Load Balancing in Cloud Computing Systems Through Formation of Coalitions in a Spatially Generalized Prisoner's Dilemma Game

Jakub Gasior
Systems Research Institute, Polish Academy of Sciences
Warsaw, Poland
E-mail: j.gasior@ibspan.waw.pl

Franciszek Seredynski
Polish-Japanese Institute of Information Technology
Warsaw, Poland
E-mail: sered@pjwstk.edu.pl
Department of Mathematics and Natural Sciences
Cardinal Stefan Wyszynski University
Warsaw, Poland
E-mail: sered@pjwstk.edu.pl

***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 and 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-Cloud computing; Cellular automata; Load-balancing; Spatial prisoner's dilemma.

I. 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 load can be defined as CPU load, memory capacity, delay, network load, etc. We formulate a purely 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 $(U_1, U_2, ..., U_k)$ 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 [1].

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 [2].

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 [12].

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. Finally, Section 5 provides some concluding remarks.

II. PRISONER'S DILEMMA AND CELLULAR AUTOMATA

The concept of the evolution of cooperation has been successfully studied using various theoretical frameworks.

 ISBN: 978-1-61208-216-5

Table I
A GENERAL PRISONER'S DILEMMA PAYOFF MATRIX

	Cooperate	Defect
Cooperate	*(R,R)*	*(S,T)*
Defect	*(T,S)*	*(P,P)*

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 [4]. As such, the best strategy for a given player is often the one that increases the payoff to the other player as well. *Table I* 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 [7]. In terms of evolutionary game theory *defection* is the unique evolutionary stable strategy (ESS) [8].

Nowak and May [3] 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 [7].

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 [2]. Thus, clusters of cooperative strategies can invade into populations of defectors that constitute an ESS in non-spatial populations [3].

III. PROBLEM FORMULATION

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.

For the sake of simplicity, it is assumed that every node placed on a two-dimensional cellular automata represents a *virtualized resource* (M_k) - an abstraction of an entity that process jobs. Computational power C_k of a certain resource M_k is defined by a number of operations per unit of time it is capable of performing. We distinguish between cooperative (job taking) nodes and selfish (non-job taking) nodes. The motivation for non-cooperative nodes to enter the cloud 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.

Job (denoted as J_k) is an equivalent of application run by the cloud clients. Every application is independent and has no link between each other whatsoever, e.g., some require more CPU time to compute complex tasks, and some may need more memory to store data, etc. Resources are sacrificed on activities performed on each individual unit of service. In order to measure direct costs of applications, every individual use of resources (i.e., CPU cost, memory cost, I/O cost) must be measured. To simplify the problem, we assume that *job* is simply an entity that, in order to be completed, requires an access to a resource during certain time p_k. For the sake of the theoretical analysis, unless otherwise stated, we assume that the jobs J_k are produced by a Poisson process. The size of a job is known immediately after the job has arrived to the system. At any given time,

 ISBN: 978-1-61208-216-5

let the local load L_k stand for the time moment when the computation of the last currently known local job ends, thus it can be defined as ratio between total size of node's queued jobs and its computational power:

$$L_k = \frac{\sum_{i=1}^{n} p_k^i}{C_k}, \quad (1)$$

where: *n* stands for the total number of jobs assigned to a single node.

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. 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 (as given by the job's meta-data, p_k) 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. However, such an assumption is unlikely to be fulfilled in real-world environments. 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.

IV. The Dynamic Load Balancing 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.

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 [1]. 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*:

$$\begin{aligned} \text{Benefit}(E_{ij}) = max(L_i, L_j) - \\ max(L_i - E_{ij}, L_j + E_{ij}), \end{aligned} \quad (2)$$

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:

$$\text{Cost}(E_{ij}) = q_{ij}. \quad (3)$$

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 1, where *A*, represents *source node*, and *B* represents *destination node*. Green line indicates original load balancing route, while red 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 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:

 ISBN: 978-1-61208-216-5

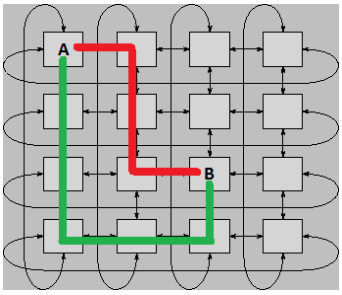

Figure 1. The issue of determining communication cost between *source node (A)* and *destination node (B)*. Green route shows original communication route according to the load balancing algorithm. Red route indicates an alternative (optimal) way back.

$$P(i) = min(P(n_i)) + 1. \quad (4)$$

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 2).

