Int. J. Appl. Math. Comput. Sci., 2015, Vol. 25, No. 4, 737–751
DOI: 10.1515/amcs-2015-0053

amcs

DECENTRALIZED JOB SCHEDULING IN THE CLOUD BASED ON A SPATIALLY GENERALIZED *PRISONER'S DILEMMA* GAME

JAKUB GĄSIOR [a,*], FRANCISZEK SEREDYŃSKI [a]

[a]Department of Mathematics and Natural Sciences
Cardinal Stefan Wyszyński University, Wóycickiego 1/3, 01-938 Warsaw, Poland
e-mail: j.gasior@uksw.edu.pl

We present in this paper a novel distributed solution to a security-aware job scheduling problem in cloud computing infrastructures. We assume that the assignment of the available resources is governed exclusively by the specialized brokers assigned to individual users submitting their jobs to the system. The goal of this scheme is allocating a limited quantity of resources to a specific number of jobs minimizing their execution failure probability and total completion time. Our approach is based on the Pareto dominance relationship and implemented at an individual user level. To select the best scheduling strategies from the resulting Pareto frontiers and construct a global scheduling solution, we developed a decision-making mechanism based on the game-theoretic model of *Spatial Prisoner's Dilemma*, realized by selfish agents operating in the two-dimensional cellular automata space. Their behavior is conditioned by the objectives of the various entities involved in the scheduling process and driven towards a Nash equilibrium solution by the employed social welfare criteria. The performance of the scheduler applied is verified by a number of numerical experiments. The related results show the effectiveness and scalability of the scheme in the presence of a large number of jobs and resources involved in the scheduling process.

Keywords: job scheduling, multiobjective optimization, genetic algorithm, *Prisoner's Dilemma*, cellular automata.

1. Introduction

The increasing demand for computational power has led to the introduction of a new type of heterogeneous, distributed computing platform, where customers do not own any part of the infrastructure. *Cloud computing* (CC) systems reduce the issue of requesting computational resources to a set of services, which can be rent from specialized organizations and provided in the form of *virtual machines* (VMs) to the clients. A scheduler has to decide how to allocate these VMs in order to guarantee a reasonable level of security-aware computation while maintaining efficient resource management of the CC environment. Obviously, the conflict between achieving good performance, in terms of the job completion time, and providing high security-assurance introduces new challenges in CC scheduling.

The main contribution of this work is a four-tier scheduling framework which effectively handles such a multi-criteria job allocation problem, taking into account not only the job completion time but also the security constraints inherent in the CC system. Our solution employs independent brokering agents assigned to individual cloud users and undertaking scheduling decisions based only on locally available information. Each broker tries to allocate a batch of jobs from the associated user in a pure selfish way, without considering actions of the other brokers and assuming the total availability of system resources.

A method employed for the scheduling purposes is *multiobjective genetic algorithm* (MOGA) optimization resulting in a Pareto-based evaluation, necessary to find the best scheduling strategies accommodating conflicting optimization objectives. In order to make a dynamic choice among the Pareto set of the proposed solutions according to the real-time needs of the user, the resulting Pareto frontier is divided into sections determining various *User Profiles*, i.e., specific job allocation strategies defining basic goals and requirements of the client.

Afterwards, the proposed *User Profiles* are used as an input in a non-cooperative scheduling game based on

*Corresponding author

a spatially generalized *Prisoner's Dilemma* (SPD) model (Nowak and May, 1992) in order to convert this local optimization problem into the one of searching for a system-wide equilibrium by a set of independent brokers. The whole process is realized in the two-dimensional *cellular automata* (CA) space, where individual brokers are mapped onto a regular square lattice. The main issues that are addressed here include: (a) incorporating the global goal of the system into the local interests of all brokers participating in the scheduling game, and (b) formulation of the local interaction rules allowing the achievement of those interests.

The remainder of this paper is organized as follows. In Section 2, we present the works related to security-aware scheduling in grid and cloud computing systems. In Section 3, we describe the proposed cloud system model. Section 4 defines the basics of MOGA optimization and the process of determining various *User Profiles*, while Section 5 reports the proposed agent-based game-theoretic scheduling scheme. The experimental evaluation of the proposed model is given in Section 6. We end the paper in Section 7 with some conclusions and indications for future work.

2. State of the art

Recently, great interest of researchers in the cloud and grid computing domains has been focused on secure scheduling, which aims to achieve an efficient assignment of tasks to trustful resources. The conflict between achieving good performance and high level of security introduces new challenges in the resource allocation domain. This problem was studied by Lin *et al.* (2004), who proposed a distributed security framework enabling access control to grid resources. Based on the overall results of access control techniques, they proposed an extension of the authentication and authorization features. On the other hand, in the work of Brandic *et al.* (2006), the security aspect was addressed as the *quality of service* (QoS) requirement defined by the grid users at runtime. The authors proposed the *location affinity* model, in which the user for security reasons may express the location affinity regarding computing resources, where certain workflows of tasks may be executed. More recently, a number of *remote data auditing* (RDA) techniques for outsourced data in the cloud was studied by Sookhak *et al.* (2015). The authors presented the taxonomy of the distributed storage auditing process including parameters such as security patterns, objective functions, auditing mode, update mode, and dynamic data structure.

The integration of security mechanisms with scheduling algorithms can be perceived as one of the most important issues in cloud scheduling. Due to the NP-hardness of the job scheduling problem, finding exact solutions to solve large-scale task scheduling problems in dynamic environments is rarely feasible. Therefore, approximation methods providing a near optimal solution are some of the most promising approaches. Heuristics and metaheuristics have shown to be particularly useful for solving a wide variety of such combinatorial and multiobjective optimization problems. For example, the issue of a multiobjective optimization was tackled by Tziritas *et al.* (2013) to solve the problem of VM placement to jointly optimize two objective functions: the total energy spent and the total network overhead. The authors proposed two methodologies for solving the aforementioned problem: one optimizing the above objective functions separately, reaching a single solution; another considering the two optimization targets and defining a set of non-dominated solutions.

Similarly, Lee and Zomaya (2012) made efforts to reduce idle power draw by putting resources into a form of sleep/power-saving mode. Two proposed energy-conscious task consolidation heuristics assigned tasks to resources on which energy consumption was explicitly or implicitly minimized without the performance degradation of these tasks. Another solution was presented by Hwang and Kesselman (2003), who proposed a failure detection service communicating with a failure handling routine as a mechanism providing the fault-tolerance method in a grid environment. It allowed both detection of potential failures as well as handling the user's security requirements without the need to constantly update the scheduling policy at local computational nodes. Insecure conditions in on-line job scheduling caused by software vulnerabilities were also analyzed by Wu and Sun (2010) by considering the heterogeneity of the fault-tolerant mechanisms employed in security-assured job scheduling.

Several works presented game theoretic models to solve job scheduling problems using the concept of a *Nash equilibrium* (NE) (Christodoulou *et al.*, 2007). The convergence time to such equilibria for several selfish scheduling problem variants was considered by Even-Dar *et al.* (2007). The authors analyzed here a variety of load balancing models, including identical, restricted, related and unrelated machines showing crucial dependence on the notion of weights assigned to individual jobs. According to a more economic-based approach, An *et al.* (2007) presented a market-based proportional resource allocation mechanism for multi-agent system investigating interactions among selfish, rational, and autonomous players in resource allocation games, each with incomplete information about other entities, and each seeking to maximize its expected utility by introducing the so-called *deal optimization* mechanism.

Similar approaches have also been recently applied to resource allocation schemes in grid and cloud architectures. Kolodziej and Xhafa (2011) proposed and evaluated four genetic-based metaheuristics as a

non-cooperative game of grid users in order to address their security requirements. In the same scenario, Khan and Ahmad (2006) performed a comparison among game-theoretic resource allocation schemes based on different design rationales: *non-cooperative*, *semi-cooperative* and *cooperative* built around a hierarchical grid infrastructure where machines are abstracted into larger computing centers labeled *federations*, each responsible for managing their own resources independently.

Li *et al.* (2009) developed a utility-driven solution for optimal resource allocation in grid systems driven by a user-centric scheme capable of outperforming deadline constraint optimization in terms of the time of job execution. Game-theoretic concepts have also been applied to the CC paradigm, for example, by Londoño *et al.* (2009), who presented the concept of collocation games. The authors proposed a number of simplified, practically motivated variants of collocation game models for which they established convergence to the NE point as well as the *price of anarchy* (PoA) bounds. Finally, Palmieri *et al.* (2013) analyzed a selfish scheduling scheme for federated cloud organizations based on independent and competing agents. The agents' behavior was conditioned by marginal costs, to force a kind of implicit coordination between the conflicting objectives of the various entities involved in the job allocation process.

3. Cloud model

In this section we formally define basic elements of the model and provide the corresponding notation, its characteristics and the type of jobs to be scheduled.

