^{(k)} = 0$, then $\theta_i^{(k)} = 0$; otherwise $\theta_i^{(k)} > 0$. This ensures each task class $k$

has at least one virtual cluster subject to $\sum_{k=1}^{K}\theta_i^{(k)}=\theta_i$. On virtual cluster *i*, there are $S(\theta_i-1)!/((\theta_i-K)!(K-1)!)$ feasible policies. For each policy we calculate $\theta_i^{(k)}=\left\{\left(\hat{J}_1(\theta_1),\hat{J}_2(\theta_1)\right),\ldots,\left(\hat{J}_1\left(\theta_{|\Theta|}\right),\hat{J}_2\left(\theta_{|\Theta|}\right)\right)\right\}$ for 10 times as a rough estimation.

Step 2 (Lay the measured performance policies): This step is to find out the dominance relationship of all of the measured performance as shown in Fig. 4(b). This step is composed of two sub-steps, i.e., first, giving all of the policies an order; second, putting the policies in different layers. They are shown in Algorithms 3 and 4, respectively.

**Algorithm 3. Order Measured Performance Policies**

**Input:**

PolicyList = {$\theta_1$, $\theta_2$, …, $\theta_{|\Theta|}$}

PerformanceList = $\left\{\left(\hat{J}_1(\theta_1),\hat{J}_2(\theta_1)\right),\ldots,\left(\hat{J}_1\left(\theta_{|\Theta|}\right),\hat{J}_2\left(\theta_{|\Theta|}\right)\right)\right\}$

**Output:**

Ordered Policy List: {PolicyList}

Equal performance list: EqualPerfList_i {i = 1, 2, … ,|Θ|}

//All the policies in EqualPerfList_i has the same performance as the policy in PolicyList(*i*)

**Procedure:**

**1.** **for** all *i* = 1 to |Θ|

**2.** **for** all *j* = 1 to |Θ|, $j \neq i$

**3.** Order PolicyList based on $\hat{J}_1$

**4.** **if** $\hat{J}_1(\theta_i)=\hat{J}_1(\theta_j)$

**5.** Order $\theta_i$ and $\theta_j$ based on $\hat{J}_2$ in PolicyList

**6.** **if** $\hat{J}_2(\theta_i)=\hat{J}_2(\theta_j)$

**7.** DeList(PolicyList, $\theta_j$)

**8.** Enlist(EqualPerfList_i, $\theta_j$)

**9.** **endfor**

**10.** **endfor**

**11.** Output(PolicyList)

**12.** Output(EqualPerfList_i)

Step 3 (Calculate the Ω type): We use $|\ell'_i|, i=1,2,\ldots mpl$, to represent the number of policies in layer $\ell'_i$. Then there are *mpl* pairs, $\left(1,|\ell'_1|\right),\left(2,|\ell'_1|+|\ell'_2|\right),\ldots\left(mpl,\sum_{i=1}^{mpl}|\ell'_i|\right)$. In this way, we derive our **Ω** type.

Step 4 (Calculate the noise level): We normalize the observed performance $\hat{J}_1(\theta_i), \hat{J}_2(\theta_i), i = 1,2,\ldots,|\Theta|$ into [-1,1], respectively, and find the maximum standard deviation in both $\hat{J}_1(\ )\left(\sigma_1^{\max}\right)$ and $\hat{J}_2(\ )\left(\sigma_2^{\max}\right)$. Then we select the larger of the two $\left(\sigma = \max\left(\sigma_1^{\max}, \sigma_2^{\max}\right)\right)$, $(1 < \sigma \le 2.5)$ as the noise level *NL*.

Step 5: We follow the steps in lines 8 to 12 in Algorithm 2 to get the selected set *S*. The MOS ensures that *S* contains at least *k* good enough policies, which are in the front *g* layers with a probability larger than $\alpha$.

**Algorithm 4. Lay Measured Performance Policies**

**Input:**

Ordered PolicyList

EqualPerfList_i {i = 1, 2, … ,|Θ|}

**Output:** $\{\ell'_1\}, \{\ell'_2\}, \ldots, \{\ell'_{mpl}\}$

**Procedure:**

**1.** *mpl*←1

**2.** **while** PolicyList ! = Null

**3.** $\{\ell'_{mpl}\} \leftarrow PolicyList(1)$

**4.** **for** all *i* = 1 to $|\ell'_{mpl}|$

**5.** **for** all *j* = 1 to |PolicyList|, $j \neq i$

**6.** **if** $\theta_i$ does not dominate $\theta_j$

**7.** $\{\ell'_{mpl}\} \leftarrow \theta_j$

**8.** DeList(PolicyList, $\theta_j$)

**9.** **endfor**

**10.** **endfor**

**11.** *mpl*←*mpl* +1

**12.** **endwhile**

**13.** **for** all *i* = 1 to |Θ|

**14.** **if** EqualPerfList_i != Null

**15.** **if** $\theta_i$ belongs to $\{\ell'_j\}$

**16.** Insert EqualPerfList_i into $\{\ell'_j\}$

**17.** **endfor**

**18.** Output $\{\ell'_1\}, \{\ell'_2\}, \ldots, \{\ell'_{mpl}\}$

# 6 Experimental Performance Results

In this section, we report and interpret the performance data based on resource allocation experiments in LIGO workflow scheduling applications.