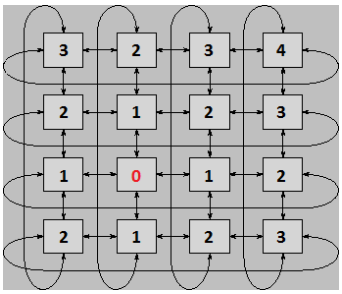

Figure 2. The gradient-based communication model. Computational nodes send results in the direction of the sender node (red) via the gradient map of proximity values. Cellular automata space comprises the von Neumann neighborhood - the four cells orthogonally surrounding a central cell on a two-dimensional square lattice.

Given this parameter, the cost function of load balancing process from Equation 3 can now be extended and denoted as:

$$\text{Cost}(E_{ij}) = q_{ij} + P(i), \quad (5)$$

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:

$$\Gamma = \sum_k \text{Benefit}(E_{ij}^k) - \mu \sum_k \text{Cost}(E_{ij}^k), \quad (6)$$

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 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 processors 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 II*.

Table II
THE PRISONER'S DILEMMA RESCALED PAYOFF MATRIX

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

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 [9]:

$$W(s_i \leftarrow s_j) = \frac{1}{1 + exp[(G_i - G_j)/K]}, \quad (7)$$

where: K characterizes the uncertainty related to the strategy adoption process, serving to avoid trapped conditions and enabling smooth transitions towards stationary states [5].

It is well known that there exists an optimal intermediate value of K at which the evolution of cooperation is most successful [6, 10], 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) [11]. This phenomenon is fully illustrated in Figure 3. 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.

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

 ISBN: 978-1-61208-216-5

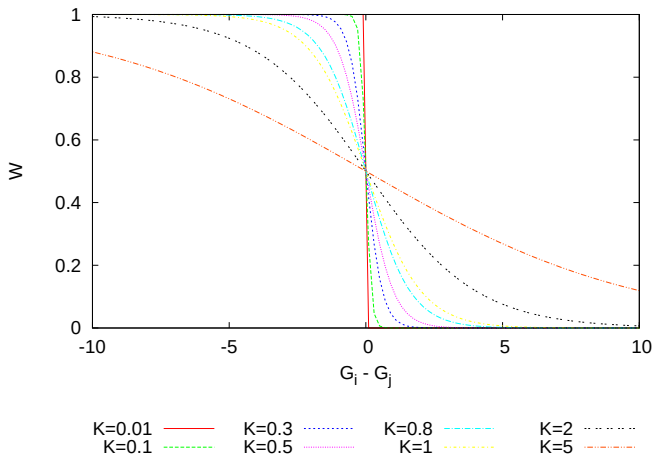

Figure 3. Strategy adaptation probability graph as a function of the payoff difference and variable K, characterizing the uncertainty related to the strategy imitation process.

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.

V. 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. 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. Currently, the model is a subject of the experimental study.

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] E. Jeannot and F. Vernier, "A practical approach of diffusion load balancing algorithms," pp. 211–221, 2006. [Online]. Available: http://dx.doi.org/10.1007/11823285_22

[2] Y. Katsumata and Y. Ishida, "On a membrane formation in a spatio-temporally generalized prisoner's dilemma," pp. 60–66, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-79992-4_8

[3] M. Nowak and R. May, "Evolutionary games and spatial chaos," *Nature 359*, pp. 826–829, 1992.

[4] M. Osborne, *An Introduction to Game Theory*. USA: Oxford University Press, 2003.

[5] M. Perc and A. Szolnoki, "Social diversity and promotion of cooperation in the spatial prisoner's dilemma game," *Physical Review E 77*, vol. 77, p. 011904, Jan 2008. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.77.011904

[6] M. Perc, "Coherence resonance in a spatial prisoner's dilemma game," *New Journal of Physics*, vol. 8, no. 2, p. 22, 2006.

[7] G. Rezaei and M. Kirley, "The effects of time-varying rewards on the evolution of cooperation," *Evolutionary Intelligence*, vol. 2, pp. 207–218, 2009, 10.1007/s12065-009-0032-1. [Online]. Available: http://dx.doi.org/10.1007/s12065-009-0032-1

[8] J. M. Smith, *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[9] G. Szabó and C. Tőke, "Evolutionary prisoner's dilemma game on a square lattice," *Phys. Rev. E*, vol. 58, pp. 69–73, Jul 1998. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.58.69

[10] G. Szabó, J. Vukov, and A. Szolnoki, "Phase diagrams for prisoner's dilemma game on two-dimensional lattices," *Physical Review E*, vol. 72, p. 047107, 2005.

[11] 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.

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