International Transactions on Systems Science and Applications
Volume 8 ▪ December 2012 ▪ pp. 41-55
sai: itssa.0008.2012.018

ISSN 1751-1461 (print)
ISSN 2051-5642 (online)

Load Balancing in the Cloud through Formation of Coalitions in the Spatial Prisoner's Dilemma Game

J. Gasior [a] and **F. Seredynski** [b]

[a] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
[b] Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract: The efficiency, in terms of load balancing and scheduling problems as well as security of both communication and computation processes, belong to the major issues related to currently built cloud computing systems. We present a general framework to study these issues. Our research goal is to develop highly parallel and distributed algorithms working in environments where only local information is available. In this paper we propose a novel approach to dynamic load balancing problem in cloud computing systems. The approach is based on the phenomena of self-organization in a game-theoretical spatially generalized Prisoner's Dilemma model defined on the two-dimensional cellular automata space. The main concept of self-organization used here is based on the formation of temporal coalitions of participants (computational nodes) of the spatial game in the iterative process of load balancing. We present the preliminary concept design for the proposed solution.

Keywords: cellular automata, cloud computing, load-balancing, spatial prisoner's dilemma.

1. Introduction

Cloud computing is one of the emerging developments in distributed, service-oriented, trusted computing. It offers the potential for sharing and aggregation of different resources such as computers, storage systems, data centers and distributed servers. The goal of a cloud-based architecture is to provide some form of elasticity, the ability to expand and contract capacity on-demand. That means there needs to be some mechanism in place to balance requests between two or more instances of client's applications. The mechanism most likely to be successful in performing such a task is a load balancer.

It provides the means by which instances of applications can be provisioned automatically, without requiring changes to the network or its configuration. It automatically handles the increases and decreases in capacity and adapts its distribution decisions based on the capacity available at the time a request is made.

In this paper, we consider the aspect of effective load balancing, i.e., the process of distributing the load among various nodes of a distributed system to improve both resource utilization and job response time. The workload can be defined as CPU load, memory capacity, delay, network load, etc. We formulate a theoretical conceptual model defined as follows: given a set of virtual resources in the Cloud ($M_1, M_2, ..., M_n$), a number of cloud clients ($O_1, O_2, ..., O_k$) and a random set of applications (also jobs or tasks) run by the clients ($J_1, J_2, ..., J_i$), find such an allocation of jobs to the resources to equalize the system workload [6].

We are interested in parallel and distributed algorithms working in environments with only limited, local information. Therefore, we propose a game-theoretical approach combining a spatially generalized Prisoner's Dilemma (SPD) model and the cellular automata (CA) paradigm. Each computational node is presented as a selfishly rational agent. Such a problem formulation is alike to a CA in the sense that the strategy first determines the rule based on the neighbors' configuration and the rule in turn determines the next action [7].

Competing players in such a system should act as a decision group choosing their actions in order to realize a global goal. Main issues that must be addressed here are: a) incorporating the global goal of the multi-agent system into the local interests of all agents participating in the game; and b) such a formulation of cellular automata's local rules, that will allow to achieve those interests [21].

The paper is organized as follows. The following section presents the basic concepts of spatial Prisoner's Dilemma game and cellular automata theory. Section 3 presents our mathematical model of cloud computing system. Section 4 details the load-balancing algorithm from the game theoretical point of view. Section 5 contains a discussion of the results obtained. Finally, Section 6 provides some concluding remarks.

2. Prisoner's Dilemma and Cellular Automata

The concept of the evolution of cooperation has been successfully studied using various theoretical frameworks. In particular the Prisoner's Dilemma (PD) is one of the most commonly employed games for that purpose, a type of non-zero sum game played by two players who can choose between two moves, either to cooperate with or defect from the other player. The problem is called the prisoner's dilemma, because it is an abstraction of the situation felt by a prisoner who can either cut a deal with the police and tell on his partner (defect) or keep silent and therefore tell nothing of the crime (cooperate). While mutual cooperation yields the highest collective payoff, which is equally shared between the two players, individual defectors will do better if the opponent decides to cooperate. The key tenet of this game is that the only concern of each individual player is to maximize his payoff during the interaction, which sets the players as naturally selfish individuals.

The dilemma arises when a selfish player realizes that he can not make a good choice without knowing what the opponent will do. Non-zero sum describes a situation where the winnings of one player are not necessarily the losses of the other [10]. As such, the best strategy for a given player is often the one that increases the payoff to the other player as well. Table 1 shows a general payoff matrix, which represents the rewards an entity obtains depending on its action and the opponent's one. In this matrix, *T* means the *Temptation* to defect, *R* is the *Reward* for mutual cooperation, *P* the *Punishment* for mutual defection and *S* the *Sucker's payoff*. To be defined as a PD, the game must accomplish the condition $T > R > P > S$.

This payoff structure ensures that there is always the temptation to defect since the gain for mutual cooperation is less than the gain for one player's defection. The outcome *(D,D)* is therefore a *Nash equilibrium* - despite the knowledge and awareness of the dilemma, both players opt to defect even though both know they are going to receive inferior scores [13].

The move D for Row player is said to strictly dominate the move C: whatever his opponent does, he is better off choosing D than C. By symmetry D also strictly dominates C for Column player. Thus two rational players will defect and receive a payoff of P, while two irrational players can cooperate and receive greater payoff R. In standard treatments, game theory assumes rationality and common knowledge. Each player is rational, knows the other is rational, knows that the other knows he is rational, etc. Each player also knows how the other values the outcomes. But since D strictly dominates C for both players, the argument for dilemma here requires only that each player knows his own payoffs. It is also worth noting that the outcome *(D,D)* of both players defecting is the game's only *strong Nash equilibrium*, i.e., it is the only outcome from which each player could only do worse by unilaterally changing its move.

