Evolutionary Scheduling of Dynamic Multitasking Workloads for Big-data Analytics in Elastic Cloud

Fan Zhang, *Senior Member, IEEE*, Junwei Cao, *Senior Member, IEEE*, Wei Tan, *Senior Member, IEEE*
Samee U. Khan, *Senior Member, IEEE*, Keqin Li, *Senior Member, IEEE* and Albert Y. Zomaya, *Fellow, IEEE*

***Abstract*—Scheduling of dynamic and multitasking workloads for big-data analytics is a challenging issue, as it requires a significant amount of parameter sweeping and iterations. Therefore, real-time scheduling becomes essential to increase the throughput of many-task computing. The difficulty lies in obtaining a series of optimal yet responsive schedules. In dynamic scenarios, such as virtual clusters in cloud, scheduling must be processed fast enough to keep pace with the unpredictable fluctuations in the workloads to optimize the overall system performance.**

In this work, ordinal optimization using rough models and fast simulation is introduced to obtain suboptimal solutions in a much shorter timeframe. While the scheduling solution for each period may not be the best, ordinal optimization can be processed fast in an iterative and evolutionary way to capture the details of big data workload dynamism. Experimental results show that our evolutionary approach compared with existing methods, such as Monte Carlo and Blind Pick can achieve higher overall average scheduling performance, such as throughput in real-world applications with dynamic workloads. Further performance improvement is seen by implementing an Optimal Computing Budget Allocating (OCBA) method that smartly allocates computing cycles to the most promising schedules.

***Index Terms*—Big-data, cloud computing, evolutionary ordinal optimization, multitasking workload, virtual clusters.**

I. Introduction

Large-scale business and scientific applications are usually composed of big-data, multitasking, time-variant, and fluctuating workloads [5], [35]. Cloud computing [2], with virtualization [3] as the key enabling technology, provides an elastic scaling-up and scaling-down provisioning mechanism. Agile and appropriate computational resource provisioning that keeps pace with the fluctuations of big-data multitasking workloads is very important in the scheduling paradigm [41]. For example, there are unprecedented amounts of requests from the customers of Amazon and eBay during the holiday seasons. Thereafter, the website traffic drops down dramatically. Over-provisioning of computing resource to constantly satisfy the requirements at the peak level leads to high and unnecessary cost, while under-provisioning leads to user churn [28].

In general, scheduling big-data multitasking workloads onto distributed computing resources is an NP-complete problem [13]. The main challenge is to maintain a reasonable tradeoff between the scheduling overhead and scheduling accuracy.

Consider a continuous scheduling scenario with multiple scheduling periods, and within each period the scheduling scheme is being repeatedly updated. If the scheduling period is too long, while a seemingly more descent solution can be achieved through elaborate workload analysis, the overall system performance may degrade. This is due to the fact that the workloads might have changed considerably during such a long time. Therefore, it is necessary that scheduling solutions are provided in an evolving fashion, so that during each iteration, a "good-enough", suboptimal, but fast-paced solution can be obtained. In the meanwhile, evolving iterations of optimization can adapt to the nature/details of system dynamism, such as dynamic workload fluctuation and resource provisioning in virtual clusters to achieve better performance.

In our previous work [40], [41], [42], [43], for the purpose of fast scheduling, we have explored the possibility of applying simulation-based optimization methods. Ordinal Optimization (OO) [15] has been applied for suboptimal but fast searching. The OO is utilized for searching suboptimal solutions in much shorter time and with reduced runs and analysis of the workloads. Targeting at a sequence of suboptimal schedules instead of an optimal one results in much lower scheduling overhead by reducing the exhaustive searching time [40]. Experiments show that the methods capture workload characteristics and lead to an overall improved performance. However, the scheduling overhead of applying those methods still refrains its applicability in highly fluctuated workloads.

In this paper, we propose an iterative and evolutionary usage of ordinal optimization to further reduce the scheduling overhead. Our major contribution consists of the following:

- The iterative OO (iOO) method published in our earlier paper [40], is applied in the new and real

Manuscript received Oct 15, 2013.

Fan Zhang is with the Kavli Institute for Astrophysics and Space Research, MIT, Cambridge, MA 02139, USA (e-mail: f_zhang@mit.edu). This work was carried out when he was a PhD student of Tsinghua University, Beijing 100084, China.

Junwei Cao is with the Research Institute of Information Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: jcao@tsinghua.edu.cn). He is the corresponding author.

Wei Tan is with the IBM T. J. Watson Research Center, Yorktown Heights, New York 10598, USA (e-mail: wtan@us.ibm.com).

Samee U. Khan is with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA (e-mail: samee.khan@ndsu.edu).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, New York 12561, USA (e-mail: lik@newpaltz.edu).

Albert Y. Zomaya is with the School of Information Technologies, University of Sydney, Sydney, NSW 2006, Australia (e-mail: albert.zomaya@sydney.edu.au)

multitasking-scheduling model in this paper. As expected, the iOO method shows up to 20% performance speedup than competing methods.

- In a series of rapidly fluctuating workload periods, we contribute a procedure of using short-phase scheduling for fine-grained workload analysis. In stable workload phases, long-term scheduling is accordingly contributed which intends to save time to analyze the workloads for accurate scheduling.
- A step further, as an extension of iOO, the evolutionary OO (eOO) analyzes the workload patterns among consecutive phases and adjusts the scheduling based on the patterns. We develop a series of algorithms to partition and merge workload for efficient scheduling.
- We use a dynamic scenario of scheduling multitask scientific workloads to a group of virtual clusters on Amazon EC2 cloud. The experiment results show that the eOO approach achieves up to 30% performance speedup regarding task throughput, compared with Monte Carlo [23] and Blind Pick. As far as we know, this is the first time an OO is applied in an evolutionary way for dynamic optimization scenarios to meet special requirements of cloud workload scheduling.

The rest of the paper is organized as follows. Section II provides a brief description of the dynamic multitasking workload scheduling problem and existing methods. In Section III, we describe our proposed eOO technique that iteratively applies the OO for scheduling workloads on virtual clusters of a cloud. The experimental results, performance evaluation, and comparative analysis are presented in Section IV that also includes detailed information on the applications and system configurations. The related work is reviewed in Section V, and we provide concluding remarks in Section VI.

II. Dynamic Workload Scheduling

In this section, first, we introduce our dynamic multitasking-scheduling model. Thereafter, the necessity of using simulations to is presented. To follow, three existing approaches, Monte Carlo, Blind Pick, and iterative Ordinal Optimization, are introduced. To each readership, Table I summarizes the most frequently used symbols, notations, and definitions.

A. Dynamic Multitasking Workload Scheduling Model

In this model, we define the *task class* as a set of tasks that are of the same type and can be executed concurrently. Suppose there are C task classes in all, and the index is denoted as c, $c \in [1,C]$. Tasks within one task class can be either interdependent, such as scientific workflow, or independent of each other, such as scientific simulations with multiple parameters. Tasks across different task classes are independent.

Virtual Machines (VMs) are the scheduling and computing units built on top of the physical machines. Given a fixed number of VMs, our scheduling method organizes them into C groups, each group serves for one task class. Each group is called a Virtual Cluster (VC). All of the tasks in the c-th task class are assigned to the c-th VC. In this way, we must partition all of the VMs into C VCs to serve the C task classes.

TABLE I. Notations of Workload Scheduling

Notations	Definition and Description
T_i	Throughput at the i-th scheduling period
S_i	The i-th scheduling period
$\Delta\delta_c(t)$	New generated workload for VC_c at time t
$\underline{\delta}_c(t)$	Remaining workload for VC_c at time t
$\delta_c(t)$	Total workload for VC_c at time t
$CET_c(t_{i,ni}, t_{i,ni+1})$	Makespan in VC_c from time $t_{i,ni}$ to $t_{i,ni+1}$
$EET(t_{i,ni}, t_{i,ni+1})$	Makespan from time $t_{i,ni}$ to $t_{i,ni+1}$
c or C	Index of a VC or the number of VCs
i or I	Index of time or the last scheduling point
$\theta_c(t_i)$	Number of VMs in VC_c that are allocated at time t_i for the S_i stage
θ	Number of VMs available for scheduling
$p_c(t_i)$	Expected execution time in VC_c at time t_i
$\beta_c(t_i)$	Job processing rate in VC_c at time t_i
$r_c(t_i)$	Remaining execution time in VC_c at time t_i

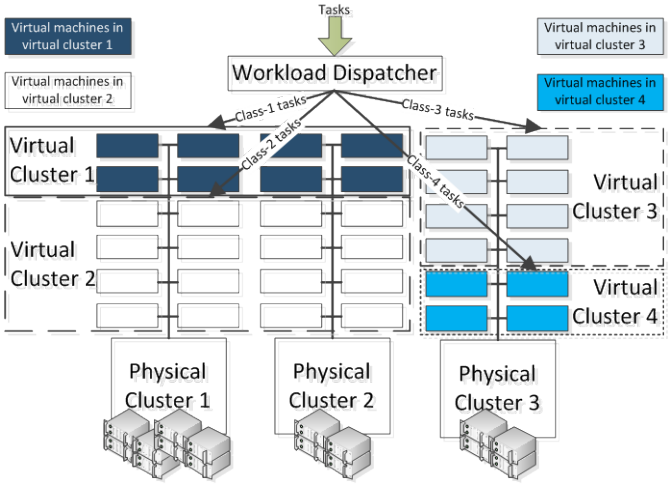


Figure 1. Virtual cluster resource allocation model for workload execution in a virtualized cloud platform. Three physical clusters with each cluster having twelve VMs are shown. The 36 VMs are topologically partitioned into four VCs. The workload dispatcher assigns the class-c task to to the c-th VC to execute.