3.1. System and user model. The system model is an extension of the architecture introduced by Tchernykh *et al.* (2010) and consists of a set of geographically distributed cloud nodes $M_1, M_2, \ldots, M_m$ connected to each other via a wide area network. Each node M_i is described by a parameter m_i, which denotes the number of identical processors P_i and its computational power s_i, characterized by a number of operations per unit of time it is capable of performing. Figure 1(a) depicts an example set of parallel machines in the CC system.

Individual users $(U_1, U_2, \ldots, U_n)$ submit to the system batches of parallel jobs for execution. Users are expected to pay appropriate fees to the cloud provider dependent on the QoS requested. Job J_k^j is the j-th job produced (and owned) by user U_k. J_k stands for the set of all jobs produced by user U_k, while $n_k = |J_k|$ is the number of such jobs. Each job has varied parameters defined as the quadruple $\langle r_k^j, \text{size}_k^j, t_k^j, d_k^j \rangle$, specifying its release date $r_k^j = 0$; its size $1 \leq size_k^j \leq m_m$, which is referred to as its processor requirements or *degree of parallelism*; its execution requirements t_k^j defined by a number of operations and deadline d_k^j.

Rigid jobs require a fixed number of processors for parallel execution: this number is not changed until the job is completed. We assume that job J_k^j can only run on machine M_i if $\text{size}_k^j \leq m_i$ holds; that is, we do not allow multi-site execution and co-allocation of processors from different machines. We assume a space sharing scheduling approach, therefore a parallel job J_k^j is executed on exactly size_k^j disjoint processors without preemptions. Let $p_k^{i,j} = t_k^j / s_i$ define job J_k^j's execution time on machine M_i. Further, $W_k^{i,j} = p_k^{i,j} \times \text{size}_k^j$ denotes the work of job J_k^j, also called its *area* or *resource consumption*. Similarly, the total work of a given job set Z is equal to $W_Z = \sum_{J_k^j \in Z} W_k$. Figure 1(b) shows an example of the multi-threaded job model.

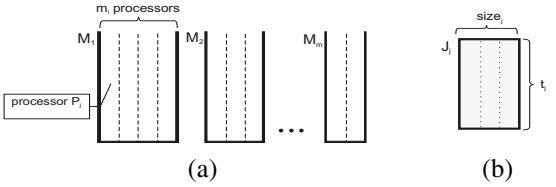


Fig. 1. Example of a cloud computing system: set of parallel machines (a), multi-threaded job model (b).

In this problem, some assumptions must be respected: the scheduler is clairvoyant and working in off-line mode. Jobs are scheduled independently, and there is no communication between them. All jobs and resources are available from time zero.

3.2. Security and pricing model. We consider a security-driven scheduling scheme to address the reliability issues in the CC environment. Such systems may often display potential vulnerabilities caused by software errors, gaps in data transitions or even malicious intent by external agents. While assigning jobs to cloud sites, the scheduler should consider such security threats and question trustworthiness of remote computer platforms. Unfortunately, there are no effective methods to assess the trust level of resources or to specify the security demand parameters for the user's jobs. Therefore, we apply a modified version of the approach presented by Song *et al.* (2006) to match a job's security requirements with security index defined for each cloud site.

During job submission, users define a *Security Demand* (SD) for each job dependent on data sensitivity, execution environment, access control or required level of authentication. On the other hand, the defense capability of a resource can be attributed to the available intrusion detection mechanisms or its response capacity. This capability is modeled by a *Security Level* (SL) factor,

evaluating the risk existing in the allocation of a submitted job to a specific machine and defined as a result of a four-step fuzzy-logic process (Song *et al.*, 2006).

The SD is a real fraction in the range $[0, 1]$ with 0 representing the lowest and 1 the highest security requirement. The SL is in the same range with 0 for the most risky resource site and 1 for a risk-free or fully trusted site. Specifically, we define a job execution model as a function of the difference between the job security demand and site trust. The probability of a *successful job completion* regarding the allocation of job J_k^j with a specific SD_j value to the machine M_i with a security level value SL_i is

$$P_{\text{Success}}^{i,j} = \begin{cases} 1, & SD_j \leq SL_i, \\ e^{-(SD_j - SL_i)}, & SD_j > SL_i. \end{cases} \quad (1)$$

Meeting the security assurance condition ($SD_j \leq SL_i$) for a given job–machine pair guarantees successful execution of that particular job. Such a scheduling will be further called *Secure Job Allocation*. On the other hand, successful execution of a job assigned to machine without meeting this condition ($SD_j > SL_i$) will be dependent on the calculated probability and further referred to as *Risky Job Allocation*.

Depending on the QoS requested, a specific pricing function defines the fee the provider charges to the customers. For simplicity we assume that *Risky Job Allocation* costs $1 \frac{Unit(s)}{Second}$, while *Secure Job Allocation* costs $5 \frac{Unit(s)}{Second}$ in cloud renting fees. *Units* represent stipulated currency used by the brokers to make conscious financial decisions regarding their scheduling choice.

3.3. Problem formulation. In the following, we present the mathematical formalism of our problem and the objective functions used for evaluating a candidate solution (scheduling). Let us denote S_k as user U_k's schedule. The completion time of jobs on machine M_i in schedule S_k is denoted by $C_i(S_k)$. Two different objectives are considered in this work:

- The minimization of the *Maximum Completion Time* or *Makespan*, $C_{\max}^k$, which means the time of completion of the last user U_k's job. We consider minimization of the time $C_i(S_k)$ on each machine M_i over the system in such a way that the *Makespan* is defined as $C_{\max}^k = \max_i\{C_i(S_k)\}$.

- The maximization of the *Security Assurance Level*, $\overline{P_{\text{Success}}^k}$, defined as an average successful job completion probability of each user U_k's job allocation in schedule S_k, that is, $\overline{P_{\text{Success}}^k} = \text{avg}_{(i,j)\in S_k}\{P_{\text{Success}}^{i,j}\}$.

The purpose of scheduling is to distribute user U_k's jobs among the available machines and schedule them to minimize the *Makespan*, $C_{\max}^k$, and maximize the *Security Assurance Level*, $\overline{P_{\text{Success}}^k}$. Therefore, the *multiobjective optimization problem* (MOP) considered in this work can be formulated as follows:

$$\text{Minimize}\left(C_{\max}^k, 1 - \overline{P_{\text{Success}}^k}\right). \quad (2)$$

We assume that there is no centralized control and the assignment of the resources available within the cloud is governed exclusively by the brokers assigned to individual users submitting their jobs to the cloud. To develop a truly distributed multiobjective scheduling algorithm, we propose a four-stage procedure (Fig. 2). At the first stage, a batch of jobs J_k submitted by user U_k is assigned to broker B_k, responsible for allocation of user U_k's jobs in the system. Afterwards, a Pareto frontier is calculated for each broker B_k and a batch of jobs J_k submitted by user U_k under the assumption that all cloud resources belong exclusively to broker B_k using the *NSGA-II* algorithm (Deb *et al.*, 2000).

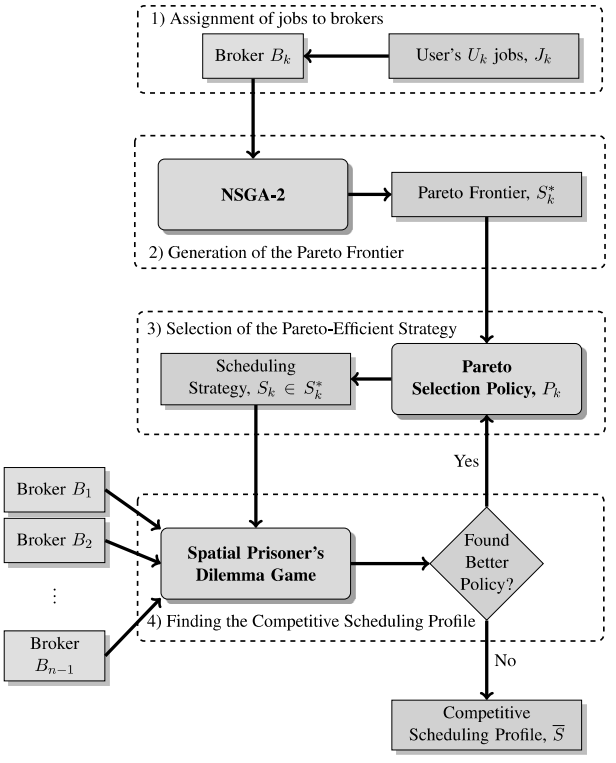


Fig. 2. Flowchart representing the distributed scheduling algorithm's steps from broker B_k's perspective.

At the third stage, some specific job allocation solutions (*Scheduling Strategies*) from the Pareto frontier are selected according to the Pareto *Selection Policies*

characterizing the user's preferences, and these solutions will be subsequently used by broker B_k representing user U_k's interests. At the final stage, all brokers employ previously selected *Scheduling Strategies* in an iterated, non-cooperative SPD game (Nowak and May, 1992) realized in a two-dimensional CA space, where they try to find scheduling strategies providing a competitive scheduling solution (NE point) in their selfish attempts to obtain cloud resources. These stages will be further explained in the subsequent sections.

