Cooperative Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds

Rubing Duan†, Radu Prodan*
†Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore
*Institute of Computer Science, University of Innsbruck, Austria
Email: radu@dps.uibk.ac.at, duanr@ihpc.a-star.edu.sg

***Abstract*—We address the problem of scheduling a class of large-scale applications inspired from real-world on hybrid Clouds, characterized by a large number of homogeneous and concurrent tasks that are the main sources of bottlenecks but open great potential for optimization. We formulate the scheduling problem 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.**

I. Introduction

Distributed computing systems such as clouds and grids have evolved towards a worldwide infrastructure providing dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. To program such a large and scalable infrastructure, loosely coupled-based coordination models of legacy software components such as *bags-of-tasks (BoT)* and *workflows* have emerged as successful programming paradigms in the scientific community.

One of the most challenging NP-complete problems 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 expected to produce $5-6$ petabytes of data per year, which must be accommodated and efficiently handled through appropriate scheduling bandwidth and storage constraints.

In this paper, we address these issues by proposing a communication and storage-aware multi-objective scheduling scheme for an important class of applications characterized by large sets of independent and homogeneous tasks, interconnected through control flow and data flow dependencies, as follows: (1) *multi-objective scheduling* minimizes the expected execution time and economic cost of applications based on a sequential cooperative game theoretic algorithm, and (2) *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.

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

II. Related Work

Several researchers in performance-oriented distributed computing have focused on system-level load balancing [5], [17] or resource allocation [6], [14], aiming to introduce economic and game theoretic aspects into computational questions. Penmatsa et al. [17] formulated the scheduling problem as a cooperative game where Grid sites try to minimize the expected response time of tasks, while Kwok et al. [14] investigated the impact of selfish behaviors of individual machine by taking into account the non-cooperativeness of machines. Ghosh et al. [8] proposed a strategy that formulates an incomplete information, alternating-offers bargaining

 DOI: https://doi.org/10.1109/CloudCom.2014.58

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. ICENI [19] addressed the scheduling problem using a game theoretic 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.

The work in [15] 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 [9], 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. Their results showed that the proposed strategies outperform well-known single directed acyclic graph (DAG) scheduling algorithms. The authors in [2] proposed four heuristics for scheduling multiple workflows using a clustering technique. They concluded that 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 [20]. 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. The work in [21] addressed online scheduling of multiple DAGs in an optical Grid environment based on two aggregation strategies scheduled using HEFT: first come first serve and service on time. In contrast to all these works targeting a static set of input workflows, our work targets workflow-interconnected BoT.

The research domain of hybrid cloud computing is relatively young. Bittencourt et al. [1] 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 [13] for building a virtual site distributed on several Clouds. In [3], 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 [16], [10].

III. System and Application 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.

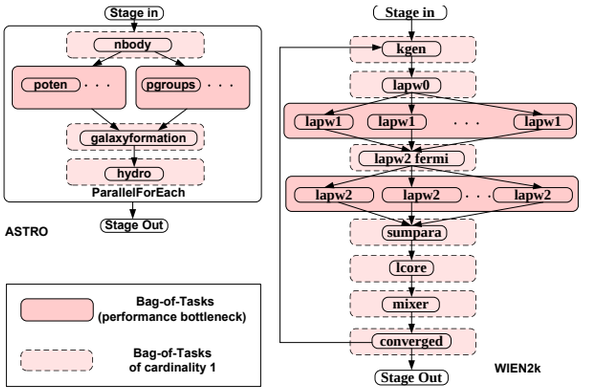


Fig. 1: Real-world application examples.

A. Application Model

We focus on large-scale workflows characterized by a high number (thousands to millions) of homogeneous parallel (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 $\mathcal{DD} = \left(\mathcal{T}_s <^d \mathcal{T}_d | \{\mathcal{T}_s, \mathcal{T}_{d,}\} \subset \mathcal{BS}\right)$ 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 *bag-of-tasks (BoT)* $\mathcal{T}_k$ as a homogeneous set of parallel atomic sequential *tasks* $\mathcal{T}_k = \bigcup_{j=1}^{K_k} \mathtt{t}_{kj}, k \in [1..K]$ 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, Figure 1 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 [18] 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. The `lapw1` and `lapw2` BoTs can be solved in parallel by a fixed number of homogeneous tasks called k-points (see Figure 1). A final task named converged applied on several output files tests whether the problem convergence criterion is fulfilled.