## 6.1 Design of the LIGO Experiments

Our experiments are carried out using seven servers at the Tsinghua University. Each sever is equipped with Intel T5870 dual-core CPU, 4GB DRAM, and 320GB hard disk, respectively. We deploy at most 20 virtual machines on each physical server with VMWare workstation 6.0.0. Each virtual machine runs with windows XP sp3.

Our many task verification programs are written in Java. There are $K = 7$ task classes in LIGO workload. We have to evaluate 27,132 scheduling policies. Each task class requires allocating with up to 20 virtual machines. We experiment 1000 times per policy to generate the ideal performance. The average time and average cost out of 1000 results are reported.

## 6.2 Experimental Results

We plot the experimental results of each allocation policy in Fig. 7. The black dots are ($J_1$, $J_2$) pair for each policy. The circles are the Pareto front of the policy space.

Fig. 7(a) describes the execution time and total cost of the measured performance policies. The total cost is defined as the memory used. There are 152 layers. The Pareto front layer is denoted with the blue circles in the first layer.

In Fig. 7(b), we plot the ideal performance policies and their Pareto front. There are 192 policies in the Pareto front layer with a total of 147 layers, which is very close to 152 layers in the measured performance.

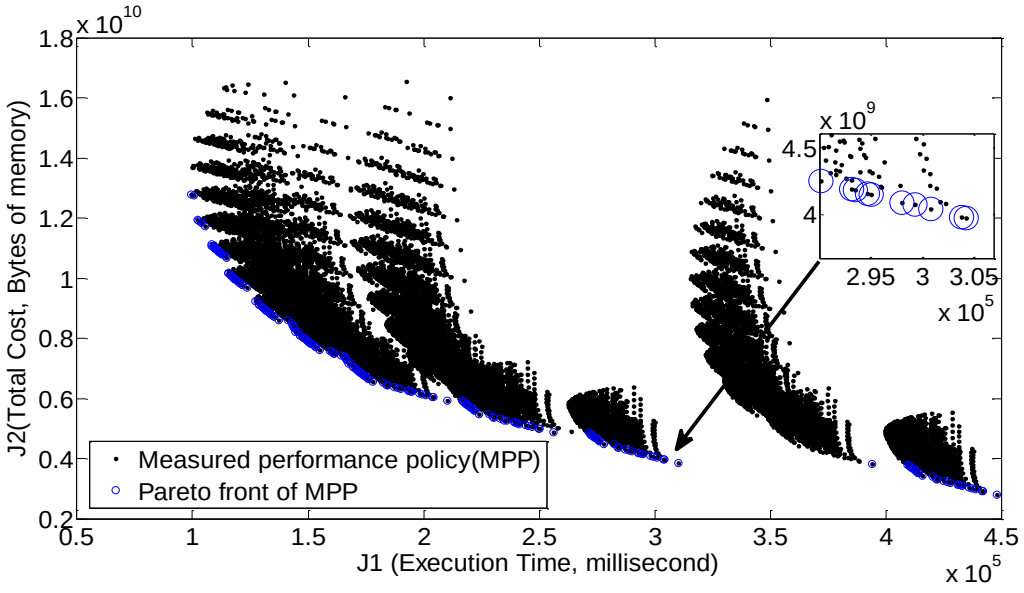
In real experiments, it is not possible to generate all these ideal performance because of the very long experiment time; instead we use the measured performance only. Later on, we will find out the discrepancy between them and discuss how to bridge the gap gracefully by analyzing the properties of this problem, such as the Ω type, and the noise level.

Our following MOS experiments reveal how many ideal scheduling policies (blue dots in Fig. 7(b)) are included in our selected good policies $S$.