Table 1. General Prisoner's Dilemma Payoff Matrix

Action	C	D
C	*(R,R)*	*(S,T)*
D	*(T,S)*	*(P,P)*

However, if there can be ties in rankings of the payoffs, original PD game condition can be weakened without destroying the nature of the dilemma. As in our case we have $T > R > P \geq S$, then, for each player, although D does not strictly dominate C, it still weakly dominates in the sense that each player always does at least as well, and sometimes better, by playing C. Under these conditions it still seems rational to play D, which again results in the payoff that neither player prefers. Let us call such a game a *weak PD*. Note that in a weak PD mutual defection is no longer a *Nash equilibrium* in the strong sense defined above. It is still, however, the only *Nash equilibrium* in the weaker sense, that neither player can improve its position by unilaterally changing its move. In terms of evolutionary game theory D is the unique evolutionary stable strategy (ESS) [16].

2.1. Spatial Prisoner's Dilemma Game

Nowak and May [9] have proposed a way to escape from the dilemma. A variation of Prisoner's Dilemma game working in the two-dimensional cellular automata space where agents are mapped onto a regular square lattice with periodic boundary conditions. In every round, players interact with the immediate neighbors according to a strategy. The fitness of each individual is determined by summing the payoffs in games against each of its neighbors. The scores in the neighborhood, including the individual's own score, are typically ranked. In the next round, all individuals update their strategy deterministically. This approach is typical for cellular automata models. From a biological perspective, the utility of an individual is interpreted in terms of reproductive success. Alternatively, from an economic perspective, the utility refers to individuals adapting their strategy to mimic a successful neighbor [13].

Nowak and May have shown that such spatial structure enables the maintenance of cooperation for the simple Prisoner's Dilemma, in contrast to the classical, spatially unstructured Prisoner's Dilemma where defection is always favored. It was determined that players do not need to play the game with the whole population. By making this assumption, different equilibria are likely to be established in different neighborhoods. More importantly, the spatial structure allows cooperators to build clusters in which the benefits of mutual cooperation can outweigh losses against defectors [7]. Thus, clusters of cooperative strategies can invade into populations of defectors that constitute an ESS in non-spatial populations [9].

2.2. Spatial Strategies in the Prisoner's Dilemma Game

We further generalized the SPD model by introducing a spatial strategy. A player is placed at each cell of a two-dimensional grid. Each player has an action and a strategy, and receives a score. The spatial strategy determines the next action depending on the spatial pattern of the actions of the neighbors. Each player plays PD with his neighbors, and changes that strategy to the one that earns the highest total score among the neighbors. To specify a spatial strategy, the actions of the eight neighbors and the player must be considered. For simplicity, we restrict ourselves to a *totalistic spatial strategy* that depends on the number of *D (defect)* actions of the neighbors, not on their positions [7]. 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 $l(l = 0, 1, ..., 8)$. As a typical strategy, we define *k-D* such that it takes action *D* if $l > k$ and *C* otherwise. This *k-D* can be regarded as a spatial version of TFT where *k* indicates how many *D* actions in the neighborhood are tolerated [3]. Figure 1 illustrates an example of the action update with a *k-D* strategy. Central node changes its action to *C* because the number of defectors in the neighborhood does not exceed *k = 7*.

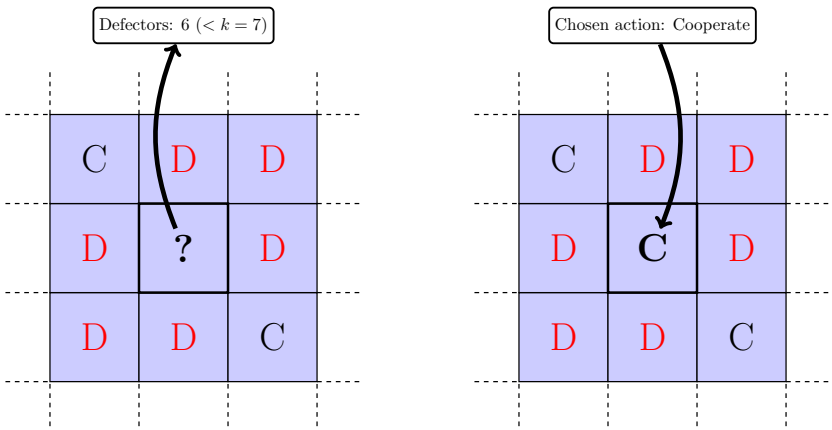


Figure 1. An example of an action update of a *k-D* strategy where *k = 7*.

2.3. Strategy Adaptation Process

After *s* (*strategy update cycle*) steps of interactions with the neighbors, all nodes are presented with an opportunity to update their strategy in a similar manner to the standard SPD game. The present set of strategy imitation rules is based on pairwise comparison of payoffs between two neighboring agents. In each subsequent elementary step of the evolutionary process we choose two neighboring players (*i* and *j*) at random, we determine their payoff G_i and G_j, and player *i* adopts the strategy s_j with a probability given by the Fermi-Dirac distribution function as proposed in [17]:

$$W(s_i \leftarrow s_j) = \frac{1}{1 + exp[(G_i - G_j)/K]}, \tag{1}$$

where: K characterizes the uncertainty related to the strategy adoption process, serving to avoid trapped conditions and enabling smooth transitions towards stationary states [11].