Consider a virtualized cloud platform with C = 4 VCs as illustrated in Fig. 1. The workload dispatcher directs the tasks to the related VC for execution. A resource-reservation *schedule* specifies the sets of VMs to be provisioned at continuous time periods. For example, the i-th schedule $\theta(t_i)$ is represented by a set of VMs allocated in C clusters in a *schedule space* U. A C-dimensional vector represents this schedule:

$$\theta(t_i)=[\theta_1(t_i), \theta_2(t_i),\ldots, \theta_c(t_i),\ldots \theta_C(t_i)], \quad (1)$$

where $\theta_c(t_i)$ is the number of VMs assigned in VC c, and θ is the total number of VMs that can be allocated. Therefore, we have $\Sigma^{C}_{c=1} \theta_c(t_i) = \theta$.

In Fig. 2, we show stacked workloads that are dispatched to the C VCs in a timeline. From t_{i-1} (or also named $t_{i-1,0}$), schedule $\theta(t_{i-1})$ is applied until t_i (or $t_{i,0}$), where a new schedule $\theta(t_i)$ is used. The new schedule $\theta(t_i)$ is generated during the previous simulation stage from t_{i-1} to t_i. This stage is named S_{i-1} as shown in the figure. Between t_i and t_{i+1}, new workloads (at time points $t_{i,1}$, $t_{i,2}$,…) arrive that are also shown in the figure. Therefore, the

dynamic workload scheduling model is built on such a sequentially overlapped simulation-execution phases. In each phase, one schedule is applied and in the meanwhile, the workloads of the subsequent stage are analyzed to simulate and generate the schedule of the following stages.

We use $\Delta\delta_c(t)$ to denote the newly produced workload of the i-th task class at any time t. The $\Delta\delta_c(t)$ is an incremental value, which represents the accumulated unfinished workloads from the previous stages. As shown in Fig. 2, $\Delta\delta_c(t_{i+1})$ denotes the workload generated at t_{i+1} for the i-th VC.

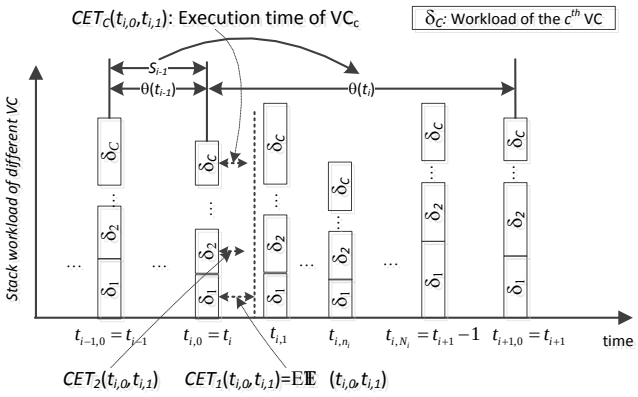


Figure 2. Demonstration of the stacked workload of different VC, given in a timeline. Each rectangular box represents the workload generated for the corresponding virtual cluster. Between schedule periods $[t_i, t_{i+1}]$, there are time points at which new workloads are generated (or arrive). The time points are represented as $t_{i,1}, t_{i,2},\ldots t_{i,Ni}$.

The real workload at t is the sum of the remaining workload of the previous stages and the newly generated workload at time t. We use $\delta_c(t)$ and $\underline{\delta}_c(t)$ to denote the real and remaining workload at time t for the c-th VC, which gives us:

$$\delta_c(t) = \underline{\delta}_c(t) + \Delta\delta_c(t)\ c\in[1,C].$$

The aforementioned results are $\delta(t) = [\delta_1(t), \delta_2(t), \ldots, \delta_c(t), \ldots, \delta_C(t)]^T$.

The tasks arriving at different VCs are time-variant. Our target is to look for a time series of schedules:

$$\theta(t) = [\theta(t_1), \theta(t_2),\ldots,\theta(t_i),\ldots,\theta(t_I)], \quad (2)$$

to maximize the throughput of the whole workload. The throughput is defined as the ratio of the total number of tasks finished and the time spent by the VCs to process the tasks. To calculate throughput, we need to find out the schedule time points, as well as the corresponding schedules for each time point.

From time point t_i to t_{i+1}, the phrasal optimal throughput given the workload and phase can be represented by the following:

$$T_i^* = \max T(\theta(t_i)|\ \delta(t), S_{i-1}) = \max T(\theta(t_i)|\ \delta(t), t_i - t_{i-1})$$
$$t\in[t_i,t_{i-1}),\ \theta(t_i)\in U$$

We first introduce the concepts of *Class Execution Time* (*CET*) and *Effective Execution Time* (*EET*). Suppose that $t_{i,0}$ and $t_{i,1}$ are the time points of two consecutive workload stages, as shown in Fig. 2. $CET_c(t_{i,0}, t_{i,1})$ is the runtime from the starting point $t_{i,0}$ to the finishing time when all the tasks in task class c are finished. We show $CET_c(t_{i,0}, t_{i,1})$, $c = \{1, 2, C\}$ in Fig. 2 by the three double end-arrow dashed lines. If $CET_c(t_{i,0}, t_{i,1})$ is larger than the scheduling period length $t_{i,1} - t_{i,0}$, which means all the tasks in this task class cannot be finished within $[t_{i,0}, t_{i,1}]$, then $t_{i,1} - t_{i,0}$ is used as *CET* and the rest tasks are rolled over to $\Delta\delta_c(t_{i,1})$. $EET(t_{i,0}, t_{i,1})$, on the other hand, means the longest execution time across all of the task classes within $[t_{i,0}, t_{i,1}]$. It can be formally defined as follows.

$$EET\left(t_{i,n_i}, t_{i,n_i+1}\right) = \begin{cases} \max\limits_{c\in[1,C]} CET_c\left(t_{i,n_i}, t_{i,n_i+1}\right) & \forall CET_c\left(t_{i,n_i}, t_{i,n_i+1}\right) < t_{i,n_i+1} - t_{i,n_i} \\ t_{i,n_i+1} - t_{i,n_i} & \exists CET_c\left(t_{i,n_i}, t_{i,n_i+1}\right) \geq t_{i,n_i+1} - t_{i,n_i} \end{cases}$$

As shown in Fig. 2, $EET\ (t_{i,1}, t_{i,0})$ equals to $CET_1\ (t_{i,1}, t_{i,0})$. Therefore, the period throughput from t_i to t_{i+1} is calculated by:

$$T_i^* = \frac{\sum_{j=1}^{N_i-1}\sum_{c=1}^{C}\left(\delta_c\left(t_{j-1}\right) - \delta_c\left(t_j\right)\right)}{\sum_{j=0}^{N_i-1} EET\left(t_{i,j}, t_{i,j+1}\right)}$$

Let $\delta_c(t_i)$ be the number of tasks in the c-thVC at time t_i, $p_c(t_i)$ be the expected execution time for each VM in the c-th VC at time t_i. Consequently, $\beta_c(t_i) = \theta_c(t_i)/p_c(t_i)$ represent the corresponding job processing rate, where $\theta_c(t_i)$ is the number of VMs in VC_c that are allocated at time t_i and $r_c(t_i) = \delta_c(t_i)/\beta_c(t_i)$ is the remaining execution time in the c-thVC at time t_i.

The task throughput is used as the performance metric for the schedules. Given a schedule $\theta(t) = [\theta(t_1), \theta(t_2),\ldots,\theta(t_i),\ldots,\theta(t_I)]$, the task throughput is calculated as the total number of finished tasks divided by the total task execution time (or the *makespan*). At different time periods, different schedules may be applied. All of the candidate schedules at each successive time period form a schedule space U. The *cardinality* of U is calculated by the following expression:

$$u = (\theta - 1)!/[(\theta - C)!(C - 1)!], \quad (3)$$

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

For example, if we use $\theta = 20$ VMs in $C = 7$ VCs for seven task classes, then we need to assess $u = 27{,}132$ possible schedules to search for the best schedule at each time t_i. Each schedule $\theta(t_i)$ takes the form of seven dimensional tuple, having a sum equal to 20, such as [2,3,3,5,2,3,2] or [6,2,1,3,2,4,2]. The search space U is very large given the large number of VMs θ, which leads to the ineffectiveness of the scheduling method. Therefore, the schedule search space must be reduced significantly, especially in a dynamic environment.

B. Simulation-based Optimization Methods

In this section, we introduce a series of existing simulation-based optimization solutions for this problem, namely Monte Carlo Simulation, Blind Pick, and iOO.

Simulations are needed due to the fact that the expected execution time $p_c(t_i)$ is unknown in real-time. The value is stochastic and may be subjected to unknown runtime conditions, such as the contention of other virtual resources. Moreover, the random runtime workload distribution in the c-thVC also has a huge impact on its value.

For a given number of cloud VM instances, configuration (2ECU in Amazon EC2, 1.7GB Memory), VM utilization value, and failure rate, we evaluate the execution time distribution of a

particular task class with a number of tasks, and correspondingly create a table.

For example, the table tells that the execution time is a normal distribution ($X \sim N(20, 5^2)$) when running [100,110] tasks of task class two with 10 VMs, each being small Amazon EC2 instance, failure rate being 0 and utilization being in a range of [70%, 80%]. Then, before each experiment runs, if we have profiled the average VM utilization is 75%, and there are 105 tasks and 10 VMs, then we must sample the normal distribution above to estimate $p_c(t_i)$, and apply this sample value to estimate the throughput of each schedule, and finally choose the best schedule for use. Algorithm 1 below introduces the process.