4. Multiobjective scheduling framework

This section provides a brief overview of the *multiobjective genetic algorithm* (MOGA) and the Pareto *Selection Policies* applied in our study.

4.1. Multiobjective genetic algorithm. *Genetic algorithms* (GAs) are meta-heuristics mimicking the process of natural selection by applying a set of genetic operators on the population of candidate solutions. In our job scheduling model, each solution (chromosome) represents a schedule allocating a batch of user U_k's jobs on a group of machines $M_1, M_2, \ldots, M_m$ with the assumption of their total availability. We employ an encoding scheme where each gene consists of a pair of values (J_k^j, M_m), indicating that job J_k^j is assigned to machine M_m, where j is the index of a job in user U_k's batch of jobs J_k and m is the index of the machine. Jobs assigned to a given machine are ordered by the local list scheduling algorithm (Switalski and Seredynski, 2011). The main idea of this algorithm is to schedule jobs locally in a way to minimize the idle period of a processor's cycle.

At the beginning, jobs are sorted according to their work W. If some jobs have equal work, then they are ordered by a sequence given by the MOGA scheduler. Figure 3 shows an example of the local scheduling scheme. Let us assume that a subset of seven jobs was assigned to a machine M_i with $m_i = 8$ processors by the MOGA. We calculate for each job its work. Let us assume that work for this subset is as follows: $W_1 = 12, W_2 = 12, W_3 = 6, W_4 = 1, W_5 = 15, W_6 = 24, W_7 = 2$.

These jobs are next sorted according to their work as depicted in Fig. 3(a). This sorted substring is used directly to assign them to processors of the target machine M_i. The job with the largest work is assigned first with subsequent jobs allocated in descending order as presented in Fig. 3(b).

4.2. Pareto front generation. As stated before, we apply in this work a Pareto optimality approach to solve the MOP defined in Eqn. (2). In particular, we employ the second version of the *nondominated sorting genetic algorithm (NSGA-II)* (Deb *et al.*, 2000), resulting in a Pareto frontier of non-dominated scheduling solutions.

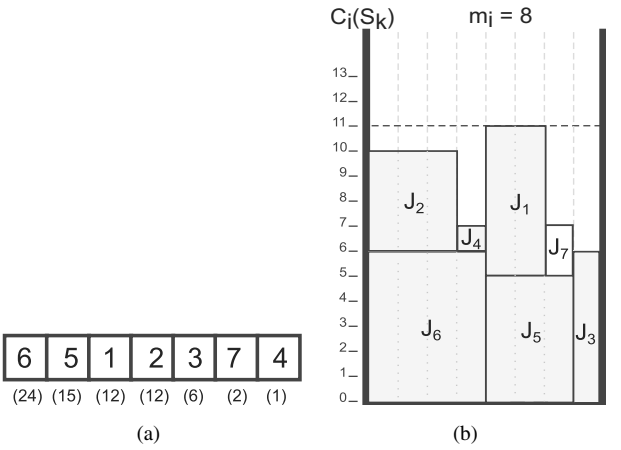


Fig. 3. List scheduling algorithm on a local machine M_i: representation of the local scheduling solution and its construction. Values in brackets represent each job's *resource consumption* value (a), final allocation of jobs in the target machine M_i (b).

We wish to further restrict the resulting solution search space to Pareto-efficient scheduling strategies, from among which the user can make an educated choice and select whatever solutions are best suited to one's preferences at the time. However, the user usually lacks *a priori* knowledge about possible trade-offs between job completion time and scheduling costs, or if the selection of a particular allocation strategy will accomplish execution of all submitted jobs in the required time frame.

Therefore, it is necessary to introduce additional mechanisms limiting a set of possible scheduling strategies from the Pareto frontier in order to provide the user with the most feasible solutions. We propose multiple *Selection Policies* aiming to achieve this goal. Each one of these *Policies* is superior in optimizing one specific objective or their combination and defined according to the following template:

$$\text{Optimize}\left\{\alpha \times C_{\max}^k + \beta \times \overline{P_{\text{Success}}^k} + \gamma \times \overline{\text{Cost}^k}\right\}. \quad (3)$$

By specifying the basic goals and requirements of the cloud user, we distinguish four such *Selection Policies*:

- **Maximum Reliability,** selecting a strategy from the Pareto frontier yielding the maximum *Security Assurance Level* (Max$\{\alpha \sim 0, \beta \sim 1, \gamma \sim 0\}$).
- **Minimum Cost,** selecting a strategy from the Pareto frontier yielding the minimum fees for the cloud provider (Min$\{\alpha \sim 0, \beta \sim 0, \gamma \sim 1\}$).
- **Minimum Cost with Deadline,** selecting a strategy

from the Pareto frontier yielding the minimum fees for the cloud provider while meeting the deadline required by the user (Min$\{\alpha \sim 0, \beta \sim 0, \gamma \sim 1\} | C^k_{\max} < d^j_k$).

- **Optimum,** selecting a strategy from the Pareto frontier minimizing the weighed sum of the three objectives (Min$\{\alpha \sim 0.33, \beta \sim 0.33, \gamma \sim 0.33\}$).

We depict in Fig. 4 an example Pareto frontier generated by broker B_k consisting of the viable, non-dominated scheduling solutions ($S_k \in S^*_k$) with highlighted *Scheduling Strategies* corresponding to the aforementioned Pareto *Selection Policies*. After the broker decides which strategy in the frontier to use, the system passes the input parameters of the chosen allocation strategy to the scheduler, which then submits them to the resource queue.

However, those policies are purely selfish and do not consider the impact of interactions with other users of the CC system. In this work, we consider multiple users competing for a limited number of resources with the help of specialized brokers. These brokers (or agents) are interested in maximizing their own welfare, according to a pure strategic behavior, and hence their unique goal is determining a scheduling strategy aiming at optimizing their own welfare. A natural framework in which to study such a problem is the classic game theory.

5. Game-theoretic scheduling scheme

This section provides a complete overview of our proposed agent-based game-theoretic distributed scheduling scheme.

5.1. Non-cooperative scheduling game model. In a game-theoretic context, our optimization problem, because of the implicit need of a coordination mechanism, can be modeled as a non-cooperative strategic game of independent, autonomous agents. These entities do not operate according to a common strategy, but act in a purely selfish manner, aiming at choosing an optimal strategy of mapping jobs to machines in order to maximize their own objective function. Our non-cooperative game can be formally defined as a triple $(B, \overline{S}, \Xi)$ that consists of

- a finite set $B = \{B_1, \ldots, B_n\}$ of brokers;
- for each broker $B_k \in B$, a set of available Pareto *Selection Policies* $S_k \in S^*_k$, $\overline{S} = (S_1, \ldots, S_k, \ldots, S_n)$;
- for each broker $B_k \in B$, an outcome (*Utility*) function $\Xi_k(S_k \in \overline{S}) \to \mathbb{R}$.

The individual component strategies $S_k \in S^*_k$, where S^*_k denotes the set of all the possible scheduling strategies

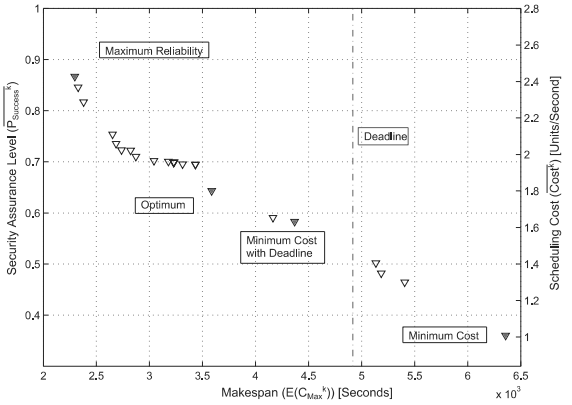


Fig. 4. Pareto frontier generated by broker B_k assigned to user U_k consisting of the viable, non-dominated scheduling solutions ($S_k \in S^*_k$). Strategies selected by the Pareto *Selection Policies* (*Maximum Reliability, Minimum Cost, Minimum Cost with Deadline* and *Optimum* policy) are marked as filled triangles.

available for broker B_k, characterize the actions to be chosen by the associated agent for any possible scenario that it can experience. All these strategies can be found on the Pareto frontier generated by the scheduler as depicted in Fig. 4. For each broker B_k, the result of the game is determined by the *Utility* function. It is a function of a broker's own strategy S_k, as well as strategies of other brokers, i.e., $\overline{S} = (S_1, \ldots, S_k, \ldots, S_n)$. Without loss of generality, we assume that the *Utility* function $\Xi_k(S_k \in \overline{S})$ in this model is to be minimized.