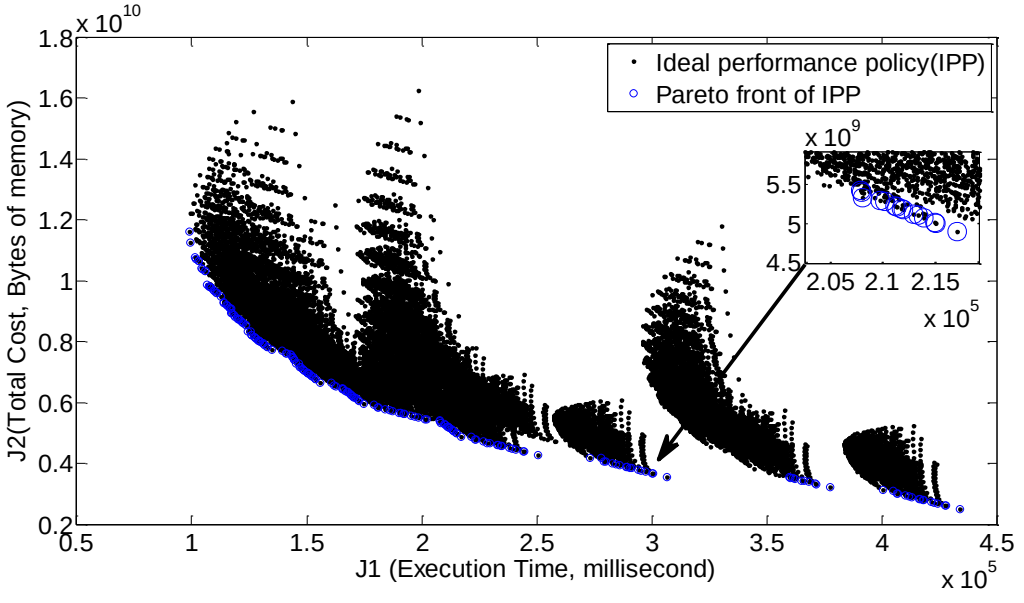
We aggregate the number of policies in each front $x$ layer as step 3 in Section 6.1 and get the Ω type in Fig. 8. The Ω type is a little bit steep in the front 60 layers and then becomes even steeper afterwards. This means the good enough policies are easy to search compared with neutral and flat type. Steep Ω type is chosen for this type of application.

To calculate the noise level, we first normalize the 27,132 measured execution time into [0,1] and find the standard deviation of the measured performance is 0.4831. The same is done on total cost and we get 0.3772 of that value. A small

noise level is therefore chosen. Based on the Ω type and the noise level, we look up the table [11] and get the coefficients of the regressed function as ($Z_0$, $\rho$, $\beta$, $\eta$) = (-0.7564, 0.9156, -0.8748, 0.6250).




(a) Measured performance




(b) Ideal performance

Figure 7. In a dual-objective scheduling example: the execution time in ms vs. total cost in memory byte. The measured performance is shown with ten experiments for each scheduling policy in Part (a). The Ideal performance results are plotted in Part (b), where 1000 experiments for each scheduling policy. In total, there are 27,132 resource allocation policies to be tested and selected from for the workflow scheduling of LIGO workload in a virtualized cloud platform. Each dot represents one scheduling policy. The circled dots form the Pareto front, which is used for the vectorized ordinal optimization process.

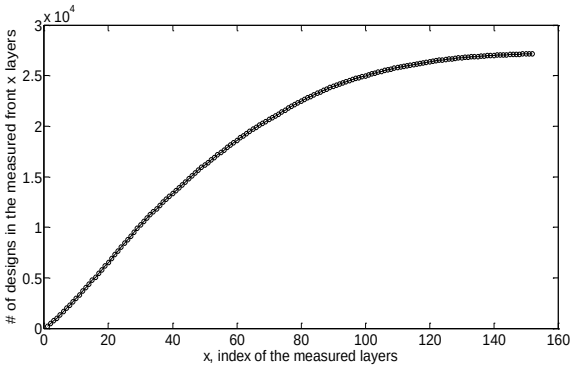



Figure 8. The Ω type of measured performance. It is a steep type corresponding to the good-enough policies (Pareto front of Fig.7(b) is easy to search from by using the MOS scheme).

Because we have so many good-enough policies, we choose $g$ = 1, namely, the Pareto front as our good enough set. Also, we choose alignment probability $\alpha$ = 98%. Based on Eq. (10.a) and (10.b), we have the following,

$$g' = \max\left\{1,\ \left\lfloor (100/152)\times 1 \right\rfloor\right\} = 1$$

and

$$k' = \max\left\{1,\ (10{,}000/27{,}132)\times k\right\}\begin{cases} = 1 & (1 \le k < 4) \\ \approx 0.369k & (k \ge 4) \end{cases}$$

Given an alignment level $k$, we can derive the size of initial selection layer based on Eq. (11) as:

$$s'(k', g') = e^{z_0}(k')^{\rho}(g')^{\beta} + \eta$$

We adjust $s'$ to $s$ based on Eq. (12) using $s = \lceil 152/100 \times s' \rceil$. Policies of the front $s$ layers in measured performance are our selection set $S$. To avoid confusion, we use $s$ in the front sections to represent the number of the selected layers, while we use $z$ here to represent how many policies are included in the front $s$ layers. The MOS works fine mainly attributed to the good Ω type in Fig.8 with the small noise level.