To yield such an estimation, however, multiple simulation runs must be conducted and an observed average is to be used for $p_c(t_i)$. This is a typical Monte Carlo simulation method that is known to be time consuming.

Monte Carlo Method: Suppose that the overhead of using the Monte Carlo method is denoted by O_M, which is the length of thet scheduling period. Thereafter, we search through all of the time points between t_i and t_{i+1}, when new workloads are generated, to calculate the *CET* for each VC as shown in Line 14 and 18. This step is followed by calculating the *EET* during the scheduling period, as shown in Line 20. Following this, the Makespan and throughput are calculated.

Blind Pick Method: Instead of searching through the whole schedule space U as done in the Monte Carlo method, the Blind Pick (BP) method, randomly selects a portion of the schedules only within U for evaluation. The ratio to be selected is defined by a value α ($0 < \alpha \leq 1$). The algorithm of applying BP is exactly the same as Monte Carlo, except for the scheduling sample space U and scheduling overhead. The length of the scheduling period is $\alpha \times O_M$.

Iterative Ordinal Optimization Method: The OO, a sub-optimal low overhead scheduling method is thoroughly introduced in [15]. Different from the aforementioned methods, the OO uses a rough model (n repeated runs) to generate a rough order of all of the schedules in U, and uses the accurate model (N repeated runs, $n << N$) to evaluate the top-s schedules in the rough order list. For the values of s, the readers are referred to Section III.B.

The OO method contains two steps. The rough evaluation step evaluates u schedules, each one being repeated n times, and the overhead becomes $u \times O \times n$. Suppose O denotes the time needed to evaluate one schedule, then the second stage will take $s \times O \times N$ amounts of time.

We have proven in our previous paper [40] that the OO simulation time is shorter than other two methods that enable us to develop an iterative OO (iOO) method. In other words, the iOO applies the OO method iteratively in a multi-stage scheduling manner, with each stage being a new OO routine.

The iOO approach exhibited superior performance in the research paper, especially in the highly fluctuated workload cases due to the reduced overhead, resulting in shorter scheduling periods and fine-grained scheduling. The length of the iOO scheduling period can be determined as $u \times O \times n + s \times O \times N$.

However, the iOO does not consider the characteristics of workloads, namely the similarity of the workloads in consequent stages of a multi-stage scheduling problem. In the next section, we introduce the evolutionary Ordinal Optimization (eOO) that extends the iOO method along those lines.

Algorithm 1. Monte Carlo Simulation Method

Input:

$U = \{\theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_c, \ldots \theta_C]\}$ //All of the possible schedules

$\delta = \{[\delta(t_{0,0}), \delta(t_{0,1}), \ldots, \delta(t_{i,0}), \ldots, \delta(t_{I,0}), \ldots, \delta(t_{I,NI})]\}$ //Workloads

α //The ratio of the search space U should be sampled

Output:

The best $\theta^*(t)$ for use at each schedule time t

Procedure:

1. Random sample U as per the ratio α; //Generate search space
2. Estimate the overhead time of using Monte Carlo method: O_M;
3. Initialize schedule period valuable $i = 1$;
4. **while** ($i \times O_M < t_I$) **do** //Not the last schedule period
5. $t_i = (i\text{-}1) \times O_M$, $t_{i+1} = i \times O_M$;
6. Load workload $\boldsymbol{\delta_c(t_i)}$ between $[t_i, t_{i+1}]$; //$\delta_c(t_{i,0})$, $\delta_c(t_{i,1})$,… $\delta_c(t_{i,Ni})$
7. **for** each θ in U **do** // Search Each Schedule
8. **for** each j in $[0, N_i]$ **do** //Each time point workload generated
9. **for** each VC c in $[1,C]$ **do** //Each virtual cluster
10. Simulate $p_c(t_i)$ $\boldsymbol{N}$ times to get an average $P_c(t_i)$;
11. $\delta_c(t_{i,j}) = \underline{\delta}_c(t_{i,j}) + \Delta\delta_c(t_{i,j})$; //Current tasks for VC_c
12. **if** $(\delta_c(t_{i,j}) - (t_{i,j+1} - t_{i,j}) \times P_c(t_i) > 0)$ //Not all the tasks finished
13. $\underline{\delta}_c(t_{i,j+1}) = \delta_c(t_{i,j}) - (t_{i,j+1} - t_{i,j}) \times P_c(t_i)$; //Remaining tasks
14. $CET_c(t_{i,j}, t_{i,j+1}) = t_{i,j+1} - t_{i,j}$;
15. **else** //All the tasks finished
16. $\underline{\delta}_c(t_{i,j+1}) = 0$;
17. **end if**;
18. $CET_c(t_{i,j}, t_{i,j+1}) = (\delta_c(t_{i,j}) - (t_{i,j+1} - t_{i,j}) \times P_c(t_i))/P_c(t_i)$;
19. **end for**;
20. $EET(t_{i,j}, t_{i,j+1}) = \max(CET_c(t_{i,j}, t_{i,j+1}))$;
21. Makespan$(\theta(t_i)) = EET(t_{i,0}, t_{i,1}) + \ldots + EET(t_{i,Ni-1}, t_{i,Ni})$;
22. **end for**;
23. **if** (Makespan$(\theta(t_i))$ < Makespan$(\theta^*(t_i))$) //A better schedule
24. $\theta^*(t_i) = \theta(t_i)$;
25. $T^*(t_i) = (\delta_c(t_{i,0}) + \delta_c(t_{i,1}) + \ldots + \delta_c(t_{i,Ni}) - \underline{\delta}_c(t_{i,1}) + \ldots + \underline{\delta}_c(t_{i,Ni}))/\theta^*(t_i)$;
26. **end if**;
27. **end for**;
28. $i = i+1$; //Next schedule period
29. **end while**

III. Evolutionary Ordinal Optimization

An intuitive interpretation of the eOO is the assumption that the simulation runs must be allocated based on the characteristics of the workloads. For example, fluctuating workload requires more intermediate scheduling time points. Because each of the short scheduling phases satisfies small periods of the workload, the eOO delivers better overall performance.

However, one disadvantage of the eOO method is that the short simulation phase leads to less simulation runs of the $p_c(t_i)$, which on the contrary, might decrease the performance of each of the single schedule period. In this section, we demonstrate the methodologies and effectiveness of the eOO method, and propose a solution to partition and merge the scheduling periods based on the fluctuations within the workload.

A. Evolutionary Ordinal Optimization

Before we step into any further detail, we define the concept of *workload pattern*. At time t_0 in Fig. 3, one batch of workload for the seven VCs arrives. This workload pattern is very similar

to the workload patterns at t_1 and t_2. To give a formal definition, suppose that there are two batches of workloads at time t', $t''\in[t_i, t_{i+1}]$, i.e., $\delta(t') = [\delta_1(t'),\ldots,\delta_c(t'),\ldots,\delta_C(t')]$ and $\delta(t'') = [\delta_1(t''),\ldots,\delta_c(t''),\ldots,\delta_C(t'')]$, respectively, where $\delta_c(t')$ denotes the tasks for VC c at time t'. The similarity pattern between $\delta(t')$ and $\delta(t'')$ can be defined as:

$$\mathrm{Sim}(\delta(t'),\delta(t'')) = \frac{\sum_{c=1}^{C}\left(\delta_c(t')\times\delta_c(t'')\right)}{\sqrt{\sum_{c=1}^{C}\delta_c(t')^2\times\sum_{c=1}^{C}\delta_c(t'')^2}}. \quad (4)$$

Intuitively, a high similarity pattern leads to the merger of two scheduling periods, while a low similarity pattern leads to a scheduling period partition. In Fig. 3, we would merge $[t_0, t_1]$, $[t_1, t_2]$, and $[t_2, t_3]$ as a single scheduling period, when supposing that t is the scheduling period of any method proposed in the previous section. However, such a merge does not apply to the period $[t_3, t_3+t]$, where the two batches of workloads are dramatically different from each other. Therefore, we must divide the scheduling period adaptively into smaller periods for separate scheduling.

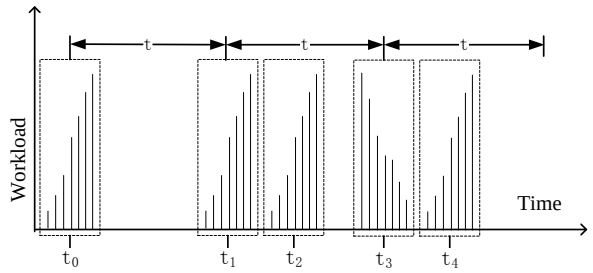


Figure 3. Demonstration of various workload characteristics. The dashed-line rectangular box with seven solid lines are the seven workloads for the $C = 7$ VCs. There are five batches of workloads arrive at time t_0, t_1, t_2, t_3 and t_4 for each of the virtual clusters.

The major tenets of the eOO lies in the automatic scheduling period partitioning and merging as introduced below.

(1) A pairwise comparison of the workload pattern must be made between each of the pairs of the workload in each scheduling period, such as $[t_i, t_{i+1}]$. If the maximum similarity across all the workload patterns is sufficiently small, then the interval must be sliced into two, or more, consecutive scheduling periods. Otherwise the period is just kept as it is.

(2) Consider two workload patterns, $\delta(t')$ and $\delta(t'')$ that are extracted from two consecutive scheduling periods $[t_i, t_{i+1}]$ and $[t_{i+1}, t_{i+2}]$, respectively, where $t'\in[t_i, t_{i+1}]$ and $t''\in[t_{i+1}, t_{i+2}]$. We make similarity analysis among all of the possible pairs as mentioned above. If the least similarity value is sufficiently large, then the two scheduling periods must be merged. Such a merge operation leads to a sufficiently large simulation period of $[t_i, t_{i+2}]$ that promises a longer simulation period for the subsequent period.