5.2. Broker's utility function. Usually the costs of job scheduling are limited to the actual costs of job execution; however, in utility-based and security-assured scheduling some additional costs, e.g., resource utilization costs or fees for the secure allocation of jobs to the machines, must also be considered (Kolodziej and Xhafa, 2011). In our model, the *Utility* function $\Xi(S_k)$ of an agent B_k selecting a *Scheduling Strategy* S_k is defined as follows:

$$\Xi_k(S_k) = \sum_{(i,j) \in S_k} \left[\frac{W^{i,j}_k \times \text{Cost}^{i,j}_k}{P^{i,j}_{\text{Success}}} \right] \times \frac{C^k_{\max}}{\Phi^k_{\text{RUR}}}, \quad (4)$$

where

- *Scheduling Strategy* (S_k) defines a complete set of associations among the jobs belonging to a specific user U_k and the available machines. Each job J^j_k to be completed on a machine M_i requires a processing time defined as $p^{i,j}_k = t^j_k / s_i$, while $C^k_{\max}$ denotes the time of completion of the latest job;

- *Scheduling Cost* ($\text{Cost}_k^{i,j}$) denotes a *Fee* for a cloud provider, dependent on the QoS requested (see Section 3.2) and proportional to both job processing time and the number of threads of job J_k^j, i.e., its *work* $W_k^{i,j} = p_k^{i,j} \times \text{size}_k^j$;
- *Job Completion Probability* ($P_{\text{Success}}^{i,j}$) associated with each job allocation in schedule S_k denotes the confidence level that the job will be completed without interruptions and is a direct result of the *Security Demand* and *Security Level* factors describing the job and machine, respectively;
- *Resource Utilization Ratio* (Φ_{RUR}^k) defines the ratio of an actual time of job execution to a duration of the *Allocated Time Slot* reserved on the target machine. The aim of this factor is mitigating performance degradation due to load imbalance resulting from selfish and non-cooperative behavior of the independent brokers.

Each broker selects its *Scheduling Strategy* S_k in a pure selfish way according to one of the previously defined Pareto *Selection Policies*. Brokers aim at designing the best schedule minimizing their own *Utility*, that is, the one using the most powerful and reliable machines and minimizing the associated scheduling costs and load imbalance. In order to visualize the trade-offs between these objectives let us consider the following simplified scenario, where

- the system consists of $m = 2$ CC nodes; node M_1 is perfectly reliable ($SL = 1$), while node M_2 offers only partial fault-tolerance ($SL = 0.5$);
- users ($n = 2$) submit to the system a batch of $n_k = 5$ distinct jobs with $SD = 0.7$.

Brokers assigned to individual users generate the Pareto frontiers and select *Scheduling Strategies* according to previously defined Pareto *Selection Policies*. *Gantt* charts representing possible job allocations resulting from local *Scheduling Strategies* of brokers B_1 and B_2 are visualized in Figs. 5 and 6, respectively.

As expected, the *Maximum Reliability* policy allocates most of the jobs to a more reliable machine, M_1. Analogously, the *Minimum Cost* policy allocates only one job to machine M_1, minimizing the associated scheduling costs. The remaining policies, i.e., the *Minimum Cost with Deadline* and *Optimum*, allocate submitted jobs in a way to achieve a compromise between various conflicting objectives inherent in the presented scheduling problem.

5.3. Construction of the competitive scheduling solution. Obviously, brokers are not isolated and their actions influence and are influenced by those of other brokers. Accordingly, they must be forced to interact in order to generate a *competitive schedule* in which all the users will have their jobs processed in the cloud (Palmieri *et al.*, 2013). This can be achieved by incorporating the global goal of the system into the local interests of all agents and such a formulation of the local interaction rules that will allow to achieve those interests.

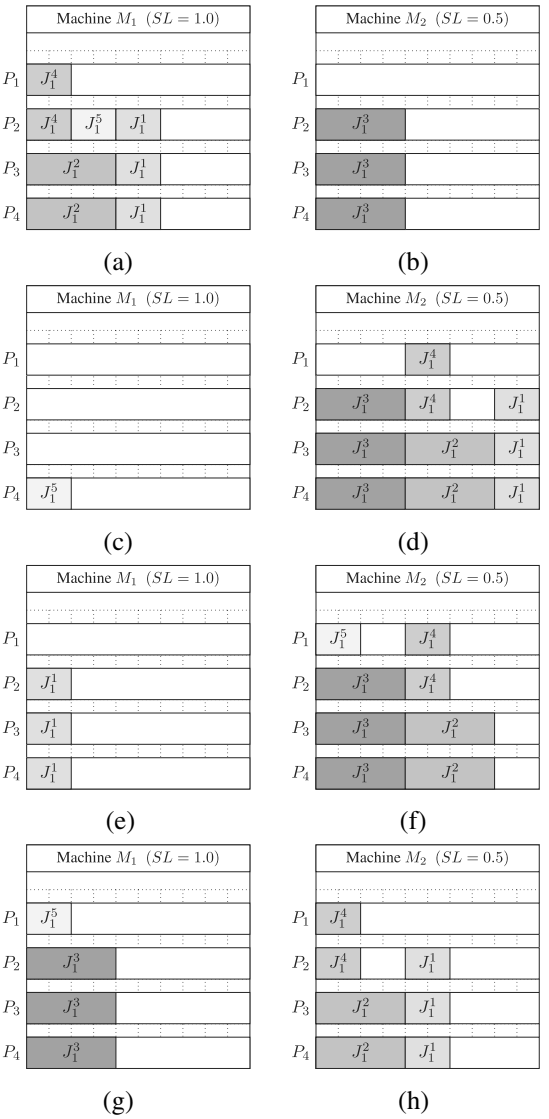


Fig. 5. Gantt charts of *Local Scheduling Strategies* S_1 of broker B_1 according to various Pareto *Selection Policies: Maximum Reliability* (a)–(b), *Minimum Cost* (c)–(d), *Minimum Cost with Deadline* (e)–(f), *Optimum* (g)–(h).

By gathering information about the component *Scheduling Strategies* proposed by other participating players, each broker B_k is able to construct a tentative scheduling profile $\overline{S} = (S_1, \ldots, S_k, \ldots, S_n)$ containing all the solutions proposed by players operating in the cloud. Such a schedule $\overline{S}$, further referred to as a *Competitive Scheduling Profile*, represents a combination of job

allocation strategies for all the brokers and hence defines a specific state of the scheduling game.

For example, Fig. 7 depicts one such a *Competitive Profile* constructed by combining local *Scheduling Strategies* found by the *Minimum Cost* and *Maximum Reliability* Pareto *Selection Policies* employed by brokers B_1 and B_2, respectively. Due to the lack of any central coordination mechanism, such a *Competitive Schedule* resulting from the above interaction may not necessarily share the available cloud resources in the most efficient way.

Thus, we are interested in conditioning the *Scheduling Strategies* S_k unilaterally chosen by each broker to obtain a *Competitive Scheduling Profile* $\overline{S}$ that optimizes the *Social Utility*, i.e., provides a form of compromise, sharing the minimal achievable *Utility* for all the brokers. This means finding a *Competitive Profile* $\overline{S}$ so that

$$\overline{S} \leftarrow \arg\min \left[\Xi(\overline{S})\right], \tag{5}$$

which is a solution presenting the minimum *Social Utility*, where the *Social Utility* associated to a single *Scheduling Profile* $\overline{S}$ can be expressed by the following formula:

$$\Xi(\overline{S}) = \frac{1}{n}\sum_{k=1}^{n} \Xi_k(\overline{S}). \tag{6}$$

The process of construction of such a *Competitive Scheduling Profile* (described in greater detail in the following section) is realized through a sequence of steps, where each broker, starting from the initial job allocation strategy S_k, proposes a new local scheduling solution S'_k minimizing his *Utility* score, that is,

$$S'_k \leftarrow \left(S_k \in S^*_k\right) \;\middle|\; \arg\max \left[\Gamma(\overline{S}, \overline{S'})\right], \tag{7}$$

where $\Gamma(\overline{S}, \overline{S'})$ denotes the *Utility Gain* metric, defining the relative difference between the previous $(\Xi_k(\overline{S}))$ and current $(\Xi_k(\overline{S'}))$ *Utility* score, i.e.,

$$\Gamma(\overline{S}, \overline{S'}) = \frac{\Xi_k(\overline{S}) - \Xi_k(\overline{S'})}{\Xi_k(\overline{S})}. \tag{8}$$

Obviously, each change in the local job allocation strategy by an individual broker may create new conflicts and influence the *Utilities* of several other brokers, thus implying reconstruction of the *Competitive Scheduling Profile* $\overline{S'}$ and recalculation of the associated *Utility* scores. The *Profile* $\overline{S}$ will not present a valid solution of the game until conflicts in resource assignment among the different broker's schedules are resolved. Thus, the goal of the scheduler is transforming this *Competitive Scheduling Profile* $\overline{S}$ into a pure NE (Nowak and May, 1992), defining a fundamental point of stability within the system, such that no broker can unilaterally perform any action (modification of his Pareto *Selection Policy*) to further improve his *Utility* score.