It is well known that there exists an optimal intermediate value of K at which the evolution of cooperation is most successful [12][18], yet in general the outcome of the PD game is robust to variations of K. For $K \ll 1$, selection is weak and the payoffs are only a small perturbation of random drift. For $K \gg 1$, selection is strong and the individual with the lower payoff will change its strategy. In statistical physics, K is the inverse temperature: for $K \to 0$, the dynamics of the system is dominated by stochasticity (the temperature of selection is high), whereas in the limit $K \to \infty$ stochastic effects can be neglected (the temperature of selection is zero) [20]. This phenomenon is fully illustrated in Figure 2. Without much loss of generality, we use $K = 0.1$, meaning that it is very likely that the better performing players will pass their strategy to other players, yet it is not impossible that players will occasionally learn also from the less successful neighbors.

2.4. Dynamic Payoff Values

It is well documented that the level of emergent cooperation is sensitive to many parameters, including the magnitude of the rewards/punishments, population size, initial conditions and update rules [4][8][9]. Typically, the magnitude of the rewards/punishments for playing a particular strategy are fixed in a given game (depending upon the strategy played by the opposing agent) and the same fixed number of agents are active throughout the game period. However, such restrictions are unlikely to be fulfilled in real-life situations where diversity is ubiquitous [15]. If varying levels of diversity are introduced into the game via dynamic payoff values the trajectory of the population will be altered. This in turn will impact on the level of cooperation found in the population.

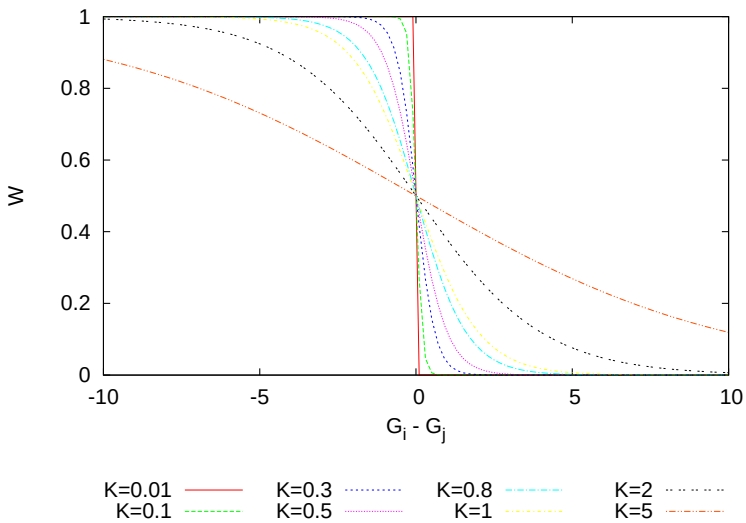


Figure 2. Strategy adaptation probability graph as a function of the payoff difference and variable K, characterizing the uncertainty related to the strategy imitation process.

In this paper, we wish to extend the scope of stochastic effects on the evolution of cooperation in the spatial prisoner's dilemma game by introducing social diversities between agents as well as a dynamic reward system. The scaling factors represent extrinsic differences amongst players, and as such determine their social status on the spatial grid. In our model, resulting payoff matrix is related both to spatial and temporal zones. It is possible that the payoffs for agents and their opponents are not equal - reminiscent of what happens in real-life scenarios. Previous studies suggested that the effect of asymmetries in the interactions between agents had a direct impact on the proportion of agents cooperating in the population [2]. As expected, the model is very sensitive to the values of T used. More detailed informations about our dynamic reward system are presented in Section 4.3.

3. Cloud Model

In this section we formally define basic elements of the model and provide corresponding notation. Then, we define possible characteristics of the model that change the available information and the type of jobs to be scheduled. The model provides a complete representation of system resources. Additionally, it includes information about resource characteristics and availability, as well as the topological information about the network configuration. It provides information used by task assignment algorithms within the load balancing framework.

3.1. Solution Strategy

The Cloud computing load balancing framework consists of (1) a cloud system model, (2) user and application model, (3) a performance model, and (4) a load balancing policy. The system and application model provide information to influence the task assignment decision. We assume that a cloud system may run many jobs concurrently, and this may influence the system state. The model will include information about the current state of the system.

The performance model provides a method of evaluating system and application information, and produces estimates of the performance of a given task on a particular resource. The load balancing policy uses the performance estimate to rank placement decisions according to its policy. Each unit of the framework must be extensible. This allows improvements to be added to the framework as they are needed.

3.2. Elements of the Model

We address an on-line load balancing problem with the objective of minimizing the makespan: n parallel rigid jobs must be scheduled on m parallel machines. For the sake of simplicity, we assume that each cell (node) placed on a two-dimensional square lattice represent a **Cloud's virtual resource** (M_k) - an abstraction of an entity that process jobs (e.g., a supercomputer, a cluster). A resource is composed of one, or more processors that do the actual work. By m_k we denote the number of identical processors of machine M_k. Without loss of generality we index the parallel machines in ascending order of their sizes $m_1 \leq m_2 \leq ... \leq m_m$.

A resource might be a workstation, a vector multiprocessor, a mesh multicomputer, or a network attached storage server. Each resource has a number of characteristics that may vary over time. We can enumerate typical characteristics of a Cloud's resource in different categories, such as: type and quantity of machine's processors and memory, processor's workload, version of operating system, type of storage and communication devices, as well as network location. Clearly, the characteristics that define resource type cannot be fully enumerated, since there is a potentially infinite set of different characteristics. Thus, a system model must necessarily be extensible enough to support arbitrary resource characteristics. In our experiments, we will discuss multiple variants of cloud system's model, where:

- number of available virtual resources in the systems is constant throughout simulation, similarly to typical Grid environment,
- number of available virtual resources is changing dynamically during simulation,
- number of available virtual resources in the systems is constant throughout simulation, however, we additionally distinguish between cooperative (job taking) nodes and selfish, defective (non-job taking) nodes.