In Algorithm 2 and Algorithm 3, we show the details of the scheduling period merge and partition based on the workload similarities. In general, for Algorithm 2, if the maximum similarity value between any two pairs of workloads in $[t_i, t_{i+1}]$ is less than α, then the interval will be partitioned into two parts. For the aforementioned case, we use a recursive algorithm, as shown in Line 9 and Line 10, to further analyze the two sub-intervals. The analysis stops when either the sub-interval is smaller than a threshold value w, or all of the workload patterns are similar in the interval that is being checked. The algorithm outputs all of the decision time points *DP*.

Algorithm 2. Scheduling Period Partition

Input: All workloads in $[t_i,t_{i+1}]$ denoted by $\delta(t_{i,}t_{i+1})=\{\delta(t_{i,0}), \delta(t_{i,1}),\ldots \delta(t_{i,Ni})\}$.

Output: Decision point (*DP*) set in $[t_i, t_{i+1}]$.

Procedure:

```
1.  DP ← null;
2.  if ((t_{i+1} - t_i) < w)   //Interval is too small, canot be partitioned
    anymore
3.     Add t_i into DP;
4.     return;
5.  end if;
6.  for each workload δ(t_l) in δ(t_i,t_{i+1}) do
7.     for each workload δ(t_k) != δ(t_l) in δ(t_i,t_{i+1}) && k > l do
8.        if (Max(Sim(δ(t_l), δ(t_k)) < α)          //small similarity
9.           Partition(t_i, t_i + (t_{i+1}- t_i)/2);      //partition left half
10.       Partition(t_i + (t_{i+1}- t_i)/2, t_{i+1});     //partition right half
11.    end if;
12.  end for;
13. end for;
14. Sort and return DP.
```

Algorithm 3: **Scheduling Period Merge**
Input: Output of Algorithm 2: *DP*.
Output: New decision point set (*NDP*).
Procedure:

```
1.  for each dp in DP do          // Each existing decision point dp
2.     Find dps associated with dp;          //decision point stage
3.     if (Width(dps) > W)
4.        Goto Step 1;          //large decision stage, no more
       merge
5.     end if;
6.     if (dp is not the last stage)
7.        Find dpn after dp;          //next decision point
8.        Find dpns associated with dpn;
9.        for each workload δ(t_l) in ds do
10.          for each workload δ(t_k) in dpns do
11.             if (Min(Sim(δ(t_i), δ(t_j))) > β)          //large similarity
12.                Remove dpn from DP;
13.                Associate dp with (dps+dpns);          //merge stages
14.                Goto Step 4;
15.             else
16.                Goto Step 1;
17.             end if;
18.          end for;
19.       end for;
20.    end if;
21. end for;
22. Return (NDP←DP).
```

In Algorithm 3, the *DP* is used as input to find out the periods that can be merged. Initially (Line 3 and Line 4), one *dps* is selected from the *DP*. If the associated period *dps* is long enough, then there is no need for a merge with the subsequent period. Otherwise, a comparison is made between the *dps* and the following *dpns*. If the minimum similarity value between all

of the pairs of the two workloads (one from the *dps* and the other one from the *dpns* is larger than β), then the two intervals will be merged. For the consecutive periods of highly similar workloads, simulation is only needed the first period and all the rest periods apply the same schedule.

After the execution of Algorithm 3, we find all of the decision points during the whole application life cycle. Now, we allocate computations by judicially selecting the size of N in the Monte Carlo based on the time allowed during the current scheduling period. We can typically see that the length of the eOO method is not fixed, which is totally different from other three approaches. For a merged period, longer simulation time is possible for its following period. For a period that is partitioned into a few sub-periods, each period

Essentially, Algorithm 2 and Algorithm 3 are adaptive methods that allocate simulation time based on the fluctuation of workloads of multiple stages. In Section III.D, we will introduce a method that allocates simulation time within a stage.

B. Parameter Configuration

Unknown parameters must be finalized before the experimental runs. These are the simulation runs N and n, size of selected set s_i in iOO, size of the selected set s_e using eOO, size of the selected set s_{BP} using BP, α and β in Algorithm 3 and Algorithm 4.

The BF Monte Carlo repeats itself N times to estimate the average value of each $p_c(t_i)$. As a common practice to achieve reliable simulation accuracy, Monte Carlo usually applies a large simulation runs, say N = 1000, for each single individual schedule for large-scale scientific workflow scheduling problems [42].

The rough model n of the iOO is also determined by the statistic theory. Based on the central tendency theory, it takes two orders of magnitudes more runs to improve one order of magnitude of the estimation accuracy. When applying n = 10 runs in the rough model, which is two orders of magnitude less simulation runs than N, would lead to one order of higher estimation error. For example, suppose the estimation error of BF Monte Carlo stays in a range from 0.1 to 0.9, the estimation error of iOO would be in a range from 1 to 9. Details analysis of and a formal proof of the relationship between the simulation time and simulation accuracy can also be referred to the work [42].

Selection set size of each method, BP, iOO and eOO is different. The BP method picks a fixed ratio α that is normally larger than 80%. As we mentioned earlier, the iOO and eOO are two-stage scheduling algorithms. Their selection sets are determined by the time left for the second scheduling period. For example, if each scheduling period of iOO equals to t = 100 seconds and the rough evaluation of all schedules takes 10 seconds. Suppose a precise evaluation of one schedule takes 3 seconds, the maximum selection set size for iOO would be 30.

Around ten thousand groups of simulated data were generated randomly based on the distribution of the workloads. We plot the histogram of the statistic analysis of these workloads as shown in Fig. 4. Six consecutive ranges of pair-wise similarity degree values are used as the x-axis and the total number of workload pairs that shows the similarity degree is plotted as the y-axis. This method classifies the similarities among all the workloads in a few categories, and identifies quantitatively the similar value threshed we use to determine whether one pair of workload being similar or not. We set the upper third quantile of the distribution as α and the lower third quantile as β. The results are reported in Fig. 4.

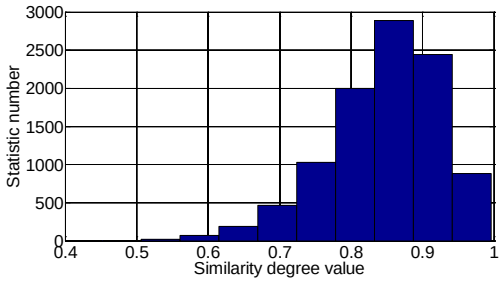


Figure 4. The cumulative number of workloads and the similarities between all of the workload pairs over the similarity degree values.

C. The eOO for Dynamic Workload Scheduling

If the scheduling overhead is too high and the scheduling itself takes longer than the period of workload and resource changing, then the dynamic workload in the multitasking scheduling problem is difficult to capture. However, applying the OO in an evolutionary manner, leads to the full investigation of the workload patterns. For the highly dynamic and fluctuating workloads, the eOO adaptively partitions into small pieces, to generate the fine-grained schedules. Although the short-period scheduling means less accuracy, the gain in the adaptivity to such kinds of workloads, outweighs the minor inaccuracies in the solutions.

On the contrary, the eOO produces course-grained schedules for stable workloads. In this way, the merged stages lead to more simulation time for a subsequent stage that improves the accuracy. We illustrate the method graphically in Fig. 5.

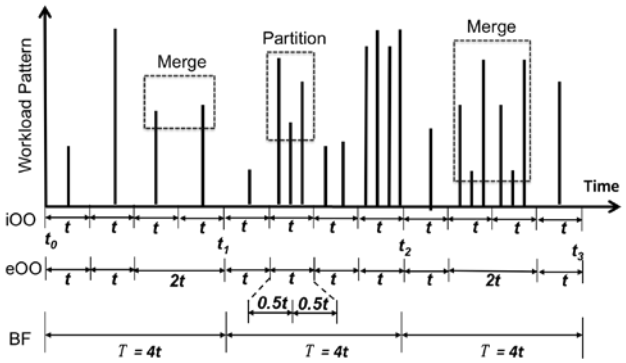


Figure 5. Illustration of OO adaption to dynamic workloads.

Let T be the time overhead of scheduling, using the BF or Monte Carlo, and let t be the time overhead of that of the iOO. For example, if at time t_0, the BF is used for simulations, then it is not until t_1 that the BF can generate the optimal schedule for $[t_1, t_2]$. While the solution is optimal at time t_1, no further and consecutive solutions can be generated again during t_1 and t_2. As for the iOO at time t_1, workload is used to generate an acceptable schedule at time t_1+t, and then t_1+2t,…. The aforementioned process is conducted repeatedly to capture a

finer granularity variation of the workload to improve the overall performance.

As an improvement of iOO, the eOO takes the workload pattern into account. In $[t_1+t, t_1+2t]$, workload patterns evolve significantly. In eOO, this interval is partitioned into two parts shown in the solid-rectangular box of Fig. 5. In $[t_0+2t, t_0+4t]$ or $[t_2, t_2+2t]$, the inter-workload patterns are similar; and as a result two intervals are merged into one.

The eOO scheduling is performed with a finer granularity, namely with $0.5t$ per scheduling period when partitioned. In case of a merge, the eOO scheduling is performed with a coarse granularity period, namely with $2t$ per scheduling period. This adaptability makes the scheduling method more accuracy with better performance.

D. Optimal Computing Budget Allocation based Scheduling

All of the scheduling methods, namely Monte Carlo, Blind Pick, iOO, eOO can be further improved by combining them with the Optimal Computing Budget Allocation (OCBA) [7].