ASTRO [12] 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.

B. Hybrid cloud model

A hybrid cloud computer consists of minimum of one private and one public cloud. 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.

IV. 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 III-B. 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 III-A (see Definition 3.1) and then propose and experimentally validate a game theory-based algorithm to efficiently address it.

A. 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:

$$\underset{x \in S}{Minimize} F(x) = (f(x), c(x)), h_i(x) \leq \lambda_{x,i}, g_i(x) \leq sl_i, i \in [1; M], \quad (1)$$

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 $\mathtt{s}_i$, and sl_i is the storage limit on site $\mathtt{s}_i$.

We assume the availability of an *expected time to compute (ETC)* [7] matrix which delivers the *expected execution time* p_{ki} of tasks in each BoT $k \in [1..K]$ on each site $\mathtt{s}_i, i \in [1..M]$. We define the expected execution time p_{ki} based on the computation time (pc_{ki}) and communication time (po_{ki}):

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

where po_{ki} can be expressed as:

$$po_{ki} = \frac{d_{ki}}{b_{ki}}, \quad (3)$$

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

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

where θ_{ki} is the number of processors allocated to BoT $\mathcal{T}_k$ on site $\mathtt{s}_i$. 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.

B. 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 $\mathtt{s}_i$. 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:

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

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

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

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

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 $\mathtt{s}_i$, φ_i is the price of site $\mathtt{s}_i$, $\mathcal{B}$ 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 $\mathtt{s}_i$ (which embodies the fairness of sharing resources), defined as the product between the number of processors m_i on $\mathtt{s}_i$ and the ratio between the weighted aggregated execution times of BoT $\mathcal{T}_k$ on $\mathtt{s}_i$ and the aggregated execution time of all BoTs on $\mathtt{s}_i$:

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

where w_{ki} is the weight of site $\mathtt{s}_i$ 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 $\mathtt{s}_i$ 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 $\mathtt{s}_i$ 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 $\mathtt{s}_i$ 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 $f_k\left(\Delta^0, \mathcal{B}^0\right)$, 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 $\mathcal{B}^0$ 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).

To circumvent the high complexity of this problem, we approximate the solution by further formulating this problem as *a sequential game* 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 directly achievable, we derive intermediate solutions in a set of *game stages*, based on the following inequality sequence:

$$\begin{aligned}\sum_{k=1}^{K} F_k^{S(1)}\left(\Delta^0, \mathcal{B}^0\right) \geq \sum_{k=1}^{K} F_k^{S(2)}\left(\Delta^{S(1)}, \mathcal{B}^{S(1)}\right) \geq \dots \\ \geq \sum_{k=1}^{K} F_k^{S(l)}\left(\Delta^{S(l-1)}, \mathcal{B}^{S(l-1)}\right) \geq \sum_{k=1}^{K} F_k\left(\Delta^*, \mathcal{B}^*\right),\end{aligned} \tag{16}$$

where S denotes the stage of the sequential game, and $S(l)$ the l^{th} 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.

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:

$$\delta_{ki} = \delta_k \cdot \frac{\frac{m_i}{p_{ki}}}{\sum_{i=1}^{M} \frac{m_i}{p_{ki}}}; \tag{17}$$

$$b_{ki} = \lambda_{x,i} \cdot \frac{d_{ki}}{\sum_{i=1}^{K} d_{ki}}. \tag{18}$$

From Equations 17 and 18, we have the initial task distribution Δ^0 and the bandwidth allocation $\mathcal{B}^0$.