The motivation for non-cooperative nodes to enter the grid is to just use resources to fulfill their own processing tasks in the role of clients and refuse to contribute as a worker (although they could due to their capabilities). Note that if nodes do not benefit from cooperation incentives (e.g., the possibility to submit jobs to others in the future), selfishness will be the optimal strategy for each node. This variant is based on our spatially generalized PD game.

Organization (denoted as O_k) is an administrative entity (such as a faculty, a laboratory or a company) that groups users and provides access to a resource. Organizations are independent. $O = O_1, ..., O_N$ denotes the set of organizations in the model. **Job** (denoted as J_k^i) is user's program or application which is executed in the Cloud. Each job may have various parameters such as required data, desired completion time often called the deadline, expected execution time, job priority, etc. Another parameter is the number of requested CPUs. Jobs requesting more than one CPU are called parallel jobs while single CPU jobs are called sequential. Job may also have various constraints such as required hardware architecture or specific software available on the target machine. One job may consist of more tasks which may have defined dependencies between each other. Such jobs are called work flows. Those dependencies may result in a complicated structures although one of the most common description is based on *Directed Acyclic Graph (DAG)* notation [5]. In such situation each task in such job can start its execution only when all preceding tasks in DAG are already finished. However this notation only covers simpler work flows. If the work flow consists of e.g., two tasks that interact together in time then DAG notation is not sufficient. Index i is used only to distinguish between jobs on a processor and does not imply the arrival or execution order.

We define the job as a tuple $< r_k^i, size_k^i, p_k^i, d_k^i >$: its release date $r_k^i \geq 0$, its size $1 \leq size_k^i \leq n_m$, that is referred to as its processor requirements or *degree of parallelism*, its execution time p_k^i and deadline d_k^i. We assume that job J_k^i can only run on machine M_k if $size_k^i \leq m_k$ holds, that is, we do not allow multi-site execution and co-allocation of processors from different machines. J_k^i is ith job produced (and owned) by organization O_k. J_k stands for the set of all jobs produced by O_k. $n_k = |J_k|$ is the number of such jobs. Finally $g(J_k^i) = M_k$ denotes that job J_k^i is allocated to

machine M_k. Let Z_k be the job set allocated to machine M_k. For resource M_k, jobs produced by the resource's organization O_k are called *local jobs*. All other jobs assigned for execution on this resource are called *foreign jobs*. For organization O_k, *remote jobs* are the jobs produced by this organization which are to be executed on non-local processors.

We assume a space sharing scheduling approach as this is typically applied on many parallel computers. Therefore, a parallel job J_k^i is executed on exactly $size_k^i$ disjoint processors without preemptions. Let $p_{max} = max_{1,n} p_k^i$. Further, $W_k^i = p_k^i * size_k^i$ is the work of job J_k^i, also called its area or its resource consumption. Similarly, the total work of a given job set Z is $W_Z = \sum_{J_k^i \in Z} W_k$. All strategies are analyzed according to their competitive factor for makespan optimization [19].

3.3. Performance Metrics

Typical way of assessing systems' performance is measuring the completion time of submitted jobs [1]. There exist various levels of aggregation, on which the performance can be measured: the level of individual jobs, the level of organizations or the level of the complete system.

According to usual notation, C_k^i denotes the completion (finish) time of an individual job J_k^i. To measure the performance experienced by organization O_k, we may compute two aggregated measures. The sum of completion times is the sum of completion times of jobs J_k owned by the organization: $C_k = \sum_i C_k^i$. The makespan (maximum completion time) is the time when the last organization's job finishes $C_{max}(O_k) = max_i C_k^i$.

In a system with non-zero release dates, the completion time of the job is not appropriate to evaluate the actual performance. A job released early and completed late has similar completion time as a job released late and completed immediately. Yet, the performance experienced by the first job is low comparing to the latter. Consequently, it is usual to measure instead the flowtime F_k^i, the time job J_k^i spends in the system. Flow time is defined as the difference between the completion time and the release date, $F_k^i = C_k^i - r_k^i$.

4. Problem Formulation

The approach we follow is a Spatial Prisoner's Dilemma game where every agent (node) may interact with z agents belonging to his neighborhood at a time. For the spatial distribution of the cells we consider a two-dimensional square lattice of size *N x N* with fixed boundary conditions, in which every cell is ruled by an agent. If we let every node in the system to interact with the remaining nodes, we have a panmictic population, i.e., a population where all the individuals are potential partners. But, in many real contexts like geography, biology, MANETs or social networks, each node interacts mainly with a set of local neighbors. In our case, we consider two different neighborhoods: a) the *von Neumann neighborhood* (*z = 4* neighbor cells: the cell above and below, right and left from a given cell) and b) the *Moore neighborhood* (*z = 8* neighbor cells: von Neumann neighborhood + diagonals).

Each node has its own spatial strategy and action. The spatial strategy determines the next action depending upon the spatial pattern of actions chosen by the neighbors. Each node gains a payoff corresponding to their actions after they play the PD game with their neighbors. The payoffs P_i and P_j of both players acquired during each iteration cycle are calculated in accordance with the standard prisoner's dilemma scheme, according to which the temptation to defect $T > 1$, reward for mutual cooperation $R = 1$, punishment for mutual defection $P = 0$, and the sucker's payoff $S = 0$. Although this formulation of the game has $P = S$ rather than $P > S$, it captures succinctly the essential social dilemma, and accordingly, the presented results can be considered fully relevant and without loss of generality with respect to more elaborated formulations of the payoffs.

4.1. Scope of the Model

From the perspective of resource management algorithms, our cloud model is an agreement between selfish, independent organizations to share their resources. We assume that each user is local to some organization. Consequently, we may assume that each job that has been produced by a user is owned by the user's organization. Organizations are administratively independent. There is no central, administrative entity that forces them to cooperate. A cloud is rather a possibility than an obligation. If an organization is not satisfied with the performance of the system perceived by its local users, it can eventually reject the agreement and quit the system [14].

