Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds

Rubing Duan, Radu Prodan, and Xiaorong Li, *Member, IEEE Computer Society*

Abstract—Scheduling multiple large-scale parallel workflow applications on heterogeneous computing systems like hybrid clouds is a fundamental NP-complete problem that is critical to meeting various types of QoS (Quality of Service) requirements. This paper addresses the scheduling problem of large-scale applications inspired from real-world, characterized by a huge number of homogeneous and concurrent bags-of-tasks that are the main sources of bottlenecks but open great potential for optimization. The scheduling problem is formulated as a new sequential cooperative game and propose a communication and storage-aware multi-objective algorithm that optimizes two user objectives (execution time and economic cost) while fulfilling two constraints (network bandwidth and storage requirements). We present comprehensive experiments using both simulation and real-world applications that demonstrate the efficiency and effectiveness of our approach in terms of algorithm complexity, makespan, cost, system-level efficiency, fairness, and other aspects compared with other related algorithms.

Index Terms—Multi-objective scheduling, game theory, bags-of-tasks, hybrid clouds

1 Introduction

Distributed computing systems such as clouds and grids have evolved over decade to support various types of scientific applications with dependable, consistent, pervasive, and inexpensive access to geographically-distributed high-end computational capabilities. To program such a large and scalable infrastructure like hybrid clouds, loosely coupled-based coordination models of legacy software components such as *bags-of-tasks (BoT)* and *workflows* [1] have emerged as one of the most successful programming paradigms in the scientific community.

One of the most challenging NP-complete problems [2] that researchers try to address is how to schedule large-scale scientific applications to distributed and heterogeneous resources such that certain objective functions such as total execution time (called from hereafter *makespan*) in academic Grids or economic cost (in short *cost* from hereon) in business or market-oriented clouds are optimized, and certain execution constraints such as communication cost and storage requirements are considered and fulfilled. From the end-users' perspective, both minimizing cost or execution time are preferred functionalities, whereas from the system's perspective system-level efficiency and fairness can be considered as a good motivation such that the applications with more amount of computation should be allocated with more resources. Currently, only a few schemes can deal with both perspectives, such as optimizing user objectives (e.g., makespan, cost) while fulfilling other constraints, and providing a good efficiency and fairness to all users. On the other hand, many applications can generate huge data sets in a relatively short time, such as the Large Hadron Collider [3] expected to produce 5-6 petabytes of data per year, which must be accommodated and efficiently handled through appropriate scheduling bandwidth and storage constraints.

Since there may be many different types of applications or tasks in the clouds which are competitors for the use of available resources, several issues arise and ought to be dealt with such as the efficient resource allocation for different applications taking into account their individual performance, cost, bandwidth, storage, and other potential constraint requirements, and the fair use of resources from the system perspective. With respect to hybrid clouds, Google launched in 2012 the Google compute engine (GCE) cloud, which will provide hosted Linux virtual machines. Google opens up its infrastructure for developers and is becoming a formidable competitor to the Amazon elastic compute cloud (EC2). Table 1 shows a comparison of the pricing between EC2 and GCE revealing that Google has better prices overall. On the other hand, Fig. 1 shows the bandwidth between different Amazon and Google cloud services demonstrating that Google cloud storage (GCS) is slower than Amazon simple storage service (S3), and GCE transfer is slower than EC2. The interesting observation is that transfer from S3 to GCE is even 2.3 times faster than from GCS to GCE. Even though EC2 prevails in this early stage, nothing prevents Google from releasing larger instances with faster CPUs soon. Therefore, we expect that as more cloud services are being used, the optimization on hybrid clouds becomes an increasingly complex undertaking.

In this paper, we address these issues by proposing a communication and storage-aware multi-objective scheduling scheme for an important class of workflow applications characterized by large sets of independent and homogeneous BoTs, interconnected through data flow dependencies:

- *multi-objective scheduling* minimizes the expected execution time and economic cost of applications based on a sequential cooperative game theoretic algorithm;

- R. Duan and X. Li are with Institute of High Performance Computing, A*STAR, Singapore. E-mail: duanr@ihpc.a-star.edu.sg.
- R. Prodan is with University of Innsbruck, 6020 Innsbruck, Austria.

Manuscript received 12 June 2013; revised 28 Oct. 2013; accepted 14 Dec. 2013; date of publication 27 Jan. 2014; date of current version 30 Apr. 2014.
Recommended for acceptance by C. De Rose.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2303077

TABLE 1
Amazon EC2 versus Google Cloud
(Prices as of December 1, 2012)

Provider	*Name*	*Virtual cores*	*Memory (GB)*	*Compute Unit*	*HDD (GB)*	*\$/ hour*	*\$/unit /hour*
Google	`n1-standard-1-d`	1	3.75	2.75	420	0.145	0.053
Amazon	`m1.medium`	1	3.75	2	410	0.160	0.080
Google	`n1-standard-2-d`	2	7.5	5.5	870	0.290	0.053
Amazon	`m1.large`	2	7.5	4	850	0.320	0.080
Google	`n1-standard-4-d`	4	15	11	1770	0.580	0.053
Amazon	`m1.xlarge`	4	15	8	1690	0.640	0.080
Google	`n1-standard-8-d`	8	30	22	2×1770	1.160	0.053
Amazon	`m2.xlarge`	4	34.2	13	850	0.900	0.069
Amazon	`c1.xlarge`	8	7	20	1690	0.660	0.033
Amazon	`cc1.4xlarge`	8	23	33.5	1690	1.300	0.039
Amazon	`cc1.8xlarge`	2×8	60.5	88	3370	2.400	0.027

- *communication and storage-aware scheduling* minimizes the makespan and cost of applications while taking into account their bandwidth and storage constraints for transferring the produced data.

The main advantages of our game theoretic algorithm are its faster convergence by using competitors and environment information to determine the most promising search direction by creating logical movements, its minimum requirements regarding the problem formulation, and its easy customisation to for new objectives. We compare the performance of our approach with six related heuristics and show that, for the applications with large BoTs, our algorithm is superior in complexity (orders-of-magnitude improvement), quality of result (optimal in certain known cases), system-level efficiency and fairness. Our algorithm may not be suited for applications with complex dependencies between BoTs (such as the hydrological Invmod, meteorological MeteoAG [4] or the bioinformatics association test [5] workflows), for which the scheduling problem cannot be properly formulated as a typical and solvable game, consisting of phases specifically defined so that game players can bargain with limited dependencies between each other.

The paper is structured as follows. Section 2 reviews the most relevant related work. Motivated by real-world applications and real heterogeneous computing testbeds, we introduce in Section 3 the application and the hybrid cloud computing models, followed by the paper's problem definition. Section 4 describes the communication and storage-aware multi-objective algorithm in detail. In Section 5, we validate and compare our algorithm against related methods through simulated and real-world experiments in a hybrid cloud environment. Section 6 concludes the paper and discusses some future work.

2 Related Work

Scheduling large-scale workflow applications is one of the most important and difficult research topics in cloud computing that led to the development of many different approaches and algorithms. In this section, we can therefore cover only a part of the important or relevant areas of related work: multi-objective scheduling, game theoretic scheduling, single workflow scheduling, multiple workflow scheduling and hybrid cloud computing.

2.1 Multi-Objective Scheduling

Many real world scheduling problems are multi-objective by nature, i.e., it is common that in these problems several

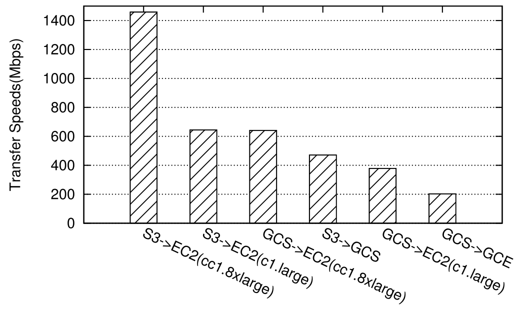


Fig. 1. Bandwidth on public clouds.

objectives exist such as minimizing the makespan of the schedule, minimizing the economic cost, satisfying the limitation of storage and network resources, etc. Over the years, there have been several approaches used to deal with the various objectives in these problems.

Traditionally, the most common way has been to analytically combine the multiple objectives into a single aggregate objective function by assigning weights to different objectives [6], [7], [8], [9]. However, in many real scenarios involving multiobjective scheduling problems, it is preferable to present various compromise solutions to the users so that the most adequate schedule can be chosen. Furthermore, the aggregate objective function is often nonlinear in the objectives, and the solution obtained will depend on the relative values of the weights specified. Therefore, the weighted sum method is essentially subjective and needs a priori information on the users' preferences.

This has increased the interest for investigating Pareto optimization methods for finding a set of compromise solutions that represent good approximations to the Pareto optimal front. For example, Fard et al. [10] proposed a method based on Pareto dominance constraints vector that has been applied to four objectives as part of the goal function (i.e., makespan, cost, energy consumption, and reliability). Our approach is distinct from this work by having lower time complexity and considering two different objectives (makespan and cost) and two constraints (bandwidth, and storage resources).

Many meta-heuristic approaches that were originally applied to single-objective optimization problems have been extended to produce multi-objective optimization variants. Among these, multi-objective evolutionary algorithms have received particular attention since several researchers argue that these methods are particularly well suited to deal with multi-objective optimization problems. Also, some multi-objective meta-heuristics such as simulated annealing, genetic algorithms (GA), tabu search and particle swarm optimization (PSO) have been proposed recently to address scheduling problems. There are also several efforts that move in the same direction as our work but try to address smaller and simpler version of the problem. Yu and Buyya introduced in [11] a new type of genetic algorithm for large-scale heterogeneous environments for which the existing genetic operation algorithms cannot be directly applied. Due to their high time complexity, GAs are not practical for large-scale applications. The algorithm in [12] introduced economic cost as a part of the objective function for data and computation scheduling, but does not consider storage constraints and cannot globally address the performance and cost optimization problem. Pandey et al. [13] proposed a PSO-based heuristic to schedule applications to cloud resources that take into account both computation cost and transmission cost. The main disadvantage of evolutionary algorithms is their high computational cost due to their slow convergence rate. Our method, in contrast, has a much faster convergence rate due to effective utilization of global resource information.

2.2 Game Theoretic Scheduling

In terms of game theoretic algorithms, other researchers in performance-oriented distributed computing have focused on system-level load balancing [14], [15] or resource allocation [16], [17], aiming to introduce economic and game theoretic aspects into computational questions. Penmatsa and Chronopoulos [15] formulated the scheduling problem as a cooperative game where Grid sites try to minimize the expected response time of tasks, while Kwok et al. [17] investigated the impact of selfish behaviors of individual machine by taking into account the non-cooperativeness of machines. Ghosh et al. [18] proposed a strategy that formulates an incomplete information, alternating-offers bargaining game on two variables: price per unit resource and percentage of bandwidth allocated. Compared with Ghosh's work, we used a more practical pricing model similar to the one used by Cloud resource providers such as amazon elastic compute cloud. The ICENI project [19] addressed the scheduling problem using a game theory algorithm that eliminates strictly dominated strategies where the least optimal solutions are continuously discarded. The feasibility of this algorithm is questionable due to its high time complexity. Apart from the game theory algorithm, ICENI provides scheduling solutions using random, best of n-random, and simulated annealing.