We plot different size of $k$ and its corresponding number of the selected layer $s$ in Fig. 9(a). Also, we plot in Fig. 9(b) different size of $k$ and its corresponding number of set $z$. This figure shows the results that if we want to find $k$ policies in the first one layer (Pareto front, $g$ = 1) with a probability larger than 98%, how many policies should we evaluate. Typical ($k$, $z$) pairs are (10, 995), (30, 2291), (50, 3668), …, (190, 12164).

When $k$ is small, the selection set $z$ is also very small. This is why MOS reduces at least one magnitude (from 27,132 to 995) of computation. We can also see that if we want to find all the 192 policies in Pareto front layer, we should test almost half of the policy space (22, 12164).

Fig. 10 shows that our MOS can generate many good-enough policies. This is seen by the overlap of Pareto front of

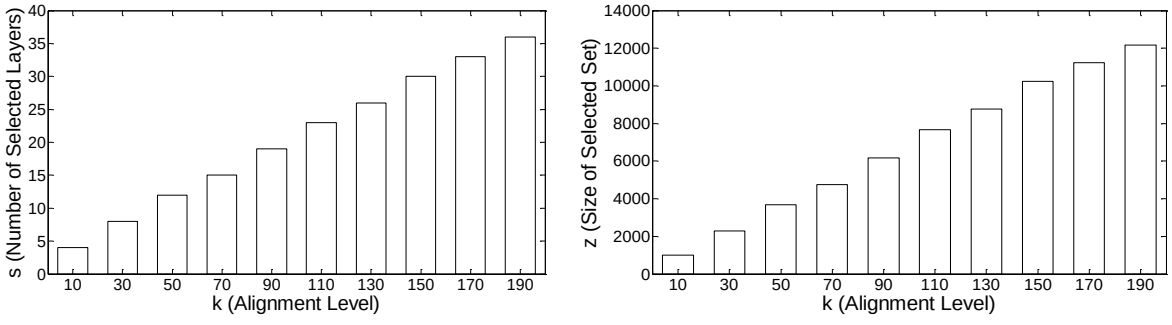
the ideal performance policies (denoted by the blue circles in the first layer) and the selected policies (denoted by the red circles) based on MOS in the Pareto front. Also, we see that the MOS finds many satisfactory results, which are those policies quite close to the Pareto front, though they are not good enough policies.

### 6.3 Comparison with Monte Carlo and Blink Pick Methods

Monte Carlo [23] is a typical cardinal optimization method. For one resource allocation policy $\theta_i^{(k)}$, we implement the $K$ task classes on the $S$ virtual clusters. In view of the fact that there are as many as $2KS$ number of uncertainties (execution time $t_i^{(k)}$ and operational cost $c_i^{(k)}$), the Monte Carlo method is firstly used to randomly sample the execution time and operational cost. We take the average value to estimate the scheduling search time over a 1000 repeated runs.

Blind Pick (BP), though it is named blind searching, is still a competitive method. We use different percentage of alignment probability (AP) to use this method in different scenarios. The AP is defined as the ratio of the search space that BP samples. The larger AP we use, the more policies we need to sample. If AP equals to one, it is the same as Monte Carlo simulation.

The performance metric is the number of the selection set size $z$ given a fixed number of good-enough policies required to be found over different methods. A good method shows its advantage by producing a small selection set which still includes the required good-enough policies. In Ho's book [12], various selection rules are compared and they concluded that there is no best selection rule for sure in all circumstances. In this paper, we prove a steep Ω type with a small noise level in our benchmark application and justify the use of the MOS method.




(a) Case of $s$ select layers (b) Case of $z$ selected sets

Figure 9. Number of selected policies in using the MOS method varies with the alignment level in the LIGO experiments.