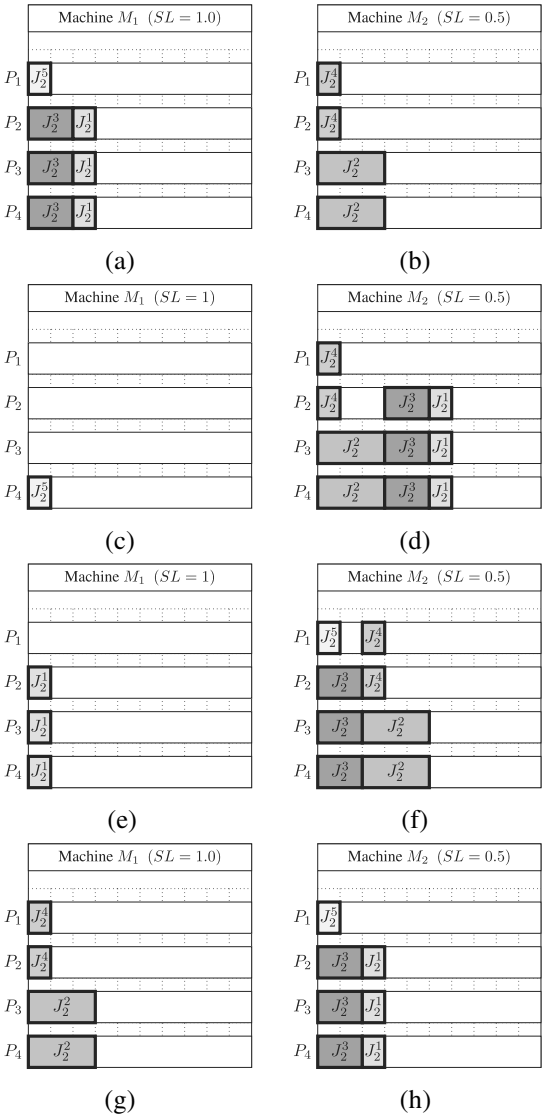


Fig. 6. Gantt charts of *Local Scheduling Strategies* S_2 of broker B_2 according to various Pareto *Selection Policies: Maximum Reliability* (a)–(b), *Minimum Cost* (c)–(d), *Minimum Cost with Deadline* (e)–(f), *Optimum* (g)–(h).

5.4. Rules of the distributed scheduling game. We now proceed with a more detailed description of our distributed scheduling game and the aforementioned process of acquiring a *Competitive Scheduling Profile* optimizing the *Social Utility* metric. The whole scheme is realized using a modified version of the SPD game model (Nowak and May, 1992). The key tenet of this game is that the only concern of each broker is to maximize his payoff during the interaction, which sets the players as naturally selfish individuals. The pseudo-code of the whole process is presented in Algorithm 1. We employ a

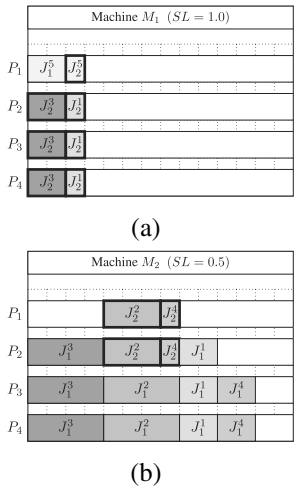


Fig. 7. Gantt chart of the *Competitive Scheduling Profile* $\overline{S} = (S_1, S_2)$ constructed by combining local *Scheduling Strategies* of both participating brokers: $\overline{S} =$ (*Minimum Cost*, *Maximum Reliability*).

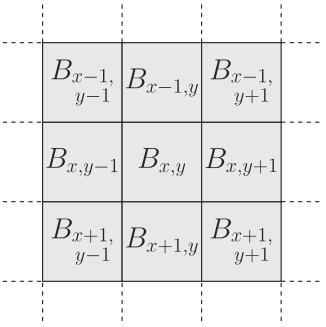


Fig. 8. Cellular automata space lattice. Brokers mapped onto cells interact with other players in their local *Moore* neighborhood.

variation of the PD game working in the two-dimensional CA space, where brokers are mapped onto a rectangular lattice with periodic boundary conditions and interact with a set of local neighbors. In our case, we consider the *Moore* neighborhood (formed by $z = 8$ cells surrounding a given cell) depicted in Fig. 8.

At the first step of the game, we assign randomly to each broker B_k an initial SPD game *Action*, $A_k = \{C, D\}$, SPD game *Spatial Strategy*, σ_k, and one of the available Pareto *Selection Policies*, P_k (Lines 10–12). A broker's *Spatial Strategy* determines the *Action* selected in the following round of interactions (Line 18). To specify a *Spatial Strategy*, the actions of the eight neighbors and the player itself must be considered. For simplicity's sake, we restrict ourselves to a *Totalistic Spatial Strategy* that depends on the number of *D (defect)* actions of the neighbors, not on their positions (Katsumata and Ishida, 2008). To represent a strategy, a bit sequence is used whose l-th element is 1(0) if action *C(D)* is taken when the number of *D* actions in the neighborhood is equal to $l\,(l = 0, 1, \ldots, 8)$. As a typical strategy, we define σ_k such that it takes action *D* if $l > k$ and *C* otherwise. This σ_k strategy can be regarded as a spatial version of *Tit-for-Tat* (TFT), where k indicates how many *D* actions in the neighborhood are tolerated.

Table 1. Distributed scheduling game payoff matrix for the row player.

Action	Cooperate (C)	Defect (D)
Cooperate (C)	$\Gamma/2$	$\Gamma/4$
Defect (D)	Γ	0

Action (C) is considered an equivalent of the *Cooperation* in a classic PD game (Line 20) and denotes a situation where brokers try to cope with potential allocation conflicts by selecting a new Pareto *Selection Policy* P'_k and a corresponding *Scheduling Strategy* S'_k from the Pareto frontier. On the other hand, action (D) means that a broker keeps his current *Selection Policy* and declines to participate in the search for a scheduling compromise, which is considered an equivalent of the *Defection* in a classic PD game (Line 22).

Depending on their actions, brokers are rewarded according to the payoff matrix parameterized in terms of the *Utility Gain* metric, Eqn. (8), and presented in Table 1. The best outcome from a selfish broker's perspective is of course *Defection* (D) (free-riding) in situations where other participating players update their *Selection Policies* in order to achieve a compromise scheduling solution (i.e., choose *Cooperation* (C)). In such a case, *Defector* (D) can acquire a *Temptation* payoff equal to the *Utility Gain* score, while *Cooperating* (C) players are being penalized with the *Sucker's Payoff*.

The *Cumulative Payoff*, G_k, of each individual is determined by summing the acquired payoffs from games with $z = 8$ agents belonging to his local neighborhood $N(x, y)$ as follows (Line 27):

$$G_k = G_k^1 + G_k^2 + \cdots + G_k^z = \sum_{i=1}^{z} G_k^i. \tag{9}$$

Afterwards, all individuals update their SPD game *Spatial Strategies*, σ_k (Line 28). Brokers in a local neighborhood are ranked in ascending order of their cumulative payoffs $G_1 \leq G_2 \leq \cdots \leq G_z$. Player B_k then adopts player B_z's *Spatial Strategy* σ_z with a probability given by the Fermi–Dirac distribution function as proposed by Szabó *et al.* (2005):

$$W(\sigma_k \leftarrow \sigma_z) = \frac{1}{1 + \exp\left[(G_k - G_z)/K\right]}, \tag{10}$$

Algorithm 1. *SPD-NSGA-II*: Distributed scheduling algorithm

1. **Input:** $B(B_1, B_2, \ldots, B_n)$: Set of brokers
2. **Input:** $J(J_1, J_2, \ldots, J_n)$: Batches of jobs
3. **Input:** $M(M_1, M_2, \ldots, M_m)$: Cloud nodes
4. **Input:** $P(P_1, P_2, \ldots, P_n)$: Pareto *Selection Policies*
5. **Input:** $A(A_1, A_2, \ldots, A_n)$: SPD game *Actions*
6. **Input:** $\sigma(\sigma_1, \sigma_2, \ldots, \sigma_n)$: SPD game *Spatial Strategies*
7. **Output:** $\overline{S}$: *Competitive Scheduling Profile*

8. *Initialize Iteration Counter,* $T \leftarrow 0$
9. **for all** $B_k \in B$ **do**
10. *Randomly Assign:* Action, A_k
11. *Randomly Assign:* Spatial Strategy, σ_k
12. *Randomly Assign: Pareto* Selection Policy, P_k

13. *Generate the Pareto set of* Scheduling Strategies, $S_k^* \leftarrow MOGA(J_k, M)$
14. *Select initial* Scheduling Strategy, $S_k \leftarrow P_k(S_k^*)$
15. *Construct* Competitive Scheduling Profile, $\overline{S} \leftarrow \{S_1, \ldots, S_k, \ldots, S_n\}$
16. *Calculate initial* Utility, $\Xi_k(\overline{S'})$