2.3 Multiple Workflow Scheduling

The work in [20] proposes static and dynamic scheduling and resource provisioning strategies for workflow ensembles, defined as a group of inter-related workflows with different priorities. The paper discusses strategies for admission or rejection of a workflow execution based on its structure and task runtime estimates, and analyses their impact on fulfilling deadline and cost constraints. In [21], the authors present an experimental study of various deterministic non-preemptive scheduling strategies for multiple workflows in Grids that take into account both dynamic site state information and workflow properties. They conducted a comprehensive evaluation of twenty five scheduling strategies depending on the type and amount of information required in two and four stages: labeling, adaptive allocation, prioritization, and parallel machine scheduling. Their results showed that the proposed strategies outperform well-known single directed acyclic graph (DAG) scheduling algorithms. The authors in [22] proposed four heuristics for scheduling multiple workflows using a clustering technique. They concluded that interleaving the workflows leads to good average makespan and provides fairness when multiple workflows share the same set of resources. An approach for scheduling multiple DAGs on heterogeneous systems to achieve fairness defined as a basis of the slowdown experienced because of competition for resources is presented in [23]. The work considers two fairness policies based on finishing and concurrent times that arrange the DAGs in ascending order of their slowdown value, selects independent tasks from the DAG with minimum slowdown, and schedules them using the heterogeneous earliest finishing time (HEFT) algorithm [24]. In contrast to all these works targeting a static set of input workflows, our work targets workflow-interconnected BoT.

The work in [25] addressed online scheduling of multiple DAGs in an optical Grid environment based on two DAG aggregation strategies: first come first serve that appends arriving DAGs to an exit node of the single DAG, and service on time that appends arriving DAGs to a task whose predecessors have not completed execution. The resulting DAG is then scheduled using HEFT.

2.4 Hybrid Cloud Scheduling

The research domain of hybrid cloud computing is relatively young. Bittencourt et al. [26] provide a brief survey of scheduling algorithms for hybrid clouds and the impact of communication networks on scheduling decisions. The concept of sky computing has been introduced in [27] for building a virtual site distributed on several Clouds. In [28], important challenges and architectural elements for utility-oriented federation of multiple cloud computing environments are investigated. The role of cloud brokers responsible for splitting user requests to multiple providers with the goal of decreasing the cost for users is discussed in [29], [30].

2.5 Contribution

The approach presented in this paper successfully applies game theoretic concepts for scheduling multiple large-scale applications onto hybrid clouds. Our work differs from the related work in that we present a more realistic and feasible system model and considered an important class of large-scale applications characterized by a large number of homogeneous independent tasks interconnected through simple control flow and data flow dependencies. We formulated the scheduling problem as a new sequential cooperative game among several application managers controlling the execution of individual applications and introduced one multiobjective algorithm for makespan, cost, communication- and storage-aware optimization which, compared with other systems, consider intra- and inter-application cooperation. To the best of our knowledge, no workflow system deals with the concept of hybrid elastic clouds, large-scale applications and multiple dimensional optimization.

3 Model

We describe in this section the abstract application and hybrid cloud models used in this paper, motivated by real-world applications and real cloud testbeds.

3.1 Application Model

We focus on large-scale workflows characterized by a high number (thousands to millions) of homogeneous parallel

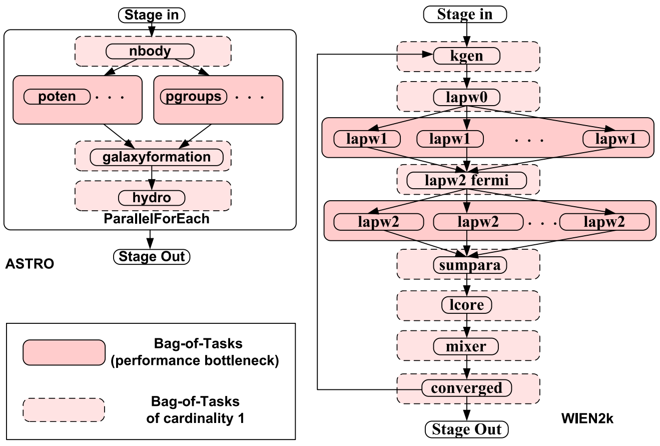


Fig. 2. Real-world application examples.

(independent) tasks that dominate their performance, interconnected through control and data flow dependencies.

Definition 3.1. *Let $\mathcal{W} = (\mathcal{BS}, \mathcal{DD})$ denote a* workflow *application modeled as a DAG, where $\mathcal{BS} = \bigcup_{k=1}^{K} \mathcal{T}_k$ is the set of K heterogeneous* bags-of-tasks (BoTs) *and $\mathcal{DD} = (\mathcal{T}_s <^d \mathcal{T}_d | \{\mathcal{T}_s, \mathcal{T}_{d,}\} \subset \mathcal{BS})$ is the set of data flow dependencies. We call $\mathcal{T}_s$ the* predecessor *of $\mathcal{T}_d$ and write: $\mathcal{T}_s = pred(\mathcal{T}_d)$. We define a BoT $\mathcal{T}_k$ as a homogeneous set of parallel atomic sequential* tasks *$\mathcal{T}_k = \bigcup_{j=1}^{K_k} t_{kj}, k \in [1 \cdots K]$ that have the same* task type *and can be concurrently executed, where K_k is the* cardinality *of BoT.*

A task type refers to an abstract functional description of tasks. For example, Fig. 2 depicts two real-world applications that we use as case study in our work and which had an impact on our proposed application model: WIEN2k from theoretical chemistry and ASTRO from astronomy domains. Examples of task types are matrix multiplication, Fast Fourier Transform, or `poten`, `pgroups`, `lapw1`, and `lapw2` for our pilot applications. While the tasks within a BoT are homogeneous and have the same type (e.g.,`lapw1`), the BoT themselves are heterogeneous in terms of the contained number of tasks and their type (e.g.,`lapw1` versus`lapw2`).

WIEN2k [31] is a program package for performing electronic structure calculations of solids using density functional theory based on the full-potential (linearized) augmented plane-wave ((L)APW) and local orbital method. We have ported this application onto the hybrid clouds by splitting the monolithic code into several coarse-grain tasks coordinated in a simple workflow illustrated in Fig. 2. The `lapw1` and `lapw2` BoTs can be solved in parallel by a fixed number of homogeneous tasks called k-points. A final task named converged applied on several output files tests whether the problem convergence criterion is fulfilled.

ASTRO [32] is an astronomical application that solves numerical simulations of the movements and interactions of galaxy clusters using an N-Body system. The computation starts with the state of the universe at some time in the past and is done until the current time. Galaxy potentials are computed for each time step. Finally, the hydrodynamic behavior and processes are calculated.

The sources of performance bottlenecks in these applications are homogeneous BoTs such as `lapw1` and `lapw2` in WIEN2k, or `poten` and `pgroups` in ASTRO. The number of grid cells (i.e., `pgroups` and `poten` tasks) of a real simulation in ASTRO is 128^3, while the number of `lapw1` and `lapw2` parallel tasks in WIEN2k may be of tens of thousand for a good density of states. Currently, most related work only considers applications with tens or hundreds of tasks, which are an order of magnitude lower than the size of our applications. Sequential tasks are relatively trivial in large-scale applications and can be served and scheduled on-demand on the fastest or cheapest available processor.

3.2 Hybrid Cloud Model

A hybrid cloud computer consists of minimum of one private and one public cloud. For example, an organization might use a public cloud service, such as amazon elastic compute cloud (EC2) or Google compute engine as an extension to its in-house private cloud. The hybrid approach allows taking advantage of scalability and cost efficiency a public cloud offers without exposing mission critical applications and data to third party vulnerabilities.

Tasks or jobs arriving at each cloud site may belong to multiple applications. The execution of each application is controlled by one *application manager* which competes with the other application managers for resources. Most other scheduling approaches in the related work assume direct mapping of user jobs or tasks to individual processors which we consider inappropriate for cloud computing where sites are usually managed by locally administered queuing systems. To support this more realistic model, the application manager maintains locally one queue for each cloud site in order to schedule and limit the number of job submissions based on the site's task processing rate. From the local queue, the jobs are submitted by the application managers to the gatekeepers of the remote public cloud sites. We assume without loss of generality that resources within a cloud site are homogeneous, while different sites are heterogeneous.

4 Multi-Objective Scheduling

The goal of this paper is to design a new algorithm for scheduling a set of applications defined according to Definition 3.1 and consisting of a huge number of tasks (for which existing algorithms do not scale) in an environment modeled in Section 3.2. Our algorithm aims to optimize two objective functions: aggregated makespan and aggregated cost, while optionally fulfilling bandwidth and storage constraints. In this section, we first formally formulate the multi-objective scheduling problem for an important class of large-scale applications introduced in Section 3.1 (see Definition 3.1) and then propose and experimentally validate a game theory-based algorithm to efficiently address it.

4.1 Problem Formulation

Definition 4.1. *Suppose we have a set of n applications (modelled as in Definition 3.1 and ignoring their arrival times) consisting of tasks that can be categorized into K different BoTs, and a cloud environment consisting of M sites. The* makespan *of an application $\mathcal{A}_i, i \in [1, n]$ is the maximum completion time of its BoTs. The objective of the* multi-objective scheduling *problem is to find a solution that assigns all tasks to the sites such that the makespan and economic cost of all applications $F(x)$ are minimized, and the bandwidth and storage requirements are fulfilled*

$$\begin{aligned} \underset{x}{Minimize}\, F(x) &= (f(x), c(x)), \\ s.t.\ \ h_i(x) &\leq \lambda_{x,i}, i \in [1, M], \\ g_i(x) &\leq sl_i, i \in [1, M], \\ x &\in S, \end{aligned} \tag{1}$$

where x is a solution, S is the set of feasible solutions, $F(x)$ is the image of x in the multi-objective space, $f(x)$ is the performance objective, $c(x)$ is the economic cost objective, $g(x)$ is the storage use function, $h(x)$ is the bandwidth use function, $\lambda_{x,i}$ is the input data bandwidth to site s_i, and sl_i is the storage limit on site s_i.