The organizations are selfish in the game-theoretic meaning. Each organization is concerned only with the performance of the locally-produced jobs. Consequently, organizations are not jealous of results achieved by others, if they do not have a direct, negative impact on the locally perceived performance. Organizations behave rationally. They take all the possible actions to optimize the performance of the local jobs.

To describe our model in terms of previously defined characteristics, we assume that our system works on-line; the scheduler is clairvoyant; each resource is composed of one, or more identical processors; jobs follow the divisible load model; preemption is not allowed. Finally, as a performance measure, each organization O_k calculates the sum of flowtimes F_k of locally produced jobs J_k.

For the sake of the theoretical analysis, unless otherwise stated, we assume that the jobs J_k are aperiodic; i.e., jobs arrival times are not known a priori. Jobs are produced by a Poisson process with a known mean time between arrivals λ_k. Such assumption is commonly used in queuing theory due to the fact that the probability of arrival at each time moment is constant [14]. At any given moment, let the local load L_k stand for the time moment when the computation of the last currently known local job ends on cloud's resourceM_k.

Informally, the goal of the scheduler is to find the allocation and the time of execution for each job. The distribution of the tasks must be done in such a way that the system's throughput is optimized. To do so, an optimal trade-off between the processing and communication overhead and the degree of knowledge used in the balancing process must be sought. All scheduling and load balancing decisions are taken locally by the agents. The algorithm analyzes the node's status in terms of its utilization and capabilities. This status is matched against the job's requirements (given by the job's meta-data) considering user-configurable policies that define the desired degree of resource contribution. Subsequently, each node may begin execution of assigned tasks, or split them among its neighbors.

Ideally, each node should receive the same (or nearly the same) number of tasks. If the same amount of work is associated with all the nodes, equal distribution of tasks ensures a good load balance. This statement holds true assuming that communication cost between neighbor nodes is negligible and balancing the workload is always beneficial and leads to a reduction of the execution time. However, such an assumption is unlikely to be fulfilled in real-world environments, since transferring workload from one node to another requires additional communication time. If the network is slow the communication cost can be higher that the execution time gain. Thus, we introduce one more parameter defining the amount of time needed to transfer the workload from one node to another and denote it as q_{ij}, where: *i* and *j* stand for identifiers of nodes participating in the exchange. For simplicity's sake, we assume that communication cost between neighbor nodes is equal to *one*, and grows linearly with each additional cell, except, of course a node may communicate with itself at no cost.

It is important to note that, in this work we make very few assumptions. We can deal with either static or dynamic load. The network topology can be of any type as long as it is connected. Nodes and networks can be homogeneous or heterogeneous. Load balancing algorithms are operating in a fully localized, distributed fashion. The required knowledge is limited to the computation speed, local workload of the neighbors and the computation time per one unit of load. All these information are supposed to be given, calculated or estimated.

4.2. Dynamic Load Balancing Algorithm

This method is a local neighborhood diffusion approach which employs overlapping balancing domains to achieve global balancing and hence the model is called Gradient-Based Neighborhood Averaging (GBNA). The scheme is distributed and asynchronous. Each agent acts independently, sending excessive workload to deficient neighbors. Nodes are limited to load information from within their own domain; all nodes inform their nearest neighbors of their load levels and update this information throughout program execution. Each node acts independently, apportioning excess load to deficient neighbors. The GBNA is defined on a two-dimensional grid, where every cell is one site and is also discrete and discontinuous concerning space and time. There is a surface *(A)* with two indices *(x,y)*. Every gradient to the cell's neighbors has to be checked if it exceeds the threshold (τ). Formalized:

$$A(x, y \pm 1) - A(x, y) > \tau,$$

$$A(x \pm 1, y) - A(x, y) > \tau. \tag{2}$$

The cell can get unstable in different directions, from one up to four (eight in a case of the Moore neighborhood). The number of unstable directions is labeled as η and the unstable neighboring cells are labeled as A_i, whereas $i = 1...\eta$. The relaxation rule for the model is defined as follows:

$$A(x, y) = A(x, y) - \xi_i,$$

$$A_i = A_i + \xi_i, \tag{3}$$

where: ξ_i is the number of tasks shifted in each direction *i*. ξ_i has to be an integer so it is rounded down which is symbolized by the *floor* operation:

$$\xi_i = \left(\frac{A(x, y) - A_i}{\eta + 1} \right)_{floor}. \tag{4}$$

The example shown in Figure 3 has different gradients and gets unstable in all 4 directions shifting one or two tasks. Finally it gets stable again and most importantly, the main cell in the middle has now small gradients to all its surrounding cells (≤ 2). There exists a lower limit for the threshold because of Formula 4. If $\tau \leq \eta$ then ξ will be 0 for all i. So if the threshold is smaller or equal to the numbers of unstable directions, no tasks will be shifted. This means, although a node becomes unstable it may not shift anything to its neighbors because there are too few tasks to shift to all surrounding nodes. This leads to an infinite loop. Because of $\eta \leq 4$ it follows that $\tau \geq 5$.

4.3. The Optimization Problem

We wish to distribute the workload among resources of the system to minimize both: a) *load imbalance* and b) *communication cost* between them. For that purpose, a set of cellular automata's local rules must be evolved according to a specific *utility function*. Let us start by defining the cost and the benefit of a load balancing process. The cost is the time lost by exchanging the workload, due to communication. The benefit is the time gained by exchanging the workload, due to a better balance and faster execution of tasks.