17. **while** $T < T_{Max}$ **do**
18. *Select* Action *based on* Spatial Strategy, $A_k \leftarrow \sigma_k(A_1, \ldots, A_z)$

19. **if** $A_k = C$ **then**
20. *Update* Scheduling Strategy, $S'_k \leftarrow P'_k(S_k^*) \mid \arg\max\left[\Gamma(\overline{S}, \overline{S'})\right]$
21. **else**
22. *Keep* Scheduling Strategy, $S_k \leftarrow P_k(S_k^*)$
23. **end if**

24. *Reconstruct* Competitive Scheduling Profile, $\overline{S'} \leftarrow \{S'_1, \ldots, S'_k, \ldots, S'_n\}$
25. *Calculate* Utility, $\Xi_k(\overline{S'})$
26. *Calculate* Utility Gain, $\Gamma(\overline{S}, \overline{S'}) \leftarrow \frac{\Xi_k(\overline{S}) - \Xi_k(\overline{S'})}{\Xi_k(\overline{S})}$
27. *Calculate* Cumulative Payoff, $G_k \leftarrow \sum_{i=1}^{z} G_k^i$
28. *Update* Spatial Strategy, $W(\sigma_k \leftarrow \sigma_z)$
29. *Update* Iteration Counter, $T \leftarrow T + 1$
30. **end while**

31. **end for**
32. **return** Competitive Scheduling Profile, $\overline{S} \leftarrow \overline{S'}$

where K is a factor controlling the intensity of the strategy imitation process. It helps to avoid trapped conditions and enables smooth transition towards stationary game states (Perc and Szolnoki, 2008). Without much loss of generality, we use in our work $K = 5$.

An equilibrium is reached when further modifications of job allocation strategies are no longer profitable to participating brokers. This is experienced when no further improvement can be achieved in the acquired *Utility* score and a valid *Competitive Scheduling Profile* $\overline{S}$ is obtained. Optionally, the process terminates when a fixed maximum number of iterations T_{Max} is reached (Line 17).

To better illustrate this process, let us consider a simple scenario in a previously described small system model. Let us assume the following starting conditions:

- broker B_1 starts with the *Minimum Cost* Pareto *Selection Policy* and action (*C*);
- broker B_2 starts with the *Optimum* Pareto *Selection Policy* and action (*C*).

After constructing the initial *Competitive Scheduling Profile* depicted in Fig. 9(a), both brokers calculate their starting *Utility* scores. We present their values in Table 2, depicting *Utility* scores of both brokers ($\Xi_1(\overline{S})$ and $\Xi_2(\overline{S})$, respectively), as well as the *Social Utility* $\Xi(\overline{S})$ for the whole system. Depending on their assigned SPD game actions, brokers will proceed to adapt their Pareto *Selection Policies* or keep their current job allocation. In our case (action C), both brokers will adjust their Pareto *Selection Policies* in an attempt to maximize their *Utility Gain* scores (Eqn. (8)). This process is akin to the sampling of the available Pareto solution space. Knowing current job allocations of other competing players and a local set of *Scheduling Strategies* resulting from available Pareto *Selection Policies* (visualized earlier in Figs. 5 and 6), each broker is changing his job allocation scheme to the one offering a minimal *Utility* under current workload conditions and assuming that other players will keep their job allocations. Of course, it is only a supposition because the whole scheduling process is highly dynamic and potential changes in *Selection Policies* occur at the same time. Nonetheless, we will show that the scheme is robust enough to converge to an optimal, competitive system-wide scheduling solution.

Accordingly, in our example, broker B_1 will virtually match each of his available *Scheduling Strategies* (visualized in Fig. 5) to the current job allocation of his opponent and select the one that optimizes his individual *Utility* score under present workload conditions. Broker B_2 will perform a similar analysis and select an appropriate *Scheduling Strategy* optimizing his *Utility* score. In our example, broker B_1 changes his Pareto *Selection Policy* to *Maximum Reliability*, while broker B_2 changes his *Selection Policy* to *Minimum Cost*, which results in a reconstruction of the *Competitive Scheduling Profile* and recalculation of the *Utility* scores for every participating broker.

As presented in Table 2, modification of the *Scheduling Strategy* was profitable only to broker B_2, resulting

Table 2. Utility scores measuring the local scheduling performance of brokers B_1 ($\Xi_1(\overline{S})$) and B_2 ($\Xi_2(\overline{S})$) and competitive system-wide scheduling performance ($\Xi(\overline{S})$).

	$\Xi_1(\overline{S})$	$\Xi_2(\overline{S})$	$\Xi(\overline{S})$
$\overline{S}$ = (*Minimum Cost*, *Optimum*)	467.8447	929.7054	698.7751
$\overline{S}$ = (*Maximum Reliability*, *Minimum Cost*)	500.2737	697.2790	589.7764
$\overline{S}$ = (*Maximum Reliability*, *Optimum*)	392.7120	561.4136	477.0628

in lower job completion times and reallocation of most of the jobs to machine M_2. This, in effect, lowered the scheduling fees for the cloud provider and improved the broker's welfare. A new job allocation of broker B_1 (see Fig. 9(b)) resulted in slightly better job completion times, but the radical shift of the Pareto *Selection Policy* from *Minimum Cost* to *Maximum Reliability* led to rescheduling of jobs to the more reliable machine M_1 and a significant increase of the scheduling fees. In effect, it worsened broker B_1's welfare measured by his *Utility* score ($\Xi_1(\overline{S})$).

In the following game iteration, broker B_1 keeps his current Pareto *Selection Policy*, while broker B_2 adapts his policy in an attempt to further maximize his welfare (i.e., changes his Pareto *Selection Policy* to *Optimum*). As shown in Table 2, it causes further improvement of broker B_2's *Utility* score as well as an inadvertent improvement of broker B_1's *Utility* due to the rescheduling of jobs and more efficient distribution of workload. The resulting *Competitive Scheduling Profile* $\overline{S}$ = (*Maximum Reliability*, *Optimum*) represents an NE point in this scenario, because no broker can perform any further adaptation of his Pareto *Selection Policy* to further improve his *Utility* score.

It is important to note that though individual brokers were realizing their selfish goals of maximizing individual welfare the overall system-wide scheduling performance measured by the *Social Utility* $\Xi(\overline{S})$ metric was also inadvertently improved during the course of this interaction, as depicted in Table 2. This confirms our hypothesis that the global MOP of the system can be solved only through local interactions of a number of distributed and independent actors.

Of course, the above scenario represents only one example in an extremely simplified system model. The complexity of the problem increases significantly with the number of brokers participating in the game as well as the sizes of the system and workload submitted for scheduling. To prove the efficiency and scalability of the proposed approach, it is therefore necessary to perform a number of experiments in testbeds imitating large-scale CC systems. Details of these experiments, their results and discussion are provided in the following section.

6. Experimental analysis and performance evaluation

In this section we present and analyze the results obtained from our experimental study.

6.1. Simulation testbed. Our experiments employed multiple independent users competing for a limited number of cloud resources. The scheduling game was conducted on a rectangular CA lattice. The initial SPD game *Actions* and *Spatial Strategies* as well as Pareto *Selection Policies* were equally distributed between the participating players. Each experiment was repeated 50 times under the same system and workload configuration to guarantee statistical significance of the results and to construct an approximate Pareto frontier by gathering non-dominated solutions in all executions. The maximum number of game iterations was fixed at a value of $T_{\text{Max}} = 200$ steps. Table 3 summarizes key simulation parameters used in the experiment. To comprehensively evaluate the scheduling performance, we used the following metrics:

- ***Makespan:*** the time of completion of the latest job submitted to the cloud, defined as $C_{\max} = \max_k\{C^k_{\max}\}$;
- ***Scheduling Success Rate:*** the percentage of jobs successfully completed in the cloud;
- ***Social Utility:*** calculated according to Eqn. (6) for the *Competitive Scheduling Profile*, $\overline{S}$.

6.2. Simulation results. The initial experiments were conducted in order to analyze the feasibility of the proposed solution for large-scale scheduling problems. We compare the results obtained by our MOGA-based scheduler (denoted as *SPD-NSGA-II*) with several static Pareto *Selection Policies*. Simulations employed 16 independent users competing for a limited number of cloud resources. Each user submitted to the system a randomly generated *Bag of Tasks* containing $n_k = 1000$ job instances. Jobs were then scheduled within $m = 8, 16, 32, 64$ CC nodes by independent brokering agents assigned to individual users.