We assume the availability of an *expected time to compute (ETC)* [33] matrix which delivers the *expected execution time* p_{ki} of tasks in each BoT $k \in [1, K]$ on each site s_i, $i \in [1, M]$. Communication overhead is a crucial issue for performance in a heterogeneous computing environment, in many cased being the most significant factor. For large-scale applications with a large number of tasks, communication and computation could run simultaneously and overlap, resulting in the execution time determined by either communication or computation. We therefore define the expected execution time p_{ki} based on the computation time (pc_{ki}) and communication time (po_{ki}), as follows:

$$p_{ki} = \begin{cases} pc_{ki}, & pc_{ki} \geq po_{ki}; \\ pc_{ki} + (po_{ki} - pc_{ki}) = po_{ki}, & pc_{ki} < po_{ki}. \end{cases} \tag{2}$$

where po_{ki} can be expressed as

$$po_{ki} = \frac{d_{ki}}{b_{ki}}, \tag{3}$$

where d_{ki} is the data size of tasks, and b_{ki} is the bandwidth allocated to the BoT k on site s_i. The input data bandwidth to site s_i ($\lambda_{x,i}$) is the sum of b_{ki} on site s_i:

$$\lambda_{x,i} \leq \sum_{k=1}^{K} \theta_{ki} \cdot b_{ki} = \sum_{k=1}^{K} \frac{\theta_{ki} \cdot d_{ki}}{po_{ki}}, \tag{4}$$

where θ_{ki} is the number of processors allocated to BoT $\mathcal{T}_k$ on site s_i. θ_{ki} is a real number, because the bidding on a small amount of resources makes the fine adjustment of resource allocation feasible. Our algorithm will round θ_{ki} to an integer in the end of scheduling. Based on the above analysis, we can find that for data-intensive applications, the expected execution time p_{ki} is determined by either computation time or communication time. Communication time is determined by the bandwidth allocated to each BoT.

4.2 Game Theoretic Solution

The multi-objective scheduling problem can be formulated as a cooperative game among the application managers which can theoretically generate the optimal solution, although this is hard to achieve due to the problem's high complexity. We therefore observe that the problem can be further formulated and addressed as a *sequential cooperative game* that requires the proper definition of three important parameters: the players, the strategies, and the specification of payoff.

We consider a *K-player cooperative game* in which each of the K application managers (as players) attempts at certain time instances to minimize the execution time t_k of one BoT $\mathcal{T}_k$ based on its total *number of tasks* δ_k and its *processing rate* β_{ki} on each site s_i. For clarity, we assume that each application manager handles the execution of one BoT. The objectives of each manager are to minimize the execution time and economic cost of its BoT while fulfilling storage and bandwidth constraints, expressed as

$$f_k(\Delta) = \frac{\delta_k}{\beta_k} = \frac{\delta_k}{\sum_{i=1}^{M} \frac{\theta_{ki}}{p_{ki}}}; \tag{5}$$

$$c_k(\Delta) = \sum_{i=1}^{M} p_{ki} \cdot \delta_{ki} \cdot \varphi_i; \tag{6}$$

$$h_i(\Delta, \mathcal{B}) = \sum_{k=1}^{K} \frac{\theta_{ki} \cdot d_{ki}}{p_{ki}} \leq \lambda_{x,i}; \tag{7}$$

$$g_i(\Delta) = \sum_{k=1}^{K} sr_k \cdot \theta_{ki} \leq sl_i, \tag{8}$$

where Δ is a *task distribution matrix* $(\delta_{ki})_{K \times M}$, sr_k the storage requirements of BoT $\mathcal{T}_k$, sl_i the storage limit on site s_i, φ_i is the price of site s_i, $\mathcal{B}$ is a *bandwidth allocation matrix* $(b_{ki})_{K \times M}$. We assume in Equation (6) that the users only pay for useful computation and, therefore, the price is independent on the number of processors used. Both Δ and $\mathcal{B}$ represent strategies as the embodiment of payoff in the cooperative game.

Cooperative game theory is concerned with situations when groups of players coordinate their actions, which is the most important algorithmic mechanism that makes games have "transferable utility". In other words, a player with increased utility has the ability to compensate some other players with decreased utility. When designing games with transferable utility, the main concern is to develop solutions for formalizing fair ways of sharing resources. For instance, the term θ_{ki} defined in the following represents the *resource allocation* of BoT k on site s_i (which embodies the fairness of sharing resources), defined as the product between the number of processors m_i on s_i and the ratio between the weighted aggregated execution times of BoT $\mathcal{T}_k$ on s_i and the aggregated execution time of all BoTs on s_i:

$$\theta_{ki} = m_i \cdot \frac{\delta_{ki} \cdot p_{ki} \cdot w_{ki}}{\sum_{k=1}^{K} \delta_{ki} \cdot p_{ki} \cdot w_{ki}}, \tag{9}$$

where w_{ki} is the weight of site s_i for BoT $\mathcal{T}_k$ with different definitions for performance (pw_{ki}), cost (cw_{ki}), bandwidth bw_{ki}, and storage (sw_k) optimizations, as follows:

$$pw_{ki} = \frac{\frac{\min\limits_{i\in[1,M]}\{p_{ki}\}}{p_{ki}}}{\sum_{i=1}^{M}\frac{\min\limits_{i\in[1,M]}\{p_{ki}\}}{p_{ki}}} = \frac{\frac{1}{p_{ki}}}{\sum_{i=1}^{M}\frac{1}{p_{ki}}}; \tag{10}$$

$$cw_{ki} = \begin{cases} \frac{\frac{1}{\varphi_i\cdot p_{ki}}}{\sum_{i=1}^{M}\frac{1}{\varphi_i\cdot p_{ki}}}, & \varphi_i \neq 0; \\ \frac{\frac{1}{p_{ki}}}{\sum_{i=1}^{M}\frac{1}{\varphi_i\cdot p_{ki}}}, & \varphi_i = 0; \end{cases} \tag{11}$$

$$bw_i = \frac{\frac{p_{ki}}{d_{ki}}}{\sum_{k=1}^{K}\frac{p_{ki}}{d_k}}; \tag{12}$$

$$sw_i = \frac{\frac{1}{sr_k}}{\sum_{k=1}^{K}\frac{1}{sr^k}}. \tag{13}$$

The problem of how to write weight function must be carefully considered so that better performance and cost are attainable and actually improve scheduling for the given problem. If the weight function is chosen poorly or defined imprecisely, the algorithm may be unable to find a solution to the problem. For instance, we use the performance weight pw_{ki} to enhance the fairness of allocation in terms of performance, because one site may have different suitability for different tasks for various reasons such as the locality of data, the size of memory, the CPU frequency, or the I/O speed. Intuitively, if the execution time t_{ki} on site s_i for BoT $\mathcal{T}_k$ is much shorter than on other sites, we set a higher priority for this BoT on this site and allocate more resources to it. Our algorithm dynamically chooses different weights when some objectives deviate from users' expectation or violate some constraints. We will explain the utilization of different weights when we introduce the notion of sequential game.

Based on the allocation of resources and the ratio of processing rate on site s_i to the total processing rate of the BoT, we define the *task distribution* as the product between the number of tasks in $\mathcal{T}_k$ and the ratio between the processing rate of $\mathcal{T}_k$ on site s_i with respect to the total processing rate of $\mathcal{T}_k$:

$$\delta_{ki} = \delta_k \cdot \frac{\frac{\theta_{ki}}{p_{ki}}}{\sum_{i=1}^{M}\frac{\theta_{ki}}{p_{ki}}}. \tag{14}$$

Definition 4.2. *Accordingly, we can define a* multi-objective optimization cooperative game *as consisting of*

- *Managers of K BoTs as players.*
- *A set of strategies Δ and $\mathcal{B}$ defined by the following set of constraints: (1) $\delta_{ki} \geq 0$; (2) $b_{ki} \geq 0$; (3) $\theta_{ki} \leq \delta_{ki}$; (4) $b_{ki} \leq \lambda_{x,i}$; (5) $sr_{ki} \leq sl_{x,i}$; (6) $\delta_{ki} = 0$, if $\theta_{ki} < 1$; (7) $\sum_{k=1}^{K}\theta_{ki} = m_i$; (8) $\sum_{k=1}^{K} b_{ki} = \lambda_{x,i}$; (9) $\sum_{k=1}^{K} sr_{ki} =_i$; and (10) $\sum_{i=1}^{M}\delta_{ki} = \delta_k$.*
- *For each player $k \in [1, K]$, the objective functions $f_k(\Delta)$ and $c_k(\Delta)$. The goal is to minimize simultaneously all f_k and c_k while satisfying g_k and h_k.*
- *For each player $k \in [1, K]$, the initial value of the objective function $f_k(\Delta^0, \mathcal{B}^0)$, where $\Delta^0 = (\delta_{ki})^0_{K\times M}$ is a $K \times M$ matrix filled with initial distribution of tasks, $\mathcal{B}^0 = (b_{ki})^0_{K\times M}$ is a $K \times M$ matrix filled with initial allocation of bandwidth. The term $\mathcal{B}^0$ is calculated based on Δ^0, and (Δ^*,$\mathcal{B}^*$) denotes the optimal solution of the game. An initial allocation is generated based on Δ^0 and $\mathcal{B}^0$, and does not need to be a feasible solution that fulfills all constraints.*

For large-scale applications with BoTs, the overall makespan is almost equal to the aggregated execution time divided by the number of processors. Therefore, the performance goal of our cooperative optimization game can be approximated to minimizing the aggregated makespan

$$\underset{\Delta}{argmin}\left(\sum_{k=1}^{K} f_k(\Delta)\right), \tag{15}$$

subject to the constraints (1)-(10).

The typical method for finding the extremum of a function subject to several constraints is by defining the *Lagrangian*:

$$\begin{aligned}\mathcal{L}(\delta,\eta,\theta,\iota,b,\phi) = & \sum_{k=1}^{K}(f_k + c_k + g_k + h_k) + \eta\cdot\left(\sum_{k=1}^{K}\theta_{ki} - m_i\right) \\ & + \phi\cdot\left(\sum_{k=1}^{K} b_{ki} - \lambda_{x,i}\right) + \varsigma\cdot\left(\sum_{i=1}^{M}\delta_{ki} - \delta_k\right) \\ & + \sum_{k=1}^{K}\iota_i^k\cdot\delta_{ki} + \ldots,\end{aligned} \tag{16}$$

where η, ϕ, ς, and ι denote the Lagrange multipliers. Unfortunately, the exact and optimal solution to this problem is in general difficult to obtain. Because the problem has high complexity and $3 \cdot K \cdot M$ variables, the solution depends on the distribution of tasks in the same BoT on different sites, the distribution of tasks in different BoTs on the same site, and the allocation of processors and bandwidth. In other words, the change of one variable impacts the values of all other variables. To circumvent this difficulty, we approximate the solution by further formulating this problem as *a sequential game* [34] in which players select a strategy following a certain predefined order and observe the moves of the players who preceded them. Although the optimal solution is not achievable directly from Equation (16), we derive intermediate solutions in a set of *game stages*, based on the following inequality sequence:

$$\begin{aligned}\sum_{k=1}^{K} F_k^{S(1)}\left(\Delta^0,\mathcal{B}^0\right) \geq & \sum_{k=1}^{K} F_k^{S(2)}\left(\Delta^{S(1)},\mathcal{B}^{S(1)}\right) \geq \ldots \\ \geq & \sum_{k=1}^{K} F_k^{S(l)}\left(\Delta^{S(l-1)},\mathcal{B}^{S(l-1)}\right) \\ \geq & \sum_{k=1}^{K} F_k\left(\Delta^*,\mathcal{B}^*\right),\end{aligned} \tag{17}$$

where S denotes the stage of the sequential game, and $S(l)$ the lth game stage. At each stage, the players (managers of BoTs) provide a set of strategies (task distributions) based on the allocation of resources in the last stage, and generate the new allocations by using Equation (9).