The resource allocation of the l^{th} 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 l^{th} stage $\Delta^{S(l)}$ based on the resource allocation of l^{th} stage $\Theta^{S(l)}$ using Equation 9 and Equation 14:

$$\Theta^{S(l)} = \Theta\left(\Delta^{S(l-1)}\right); \tag{19}$$

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

C. 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 [4] 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? 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.

Algorithm 1 depicts the GMO algorithm in pseudocode consisting of three phases.

Phase 1: After acquiring information about tasks and resources (ETC matrix), we generate an initial distribution of tasks Δ^0 and a weight matrix (see lines 3–11). 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.

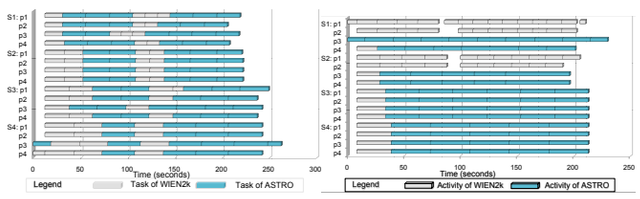
Phase 2: Every iteration of the `repeat` loop (lines 12–15) is one game stage, where every stage consists of M *sub-games* (i.e. one per site). In each sub-game, all BoTs compete for resource allocation and those with relatively large weights win the sub-game on one site and obtain more resources in 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. More specifically, we apply different weight definitions at line 13 to generate the new resource allocation matrix $\Theta^{S(1)}$. Lines 20 – 33 describe the adjustment of the weights, which is the change of players' 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.

Algorithm 1 Game-multiobjective 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; δ_k: number of tasks of BoT $k (k \in [1..K])$; ϵ: 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 Δ^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 $\wedge$ $(\text{pred}(\mathcal{T}_k) = \emptyset \vee (\mathcal{T}_j$ is completed, $\forall\ \mathcal{T}_j \in \text{pred}(\mathcal{T}_k)))$ **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 δ_{ki} by applying Equation 14 to build Δ^0
9: **end for**
10: **end for**
11: **end for**
12: **repeat** // ***Phase 2:*** *search final distribution of tasks 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 20
15: **until** $\sum_{k=1}^{|GP|} \left(f_k\left(\Delta^{S(l-1)}\right) - f_k\left(\Delta^{S(l)}\right)\right) \leq \epsilon$ AND $\sum_{k=1}^{K} \left(c_k\left(\Delta^{S(l-1)}\right) - c_k\left(\Delta^{S(l)}\right)\right) \leq \epsilon$
16: **wait** for a BoT to complete
17: $GP \leftarrow GP - \mathcal{T}, \forall \mathcal{T} \in GP \wedge \mathcal{T}$ 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** MULTIOBJECTIVE-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** $s_i \in Grid$ **do**
26: **if** $\sum_{k=1}^{K} b_{ki} \cdot \theta_{ki} > \lambda_{x,i}$ OR $\sum_{k=1}^{K} sr_{ki} \cdot \theta_{ki} > sl_i$ **then** // *Bandwidth or storage requirements are not fulfilled*
27: Recalculate θ'_{ki} by applying Equation 9 with different weights 12 or 13
28: **end if**
29: Recalculate θ''_{ki} by applying Equation 9 with different weights Equations. 10 and 11
30: **end for**
31: **end for**
32: Evaluate different Θ gains with different weights and return Θ with best gain
33: **end function**

TABLE I: 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. 2: Multi-objective scheduling for two real applications.

Based on $\Theta^{S(1)}$, we use Equation 20 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.

V. 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.

A. 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 III-A and executed in the EC2, GCE, and four parallel machines (see Table I). To quantify whether users or applications are receiving a fair share of system resources, we use the *Jain's fairness index* [11]:

$$\frac{\left(\sum_{k=1}^{K} T_k\right)^2}{K \cdot \sum_{k=1}^{K} T_k^2}, \tag{21}$$