We conducted five different experiments, including four static experiments, where each agent was employing

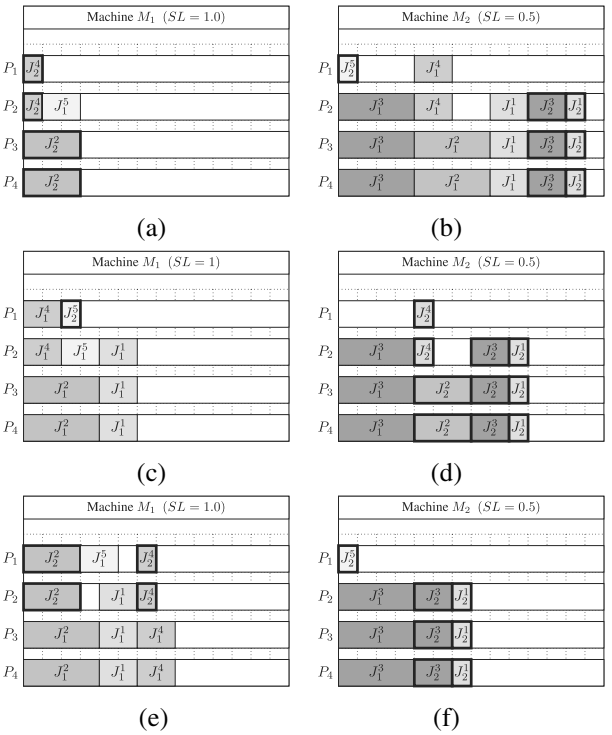


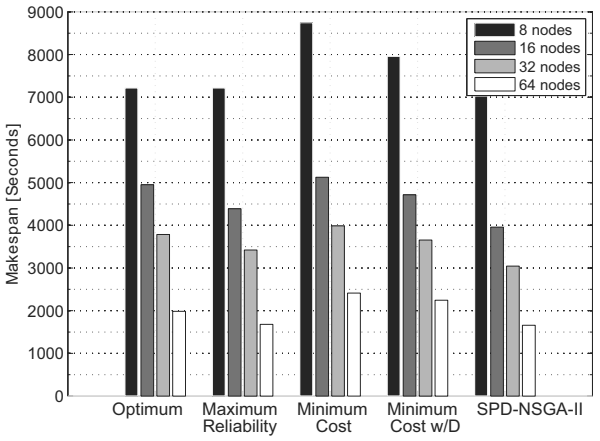
Fig. 9. Gantt charts of *Competitive Scheduling Profile* $\overline{S} = (S_1, S_2)$ constructed by combining local *Scheduling Strategies* of both participating brokers: $\overline{S}$ = (*Minimum Cost*, *Optimum*) (a), $\overline{S}$ = (*Maximum Reliability*, *Minimum Cost*) (b), $\overline{S}$ = (*Maximum Reliability*, *Optimum*) (c).

Table 3. Simulation parameter settings.

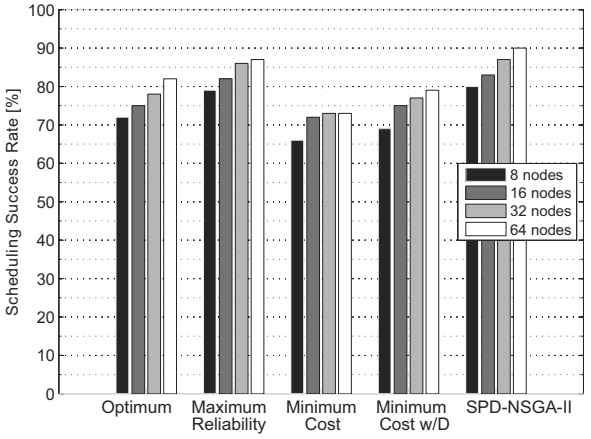
System parameters	Value setting
Number of cloud nodes (m)	8, 16, 32, 64
Average number of cores ($\overline{m_i}$)	6
Average processor speed ($\overline{s_i}$)	2
Node *Security Level* (SL)	$0.3 - 1.0$
Job parameters	**Value setting**
Average number of threads ($\overline{\text{size}_j^k}$)	4
Average execution time of a job ($\overline{t_j^k}$)	5
Job *Security Demand* (SD)	$0.6 - 0.9$

the same Pareto *Selection Policy* from the available set throughout the whole scheduling cycle. No changes in the scheduling strategy were allowed. In the fifth experiment, our proposed solution employing dynamic adaptation of the Pareto *Selection Policies* in the course of the SPD game was evaluated. The simulation results are given in Figure 10 for each proposed performance metric. Let us start with a discussion of the results achieved by the static Pareto *Selection Policies*. Not surprisingly, the *Maximum Reliability* policy achieves the best *Makespan* performance of all compared static *Policies*, due to allocation of jobs to the most reliable resources, regardless of the scheduling cost to the user. As can be seen in Fig. 10(b), it also results in one of the highest *Scheduling Success Rates*. On the other hand, job allocations selected by the *Minimum Cost* and *Minimum Cost with Deadline* policies result in a rather poor performance. The obvious reason is the assignment of jobs to resources offering the lowest *Cost*, regardless of their overall reliability which results in higher probability of failures and more frequent rescheduling events.

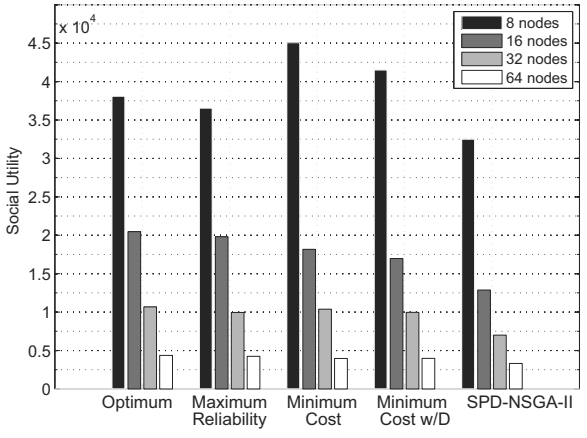
These outcomes are highly correlated with the *Scheduling Success Rate* and *Social Utility* results; i.e., *Secure Job Allocation* guarantees higher probability of success but at the same time requires higher fees for the cloud provider, which affects the broker's *Utility*. Thus, the results produced by the Pareto *Selection Policies* are directly related to their optimization objectives defined by their selection vectors (Eqn. 3). It is clear, however, that the proposed scheduling scheme (*SPD-NSGA-II*) clearly outperforms the static Pareto *Selection Policies*. *Policies* focusing on similar objectives are simply not capable of achieving a compromise solution with competing cloud users. Their similar goals and optimization criteria result in allocating jobs to the same pool of machines, which leads to an overall congestion and load imbalance, and, in the effect, inferior scheduling performance.



(a)



(b)



(c)

Fig. 10. Performance results of conducted experiments with multiple Pareto *Selection Policies* for a total of $n = 16000$ jobs scheduled within $m = 8, 16, 32$ and 64 CC nodes by 16 independent agents; *Makespan* (a), *Scheduling Success Rate* (b), *Social Utility* (c).

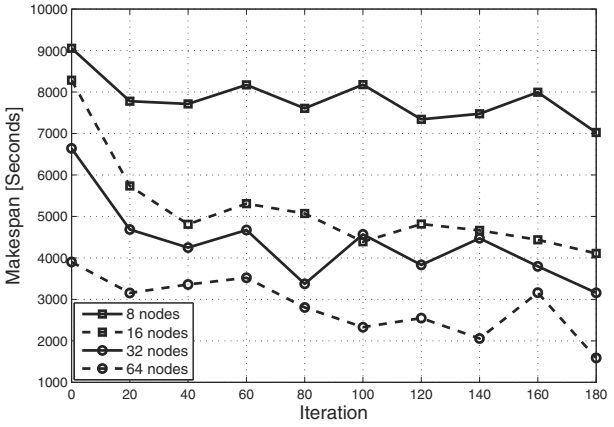
Further experiments aimed to analyze the number of iterations necessary to solve potential conflicts between the selfish strategies of the individual brokers and to converge to a global equilibrium using our proposed distributed scheduling scheme. Figure 11 shows how selected performance metrics change over time due to actions taken by the brokers during our resource allocation game. Starting from an initial job allocation resulting from random assignment of the Pareto *Selection Policies*, a steady progress towards an optimal global scheduling solution is clearly visible. Due to a high *Utility* score resulting from load imbalance and conflicting job allocations, players are compelled to modify their Pareto *Selection Policies* in a way to minimize the negative impact on their individual *Utility* function scores. We conclude that such a behavior is a result of the dynamic interactions between agents who are compelled to cooperate with one another to achieve a game equilibrium. Thus, the scheduling scheme seems to be efficient enough to achieve the desired result, that is, determining a *Competitive Scheduling Profile* that minimizes job completion times and failure probabilities by exploiting brokers' selfish needs to maximize their own *Utility Gain* scores.