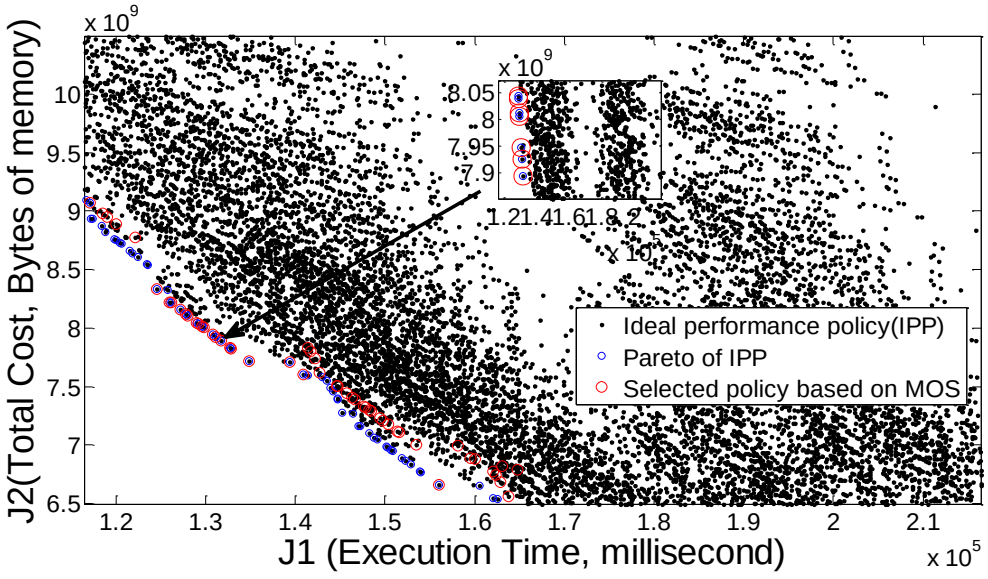



Figure 10. Bi-objective performance results of the MOS method in LIGO experiments. In total, there are 27,132 resource allocation policies to be selected from for the workflow scheduling of LIGO workload in a virtualized cloud platform. Each dot represents one scheduling policy. The circled dots form the Pareto front of the short list of good-enough policies.

We compare those methods in Figs. 11 to reveal how much search time reduction could be achieved. In order to find 10 good enough policies in the Pareto front, the Monte Carlo method has to go through the whole search space with 27,132 policies. However, the BP method with 50%, 95%, and 98% alignment probability needs to assess 1364, 2178, and 2416 policies respectively.

MOS has to assess only 995 policies, which is more than one order of magnitude search time reduction than the Monte Carlo and less than half compared with the BP method. We scale the good enough set to the whole Pareto front and the experiments revel that the search time of both blind pick and Monte Carlo is almost the same. MOS has to assess only 12,164 policies, which is also half of the search time of other methods. MOS shows its excellence in scalability in view of the increase of alignment level *k*.

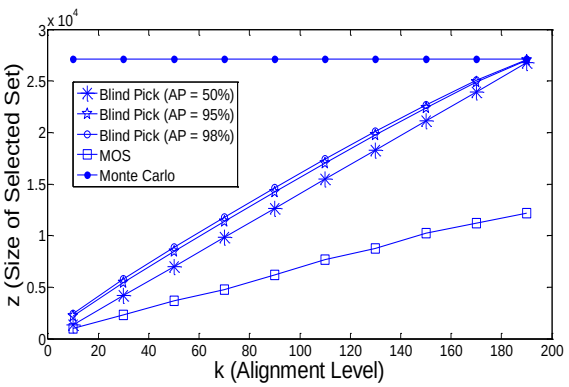



Figure 11. Comparison of MOS with the Blind Pick method under different alignment probabilities and with the Monte Carlo method. The MOS results in the least selected set. The Monte Carlo method requires the highest selected policy set.

We carried out the above experiments on different numbers of virtual nodes in our virtualized cloud platform. Seven task classes in Section 5.1 are also used in our four typical experimental environments with 16, 32, 64, and 128 virtual nodes respectively. For each environment, we have 5005, 736281, 67945521, and 5169379425 policies, which could be used to justify the scalability of those methods.

The Pareto front ($g$ = 1) is used as an indication of the good enough set. Let us take Fig. 11 as an example. There are 858 policies in the Pareto front. Thus, we set the x-axis every 85 policies a step and in all we select 10 experiment result every method. We make a similar comparison with this twenty-virtual-nodes experiment.

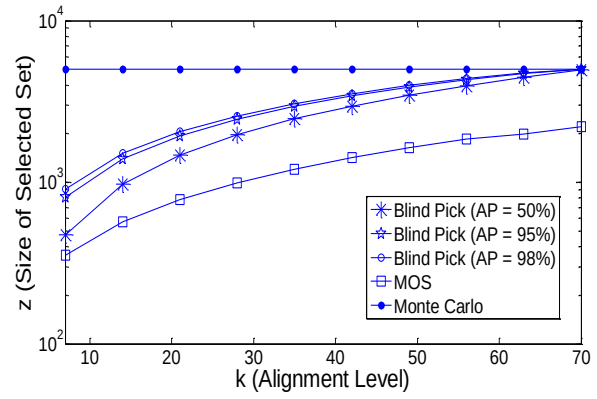
In Fig. 12(b), our MOS achieves one order of magnitude search time reduction than blind pick and nearly two orders of magnitude than Monte Carlo when $k$ is small. As the increase of alignment level $k$, MOS can maintain at least one order of magnitude search time advantage than other methods.

We scale to 128 virtual nodes to make the comparison in Fig. 12(d). The results reveal that MOS can maintain two orders of magnitude search time reduction given a small $k$. We can achieve nearly a thousand times reduction in search time, if we scale $k$ to cover the entire Pareto front.