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:

$$E = \sum_{k=1}^{K} E_k \cdot EW_k, \tag{22}$$

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

$$E_k = \frac{\max\limits_i (T_{ki}) + \min\limits_i (T_{ki}) - 2 \cdot \text{avg}\,(T_{ki})}{\max\limits_i (T_{ki}) - \min\limits_i (T_{ki})}, \tag{23}$$

where $\max\limits_i(T_{ki})$ is the makespan of $\mathcal{W}_k$ on the slowest site, $\min\limits_i(T_{ki})$ 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:

$$EW_k = \frac{(\max\limits_i(T_{ki}) - \min\limits_i(T_{ki})) \cdot \theta_k}{\sum_{k=1}^{K}(\max\limits_i(T_{ki}) - \min\limits_i(T_{ki})) \cdot \theta_k}. \tag{24}$$

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

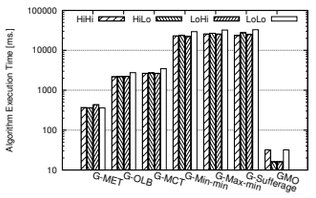
We first evaluated the performance of GMO by comparing the makespan and the fairness against G-Min-min, which outperforms the other heuristics for scheduling these two applications. As shown in Figure 2a, G-Min-min gives a fixed makespan of 258, a system-level efficiency of 0.249 and a fairness of 0.9466, while GMO improves is to 230 seconds and an almost perfect fairness above 0.99 (see Figure 2b).

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.

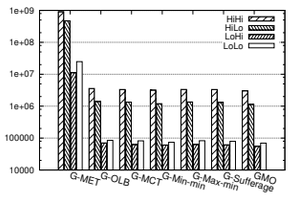
B. Simulation experiments

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 II 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 $[1; 1000]$ 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 $[10000; 20000]$, 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% 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 evaluated the algorithms in four consistent scenarios displayed in Table II: 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).

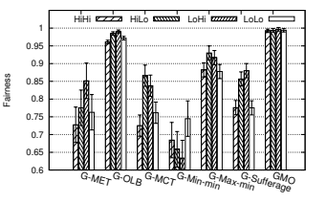
Figure 3a shows the execution times of the algorithms ranked from the fastest to the slowest as follows: GMO,



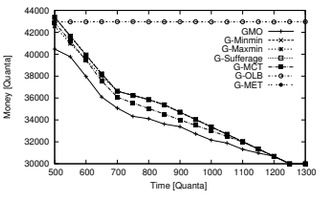
(a) Algorithm execution time for 10^5 tasks and 10^3 processors.



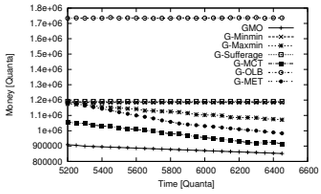
(b) Makespan.



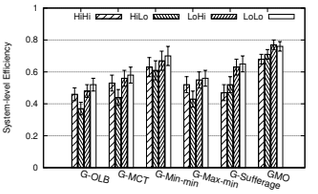
(c) Fairness.



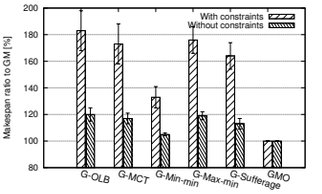
(d) Consistent multiobjective scheduling.



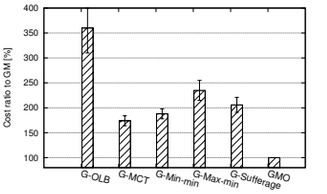
(e) Multi-objective scheduling.



(f) System-level efficiency.



(g) Makespan ratio to GMO.



(h) Cost ratio to GMO.

Fig. 3: GMO scheduling results for consistent scenarios.

TABLE II: Simulation 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	$[1; 10]$	$[1; 10]$

TABLE III: Communication- and storage-aware scheduling simulation environment.

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

G-MET, G-OLB, G-MCT, G-Min-min, G-Max-min, and G-Sufferage. Figures 3b and 3c display the 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 $5\% - 10\%$ 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 Figure 3d and Figure 3e, 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 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% better than G-Min-min for the LoHi scenario, and 5% better for the other three. We can see that when fairness is ensured, the efficiency is also improved. We can further observe in Figure 3c that GMO always achieved almost perfect fairness of 0.99 in average.