The OCBA is a simulation-based optimization approach that allocates computing resources effectively among all of the candidate schedules. Simply speaking, rather than equally allocate simulation time among all of the schedules, the OCBA allocates more simulation time to the schedules that show better overall performances in their early simulation stages. Such an exercise removes the potentially poor schedules at an early stage, and gives more observations to the promising schedules. In our experimental evaluations, we apply the OCBA method on each of the schedule stage derived by the eOO, and compare the performances of all of the methods.

IV. Evaluations, Results, and Discussions

In this section, detailed experimental design and results are presented for the virtual cluster allocation in multitasking workload scheduling applications.

A. Experimental Settings

The cloud experiments are carried out using 16, 32, 64, and up to 128 standard VM instances on the Amazon EC2 platform. Each instance is by default, a small one with 1.7 GB of memory, one EC2 Compute Unit (one virtual core with one EC2 Compute Unit), 160 GB of local instance storage, a 32-bit platform on deployed Linux Base with EBS boot, and a 32-bit architecture with Amazon EC2 Tools.

The benchmarking applications used are similar to [8]. The authors report a benchmark application suite includes bit-reversals, merge-sort, matrix multiplication, a trace-driven simulation, partitioning meshes, k-nearest neighbor classification, and AdaBoost algorithm. These seven applications correspond to our seven tasks classes, which need seven virtual clusters to host and execute.

Up to 10,000 tasks were concurrently executed. Each task corresponding to one benchmark application has a variable amount of input dataset size. The total amount of data used in the 128-VM scenarios approximately exceeds 1.6 TB.

We define four typical workload types by using, say, LFLD to represent Low Frequency Low Difference (LFLD) workload. This means the workload comes slowly while most of the workload patterns are alike. Similarly, the three other workload types are LFHD, HFLD and HFHD. In the HFHD case, the average arrival rate of the workload comes up to 500 tasks/second. The high velocity and volume of data used justifies a testing-scale big data analytic in these experiments.

Even though the data amount tested here is not extremely large, we argue that the method can still be extrapolated to a larger scale of hundred TB or even PB of data due to the loosely-coupled multitasking property.

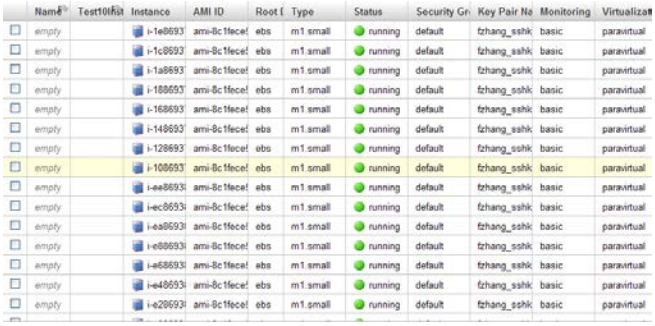


Figure 6. Snapshot of multitasking applications on Amazon EC2.

The hierarchical structure of our experiments is shown in Fig. 7. Each server notation is an Amazon EC2 instance. There are 7 VCs denoted by the dashed-rectangular boxes. There is one level-2 seed sever in each of the VC to ensure at least one VM is running to process the requests from level-1 seed server. It is also used to balance the workloads into the internal VMs. This is a dedicated VM that resides in each of the VC.

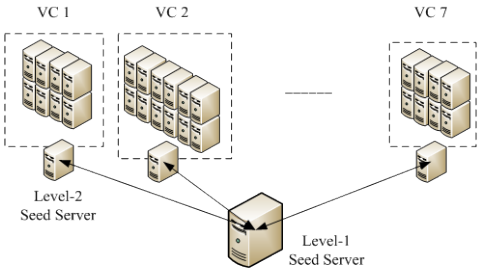


Figure 7. Architecture of the organized instances on Amazon EC2.

B. Application Scenarios

Two major application scenarios are depicted in Fig. 8 below. We describe both of them in detail.

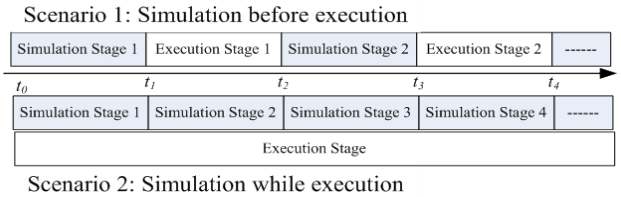


Figure 8. Two application scenarios for the benchmark.

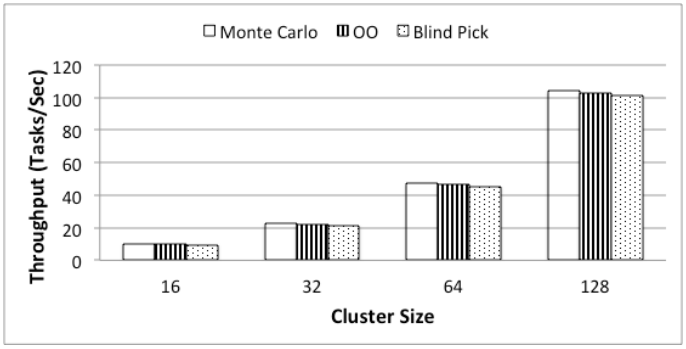
Scenario 1: Simulation before execution. In the upper part of Fig. 8, one simulation stage is conducted before each experimental run. Before execution stage 1, there is sufficient time for a simulation to find a good allocation schedule from t_0

to t_1, and the schedule is applied at t_1. No simulation is carried out during execution stage 1 until t_2 . This scenario is commonly seen in the job scheduling of traditional parallel and distributed computing systems and spot instances in Amazon EC2. For example, in using spot instances, there is sufficient time for simulation before the price reaches an acceptable level, and execution stage can follow the simulation.

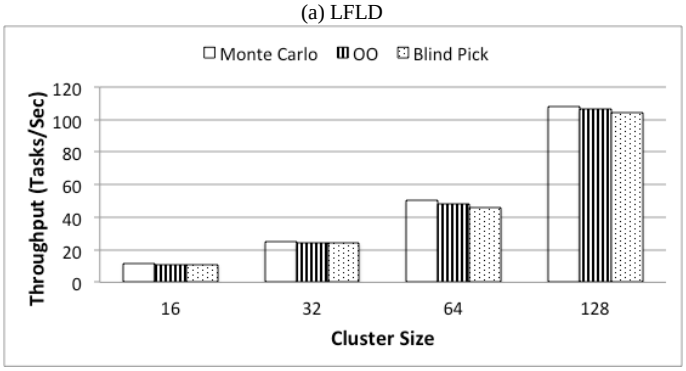
Scenario 2: Simulation while execution. In the lower part of Fig. 8, simulation and execution stages overlap. At time t_0, a random schedule is used for the time being, while the workload information between $[t_1, t_2]$ is used to generate a better quality schedule to be used at time t_1. Similar schedulings are conducted repeatedly until the end of experiment. Most of the real-time multitasking scheduling applications are carried out based on Scenario 2.

C. Comparative Studies on Scenario 1

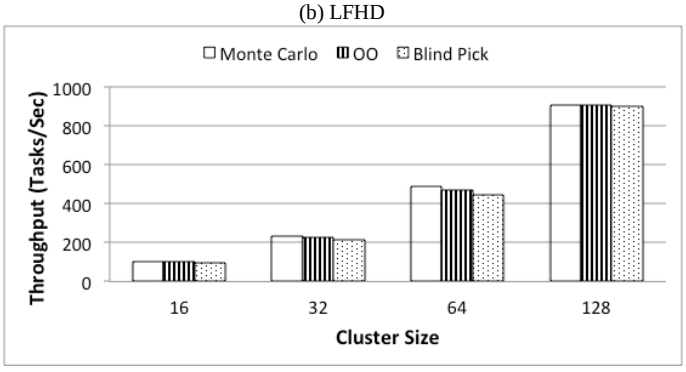
We compare the case studies of Scenario 1, on the abovementioned four types of workloads. The results are reported in Fig. 9 (a)-(d).



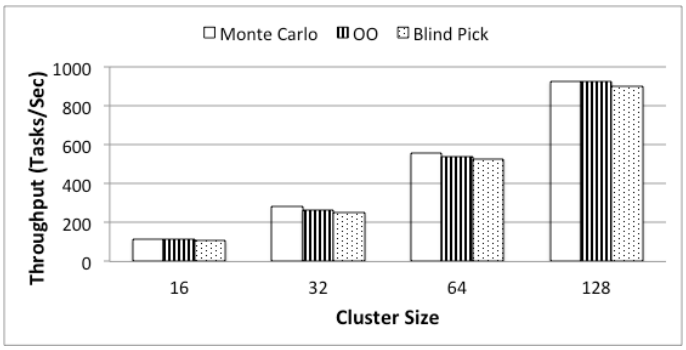
(a) LFLD



(b) LFHD



(c) HFLD



(d) HFHD

Figure 9. Comparative studies of application scenario 1.

It can be observed that the workload variation has a very limited impact on the three methods. The BF Monte Carlo is always the best while the Blind Pick is always the worst among all. The OO method lies in between. In Table 2, we demonstrate the performance difference of using OO compared with the other two methods. Positive values mean the method outperforms OO.