7. Conclusions

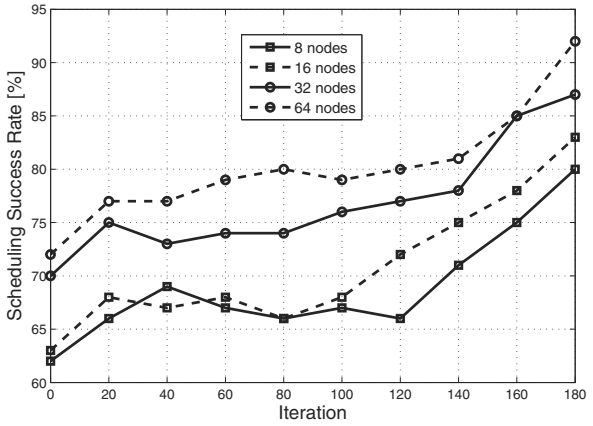
Security-driven job scheduling is crucial for achieving high performance in the cloud computing environment. However, existing scheduling algorithms largely ignore the security induced risks involved in dispatching jobs to untrustworthy resources. The paper proposes a new agent-based game-theoretic scheme for scheduling jobs within a cloud infrastructure. It combines the paradigm of MOGA-based optimization with a game-theoretic model of *Spatial Prisoner's Dilemma* game.

The scheme incorporates security-awareness into scheduling process and aims to minimize both job completion time and possible security risks. Due to its very nature, it is capable of exploring and exploiting the whole range of solution search space. By employing non-cooperative agents, we are able to use the competition among the entities involved to converge towards a Nash equilibrium solution. It allows accounting for often contradicting interests of the clients within the cloud, without the need of any centralized control. Brokers are given a level of autonomy, which grants them properties like adaptation, self-organization and resilience that make such a solution particularly attractive from the cloud scheduling perspective.

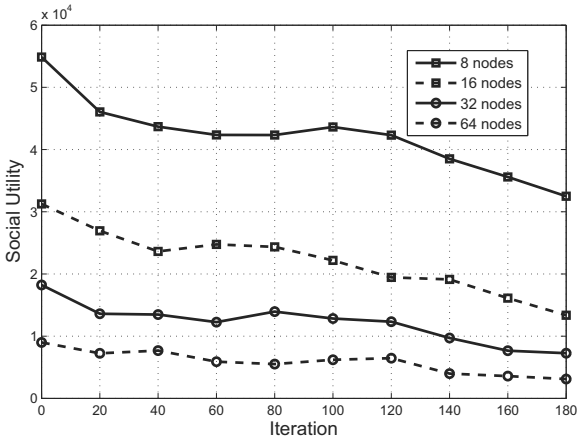
We investigated the effectiveness of the proposed approach by implementing a simple simulation environment emulating the behavior of brokering agents. The achieved results validate the feasibility of the proposed competitive approach applied in the distributed



(a)



(b)



(c)

Fig. 11. Scheduling performance over time using the *SPD-NSGA-II* scheduler for a total of $n = 16000$ jobs scheduled within $m = 8, 16, 32$ and 64 CC nodes by 16 independent agents: *Makespan* (a), *Scheduling Success Rate* (b), *Social Utility* (c).

scheduling scheme. It can lead to a satisfactory solutions both in terms of quality and scalability and provide a substantial performance gain in terms of job completion time, security assurance and scheduling costs.

References

An, B., Miao, C. and Shen, Z. (2007). Market based resource allocation with incomplete information, *in* M. Veloso (Ed.), *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, Morgan Kaufmann Publishers Inc., San Francisco, CA, pp. 1193–1198.

Brandic, I., Pllana, S. and Benkner, S. (2006). An approach for the high-level specification of QoS-aware grid workflows considering location affinity, *Workshop on Workflows in Support of Large-Scale Science, WORKS'06, Paris, France*, Vol. 14, pp. 231–250.

Christodoulou, G., Koutsoupias, E. and Vidali, A. (2007). A lower bound for scheduling mechanisms, *in* H. Gabow (Ed.), *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'07*, Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 1163–1170.

Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II, *in* M. Schoenauer *et al.* (Eds.), *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, PPSN VI*, Springer-Verlag, London, pp. 849–858.

Even-Dar, E., Kesselman, A. and Mansour, Y. (2007). Convergence time to Nash equilibrium in load balancing, *ACM Transactions on Algorithms* **3**(3): 111–132.

Hwang, S. and Kesselman, C. (2003). A flexible framework for fault tolerance in the grid, *Journal of Grid Computing* **1**(3): 251–272.

Katsumata, Y. and Ishida, Y. (2008). On a membrane formation in a spatio-temporally generalized prisoner's dilemma, *in* H. Umeo *et al.* (Eds.), *Proceedings of the 8th International Conference on Cellular Automata for Research and Industry, ACRI'08*, Springer-Verlag, Berlin/Heidelberg, pp. 60–66.

Khan, S.U. and Ahmad, I. (2006). Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation, *20th International Parallel and Distributed Processing Symposium, IPDPS 2006, Rhodes, Greece*.

Kolodziej, J. and Xhafa, F. (2011). Meeting security and user behavior requirements in grid scheduling, *Simulation Modelling Practice and Theory* **19**(1): 213–226.

Lee, Y. and Zomaya, A. (2012). Energy efficient utilization of resources in cloud computing systems, *The Journal of Supercomputing* **60**(2): 268–280.

Li, Z.-J., Cheng, C.-T. and Huang, F.-X. (2009). Utility-driven solution for optimal resource allocation in computational grid, *Computer Languages, Systems & Structures* **35**(4): 406–421.

Lin, C., Varadharajan, V., Wang, Y. and Pruthi, V. (2004). Enhancing grid security with trust management, *in* L.-J. Zhang, J. Zhang and H. Cai (Eds.), *Proceedings of the 2004 IEEE International Conference on Services Computing, SCC'04*, IEEE Computer Society, Washington, DC, pp. 303–310.

Londoño, J., Bestavros, A. and Teng, S.-H. (2009). Collocation games and their application to distributed resource management, *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, HotCloud'09, San Diego, CA, USA*.

Nowak, M.A. and May, R.M. (1992). Evolutionary games and spatial chaos, *Nature* **359**: 826.

Palmieri, F., Buonanno, L., Venticinque, S., Aversa, R. and Di Martino, B. (2013). A distributed scheduling framework based on selfish autonomous agents for federated cloud environments, *Future Generation Computer Systems* **29**(6): 1461–1472.

Perc, M. and Szolnoki, A. (2008). Social diversity and promotion of cooperation in the spatial prisoner's dilemma game, *Physical Review E* **77**: 011904.

Song, S., Hwang, K. and Kwok, Y.-K. (2006). Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling, *IEEE Transactions on Computers* **55**(6): 703–719.

Sookhak, M., Akhunzada, A., Talebian, H., Gani, A., Khan, S., Buyya, R. and Zomaya, A.Y. (2015). Remote data auditing in cloud computing environments: A survey, taxonomy, and open issues, *ACM Computing Surveys* **47**(4), Article no. 65.

Switalski, P. and Seredynski, F. (2011). An efficient evolutionary scheduling algorithm for parallel job model in grid environment, *in* V. Malyshkin (Ed.), *Proceedings of the 11th International Conference on Parallel Computing Technologies, PaCT'11*, Springer-Verlag, Berlin/Heidelberg, pp. 347–357.

Szabó, G., Vukov, J. and Szolnoki, A. (2005). Phase diagrams for Prisoner's Dilemma game on two-dimensional lattices, *Physical Review E* **72**(4).

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R. and Kuzjurin, N. (2010). On-line hierarchical job scheduling on grids with admissible allocation, *Journal of Scheduling* **13**(5): 545–552.

Tziritas, N., Xu, C.-Z., Loukopoulos, T., Khan, S. and Yu, Z. (2013). Application-aware workload consolidation to minimize both energy consumption and network load in cloud environments, *42nd International Conference on Parallel Processing (ICPP), Lyon, France*, pp. 449–457.

Wu, C.-C. and Sun, R.-Y. (2010). An integrated security-aware job scheduling strategy for large-scale computational grids, *Future Generation Computer Systems* **26**(2): 198–206.

Jakub Gąsior is a Ph.D. candidate at the Systems Research Institute, Polish Academy of Sciences in Warsaw, Poland. He received his B.E. and M.Sc. degrees from the Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, in 2009 and 2010, respectively. His research interests concern application of game theory, evolutionary metaheuristics and cellular automata models in the design and management of job scheduling, and load balancing schemes for distributed computing systems.

Franciszek Seredyński is a professor of computer science at the Department of Mathematics and Natural Sciences, Cardinal Stefan Wyszyński University in Warsaw, Poland. He received his M.Sc. and Ph.D. degrees in computer science from the State Electrotechnical University, St. Petersburg, in 1973 and 1978, respectively, and a habilitation from the Institute of Computer Science, Polish Academy of Sciences, in 1998. His research interests concern naturally inspired computational paradigms such as evolutionary algorithms and artificial immune systems, as well as their application for computer security, mobile and ad hoc networks, multiprocessor scheduling and distributed computing.

Received: 11 April 2014
Revised: 9 March 2015