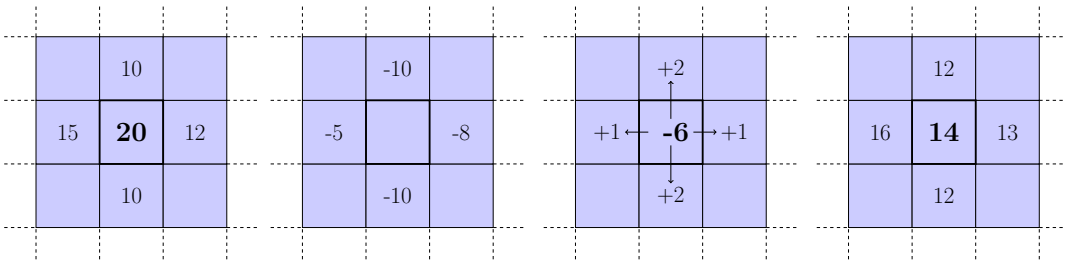


Figure 3. Example of a load balancing with the GBNA model with a threshold of 5 tasks. From left to right: (1) Starting situation with the unstable cell and its four neighbors. (2) The gradients between the main cell and the neighboring cells. (3) Number of tasks which are shifted. (4) Stable situation after the relaxation.

Let E_{ij} stand for the exchange of workload between nodes *i* and *j*. The benefit given by the exchange E_{ij} can be estimated by the computation time on *i* and *j* without the exchange minus the computation time on *i* and *j* after this exchange [6]. Intuitively, the benefit of a load exchange must be positive if the computation time is reduced by this exchange and negative in the other case. The following equation denotes the benefit of load balancing scheme, assuming that node *i* transfers workload to node *j*:

$$\begin{aligned} \text{Benefit}(E_{ij}) = max(L_i, L_j) - \\ max(L_i - E_{ij}, L_j + E_{ij}), \end{aligned} \tag{5}$$

where L_i and L_j define local loads on nodes *i* and *j*, respectively. Let us now consider the communication part of the load balancing process. The cost of communication from one node to another depends on the network architecture (i.e., network bandwidth, network traffic, buffer size). A truly portable load balancing algorithm would have no option but to send sample messages around and measure those metrics, then distribute the workload appropriately. In this paper, however, we shall avoid this question by assuming that all pairs of computational resources are equally far apart. We can make the assumption that the total communication cost is equal to the amount of time needed to transfer the workload from node *i* to node *j* (denoted as q_{ij}) and thus:

$$\text{Cost}(E_{ij}) = q_{ij}. \tag{6}$$

Additionally, we make an assumption that any node which took part in the balancing operation is obliged to return resulting data to the originating node. This issue can be solved by simply propagating the results backwards through the initial load balancing route. Such a problem formulation, however, may become ineffectual in a case of large quantities of workload being shared among many neighboring nodes. It is possible, that in such a case, there exist an alternative way back to the originating node; shorter than original load balancing route. The issue is illustrated in Figure 4, where *A*, represents *source node*, and *B* represents *destination node*. Red line indicates original load balancing route, while gray line shows the optimal way back.

We propose a simple solution to this problem by implementing a gradient-based communication model. We define the node's *proximity (P)* as the shortest distance from itself to the source (sender) node. All cells are initialized with a proximity of P_{max}, equal to the diameter of the system lattice. The proximity is set to 0 if node becomes overloaded and its state changes to *sender*. All other nodes *i* with local neighbors n_i, compute their proximity as:

$$P(i) = min(P(n_i)) + 1. \tag{7}$$

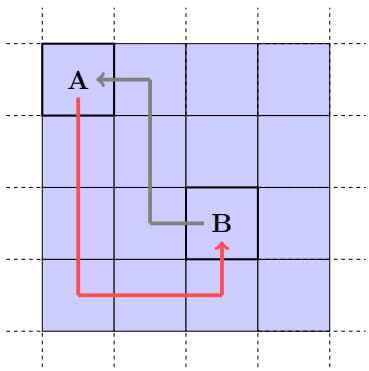


Figure 4. The issue of determining communication cost between *source node (A)* and *destination node (B)*. Red route shows original communication route according to the load balancing algorithm. Gray route indicates an alternative (optimal) way back.

The resulting proximity map is later used used to perform the migration phase. Results are routed through the system in the direction of the sender node (Figure 5).

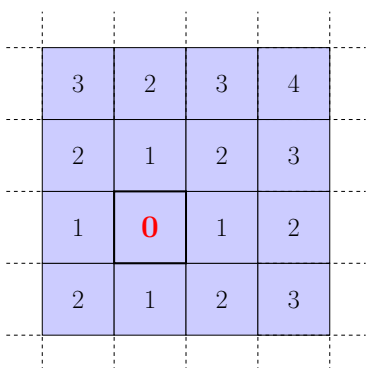


Figure 5. The gradient-based communication model. Computational nodes send results in the direction of the sender node (red) via the gradient map of proximity values. Cellular automata space comprises the von Neumann neighborhood - the four cells orthogonally surrounding a central cell on a two-dimensional square lattice.

Given this parameter, the cost function of load balancing process from Equation 6 can now be extended and denoted as:

$$\text{Cost}(E_{ij}) = q_{ij} + P(i), \tag{8}$$

assuming that node *i* is transferring its workload to node *j*. Such a formulation is possible because node's proximity is equal to the amount of time needed for propagating the results back to the originating node. Additionally, it ensures that load balancing profitability is decreasing linearly with an increase in distance from the source.