TABLE II. PERFORMANCE COMPARISON OF SCENARIO 1

Cluster Size	Methods	LFLD	LFHD	HFLD	HFHD
16	Monte Carlo	6.19%	5.50%	1.19%	0.80%
32		3.18%	3.32%	2.53%	6.48%
64		1.29%	3.51%	4.01%	2.96%
128		1.07%	1.60%	0.04%	0.13%
16	Blind Pick	-1.03%	-3.67%	-2.97%	-4.73%
32		-2.27%	-0.83%	-3.78%	-5.04%
64		-2.58%	-4.96%	-4.78%	-2.31%
128		-1.46%	-2.07%	-0.68%	-2.27%

The aforementioned behavior is straightforward. Given sufficient simulation time, the BF Monte Carlo simulates longer than any other methods to evaluate $p_c(t_i)$, which undoubtedly achieves the best performance because of its precise evaluation. Random selection of the Blind Pick method leads to the worst performance. This Method takes the least of the simulation time; however, the performance is mediocre at best. The OO method cannot take advantage of the long simulation time allowed and has improvement over Blind Pick, but not as good as BF Monte Carlo.

D. Comparative Studies on Scenario 2

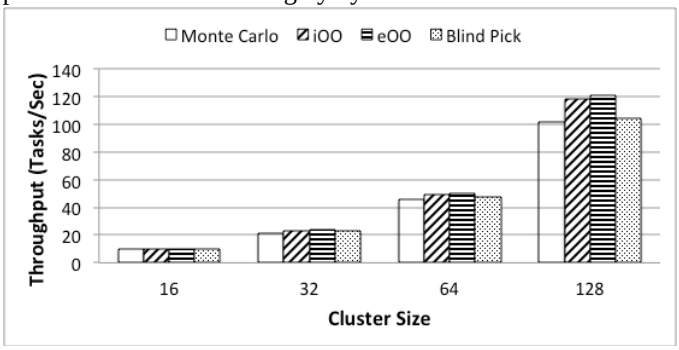
The simulation time budget in Scenario 2 is limited. Consequently, continuous simulation and scheduling are performed simultaneously.

In this scenario, the BF Monte Carlo method with extended simulation is no longer the best, with lower throughput compared with the iOO and eOO methods. From Fig. 10 (a)-(d), we compare the throughput of the methods using a variable size of cluster. The corresponding performance comparison using eOO compared with other methods are summarized in Table 3.

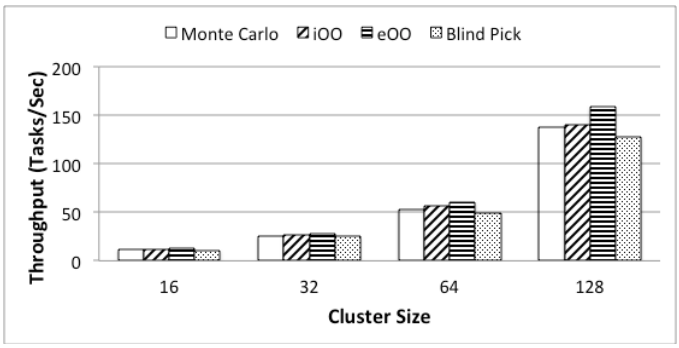
The performance of all the four methods is similar when a small number of VMs is used. For example, the four methods perform similarly in the case of 16 VMs. However, the difference grows substantially as the number of VMs increase. In the case of 128 VMs, the eOO performs 32.62% and 29.06% better than the BP and the BF.

The relative performance of the BF Monte Carlo and BP varies given different types of workloads. In the LFHD and HFHD cases, the BP performs better; while in the LFLD and HFLD ones, the BF Monte Carlo slightly outperforms the BP . This is caused by the low overhead of BP. The BF Monte Carlo with long simulation time and high overhead, is inapplicable in Scenario 2.

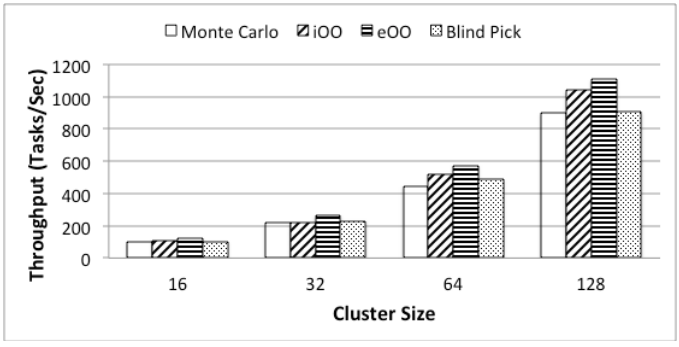
The advantage of the eOO over the iOO resides in its adaptability in mutating the scheduling periods. In Fig. 10 (b) and Fig. 10 (d), where the workloads exhibit high difference over time, the eOO performs much better with its capability to partition one period into two or merge two neighboring periods into one. This is a workload pattern-aware method; therefore, it performs much better in highly dynamic workload cases.



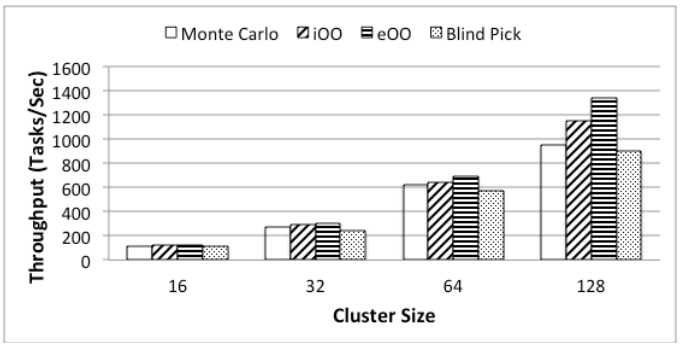
(a) LFLD



(b) LFHD



(c) HFLD



(d) HFHD

Figure 10. Comparative studies of application scenario 2.

TABLE III. PERFORMANCE COMPARISON OF SCENARIO 2

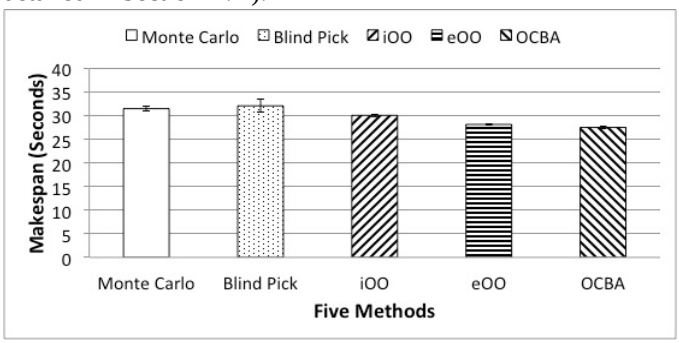
Cluster Size	Methods	LFLD	LFHD	HFLD	HFHD
16	Monte Carlo	-5.88%	-12.12%	-17.92%	-7.12%
32		-8.51%	-10.43%	-17.71%	-10.31%
64		-9.56%	-12.31%	-22.26%	-9.63%
128		-16.00%	-13.05%	-18.68%	-29.06%
16	Blind Pick	0.98%	-18.18%	-14.41%	-7.98%
32		-3.40%	-10.43%	-12.31%	-20.13%
64		-5.98%	-19.13%	-15.09%	-16.50%
128		-13.85%	-19.48%	-18.08%	-32.62%
16	iOO	-2.94%	-8.33%	-7.29%	-5.56%
32		-2.13%	-3.24%	-15.66%	-3.18%
64		-2.79%	-5.82%	-9.48%	-7.06%
128		-2.24%	-11.35%	-6.06%	-14.05%

E. Simulated Comparative Studies with OCBA

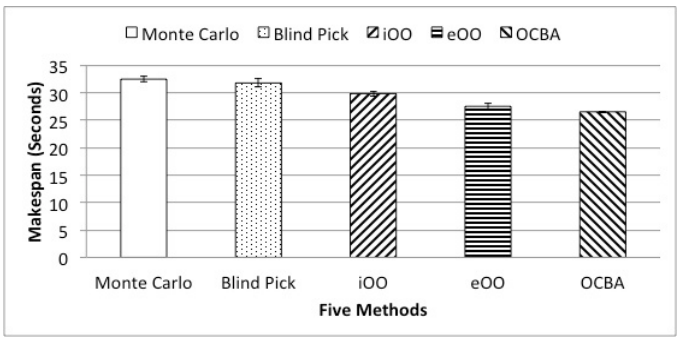
As mentioned previously (Section III.D), the OCBA attempts to judicially allocate the computing budget among all the multiple schedules, which breaks the equal budget allocation assumption implemented in the iOO and eOO methods. In simple words, the more a schedule shows better performance in a set of simulation runs, the more follow-up simulation runs the schedule tends to receive. By employing such a methodology, the potentially better schedules are evaluated more frequently and accurately, which translates into better system performance. Without explicitly mentioned, the OCBA used in this section is referred to using the eOO method by replacing its equal schedule simulation allocation time with the OCBA algorithm.

Unlike other methods, the OCBA calculates the simulation time for each schedule and adjusts it in runtime. In a real cloud-computing environment, this solution faces a difficulty by not being able to effectively predict the simulation cycles. Instead, we use simulations only to prove the effectiveness of this method. In our future works, we will explore by studying more scenarios to justify the applicability of this method in real cloud computing environments. The comparative results are shown in Fig. 11 by simulating a cluster of using 32 VMs. The performance metric is the makespan which aggregates all of the effective compute resource usage time, denoted by the *EET* (as

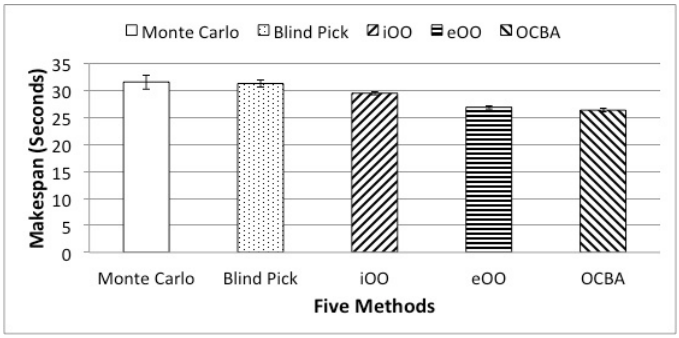
detailed in Section II.A).