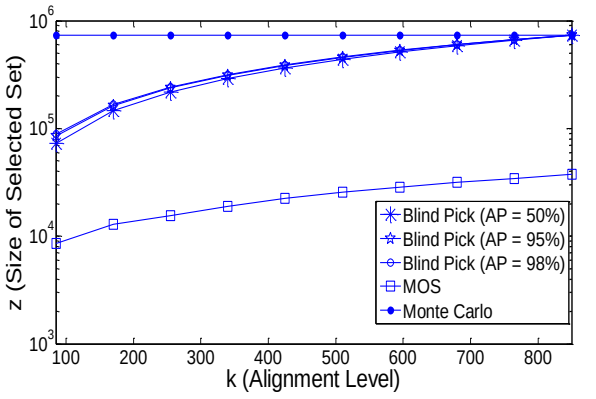
It is not occasional that the selection set does not vary as the increase of alignment level in 128 virtual nodes. Since there are so many policies in search space, the Pareto front covers only a much smaller portion. In this way, the effectiveness of "ordinal" takes more effects than traditional "cardinal" methods. If we continue to increase the number of $k$, such as taking ($g$ = 2), the selection set of MOS will increase while maintaining the search time reduction advantage.

## 7. CONCLUSIONS

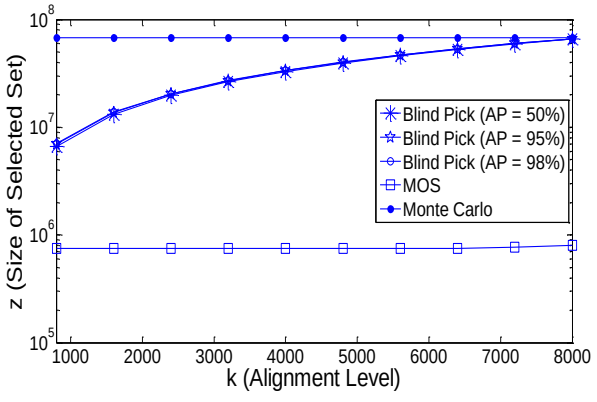
In this paper, we have extended the ordinal optimization method from single objective to multiple objectives using a *vectorized ordinal optimization* (VOO) approach. We are the very first research group proposing this VOO approach to achieve *multi-objective scheduling* (MOS) in many-task workflow applications. Many-task scheduling is often hindered by the existence of large amount of uncertainties. Our original technical contributions are summarized below.



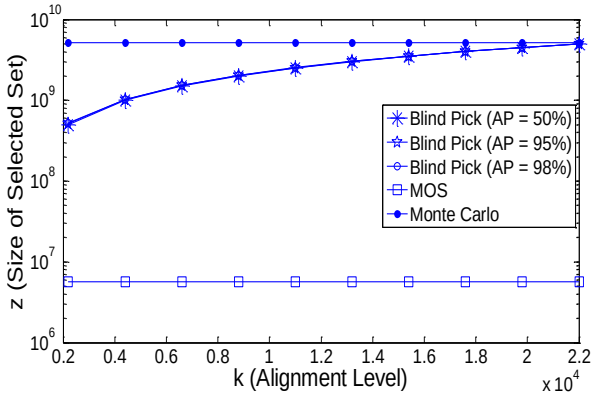

(a) 16 virtual nodes




(b) 32 virtual nodes




(c) 64 virtual nodes




(d) 128 virtual nodes

Figure 12. Experimental results on the size of the selected policy sets on cloud platforms running on 7 virtual clusters consisting of 16, 32, 64, and 128 virtual node (VMs). Again, the MOS method outperforms the Monte Carlo and all Blink-Pick scheduling methods.

(1) We proposed the VOO approach to achieving multi-objective scheduling (MOS) for optimal resource allocation in cloud computing. The OO is specified in Algorithm 1. The extension of OO, MOS scheme, is specified in Algorithm 2.

(2) We achieved problem scalability on any virtualized cloud platform. On a 16-node virtual cluster, the MOS method reduced half of searching time, compared with using the Monte Carlo and Blind Pick methods. This advantage can be scaled to a thousand times reduction in scheduling overhead as we increase the cloud cluster to 128 virtual machines.

(3) We have demonstrated the effectiveness of the MOS method on real-life LIGO workloads for earth gravitational wave analysis. We used the LIGO data analysis to prove the effectiveness of the MOS method in real-life virtualized cloud platforms from 16 to 128 nodes.

For further research, we suggest to extend the work in the following two directions.

(1) Migrating MOS to larger platform - We plan to migrate our MOS to even larger cloud platform with thousands or more virtual nodes to execute millions of tasks.

(2) Building useful tools to serve larger virtualized cloud platform - Cloud computing allocates virtual cluster resources in *software as a service* (SaaS), *platform as a service* (PaaS), and *infrastructure as a service* (IaaS) applications. Our experimental software can be tailored and prototyped toward this end.