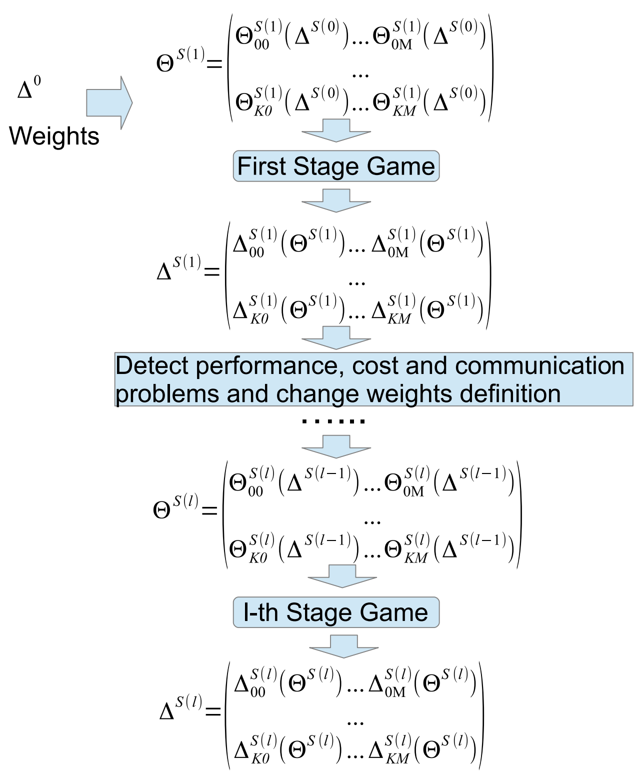

Δ^0
Weights
$\Theta^{S(1)}=\begin{pmatrix}\Theta_{00}^{S(1)}(\Delta^{S(0)})\ldots\Theta_{0M}^{S(1)}(\Delta^{S(0)})\\ \ldots \\ \Theta_{K0}^{S(1)}(\Delta^{S(0)})\ldots\Theta_{KM}^{S(1)}(\Delta^{S(0)})\end{pmatrix}$
First Stage Game
$\Delta^{S(1)}=\begin{pmatrix}\Delta_{00}^{S(1)}(\Theta^{S(1)})\ldots\Delta_{0M}^{S(1)}(\Theta^{S(1)})\\ \ldots \\ \Delta_{K0}^{S(1)}(\Theta^{S(1)})\ldots\Delta_{KM}^{S(1)}(\Theta^{S(1)})\end{pmatrix}$
Detect performance, cost and communication problems and change weights definition
......
$\Theta^{S(l)}=\begin{pmatrix}\Theta_{00}^{S(l)}(\Delta^{S(l-1)})\ldots\Theta_{0M}^{S(l)}(\Delta^{S(l-1)})\\ \ldots \\ \Theta_{K0}^{S(l)}(\Delta^{S(l-1)})\ldots\Theta_{KM}^{S(l)}(\Delta^{S(l-1)})\end{pmatrix}$
l-th Stage Game
$\Delta^{S(l)}=\begin{pmatrix}\Delta_{00}^{S(l)}(\Theta^{S(l)})\ldots\Delta_{0M}^{S(l)}(\Theta^{S(l)})\\ \ldots \\ \Delta_{K0}^{S(l)}(\Theta^{S(l)})\ldots\Delta_{KM}^{S(l)}(\Theta^{S(l)})\end{pmatrix}$

Fig. 3. Sequential game theoretic allocation data flow.

The first step in the game is to initialize the distribution of tasks $\Delta^{S(0)}$ and the allocation of bandwidth $\mathcal{B}^{S(0)}$. Every BoT is allocated an amount of processors based on the processing rate on each site. At the initial stage $S(0)$, every BoT assumes that all processors and bandwidth are available

$$\delta_{ki} = \delta_k \cdot \frac{\frac{m_i}{p_{ki}}}{\sum_{i=1}^{M} \frac{m_i}{p_{ki}}}; \tag{18}$$

$$b_{ki} = \lambda_{x,i} \cdot \frac{d_{ki}}{\sum_{i=1}^{K} d_{ki}}. \tag{19}$$

From Equations (18) and (19), we have the initial task distribution Δ^0 and the bandwidth allocation $\mathcal{B}^0$.

The resource allocation of the lth stage $\Theta^{S(l)}$, where $\Theta = (\theta_{ki})_{K\times M}$ is the *resource allocation matrix*, is calculated based on the task distribution of the last stage $\Delta^{S(l-1)}$. Accordingly, we calculate the task distribution of the lth stage $\Delta^{S(l)}$ based on the resource allocation of l^{th} stage $\Theta^{S(l)}$ using Equations (9) and (14) (see Fig. 3, where these steps fully embody the idea of sequential cooperative game):

$$\Theta^{S(l)} = \Theta\left(\Delta^{S(l-1)}\right); \tag{20}$$

$$\Delta^{S(l)} = \Delta\left(\Theta^{S(l)}\right). \tag{21}$$

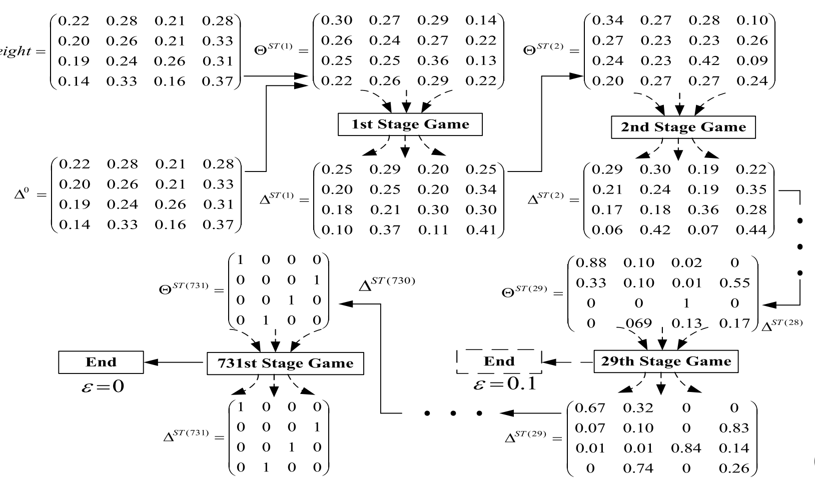
4.3 Game-Multi-Objective Algorithm

In this section, we present an algorithm called *Game-multi-objective* (GMO) which implements the game theoretical scheduling method formalized in the previous section. Our previous work [35] proposed two algorithms for single objective optimization on performance or economic cost. In extending the idea of single objective game theory-based scheduling algorithms to multi-objective cases, one major problem has to be addressed: how to select one or multiple objectives to guide the search towards a whole set of potential solutions? The most common way is to convert the multi-objective problem into a single-objective one by forming a linear combination of the objectives by assigning weights to each criterion and sum them in an aggregated function. The main disadvantage of this technique is that the weights are decided a-priori with no information about the application, infrastructure and problem being solved, which may lead to unfeasible solutions for realistic nonlinear problems. Another classical way is to switch between objectives, each time an objective and weight w_{ki} being chosen for the next competition. We use a hybrid approach of alternating and combining the objectives. GMO first optimizes the performance and communication criterion at the same time while imposing constraints on economic cost. After identifying the range of performance, GMO starts to optimize cost and communication criterion. The difficulty here is on how to establish the order in which the criteria should be optimized, because this can have an effect on the success of the search.

We accompany our presentation by a small example depicted in Fig. 4a, which presents a scenario in which GMO outperforms related heuristics in terms of makepsan. The first ETC matrix presents the expected execution times of four tasks $\{t_0, t_1, t_2, t_3\}$ on four sites $\{s_0, s_1, s_2, s_3\}$. For the sake of clarity, we restrict in this example the size of each site to one processor only. Algorithm 1 depicts the GMO algorithm in pseudocode consisting of three phases.

Phase 1: After acquiring information about tasks and resources (e.g., the ETC matrix in Fig. 4b), we generate an initial distribution of tasks Δ^0 and a weight matrix (see lines 3-11), as shown in Fig. 4a. In this simple case, we only consider the optimization for makespans. In this phase, users are also allowed to set performance constraints or to filter undesired sites by simply setting the weights of the applications for these sites to zero which prevents mapping of any tasks to those sites. To assure that all constraints are satisfied, they can be verified again in the third phase.

Phase 2: Every iteration of the `repeat` loop (lines 12-15) is one game stage, where every stage consists of M *sub-games* (i.e., one per site). In each sub-game, all BoTs compete for resource allocation and those with relatively large weights win the sub-game on one site and obtain more resources in the next stage. These BoTs, however, cannot win everywhere due to the weight definition (i.e., the weight sum of one BoT is 1), therefore, winners of the sub-game on one site must be losers on other sites and vice-versa. This process repeats until no more performance can be gained. The further processing of the algorithm depends on the evaluation result on performance and cost at line 15, where ϵ can be used to control the number of stages and the degree of optimization. The input and output data flow of each game stage has been shown in Fig. 3. More specifically, we apply different weight definitions at line 13 to generate the new resource allocation matrix $\Theta^{S(1)}$. Lines 20-34 describe the adjustment of the weights, which is the change of players'



(a) Game-multi-objective stages.

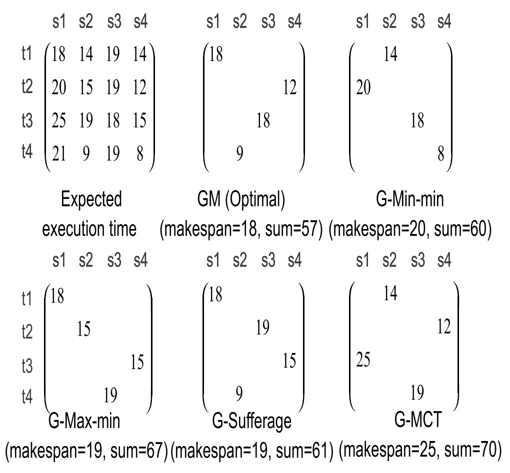

s1 s2 s3 s4
t1
t2
t3
t4
Expected execution time
GM (Optimal) (makespan=18, sum=57)
G-Min-min (makespan=20, sum=60)
G-Max-min (makespan=19, sum=67)
G-Sufferage (makespan=19, sum=61)
G-MCT (makespan=25, sum=70)
(b) Related algorithms schedule comparison.