C. Communication- and storage-aware Experiments

Table III 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 Figure 3g. 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 $5 - 10\%$ better than G-Min-min, and achieves less costs than other algorithms by at least 28% (see Figure 3g). When there are constraints, GMO improves the performance of multiple workflows by at least 33%, and decrease costs by at least 74% (see Fig. 3h). 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, Figure 3h illustrates that all algorithms need approximately twice the cost of GMO.

VI. 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 2048 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.

References

[1] Luiz F. Bittencourt, Edmundo R. M. Madeira, and Nelson L. S. Da Fonseca. Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9):42–47, 2012.

[2] Luiz Fernando Bittencourt and Edmundo R. M. Madeira. Towards the scheduling of multiple workflows on computational grids. *Journal of Grid Computing*, 8(3):419–441, September 2010.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *10th International Conference on Algorithms and Architectures for Parallel Processing*, Busan, Korea, 2010.

[4] Rubing Duan, Radu Prodan, and Thomas Fahringer. Performance and cost optimization for multiple large-scale grid workflow applications. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing (SC '07)*, New York, NY, USA, 2007. ACM Press.

[5] C.Kim et al. An algorithm for optimal load balancing in distributed computer systems. *IEEE Transactions on Computers*, 41(3):381–384, 1992.

[6] Jonathan Bredin et al. A game-theoretic formulation of multi-agent resource allocation. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 349–356, Barcelona, Catalonia, Spain, 2000. ACM Press.

[7] Tracy D. Braun et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[8] Preetam Ghosh, Kalyan Basu, and Sajal K. Das. A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. *IEEE Trans. Parallel Distrib. Syst.*, 18(3):289–306, 2007.

[9] Adán Hirales-Carbajal, Andrei Tchernykh, Ramin Yahyapour, José Luis González-García, Thomas Röblitz, and Juan Manuel Ramírez-Alcaraz. Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing*, 10(2):325–346, June 2012.

[10] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache. Cloud service delivery across multiple cloud platforms. In *2011 IEEE International Conference on Services Computing (SCC)*, 2011.

[11] 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, 1998.

[12] W. Kapferer, W. Domainko, S. Schindler, E. Van Kampen, S. Kimeswenger, M. Mair, T. Kronberger, and D. Breitschwerdt. Metal enrichment and energetics of galactic winds in galaxy clusters. *Advances in Space Research*, 36:682, 2005.

[13] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky computing. *IEEE Internet Computing*, pages 43–51, 2009.

[14] Y. Kwok, S. Song, and K. Hwang. Selfish grid computing: Game-theoretic modeling and nas performance results. In *Proceedings of the 5th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2005)*, 2005.

[15] Maciej Malawski, Gideon Juvey, Ewa Deelman, and Jarek Nabrzyski. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *International Conference on High Performance Computing, Networking, Storage and Analysis*, number 22. IEEE Computer Sociery, 2012.

[16] S.K. Nair and S. Porwal et al. Towards secure cloud bursting, brokerage and aggregation. In *8th IEEE European Conference on Web Services (ECOWS 2010)*, 2010.

[17] S. Penmatsa and A. T. Chronopoulos. Cooperative load balancing for a network of heterogeneous computers. In *21st IEEE Intl. Parallel and Distributed Processing Symposium*. IEEE Computer Sociery, 2006.

[18] K. Schwarz, P. Blaha, and G. K. H. Madsen. Electronic structure calculations of solids using the wien2k package for material sciences. *Computer Physics Communications*, 147(71), 2002.

[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] Henan Zhao and Rizos Sakellariou. Scheduling multiple DAGs onto heterogeneous systems. In *International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2006.

[21] Liying Zhu, Zhenyu Sun, Wei Guo, Yaohui Jin, Weiqiang Sun, and Weisheng Hu. Dynamic multi DAG scheduling algorithm for optical grid environment. In *Network Architectures, Management, and Applications V*, volume 6784. International Society for Optical Engineering, 2007.