We may now construct our *utility function*, Γ, as the sum of parts describing benefits and costs of the load balancing operation, respectively:

$$\Gamma = \sum_{k} \text{Benefit}(E_{ij}^{k}) - \mu \sum_{k} \text{Cost}(E_{ij}^{k}), \tag{9}$$

where: k denotes the amount of workload exchanged between neighbor nodes and μ is a parameter expressing the balance between the two aspects of load balancing scheme - communication and

Table 2. The Prisoner's Dilemma Rescaled Payoff Matrix

Action	C (Send workload)	D (Compute locally)
C (Accept)	$\Gamma/2$, $\Gamma/2$	*0, 0*
D (Reject)	$\Gamma, 0$	*0, 0*

computation. For programs with a great deal of calculation compared to communication, μ should be relatively small, and *vice versa*. As μ increases, the number of machines in use will decrease until eventually the communication is so costly that the entire calculation must be done on a single node. Score calculated according to Γ is awarded to every node taking part in the load balancing scheme. Its magnitude is strictly dependent on agent's action taken in the PD game as shown in Table 2.

It can be seen that agent's performance in the dynamic load balancing scheme directly affects its scores acquired in the PD game, by shifting the magnitude of payoff values. Thus, agent with a more effective balancing strategy will acquire higher scores in the PD game, which in turn will increase the probability of imitating that strategy by his less successful neighbors and propagating it in the system. This in turn should lead to an optimal load distribution in the cloud computing environment.

5. Experiments

In this section we report some initial results of an experimental study of the proposed load balancing model. Our SPD is done in the following way with n (a size of the lattice, 30 x 30 in our simulations) players with $k(2 \leq k \leq m)$ distinct strategies. In a match-up with m strategies, an initial configuration with m strategies is assigned randomly to all the players (900 players). The initial proportion of these m strategies is equal for these strategies. Initial actions (C or D) are assigned randomly to all the players.

Our experiments attempt to support the hypothesis that a relatively small quantity of cooperative node neighborhoods can emerge through an evolutionary procedure and thus these neighborhoods can reduce the computation and communication cost occurring in large load balancing systems. We compare flowtimes for two scenarios:

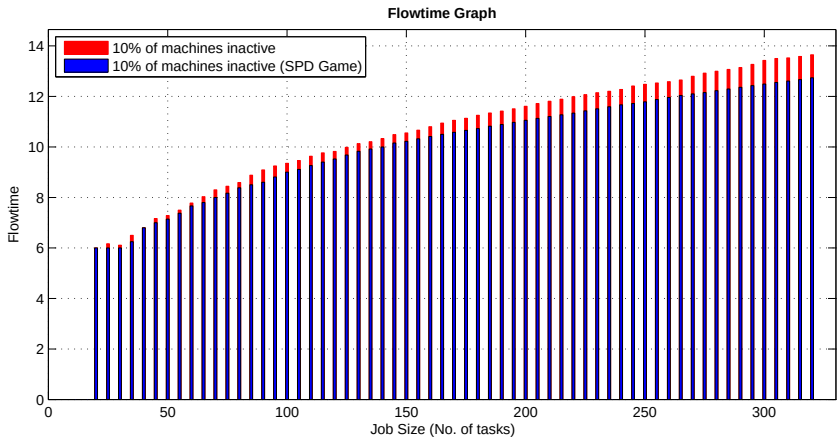
- Game lattice is initialized and the states of 10%, 30% or 50% of randomly selected nodes are set as inactive. Node's state does not change during the course of simulation.
- Game lattice is initialized and the states of 10%, 30% or 50% of randomly selected nodes are set as inactive. Node's state changes dynamically according to the rules of SPD game.

Next, we assign jobs (consisting of increasing number of individual tasks, $20 \leq p_k^i \leq 350$) to randomly selected nodes and calculate sum of flowtimes F_k of locally produced jobs J_k. Comparative results of both aforementioned scenarios are illustrated in Figure 6. From these figures, we can see that the relative improvement in achieved flowtimes is increasing accordingly to the number of initially inactive computational nodes.

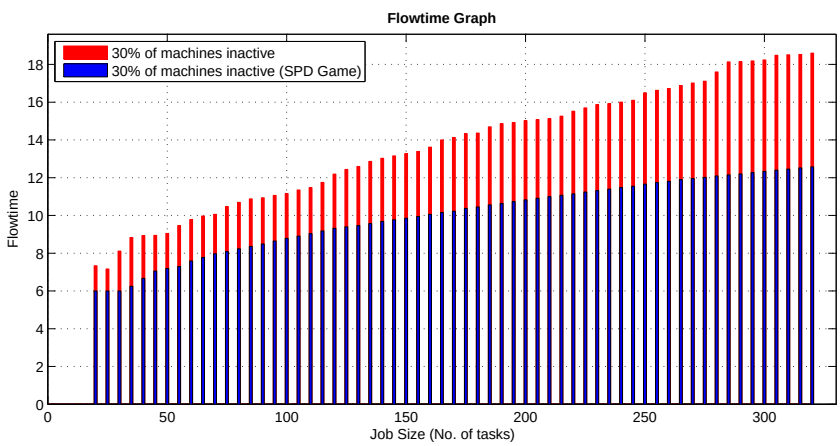
One very important conclusion is that successful strategies tend to build clusters of similar individuals because of local reproduction. With such clusters some strategies can spread in an environment in which they could not spread without clustering. Thus, that spatial structure greatly influences the evolution of cooperative behavior; there are more cooperative individuals in a spatially structured population and cooperation evolves faster. This in turn leads to a more efficient and faster load distribution in the simulated model.