## ACKNOWLEDGMENTS

The authors would like to express their heartfelt thanks to all the referees who provided critical and valuable comments to the manuscript. This work was supported in part by Ministry of Science and Technology of China under National 973 Basic Research Program (grants No. 2011CB302805, No. 2013CB228206 and No. 2011CB302505), National Natural Science Foundation of China (grant No. 61233016), and Tsinghua National Laboratory for Information Science and Technology Academic Exchange Program. Fan Zhang would like to show his gratitude to Prof. Majd F. Sakr, Qutaibah M. Malluhi and Tamer M. Elsayed for providing support under NPRP grant 09-1116-1-172 from the Qatar National Research Fund (a member of Qatar Foundation).

## REFERENCES

[1] A. Benoit, L. Marchal, J. Pineau, Y. Robert, F. Vivien, Resource-aware allocation strategies for divisible loads on large-scale systems, in: Proceedings of IEEE Int'l Parallel and Distributed Processing Symposium, IPDPS 2009, pp. 1-4.

[2] R. N. Calheiros, A. N. Toosi, C. Vecchiola and R. Buyya, A coordinator for scaling elastic applications across multiple clouds, Future Generation Computer Systems 28 (8) (2012) 1350-1362.

[3] J. Cao, K. Hwang, K. Li, and A. Zomaya, Profit maximization by optimal multiserver configuration in cloud computing, IEEE Transactions on Parallel and Distributed Systems (special issue on cloud computing) 24 (6) (2013) 1087-1096.

[4] H. Casanova et al, Heuristic for scheduling parameters sweep applications in grid environments, in: Proceedings of the 9th Heterogenous Computing Workshop, HCW 2009, pp. 349-363.

[5] D. Chen, L. Wang, X. Wu, et al, Hybrid modelling and simulation of huge crowd over a hierarchical Grid architecture, Future Generation Computer Systems 29 (5) (2013) 1309-1317.

[6] E. Deelman, C. Kesselman, et al, GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists, in: Proceeding of the 11th IEEE Int'l Symp. on High Performance Distributed Computing, HPDC 2002, pp. 225-234.

[7] A. Dogan, and F. Özgüner, Biobjective Scheduling Algorithms for Execution Time–Reliability Trade-off in Heterogeneous Computing Systems, The Computer Journal 48 (3) (2005) 300-314.

[8] R. Duan, R. Prodan, and T. Fahringer, Performance and Cost Optimization for Multiple Large-scale Grid Workflow Applications, in: Proceeding of IEEE/ACM Int'l Conf. on SuperComputing, SC 2007, pp. 1-12.

[9] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, in: IEEE Grid Computing Environments, GCE 2008, co-located with IEEE/ACM Supercomputing, SC 2008, pp. 1-10.

[10] R. Freund et al, Scheduling resources in multi-user, heterogeneous, computing environments with smartne, in Proceeding of the 7th Heterogenous Computing Workshop, HCW 1998, pp. 184-199.

[11] Y. C. Ho, R. Sreenivas, and P. Vaklili, Ordinal Optimization of Discrete Event Dynamic Systems, Journal of Discrete Event Dynamic Systems 2 (2) (1992) 61-88.

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

[13] Y. C. Ho, Q. C. Zhao and Q. S. Jia, Vector Ordinal Optimization, Journal of Optimization Theory and Applications, 125(2) (2005) 259-274.

[14] K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hilly, 1998.

[15] Q. Jia, Enhenced Ordinal Optimization: A Theoretical Study and Applications, Ph.D. Thesis, Tsinghua University, China, 2006.

[16] Q. S. Jia, Y. C. Ho, and Q. C. Zhao, Comparison of selection rules for ordinal optimization, Mathematical and Computer Modelling, 43 (9) (2006) 1150-1171.

[17] S. Lee and R. Eigenmann, Adaptive Tuning in a Dynamically Changing Resource Environment, in: Proceeding of IEEE In'l Parallel & Distributed Processing Symp, IPDPS 2008, pp. 1-5.

[18] Y. C. Lee and A. Y. Zomaya, On the Performance of a Dual-Objective Optimization Model for Workflow Applications on Grid Platforms, IEEE Transactions on Parallel and Distributed Systems 20 (9) (2009) 1273–1284.

[19] H. Li and R. Buyya, Model-driven Simulation of Grid Scheduling Strategies, in Proceeding of the 3rd IEEE Int'l Conf. on e-Science and grid Computing, escience 2007, pp. 287-294.

[20] D. Li, L. H. Lee, and Y. C. Ho, Constraint ordinal optimization, Information Sciences, 148(1-4) (2002) 201-220.

[21] K. Lu, and A. Y. Zomaya, A Hybrid Policy for Job Scheduling and Load Balancing in Heterogeneous Computational Grids, in: Proceeding of 6th IEEE In'l Parallel & Distributed Processing Symp., ISPDC 2007, pp. 121–128.

[22] M. Maheswaran, H. J. Siegel, D. Haensgen, and R. F. Freud, Dynamic mapping of A Class of Independent taskss onto Heterogeneous Computing Systems, Journal of parallel and Distributed Computing, 59 (2) (1999) 107-131.