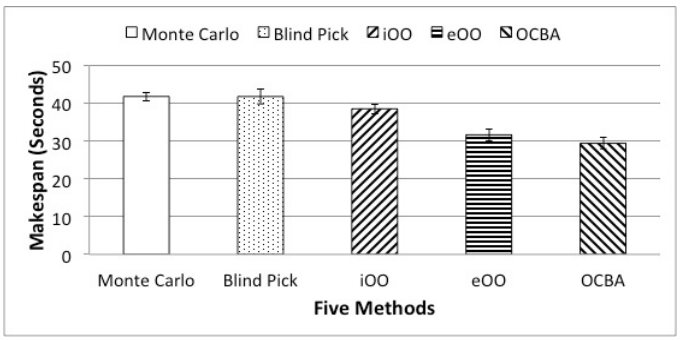
(a) LFLD



(b) LFHD



(c) HFLD



(d) HFHD

Figure 11. Comparative studies of application Scenario 2 with OCBA.

In Table 4, similar performance comparison of using OCBA compared with other methods is given. OCBA improves up to 29.69% performance than other methods. This improvement is observed mainly in HFHD workload. This validates our assumption again that, our methods are appealing to highly fluctuating workloads. We also demonstrate the effectiveness of using the OCBA method over all other methodologies.

Overall, there is a 2% to 7% speedup using the eOO with OCBA compared to the eOO method only. Compared to other methods, such as the Monte Carlo and the Blind Pick in the HFHD workload scenario, the speedup can be as much as 29%. In other less dynamic workload scenarios, the speedup value varied between 12.6% and 18.3%.

TABLE IV. PERFORMANCE COMPARISON OF USING OCBA

	LFLD	LFHD	HFLD	HFHD
Monte Carlo	12.64%	18.33%	16.54%	29.68%
Blind Pick	14.28%	16.53%	15.79%	29.65%
iOO	8.52%	11.11%	10.65%	23.73%
eOO	2.27%	3.63%	2.01%	7.14%

V. RELATED WORK

We have witnessed an escalating interest in resource allocation for multi-task scheduling [21], [32]. Many classical optimization methods, such as opportunistic load balance, minimum execution time, and minimum completion time, are described in [12]. Suffrage, min-min, max-min, and auction based optimization procedures are discussed in [6], [25]. Yu and Buyya [39] proposed economy-based methods to handle large-scale grid workflow scheduling under deadline constraints, budget allocation, and Quality of Service (QoS). Benoit *et al.* [4] designed resource-aware allocation strategies for divisible loads. Li and Buyya [18] proposed a model-driven simulation grid scheduling strategies. Zomaya et al. [20], [33] have proposed a hybrid schedule and a cooperative game framework. There are also studies on making various tradeoffs under various constraints and objectives for the workflow scheduling. Wieczorek [37] analyzed five facets that may have a major impact on the selection of an appropriate scheduling strategy, and proposed taxonomies for the multi-objective workflow scheduling. Prodan *et al.* [29] proposed a novel dynamic constraint algorithm, which outperforms many existing methods, such as SOLOS and BDLS, to optimize bi-criteria problems.

Duan *et al.* [11] suggested a low complexity game-theoretic optimization method. Dogan *et al.* [10] developed a matching and scheduling algorithm for both the execution time and the failure probability. Moretti [24] suggested the All-Pairs to improve usability, performance, and efficiency of a campus grid. Smith *et al.* [30] proposed a robust static resource allocation for distributed computing systems operating under imposed QoS constraints.

Ozisikyilmaz *et al.* [26] suggested an efficient machine learning method for system space exploration. Similar to the work described in [19], our work is also carried out by using the search size based reduction. In [28], [33], [34], petri-net and data-driven based methods are shown to compose services via the mediator, which motivates us to design the architectural cloud-scheduling platform for multitasking workloads.

A few control-based strategies are also proposed [9], [27], [45] to manipulate the VM count for delivering optimal performance. Other related work, such as [22] [44], focus on finding optimal price strategy for renting spot instances. The similarity of these

works lies to relying on their individual statistic models. Our work, however, is a simulation-based strategy, which is applied to multi-stage scheduling problems.

Based on the theory of the OO, we proposed the eOO that has advantage in handling large-scale search space to solve the multitask scheduling problem for dynamic workloads. Ever since the introduction of OO in [14], one can search for a small subset of solutions that are sufficiently good and computationally tractable. Along the OO line, many OO based heuristic methods have been proposed [17], [36]. It quickly narrows down the solution to a subset of "good enough" solutions with manageable overhead. Typical and original applications of the OO include automated manufacturing [31], communications [38], power systems [1], and distributed computing systems [43]. Different selection rules of OO are compared in [16] to discuss the relative performance. Conclusion were made that no selection rule is absolutely better than the others under all circumstances. To the best of our knowledge, this paper reports the first successful attempt to apply the OO in an iterative and evolutionary fashion for the dynamic optimization scenarios to meet special requirements of the cloud workload scheduling.

VI. Conclusions

A low-overhead method for dynamic multitasking workload scheduling in elastic virtual clusters is introduced in this work. The advantage of the proposed low-overhead scheduling scheme is summarized below.

1) Modeling of multitasking workload scheduling in wide-area elastic virtualized clusters. Cloud computing allocates virtual cluster resources on demand. This model serves the cloud computing environments and proposes simulation-based method for optimization.

2) The OO methodology is enhanced to deal with dynamically evolving workloads. iOO, with lower scheduling overhead and eOO, with adaptivity in terms of variable scheduling interval to turbulent workloads, are introduced, respectively. These two enhancements to OO are agile enough to capture system dynamism and conduct real-time scheduling. The two methods improve the overall performance in fluctuating workload scenarios evidenced by the experimental studies.

3) Proposed methods are validated by real benchmark applications with large search space, VM resource constraints, and uncertainties. Very few previous works utilizes simulation-based method for dynamic multitasking workload scheduling. Our approach is simulation based which takes consideration of these factors and is more realistic to be applied in real-world cloud systems.

Ongoing work includes the establishment of new cloud infrastructure for enabling real-time large-scale data analysis using a variety of more benchmark applications with varying workloads. We also intend to release the simulation-based optimization approaches as profiling tools to enhance performance of scheduling multitasking workload on Amazon EC2 platform.

Acknowledgements

This work was supported in part by Ministry of Science and Technology of China under National 973 Basic Research Program (grants No. 2013CB228206 and No. 2011CB302505) and National Natural Science Foundation of China (grant No. 61233016).

References

[1] E. H. Allen, *Stochastic Unit Commitment in a Deregulated Electric Utility Industry*, Ph.D. Thesis, Massachusetts Institute of Technology, 1998.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, et, al, "Above the clouds: A Berkeley View of Cloud Computing", *Technical Report No. UCB/EECS-2009-28, University of California, Berkerley*, CA, 2009.

[3] P. Barham, and B. Dragovic, "Xen and the Art of Virtualization", *Proc. of 19th ACM symposium on Operating Systems Principles*, Bolton Landing, NY, USA, pp. 164-177, 2003.

[4] 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 Symposium (IPDPS'09)*, Rome, Italy. 2009.

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

[6] H. Casanova et al. "Heuristic for Scheduling Parameters Sweep Applications in Grid Environments", *Proc. of the 9th Heterogenous Computing Workshop (HCW)*, pp. 349-363, Cancun, Mexico, May, 2000.

[7] C. H. Chen "An Effective Approach to Smartly Allocate Computing Budget for Discrete Event Simulation," *Proceedings of the 34th IEEE Conference on Decision and Control*, pp. 2598-2605, December 1995.

[8] S. Chen, L. Xiao and X. Zhang, "Adaptive and virtual reconfigurations for effective dynamic job scheduling in cluster systems", *Proceedings of the 22nd International Conference on Distributed Computing Systems* (ICDCS'02), Vienna, Austria, 2002.

[9] A. Demberel, J. Chase, and S. Babu. "Reflective Control for an Elastic Cloud Application an Automated Experiment Workbench", *the 2009 USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'09)*, San Diego, CA.

[10] A. Dogan, and F. Özgüner. "Biobjective Scheduling Algorithms for Execution Time–Reliability Trade-off in Heterogeneous Computing Systems". *The Computer Journal*, Vol. 48, No.3, pp. 300-314, 2005.

[11] 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.

[12] 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.

[13] M. Garey and D. Johnson, *Computers and Intractibility: A guide to the Theory of NP-completeness*, W. H. Freeman, 1979.

[14] 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.

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

[16] Q. S. Jia, Y. C. Ho, and Q. C. Zhao, "Comparison of Selection Rules for Ordinal Optimization", *Mathematical and Computer Modelling*, Vol. 43, No. 9-10, pp. 1150-1171, 2006.

[17] D. Li, L. H. Lee, and Y. C. Ho, "Constraint Ordinal Optimization", *Information Sciences*, Vol. 148, No. 1-4, pp. 201-220, 2002.

[18] 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.

[19] D. Lu, H. Sheng, and P. Dinda, "Size-based Scheduling Policies with Inaccurate Scheduling Information", *Proc. of 12th IEEE Int'l Symp, on Modeling, Analysis, and Simulation of Computer and Telecomm. Systems*, pp. 31-38, 2004.

[20] K. Lu, and A. Y. Zomaya, "A Hybrid Schedule for Job Scheduling and Load Balancing in Heterogeneous Computational Grids," *Proc. of 6th IEEE Int'l Parallel & Distributed Processing Symp.*, July 5–8, pp. 121–128, Hagenberg, Austria.