Fig. 4. GMO makespan scheduling example.

strategies in next game stage. Basically, players' strategies need to be changed when some constraints are not fulfilled, or when makespan or economic cost objectives are deviated from the users' expectation. For instance, when storage constraints cannot be fulfilled on one site, the balanced condition on storage resources would first be violated for the user with the largest storage requirements sr and the player with the largest sr is the first to be motivated to change strategy. It follows that, with global knowledge of all players, a weight change can be trivially computed by ordering the users in decreasing order of sr and then using storage weights to replace the old weights. Based on $\Theta^{S(1)}$, we use Equation (21) at line 14 to generate the first task distribution $\Delta^{S(1)}$. Thereafter, we repeat the iteration until we reach the upper limit of optimization. In addition, we can use ϵ to control the number of stages.

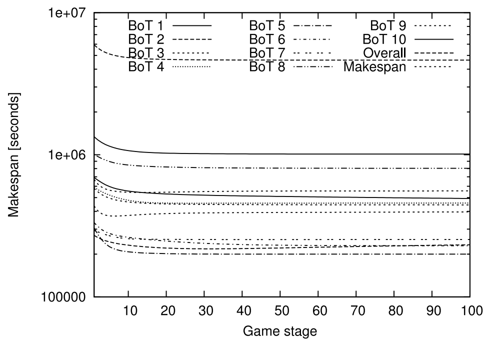
Phase 3: Finally, the earliest completed BoTs are eliminated. To utilize the released resources, we repeat the first two phases to recompute the distribution of the remaining BoTs until all applications complete.

4.4 Convergence and Algorithmic Mechanism Analysis

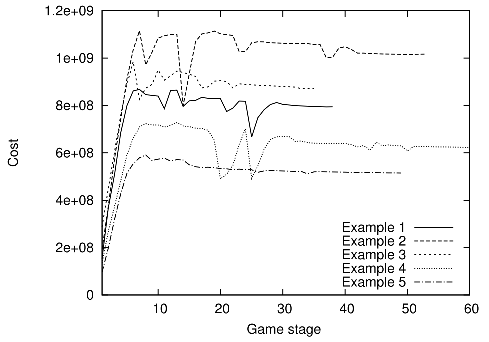
Figs. 5a and 5b plot the convergence of total execution time and cost by GMO over the number of stages. For this experiment, we randomly generated several examples that assign $10^2 \times 10^4$ tasks to $10^2 \times 10^2$ processors. Within about 30 to 40 stages, more than most examples complete their optimization process. Initially, the tasks are distributed based on the processing rate and performance/price ratio of each site. Therefore, the initial cost is always low and performance is always high. This initial cost and performance corresponds to the stage 0. The reason for the fast convergence is that, to some extent, every BoT acts according to its own self-interest to compete for the most efficient resources. After the competitions on the most efficient resources complete, the algorithmic mechanism makes the players move the workloads to less powerful sites. In other words, the mechanism makes players' privately known preferences for each resource be aggregated towards a "social choice". This mechanism ensures that a globally optimized solution can be generated when individual players behave selfishly. Finally, all BoTs reach a balance when no further improvement can be achieved.

Algorithm 1 Game-multi-objective scheduling algorithm.

```
Require: $\mathcal{AS}$: set of applications; $K$: number of BoTs; $M$ = number of sites; $m_i$:
    number of processors on site $s_i (i \in [1, m])$; $(p_{ki})_{K\times M}$: ETC matrix; $\delta_k$:
    number of tasks of BoT $k (k \in [1, K])$; $\epsilon$: optimization threshold;
Require: $bl_i$: bandwidth limit of site $s_i (i \in [1, M])$; $br_k$: bandwidth requirements
    of BoT $\mathcal{T}_k (k \in [1, K])$;
Ensure: $\Delta^{S(l)}$: task distribution matrix; $\Theta^{S(l)}$: resource allocation matrix
 1: $GP \leftarrow \emptyset$ // Initialize the set of game players
 2: repeat
 3:     for all $\mathcal{W} \in \mathcal{AS}$ do // Phase 1: Initialize $\Delta^0$ and the weight of BoTs;
    optionally apply constraints
 4:         for all $\mathcal{T}_k \in \mathcal{W} \wedge \mathcal{T}_k$ not yet scheduled $\wedge$ (pred$(\mathcal{T}_k) = \emptyset \vee (\mathcal{T}_j$
    is completed, $\forall\ \mathcal{T}_j \in$ pred$(\mathcal{T}_k)))$ do// Take the next not scheduled BoT
 5:             $GP \leftarrow GP \cup \mathcal{T}_k$ // Add $\mathcal{T}_k$ to the set of game players
 6:             for all $i \in [1; M]$ do // For every site $s_i$
 7:                 Calculate $pw_{ki}, cw_{ki}, sw_{ki}$ by applying Equations 10,11,13
 8:                 Calculate $\delta_{ki}$ by applying Equation 14 to build $\Delta^0$
 9:             end for
10:         end for
11:     end for
12:     repeat // Phase 2: search the final task distribution and resource allocation
13:         $\Theta^{S(l)} \leftarrow$ MULTIOBJECTIVE-SCHEDULE$(\Theta, \Delta, m, b, \lambda, p)$
14:         Calculate $\Delta^{S(l)} = (\delta_{ki})_{|GP|\times M}$ by applying Equation 21
15:     until $\sum_{k=1}^{|GP|}\left(f_k\left(\Delta^{S(l-1)}\right) - f_k\left(\Delta^{S(l)}\right)\right) \leq \epsilon \wedge$
    $\sum_{k=1}^{K}\left(c_k\left(\Delta^{S(l-1)}\right) - c_k\left(\Delta^{S(l)}\right)\right) \leq \epsilon$
16:     wait for a BoT to complete
17:     $GP \leftarrow GP \setminus \mathcal{T}, \forall \mathcal{T} \in GP \wedge \mathcal{T}$ completed // Phase 3: remove completed
    BoTs , release resources, and start new game by repeating Phases 1 and 2
18: until $\forall\ \mathcal{W} \in \mathcal{AS}$ completed
19:
20: function MULTI-OBJECTIVE-SCHEDULE$(\Theta, \Delta, m, b, \lambda, p)$
21:     for all $\mathcal{T}_k \in \mathcal{W}$ do
22:         Calculate all $bw_k$ by applying Equation 12
23:     end for
24:     for all $\mathcal{T}_k \in \mathcal{W}$ do
25:         for all $s_i \in [1, M]$ do
26:             if $\sum_{k=1}^{K} b_{ki} \cdot \theta_{ki} > \lambda_{x,i} \vee \sum_{k=1}^{K} sr_{ki} \cdot \theta_{ki} > sl_i$ then
    // Bandwidth or storage requirements are not fulfilled
27:                 Recalculate $\theta'_{ki}$ with Equation 9 and weights from 12 or 13
28:             end if
29:             Recalculate $\theta''_{ki}$ with Equation 9 with weights from 10 and 11
30:         end for
31:     end for
32:     Evaluate gains of different $\Theta$ with different weights
33:     return $\Theta$ with best gain
34: end function
```



(a) Makespan.



(b) Cost.

Fig. 5. Optimization convergence.

4.5 Comparison with Related Algorithms

Fig. 4a presents a small example with GMO outperforming six greedy algorithms modified based on well-known list scheduling heuristics (see Fig. 4b) of two classes.

$\mathcal{O}(n \cdot N)$ complexity algorithms:[36], [37] iterate once over the list of tasks and schedules them as follows: *greedy minimum execution time (G-MET)* assigns each task to the machine delivering its fastest execution, or in case of a draw, to the one with the lowest cost; *greedy minimum completion time (G-MCT)* assigns each task to the machine delivering its minimum completion time or cost; *greedy opportunistic load balancing (G-OLB)* minimizes the global load imbalance without considering the task execution time or cost.

$\mathcal{O}(n \cdot N^2)$ complexity algorithms: [38], [39] iterate over all tasks before selecting one for scheduling according to the following criteria: *greedy min-min (G-Min-min)* selects the task with the shortest minimum completion time or cost; *greedy max-min (G-max-min)* selects the task with the largest minimum completion time or cost; *greedy sufferage (G-sufferage)* selects the task with the largest difference between the fastest and second fastest minimum completion time or cost. The selected task is assigned to the machine that delivers its earliest completion time or cost.

To further compare the quality of the GMO solutions against the absolute optimum, we consider a small-sized problem consisting of two sites with three and two homogeneous processors each, and two BoTs containing five tasks each. We use in our simulation both consistent matrices and inconsistent matrices, outlined in Table 2:*consistent* represents that, if a site A executes a task faster and more expensive than site B, then A executes all tasks faster and more expensive than B, while *inconsistent* characterizes the situation when the site A might be faster and more expensive than B for some tasks and slower and less expensive for some others [33]. The solution delivered by GMO is optimal in both cases, while G-Min-min and G-Max-min deliver equal results for inconsistent matrices and worse results for the consistent cases. However, as we will demonstrate in the following sections, G-Min-min and G-Max-min require significantly longer scheduling times, especially in the case of large problems.

TABLE 2
GMO, G-Min-Min, and G-Max-Min Comparison for Small-Sized ETC Matrices

Processor	*Inconsistent*					*Consistent*				
	$\mathcal{T}_1$	$\mathcal{T}_2$	*GMO*	*G-Min-min*	*G-Max-min*	$\mathcal{T}_1$	$\mathcal{T}_2$	*GMO*	*G-Min-min*	*G-Max-min*
$S_1 : P_1$	15	8	8, 9	8, 15	15, 8	15	9	9, 9	9, 15	15, 9
$S_1 : P_2$	15	8	8, 9	8, 15	15, 8	15	9	9, 9	9, 15	15, 9
$S_1 : P_3$	15	8	8, 15	8, 15	15, 8	10	8	10, 10	8, 10	10, 8
$S_2 : P_1$	10	9	10, 10	9, 10	10, 9	10	8	10, 10	8, 10	10, 8
$S_2 : P_2$	10	9	10, 10	9, 10	10, 9	10	8	10, 9	8, 10	10, 8
Makespan			23	23	23			19	24	24

4.6 Complexity

The time complexity of GMO is $O(l \cdot K \cdot M)$ corresponding to the four algorithm nested loops (lines 2, 3, 4, and 6 in the first phase of Algorithm 1), where l is the number of stages of the sequential game, K is the total number of BoTs in all input workflows $\mathcal{W} \in \mathcal{AS}$, and M is the number of sites (the work performed within each stage is constant). The second phase does not have an impact on the complexity, since the work performed in each stage is constant and depends linearly on the number of sites and on a small number of game players only. Most importantly, the complexity is independent of the total number of tasks, which is a huge advantage against other related approaches.