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

[24] C. Moretti, H. Bui, K. Hollingsworth, B. Rich et al, All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids. IEEE Transactions on Parallel Distributed Systems 21 (1) (2010) 33-46.

[25] C. Moretti, K. Steinhaeuser, D. Thain, and N. V. Chawla, Scaling up Classifiers to Cloud Computers, in: IEEE International Conference on Data Mining, ICDM 2008, pp. 472-481.

[26] A. Mutz, and R. Wolski, Efficient Auction-Based Grid Reservation using Dynamic Programming, in: IEEE/ACM Int'l Conf. on SuperComputing, SC 2007, pp. 1-8.

[27] B. Ozisikyilmaz, G. Memik, and A. Choudhary, Efficient System Design Space Exploration using Machine Learning Techniques, in: Proceedings Design Automation Confonference, DAC2008, pp. 966-969.

[28] R. Prodan and M. Wieczorek, Bi-criteria scheduling of scientific Grid workflows, IEEE Transactions on Automation Science and Engineering 7 (2) (2010) 364-376.

[29] Ioan Raicu, Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing, ISBN: 978-3-639-15614-0, VDM Verlag Dr. Muller Publisher, 2009

[30] I. Raicu, I. Foster, Y. Zhao, Many-Task Computing for Grids and Supercomputers, IEEE Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS 2008, co-located with IEEE/ACM Supercomputing, SC 2008, pp. 1-11.

[31] I. Raicu, I. Foster, Y. Zhao et al, The Quest for Scalable Support of Data Intensive Workloads in Distributed Systems, in: Int'l ACM Symposiums on High Performance Distributed Computing, HPDC 2009, pp. 207-216.

[32] Z. Shen, H. Bai, and Y. Zhao, "Ordinal optimization references list." [Online]. Available: http://www.cfins.au.tsinghua.edu.cn/uploads/Resources/Complete_Ordinal_Optimization_Reference_List_v7.doc

[33] J. Smith, H. J. Siegel and A. A. Maciejewski, A Stochastic Model for Robust Resource Allocation in Heterogeneous Parallel and Distributed Computing Systems, in: Proceeding of IEEE Int'l Parallel & Distributed Processing Symp., IPDPS 2008, pp. 1-5.

[34] S. Song, K. Hwang, and Y. Kwok, Risk-Tolerant Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling, IEEE Transactions on Computers 55 (6) (2006) 703-719.

[35] C. Song, X. H. Guan, Q. C. Zhao, and Y. C. Ho, Machine Learning Approach for Determining Feasible Plans of a Remanufacturing System, IEEE Tranactions on Automation Science and Engineering 2 (3) (2005) (262 - 275).

[36] R. Subrata, A. Y. Zomaya, and B. Landfeldt, A Cooperative Game Framework for QoS Guided Job Allocation Schemes in Grids, IEEE Transactions on Computers, 57 (10) (2008) 1413–1422.

[37] S. Teng, L. H. Lee, and E. P. Chew, Multi-objective ordinal optimization for simulation optimization problems, Automatica, 43(11) (2007) 1884-1895.

[38] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer, Applying Double Auctions for Scheduling of Workflows on the Grid, in: Proceedings of IEEE/ACM Int'l Conf. on SuperComputing, SC 2008, pp. 1-11.

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

[40] D. H. Wolpert and W. G. Macready, No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1 (1) (1997) 67–82.

[41] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, Adaptive Workload Prediction of Grid Performance in Confidence Windows, IEEE Transactions on Parallel and Distributed Systems 21 (7) (2010) 925-938.

[42] K. Xu, J. Cao, L. Liu, and C. Wu, Performance Optimization of Temporal Reasoning for Grid Workflows Using Relaxed Region Analysis, in: Proceeding of the 2nd IEEE Int'l Conf. on Advanced Information Networking and Applications, AINA 2008, pp.187-194.

[43] J. Yu and R. Buyya, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, Scientific Programming Journal, 14 (1) (2006) 217-230.

[44] J. Yu, M. Kirley, and R. Buyya, Multi-objective Planning for Workflow Execution on Grids, in: Proceeding of the 8th IEEE/ACM Int'l Conf. on Grid Computing, Grid 2007, pp. 10-17.

[45] F. Zhang, J. Cao, K. Hwang, and C. Wu, Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds, in: Proceeding of the 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011, pp. 9-17.

[46] F. Zhang, J. Cao, L. Liu, and C. Wu, Fast Autotuning Configurations of Parameters in Distributed Computing Systems using OO, in: Proceeding of the 38th International Conference on Parallel Processing Workshops, ICPPW 2009, pp. 190-197.

[47] Y. Zhao, I. Raicu and I. Foster, Scientific Workflow Systems for 21st Century, New Bottle or New Wine?, in: Proceedings of the 2008 IEEE Congress on Services, SERVICE 2008, pp. 467-471.