6. Conclusion and Future Work

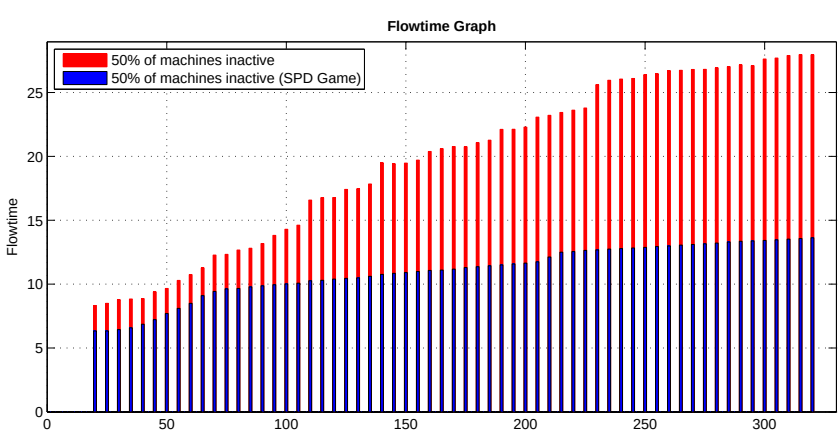
We have proposed in this paper a novel paradigm for a parallel and distributed evolutionary computation in cloud computing systems based on the model of spatio-temporal Prisoner's Dilemma game.



(a) 10% of machines inactive.



(b) 30% of machines inactive.



(c) 50% of machines inactive.

Figure 6. Flowtime graphs depicting results of simulations employing Spatial Prisoner's Dilemma game-based load balancing model. Graphs compare achieved results for a) 10%, b) 30% and c) 50% of inactive nodes, respectively. Red bars indicate results for static model; blue bars indicate results for dynamic SPD game-based model.

We presented the rules of a local interaction among agents providing a global behavior of the system as well as the analysis of costs and benefits of workload exchange. Game-theoretic approach allowed us to model organizational heterogeneity of cloud computing systems.

Our future work is threefold. Firstly, we want to further enhance our model in order to study the problem of evolution of global behavior and formation of coalitions between agents. Secondly, we intend to extend the model to enhance security of both communication and data processing. In particular, we want to focus on aspects of reputation and cryptography. This could be important, for instance, when agents have to decide which action to take against outsiders. If these outsiders have a reputation degree, such information could be used in the decision-making process. Also, reputation may turn important among members of coalitions themselves, for instance to decide when coalitions should be dissolved. Finally, we would like to port this solution to real-world scenarios that involve data networks such as P2P, sensor, and ad-hoc networks.

Acknowledgment

This contribution is supported by the Foundation for Polish Science under International PhD Projects in Intelligent Computing. Project financed from The European Union within the Innovative Economy Operational Programme 2007-2013 and European Regional Development Fund (ERDF).

References

[1] P. Brucker, *Scheduling Algorithms*. Springer, 2004.
[2] H. Fort, "On evolutionary spatial heterogeneous games," *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 7, pp. 1613–1620, 2008.
[3] P. Grim, "The greater generosity of the spatialized prisoner's dilemma," *Journal of Theoretical Biology*, vol. 173, no. 4, pp. 353–359, 1995.
[4] C. Hauert, "Fundamental clusters in spatial 2 times 2 games," *Proceedings: Biological Sciences*, vol. 268, pp. 761–769, 2001.
[5] M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment," in *Proceedings of the Seventh Heterogeneous Computing Workshop*, ser. HCW '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 70–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795689.797884
[6] E. Jeannot and F. Vernier, "A practical approach of diffusion load balancing algorithms," *Euro-Par'06 Proceedings of the 12th international conference on Parallel Processing*, 2006.
[7] Y. Katsumata and Y. Ishida, "On a membrane formation in a spatio-temporally generalized prisoner's dilemma," *Proceeding ACRI '08 Proceedings of the 8th international conference on Cellular Automata for Research and Industry*, 2008.
[8] K. Lindgren and M. Nordahl, "Evolutionary dynamics of spatial games," *Physica D*, vol. 75, 1994.
[9] M. Nowak and R. May, "Evolutionary games and spatial chaos," *Nature*, vol. 359, 1992.
[10] M. Osborne, *An Introduction to Game Theory*. USA: Oxford University Press, 2003.
[11] M. Perc and A. Szolnoki, "Social diversity and promotion of cooperation in the spatial prisoner's dilemma game," *Physical Review E 77*, 2008.
[12] M. Perc, "Coherence resonance in a spatial prisoner's dilemma game," *New Journal of Physics*, vol. 8, no. 2, p. 22, 2006.
[13] G. Rezaei and M. Kirley, "The effects of time-varying rewards on the evolution of cooperation," *Evolutionary Intelligence*, vol. 2, no. 4, pp. 207–218, 2009.
[14] K. Rzadca, "Resource management models and algorithms for multi-organizational grids," Ph.D. dissertation, Grenoble-INP and Polish Japanese Institute of Information Technology, Feb. 2008.
[15] F. Santos, M. Santos, and J. Pacheco, "Social diversity promotes the emergence of cooperation in public goods games," *Nature*, vol. 454, 2008.
[16] J. M. Smith, *Evolution and the Theory of Games*. Cambridge University Press, 1982.
[17] G. Szabó and C. Tőke, "Evolutionary prisoner's dilemma game on a square lattice," *Physical Review E*, vol. 58, no. 1, pp. 69–73, 1998.
[18] G. Szabó, J. Vukov, and A. Szolnoki, "Phase diagrams for prisoner's dilemma game on two-dimensional lattices," *Physical Review E*, vol. 72, 2005.

[19] A. Tchernykh, U. Schwiegelshohn, R. Yahyapour, and N. Kuzjurin, "On-line hierarchical job scheduling on grids with admissible allocation," *J. of Scheduling*, vol. 13, no. 5, pp. 545–552, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10951-010-0169-x

[20] A. Traulsen, M. A. Nowak, and J. M. Pacheco, "Stochastic payoff evaluation increases the temperature of selection," *Journal of Theoretical Biology*, vol. 244, no. 2, pp. 349–356, 2007.

[21] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.