For empirical evidence, Table 3 a displays the number of stages and the execution time of GMO for different computing environments and application sizes on a Dual Core Opteron 880 2.4 GHz processors with 16 GB of memory. We can observe that GMO scales well since the number of game stages does not increase as fast as the number of processors and tasks. Even for 10^6 tasks and 10^4 processors, the algorithm only needs 593 stages and 0.36 seconds to complete the optimization, in contrast to the algorithms from the Min-min family which need more than one hour. Based on this analysis, we use for GMO a maximum number of $1,000$ iterations that proves to be sufficient for convergence.

Table 3 shows more empirical results that compare GMO against Greedy algorithms when scheduling 10^5 tasks to 10^3 processors. The execution time of GMO is less than 0.3 seconds which is significantly faster than the other algorithms. The exception is MET which has asymptotic complexity of $O(m + N)$, where m is the number of processors and N is the number of tasks $(N \gg K)$, and executes for less than one second. However, the results of G-MET have serious problems because it serializes most tasks on the fastest site. G-OLB and G-MCT have asymptotic complexity of $O(m \cdot N)$, but their results are much worse than of GMO. G-Min-min, G-Max-min, and G-Sufferage have asymptotic complexity of $O(m \cdot N^2)$ and execute for an average of

TABLE 3
Algorithm Complexity and Time Analysis

(a) GMO stages and execution times.

Sites $\times$ *No. proc.*	*BoTs* $\times$ *Tasks/BoT*	*No. Stages*	*GMO [seconds]*	*G-Min-Min [seconds]*
10×10	10×10	310	0.002	0.015
10×10	10×100	334	0.002	0.022
10×10^2	$10^2 \times 10^3$	476	0.023	3.109
10×10^2	$10^2 \times 10^4$	484	0.025	29.512
$10^2 \times 10^2$	$10^2 \times 10^3$	597	0.362	485.597
$10^2 \times 10^2$	$10^2 \times 10^4$	593	0.362	> 3600
$10^2 \times 10^2$	$10^3 \times 10^3$	632	11.065	> 3600
$10^2 \times 10^2$	$10^3 \times 10^4$	627	11.856	> 3600

(b) 10^5 tasks and 10^3 processors.

Algorithm	*Time complexity*	*Exec. time [seconds]*
GMO	$O(l \cdot K \cdot M)$	< 0.3
G-MET	$O(m + N)$	< 1
G-OLB	$O(m \cdot N)$	$2 - 3$
G-MCT	$O(m \cdot N)$	$2 - 3$
G-Min-min	$O\left(m \cdot N^2\right)$	250
G-Max-min	$O\left(m \cdot N^2\right)$	250
G-Sufferage	$O\left(m \cdot N^2\right)$	250
GA-based	poor	$\gg 250$
A^*	exponential	$\gg 250$

$200 - 300$ seconds. Heterogeneous earliest finish time (HEFT) [24] algorithm, originally designed for workflows, sorts the tasks in the prioritization phase based on their upward rank and, therefore, degrades to G-Min-min by scheduling the small tasks first. GA-based [33], [40] or A* [41] solutions scale poorly as the number of tasks and processors increases and might need several hours to generate comparable solutions in terms of makespan or cost. For this reason, we consider GA as not applicable to our problem and do not study them any further (although they can decrease the makespans of G-Min-min by 5 to 10 percent, according to related work [33]). Other algorithms are similar to the ones discussed above or are not applicable to our problem, for example the work queue [42] suited for homogeneous clusters.

5 Experimental Results

In this section, we first show experimental results of two real applications on hybrid clouds to explain the effectiveness of our algorithm. To ensure the completeness of our experiments, we also evaluated and compared different algorithms over a complex simulated system and large amount of tasks based on different machine and task heterogeneity. We performed all measurements on a machine with Intel i7-2640M 2.8 GHz processors and 4 GB of RAM.

5.1 Real-World Experiments

In the following we report on the evaluation of the GMO algorithm for the WIEN2k and AstroGrid scientific applications introduced in Section 3.1 and executed in the EC2, GCE, and four parallel machines (see Table 4). To quantify whether users or applications are receiving a fair share of system resources, we use the *Jain's fairness index* [43]:

$$\frac{\left(\sum_{k=1}^{K} T_k\right)^2}{K \cdot \sum_{k=1}^{K} T_k^2}, \tag{22}$$

where K is the number of applications and T_i is the execution time of application $\mathcal{W}_i$. The fairness ranges from zero indicating the worst fairness, to one indicating the best.

To quantify the utilization of the cloud sites, we need to measure if resources are allocated to the BoTs that need them the most through a system-level efficiency equation:

$$E = \sum_{k=1}^{K} E_k \cdot EW_k, \tag{23}$$

where E_k is the scheduling efficiency of application $\mathcal{W}_k$:

$$E_k = \frac{\max_i(T_{ki}) + \min_i(T_{ki}) - 2 \cdot \operatorname{avg}(T_{ki})}{\max_i(T_{ki}) - \min_i(T_{ki})}, \tag{24}$$

where $\max_i(T_{ki})$ is the makespan of $\mathcal{W}_k$ on the slowest site, $\min_i(T_{ki})$ is its makespan on the fastest site, and EW_k is the weight of scheduling efficiency of $\mathcal{W}_k$, which is the optimization degree of a BoT compared with overall optimization degree:

$$EW_k = \frac{\left(\max_i(T_{ki}) - \min_i(T_{ki})\right) \cdot \theta_k}{\sum_{k=1}^{K}\left(\max_i(T_{ki}) - \min_i(T_{ki})\right) \cdot \theta_k}. \tag{25}$$

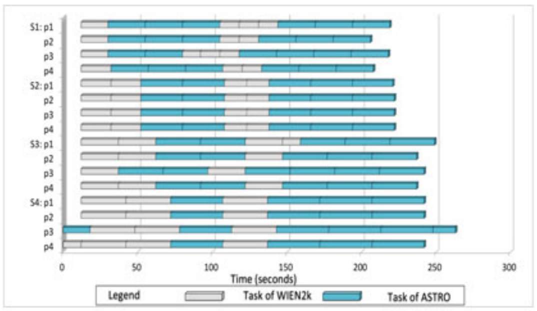
The system-level efficiency ranges from -1 indicating the worst efficiency, to 1 indicating the best.

We first evaluated the performance of GMO by comparing the makespan and the fairness against G-Min-min, which outperforms the other heuristics for scheduling these two applications. As shown in Fig. 6a, G-Min-min gives a fixed makespan of 258, a system-level efficiency of 0.249 and a fairness of 0.9466, while GMO improves is to 230 seconds and an almost perfect fairness above 0.99 (see Fig. 6b).

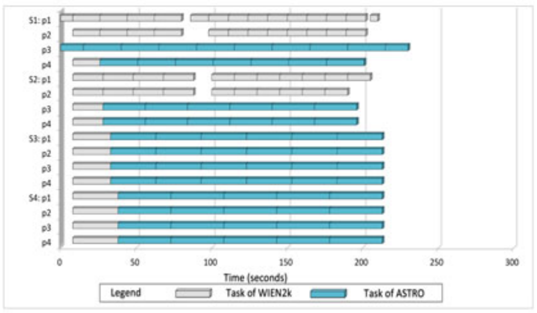
Finally, we can intuitively observe that the tasks are highly interleaved in the Gantt chart produced by G-Min-min, which makes their completion time hard to predict. On the contrary, GMO yields an execution plan in which tasks belonging to the same BoT are grouped in contiguous slots on the same sites which makes their execution more predictable. This implies that GMO is more robust to keep the makespan of each BoT under control and deliver the desired makespan with small statistical variation.

TABLE 4
The Hybrid Cloud Testbed

Site	*Name*	*Architecture*	*Processor*	*#cores* (#Total)	*Clock* [GHz]	*Resource Manager*	*Hourly price*	*Location*
S_1	GCE	Cluster	EC2 Compute Unit	$4(-)$	1.6	Fork	1	U.S.
S_2	EC2	Cluster	GCE Unit	$4(-)$	1.6	Fork	1	Singapore
S_3	Aurora	SMP	Xeon X7542	$4(2624)$	2.67	LSF	0	ACRC
S_4	fuji	Cluster	Xeon 5570	$4(3888)$	2.93	LSF	0	ACRC



(a) G-Min-min.



(b) GMO.

Fig. 6. Multi-objective scheduling for two real applications.

5.2 Simulation Experiments

For the completeness of experiments, we evaluated through simulation the performance of GMO against related algorithms for different ETC matrices generated according to different degrees of resource and task heterogeneity parameters selected from a uniform distribution in the specified ranges. Table 5 presents the details of the simulated environment. High resource heterogeneity in the range of $[1; 100]$ causes a significant difference in task execution times and cost across sites, while high task heterogeneity in the range of $[1; 1,000]$ indicates that the expected execution times of different tasks vary largely. We assume that the number of tasks in each BoT is randomly generated based on a uniform distribution in the range of $[10,000; 20,000]$, and that the number of processors on each site varies in the range of $[64; 128]$. The BoTs are organized in workflows by having 10 percent dependence probability between each pair of BoTs and excluding cycles. We chose not to simulate larger number of applications and sites for two reasons: (1) the complexity of the algorithms from the G-Min-min family that need several hours to complete making our entire simulation difficult; and (2) enlarging the simulation size will only increase the advantage of our game theoretic algorithm over the related heuristics. We used again consistent and inconsistent ETC simulation models and evaluated the algorithms in four scenarios: high task and high resource heterogeneity (HiHi), high task and low resource heterogeneity (HiLo), low task and high resource heterogeneity (LoHi), and low task and low resource heterogeneity (LoLo).

Fig. 7a shows the execution times of the algorithms which do not vary for consistent and inconsistent matrices, ranked from the fastest to the slowest as follows: GMO, G-MET, G-OLB, G-MCT, G-Min-min, G-Max-min, and G-Sufferage. We discuss in the next paragraphs the performance

TABLE 5
Simulation Environment

(a) Consistent environment.

Configuration	*No. of processors*	*No. of clusters*	*No. of tasks*	*No. of BoTs*	*Task heterogeneity*	*Resource heterogeneity*
HiHi	900	10	157118	10	[1; 1000]	[1; 100]
HiLo	989	10	147871	10	[1; 1000]	[1; 10]
LoHi	900	10	149731	10	[1; 10]	[1; 100]
LoLo	1048	10	168208	10	[1; 10]	[1; 10]

(b) Inconsistent environment.