[21] M. Maheswaran, H. J. Siegel, D. Haensgen, and R. F. Freud, "Dynamic Mapping of A Class of Independent Tasks onto Heterogeneous Computing Systems", *Journal of Parallel and Distributed Computing*, Vol. 59, No.2, pp.107-131, Feb. 1999.

[22] M. Mattess, C. Vecchiola, R. Buyya. "Managing Peak Loads by Leasing Cloud Infrastructure Services from a Spot Market", *the 12th IEEE International Conference on High Performance Computing and Communications (HPCC'10)*, Melbourne, Australia.

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

[24] C. Moretti, H. Bui, K. Hollingsworth, B. Rich, P. J. Flynn, D. Thain "All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids", *IEEE Trans. Parallel Distrib. Syst.*, Vol. 21, No.1, pp. 33-46, 2010.

[25] A. Mutz, and R. Wolski, "Efficient Auction-Based Grid Reservation using Dynamic Programming", *Proc. of IEEE/ACM Int'l Conf. on SuperComputing*, Austin, 2007.

[26] B. Ozisikyilmaz, G. Memik, and A. Choudhary. "Efficient System Design Space Exploration using Machine Learning Techniques", *Proc. of Design Automation Conf.*, June 2008.

[27] P. Padala, Kang G. Shin, et al. "Adaptive Control of Virtualized Resources in Utility Computing Environments", *2007 ACM SIGOPS Operating Systems Review (EuroSys'07)*, Lisbon, Portugal.

[28] S. Pal, S. K. Das and M. Chatterjee. "User-Satisfaction Based Differentiated Services for Wireless Data Networks", *2005 IEEE International Conference on Communications (ICC'2005)*. Seoul Korea.

[29] R. Prodan and M. Wieczorek, "Bi-criteria Scheduling of Scientific Grid Workflows". *IEEE Tran. on Automation Science and Engineering*, Vol. 7, No. 2, pp. 364-376, 2010.

[30] J. Smith, H. J. Siegel and A. A. Maciejewski. "A Stochastic Model for Robust Resource Allocation in Heterogeneous Parallel and Distributed Computing Systems", *Proc. of IEEE Int'l Parallel & Distributed Processing Symp. (IPDPS'08)*, Miami, FL. 2008.

[31] C. Song, X. H. Guan, Q. C. Zhao, and Y. C. Ho, "Machine Learning Approach for Determining Feasible Plans of a Remanufacturing System", *IEEE Tran. on Automation Science and Engineering*, Vol. 2, No. 3, pp. 262-275, 2005.

[32] D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd, "Performance-aware Workflow Management for Grid Computing", *The Computer J., Special Focus - Grid Performability*, Vol. 48, No. 3, pp. 347-357, 2005.

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

[34] W. Tan, Y. Fan, M. C. Zhou, Z. Tian, Data-Driven Service Composition in Enterprise SOA Solutions: A Petri Net Approach. *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 3, pp. 686–694, 2010.

[35] W. Tan, Y. Fan, M. C. Zhou, A petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering*, Vol. 6, No. 1, pp. 94–106, 2009.

[36] S. Teng, L. H. Lee, and E. P.Chew, "Multi-objective Ordinal Optimization for Simulation Optimization Problems", *Automatica*, Vol. 43, No.11, pp.1884-1895, 2007.

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

[38] J. E. Wieselthier, C. M. Barnhart, and A. Ephremides, "Standard Clock Simulation and Ordinal Optimization Applied to Asmission Control in Intergrated Communication Networks", *Discrete Event Dynamic System: Theory and Application*, Vol. 5, pp. 243-280, 1995.

[39] J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms", *Scientific Programming*, Vol. 14, No. 1, 2006.

[40] F. Zhang, J. Cao, K. Hwang, and C. Wu, "Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds", *Proc. 3rd IEEE Int. Conf. on Cloud Computing Technology and Science*, Athens, Greece, 2011.

[41] F. Zhang, J. Cao, C. Hong, J. J. Mulcahy, and C. Wu, "Provisioning Virtual Resources Adaptively in Elastic Compute Cloud Platforms", *Int. J. Web Services Research*, Vol. 8, No. 4, pp. 54-69, 2011.

[42] F. Zhang, J. Cao, K. Li and S. Khan, "Multi-Objective Scheduling of Many Tasks in Cloud Platforms", *Future Generation Computer Systems* (SI on Advances in Data-Intensive Modelling and Simulation), in press.

[43] F. Zhang, J. Cao, L. Liu, and C. Wu, "Performance Improvement of Distributed Systems by Autotuning of the Configuration Parameters", *Tsinghua Science and Technology*, 16(4), 440-448, 2011.

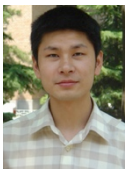
[44] Q. Zhang, E. Gurses, R. Boutaba, J. Xiao. "Dynamic Resource Allocation for Spot Markets in Clouds", *the 4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'11)*, Melbourne, Australia.

[45] H. Zhao, M. Pany, X. Liu, X. Liy and Y. Fangy. Optimal Resource Rental Planning for Elastic Applications in Cloud Market, *the 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS'12)*, Phoenix, Arizona.

BIOGRAPHIES

Fan Zhang (S'08, M'12, SM'13) is currently a postdoctoral associate with the Kavli Institute for Astrophysics and Space Research at Massachusetts Institute of Technology. He has been appointed as a visiting associate professor in the Shenzhen Institute of advanced technology, Chinese Academy of Science since Jan 2014. He received his Ph.D. in Department of Control Science and Engineering, Tsinghua University in Jan. 2012. He received the B.S. in computer science from Hubei University of Technology and M.S. in control science and engineering from Huazhong University of Science and Technology. From 2011 to 2013 he was a research scientist at Cloud Computing Laboratory, Carnegie Mellon University. An IEEE Senior Member, he received an Honorarium Research Funding Award from the University of Chicago and Argonne National Laboratory (2013), a Meritorious Service Award (2013) from IEEE Transactions on Service Computing, two IBM Ph.D. Fellowship Awards (2010 and 2011). His research interests include big-data scientific computing applications, simulation-based optimization approaches, cloud computing, and novel programming models for streaming data applications on elastic cloud platforms.

Junwei Cao (M'99, SM'05) received his Ph.D. in computer science from the University of Warwick, Coventry, UK, in 2001. He received his bachelor and master degrees in control theories and engineering in 1998 and 1996, respectively, both from Tsinghua University, Beijing, China. He is currently Professor and Deputy Director of Research Institute of Information Technology, Tsinghua University, Beijing, China. He is also Director of Open Platform and Technology Division, Tsinghua National Laboratory for Information Science and Technology, Beijing, China. He is an adjunct faculty of Electrical and Computer Engineering, North Dakota State University. Before joining Tsinghua University in 2006, he was a Research Scientist at MIT LIGO Laboratory and NEC Laboratories Europe for about 5 years. He has published over 160 papers and cited by international scholars for over 7,000 times according to Google Scholar. He has authored or edited 6 books or conference proceedings. He is an associate editor of IEEE Tractions on Cloud Computing and an editor of Journal of Computational Science. He is a General Co-chair of International Conference on Networking and Distributed Computing. His research is focused on distributed computing and networking and energy/power applications. Prof. Cao is a Senior Member of the IEEE Computer Society and a Member of the ACM and CCF.



Wei Tan (M'12, SM'13) received the B.S. degree and the Ph.D. degree from Department of Automation, Tsinghua University, China in 2002 and 2008, respectively. He is currently a research staff member in IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. From 2008 to 2010 he was a researcher at Computation Institute, University of Chicago and Argonne National Laboratory, USA. At that time he was the technical lead of the caBIG workflow system. His research interests include NoSQL, big data, cloud computing, service-oriented architecture, business and scientific workflows, and Petri nets. He has published over 50 journal and conference papers, and a monograph "Business and Scientific Workflows: A Web Service-Oriented Approach" (272 pages, Wiley-IEEE Press). He is an associate editor of IEEE Tractions on Automation Science and Engineering. He served in program committee of many conferences and co-chaired several workshops. He received the Best Paper Award from IEEE International Conference on Services Computing (2011), Pacesetter Award from Argonne National Laboratory (2010), and caBIG Teamwork Award from the National Institute of Health (2008). He is member of ACM and IEEE.

Samee U. Khan (S'02, M'07, SM'12) is an assistant professor at the North Dakota State University. He received his Ph.D. from the University of Texas at Arlington in 2007. Dr. Khan's research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in over 225 publications with two receiving best paper awards. He is a Fellow of the IET and a Fellow of the BCS.

Keqin Li (M'90, SM'96) is a SUNY distinguished professor of computer science and an Intellectual Ventures endowed visiting chair professor at Tsinghua University, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. Dr. Li has over 265 research publications and has received several Best Paper Awards for his research work. He is currently on the editorial boards of IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing and Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing, which was established, in late 2009. Professor Zomaya is the author/co-author of seven books, more than 400 papers, and the editor of nine books and 11 conference proceedings. He is the Editor-in-Chief of the IEEE Transactions on Computers and serves as an associate editor for 19 leading journals, such as, the IEEE Transactions on Parallel and Distributed Systems and Journal of Parallel and Distributed Computing. Professor Zomaya is the recipient of the Meritorious Service Award (in 2000) and the Golden Core Recognition (in 2006), both from the IEEE Computer Society. Also, he received the IEEE Technical Committee on Parallel Processing Outstanding Service Award and the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing, both in 2011. Dr. Zomaya is a Chartered Engineer, a Fellow of AAAS, IEEE, and IET (UK).