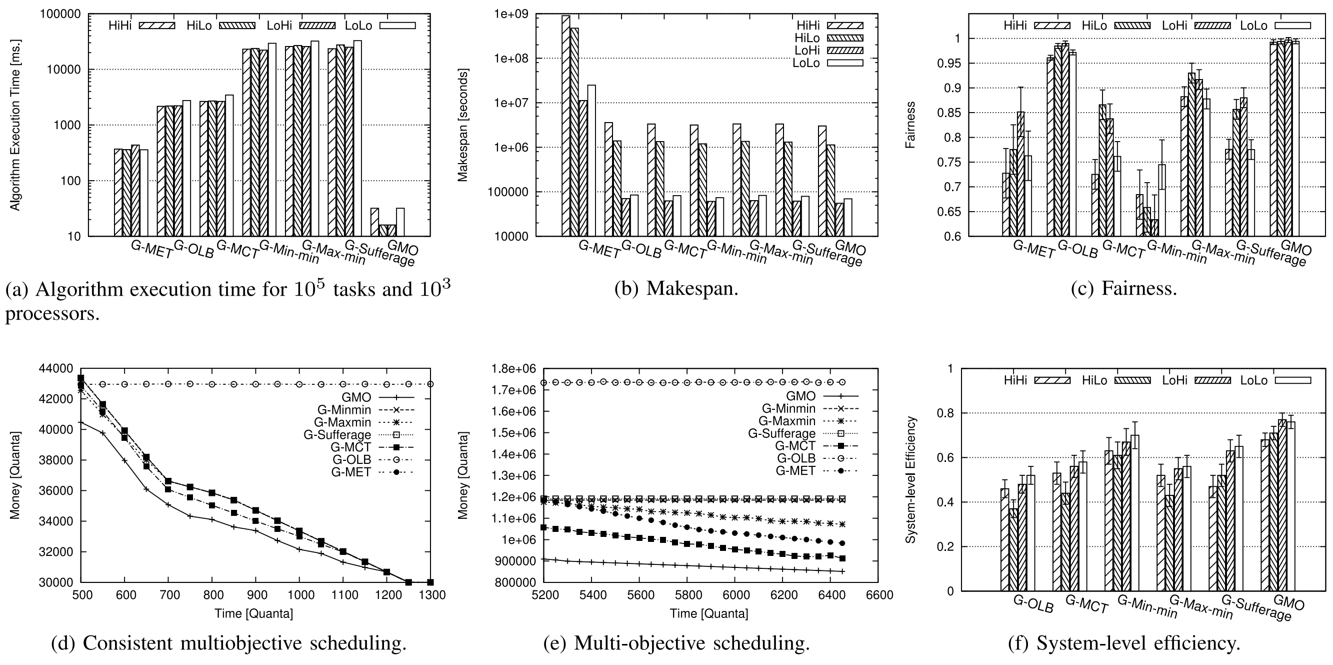
Configuration	*No. of processors*	*No. of clusters*	*No. of tasks*	*No. of BoTs*	*Task heterogeneity*	*Resource heterogeneity*
HiHi	982	10	131298	10	[1; 1000]	[1; 100]
HiLo	955	10	153395	10	[1; 1000]	[1; 10]
LoHi	955	10	173418	10	[1; 10]	[1; 100]
LoLo	1007	10	150156	10	[1; 10]	[1; 10]

of all algorithms in the order from the worst to the best in the consistent and inconsistent scenarios.

5.2.1 *Consistent Heterogeneity*

Table 5a presents the input of four consistent heterogeneous scenarios and Figs. 7b and 7c the corresponding simulation results considering the makespan and the fairness objectives. G-MET always gives the worst results because it maps all tasks to the fastest machine. G-OLB usually performs the second worst because it selects resources without considering the task execution time. G-Max-min gives poor results because it only fits the situation when a small number of tasks are much larger than the others, which is never encountered in our simulated environment generated using uniform distributions. In addition, G-Max-min offers no fairness to smaller tasks, hence, it performs worse than most algorithms. G-MCT performs quite well for high machine heterogeneity because it has a higher likelihood for selecting the fastest machine, especially for large tasks, and poorly for low machine heterogeneity because it only considers the completion time and ignores the task execution time. G-Sufferage performs quite similar to G-MCT for high machine heterogeneity and $5-10$ percent better for low machine heterogeneity scenarios because it makes more intelligent decisions by considering the task execution time. G-Min-min gives the second best results in each case due to the uniform distribution of task execution times, but looses fairness because of handling the smallest tasks first.

In Figs. 7d and 7e, we show the skyline produced by various algorithms. G-OLB gives the worst results because of no cooperation between different BoTs, the resources being selected based on their availability without considering task execution time and public clouds' prices. G-Min-min only handles the smallest tasks and ignores larger ones in the beginning. However, the smallest tasks are not the ones with the best performance/price ratio and, therefore, G-Min-min cannot perform well. G-Sufferage performs quite similar to Min-min due to similar reasons. In contrast to G-Min-min, G-Max-min gives better results because it schedules larger tasks in the beginning, which makes the large tasks gain more on the site with best performance/price ratio. G-MCT gives the second best results because it unconsciously selects tasks with average sizes, with a statistically larger likelihood of having the best performance/price


(a) Algorithm execution time for 10^5 tasks and 10^3 processors.

(b) Makespan.

(c) Fairness.

(d) Consistent multiobjective scheduling.

(e) Multi-objective scheduling.

(f) System-level efficiency.

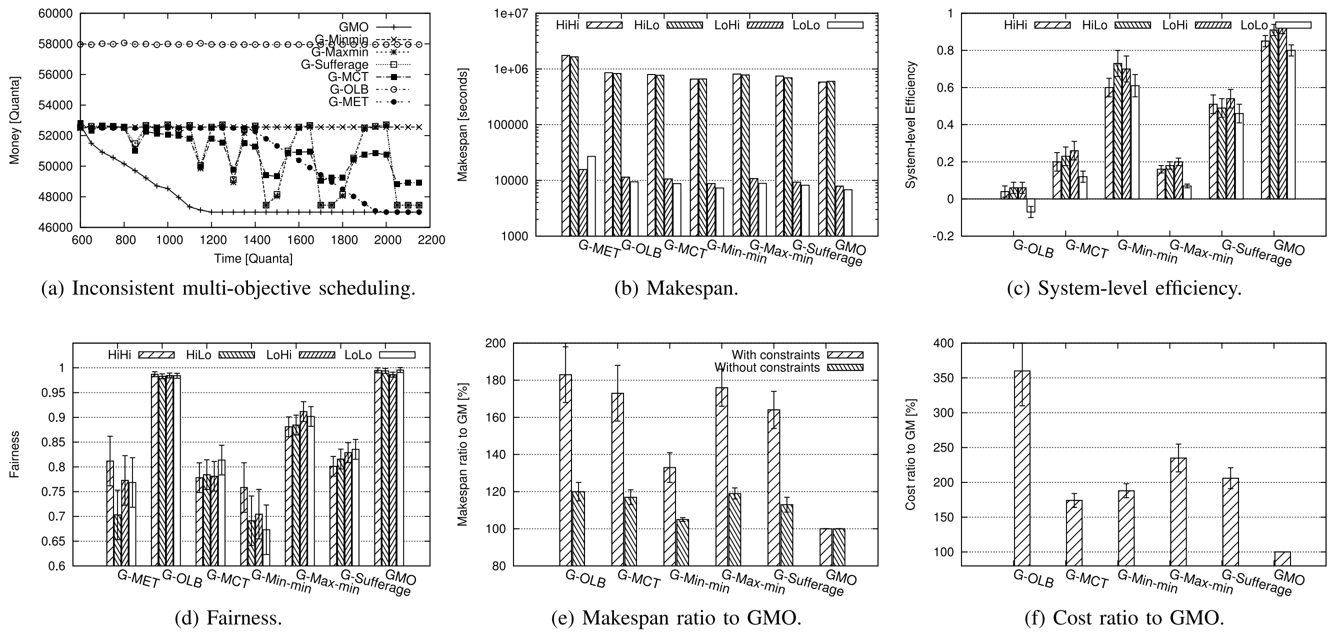
Fig. 7. GMO scheduling results for consistent scenarios.

ratio. GMO gives the best schedules in most cases, however it may not deliver the best results for workflows with complex dependencies between BoTs for which the scheduling problem cannot be formulated as a typical and solvable game.

GMO provides the best performance in all four scenarios because it takes the best global decisions. It performs about 10 percent better than G-Min-min for the LoHi scenario, and 5 percent better for the other three. We can see that when fairness is ensured, the efficiency is also improved. We can further observe in Fig. 7c that GMO always achieved almost perfect fairness of 0.99 in average.

5.2.2 Inconsistent Heterogeneity

Table 5 b presents the input of the four inconsistent scenarios investigated and Fig. 8 depicts the results. In all four cases, G-MET gives the worst results because it maps most tasks to the fastest sites. G-MET performs better than G-OLB when the fastest sites for different BoTs are evenly distributed. However, Fig. 8a shows that G-MET is more stable than other algorithms because its scheduling strategy makes tasks concentrate on several faster sites, which allows further optimization. G-OLB, G-Max-min, G-Min-min, G-MCT, and G-Sufferage perform worse for inconsistent than for consistent scenarios due to their design that cannot


(a) Inconsistent multi-objective scheduling.

(b) Makespan.

(c) System-level efficiency.

(d) Fairness.

(e) Makespan ratio to GMO.

(f) Cost ratio to GMO.

Fig. 8. GMO scheduling results for inconsistent matrices.

TABLE 6
Communication and Storage-Aware Simulation Environment

No. of processors	*No. of Clusters*	*No. of Tasks*	*BoTs*	*Task heterogeneity*	*Resource heterogeneity*	*Bandwidth heterogeneity*	*Storage heterogeneity*
1035	10	148690	10	$[1; 1000]$	$[1; 100]$	$[1; 1000]$	$[1; 100]$

effectively handle the high heterogeneity of machines in inconsistent environments. GMO still provides the best mapping for inconsistent cases for the same reason as in consistent scenarios.

5.3 Communication and Storage-Awareness

Table 6 presents the simulated computing environment, where the real values are randomly generated from a uniform distribution in the specified ranges. Since consistent matrix and inconsistent matrix generate similar results, we only present the results for inconsistent matrices that we consider more authentic for modeling a hybrid cloud environment.

As expected GMO also gives the best results as shown in Fig. 8e. The relative order of the algorithms from the best to worst is: GMO, G-Min-min, G-Sufferage, G-MCT, G-Max-min, G-OLB, and G-MET. When there are no bandwidth or storage constraints on the sites, GMO performs about $5-10$ percent better than G-Min-min, and achieves less costs than other algorithms by at least 28 percent (see Fig. 8e). When there are constraints, GMO improves the performance of multiple workflows by at least 33 percent, and decrease costs by at least 74 percent (see Fig. 8f). GMO provides the best performance because makes the best global decisions in terms of simultaneous performance and bandwidth optimization, while other heuristics can only find a compromise between the two objectives when bandwidth requirements cannot be fulfilled, resulting in a potential waste of computing power. In terms of cost, Fig. 8f illustrates that all algorithms need approximately twice the cost of GMO.

6 Conclusion

With increasing focus on large-scale applications on hybrid clouds, it is important for a workflow management service to efficiently and effectively schedule and dynamically steer execution of applications. In this paper, we analyzed the main bottlenecks of a class of applications with bags-of-tasks characterized by a large number of homogeneous tasks, and presented a communication- and storage-aware multiobjective scheduling solution based on a sequential cooperative game algorithm for four important metrics: makespan, cost, storage resource and network bandwidth. Experimental results based on simulation, as well as real applications in the hybrid cloud computing environment demonstrate that our approach delivers better solutions in terms of makespan, cost, system-level efficiency and fairness with less algorithm execution times than other greedy approaches such as G-Min-min, G-Max-min, or G-Sufferage, which proves that we have successfully overcome the drawbacks of slow convergence and random constructions of other metaheuristics. Furthermore, we observed that the larger scale the experiments are, the better results we achieve. For example, considering larger hybrid infrastructures up to 2,048 processors to schedule the applications will only increase the gap between our game theoretic algorithms and the other classical heuristics, greedy algorithms needing in this case hours to complete.

Our game theory-based scheduling algorithm possesses great potential for improvement for large-scale applications in hybrid clouds. We plan to investigate how our algorithm can adapt to other metrics such as memory, security, resource availability, or multiple virtual organizations. We further intend to extend our method with priorities in two ways: (1) search for new appropriate weights for BoTs that guarantee them a faster or slower completion; (2) assign deadlines to BoTs, partition them into sub-BoTs according to assigned deadlines, and applying our algorithm on each sub-BoT.Finally, our approach of expressing task distribution using job processing rates (see Equation (14)) can be easily extended for online scheduling that handles dynamic job arrivals.

Acknowledgments

This work was partially supported by Singapore A*STAR Thematic Strategic Research Programme, grant number 1021580036, and by Austrian Science Fund project TRP 237-N23, Standortagentur Tirol project RainCloud.

References

[1] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields, *Scientific Workflows for Grids*. Springer, 2007.

[2] O.H. Ibarra and C.E. Kim, “Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors,” *J. ACM*, vol. 24, no. 2, pp. 280-289, 1977.

[3] C.L. Smith, “The Large Hadron Collider,” *Scientific Am.*, vol. 283, no. 1, pp. 70-77, July 2000.

[4] R. Prodan, S. Ostermann, and K. Plankensteiner, “Performance Analysis of Grid Applications in the Askalon Environment,” *Proc. 10th ACM/IEEE Int'l Conf. Grid Computing*, pp. 97-104, 2009.

[5] G. Mehta, E. Deelman, J.A. Knowles, T. Chen, Y. Wang, J. Vöckler, S. Buyske, and T. Matise, “Enabling Data and Compute Intensive Workflows in Bioinformatics,” *Proc. Parallel Processing Workshops (Euro-Par '11)*, pp. 23-32, 2012.

[6] A. Doğan and F. Özgüner, “Biobjective Scheduling Algorithms for Execution Time—Reliability Trade-Off in Heterogeneous Computing Systems,” *Computer J.*, vol. 48, no. 3, pp. 300-314, May 2005.

[7] S.G. Kumar, R. Buyya, and H.J. Siegel, “Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-Off Management,” *Proc. 32nd Australasian Conf. CS*, pp. 151-160, 2009.

[8] I. Assayad, A. Girault, and H. Kalla, “A Bi-Criteria Scheduling Heuristic for Distributed Embedded Systems Under Reliability and Real-Time Constraints,” *Proc. Int'l Conf. Dependable Systems and Networks*, pp. 347-356, June 2004.

[9] M. Hakem and F. Butelle, “Reliability and Scheduling on Systems Subject to Failures,” *Proc. Int'l Conf. Parallel Processing*, pp. 38-46, Sept. 2007.

[10] H.M. Fard, R. Prodan, J.J.D. Barrionuevo, and T. Fahringer, “A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments,” *Proc. 12th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid '12)*, 2012.

[11] J. Yu and R. Buyya, “A Budget Constrained Scheduling of Workflow Applications on Utility Grids Using Genetic Algorithms,” *Proc. First Workshop on Workflows in Support of Large-Scale Science*, June 2006.

[12] S. Venugopal and R. Buyya, “A Deadline and Budget Constrained Scheduling Algorithm for E-Science Applications on Data Grids,” *Proc. Sixth Int'l Conf. Algorithms and Architectures for Parallel Processing*, 2005.

[13] S. Pandey, L. Wu, S.M. Guru, and R. Buyya, “A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments,” *Proc. 24th IEEE Int'l Conf. Advanced Information Networking and Applications*, pp. 400-407, 2010.

[14] C. Kim and H. Kameda, "An Algorithm for Optimal Load Balancing in Distributed Computer Systems," *IEEE Trans. Computers*, vol. 41, no. 3, pp. 381-384, Mar. 1992.

[15] S. Penmatsa and A.T. Chronopoulos, "Cooperative Load Balancing for a Network of Heterogeneous Computers," *Proc. 21st IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '06)*, Apr. 2006.

[16] J. Bredin et al., "A Game-Theoretic Formulation of Multi-Agent Resource Allocation," *Proc. Fourth Int'l Conf. Autonomous Agents*, pp. 349-356, citeseer.ist.psu.edu/article/bredin00gametheoretic.html, 2000.

[17] Y. Kwok, S. Song, and K. Hwang, "Selfish Grid Computing: Game-Theoretic Modeling and Nas Performance Results," *Proc. Fifth IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid '05)*, citeseer.ist.psu.edu/kwok05selfish.html, 2005.

[18] P. Ghosh, K. Basu, and S.K. Das, "A Game Theory-Based Pricing Strategy to Support Single/Multiclass Job Allocation Schemes for Bandwidth-Constrained Distributed Computing Systems," *IEEE Trans. Parallel Distributed System*, vol. 18, no. 3, pp. 289-306, Mar. 2007.

[19] L. Young, S. McGough, S. Newhouse, and J. Darlington, "Scheduling Architecture and Algorithms within the Iceni Grid Middleware," technical report, UK e-Science All Hands Meeting, EPSRC, 2003.

[20] M. Malawski, G. Juvey, E. Deelman, and J. Nabrzyski, "Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in Iaas Clouds," *Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis*, 2012.

[21] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J.L. González-García, T. Röblitz, and J.M. Ramírez-Alcaraz, "Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid," *J. Grid Computing*, vol. 10, no. 2, pp. 325-346, June 2012.

[22] L.F. Bittencourt and E.R.M. Madeira, "Towards the Scheduling of Multiple Workflows on Computational Grids," *J. Grid Computing*, vol. 8, no. 3, pp. 419-441, Sept. 2010.

[23] H. Zhao and R. Sakellariou, "Scheduling Multiple DAGs onto Heterogeneous Systems," *Proc. Int'l Parallel and Distributed Processing Symp.*, 2006.

[24] H.T. et al., "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel Distributed System*, vol. 13, no. 3, pp. 260-274, 2002.

[25] L. Zhu, Z. Sun, W. Guo, Y. Jin, W. Sun, and W. Hu, "Dynamic Multi Dag Scheduling Algorithm for Optical Grid Environment," *Network Architectures, Management, and Applications V*, 2007.

[26] L.F. Bittencourt, E.R.M. Madeira, and N.L.S.D. Fonseca, "Scheduling in Hybrid Clouds," *IEEE Comm. Magazine*, vol. 50, no. 9, pp. 42-47, Sept. 2012.

[27] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky Computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43-51, Sept./Oct. 2009.

[28] R. Buyya, R. Ranjan, and R.N. Calheiros, "Intercloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," *Proc. 10th Int'l Conf. Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2010.

[29] S. Nair et al., "Towards Secure Cloud Bursting, Brokerage and Aggregation," *Proc. IEEE Eight European Conf. Web Services (ECOWS '10)*, 2010.

[30] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache, "Cloud Service Delivery across Multiple Cloud Platforms," *Proc. IEEE Int'l Conf. Services Computing (SCC)*, 2011.

[31] K. Schwarz, P. Blaha, and G.K.H. Madsen, "Electronic Structure Calculations of Solids Using the Wien2K Package for Material Sciences," *Proc. Europhysics Conf. Computational Physics*, vol. 147, pp. 71-76, 2002.

[32] W. Kapferer, W. Domainko, S. Schindler, E.V. Kampen, S. Kimeswenger, M. Mair, T. Kronberger, and D. Breitschwerdt, "Metal Enrichment and Energetics of Galactic Winds in Galaxy Clusters," *Advances in Space Research*, vol. 36, pp. 682-684, 2005.

[33] T.D. Barun et al., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, citeseer.ist.psu.edu/braun01comparison.html, 2001.

[34] R.B. Myerson, *Game Theory: Analysis of Conflict*. Harvard Univ. Press, http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20&path=AS IN/0674341163, Sep. 1997.

[35] R. Duan, R. Prodan, and T. Fahringer, "Performance and Cost Optimization for Multiple Large-Scale Grid Workflow Applications," *Proc. ACM/IEEE Conf. Supercomputing (SC '07)*, 2007.

[36] R. Armstrong et al., "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-Time Predictions," *Proc. IEEE Seventh Heterogeneous Computing Workshop (HCW '98)*, pp. 79-87, 1998.

[37] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," *Proc. Heterogeneous Computing Workshop*, 1999.

[38] H. Casanova et al., "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. Ninth Heterogeneous Computing Workshop (HCW)*, pp. 349-363, citeseer.ist.psu.edu/casanova00heuristics.html, May. 2000.

[39] M. Maheswaran et al., "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *J. Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107-131, citeseer.ist.psu.edu/article/maheswaran99dynamic.html, 1999.

[40] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992.

[41] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. second ed. ed., Prentice-Hall, 2003.

[42] R.L. Graham, "Bounds for Certain Multiprocessor Anomalies," *Bell System Technical J.*, vol. 45, pp. 1563-1581, 1966.

[43] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems,"DEC Research Report TR-301 http://www.citebase.org/abstract?id=oai:arXiv.org:cs/9809099, 1998.

Rubing Duan received the PhD degree from the Institute of Computer Science, University of Innsbruck, Austria, in 2008. He is currently a scientist at the Institute of High Performance Computing, Singapore. His research interests include distributed software architectures, performance analysis and optimization, and scheduling for parallel and cloud computing. He participated in several national and Singapore projects. He is the author of more than 20 journal and conference papers.

Radu Prodan received the PhD degree from the Vienna University of Technology, Austria, in 2004. He is currently an associate professor at the Institute of Computer Science, University of Innsbruck, Austria. His research interests include parallel and distributed systems comprises programming methods, compiler technology, performance analysis and scheduling. He is the author of more than 70 journal and conference papers and one book. He received the IEEE best paper award.

Xiaorong Li received the PhD degree from the Department of Electrical and Computing Engineering of National University of Singapore in 2006, and the BEng degree from the Department Telecommunication Engineering of Beijing University of Posts and Telecommunication, China, in 1998. She is currently a senior scientist and the manager of Distributed Computing Group in A*STAR Institute of High Performance Computing, Singapore. Her research interests include parallel and distributed computing systems, workflow management, quality management in computer Cloud/Grid and multimedia systems. She is a member of ACM and IEEE Computer Society.