Theoretical Computer Science 411 (2010) 2459–2466

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Game authority for robust and scalable distributed selfish-computer systems☆

Shlomi Dolev[a], Elad M. Schiller[b,*], Paul G. Spirakis[c], Philippas Tsigas[b]

[a] Department of Computer Science, Ben-Gurion University of the Negev, 84105, Israel

[b] Department of Computing Science, Chalmers University of Technology and Göteborg University, Rännvägen 6B Göteborg, S-412 96, Sweden

[c] Research Academic Computer Technology Institute, N. Kazantzakis Street, University Campus, 265 00 Rio, Patras, Greece

ARTICLE INFO

Article history:
Received 24 October 2008
Received in revised form 22 April 2009
Accepted 12 February 2010
Communicated by editor J. Díaz

Keywords:
Distributed computing
Game theory
Game authority
Self-stabilization

ABSTRACT

Distributed algorithm designers often assume that system processes execute the same predefined software. Alternatively, when they do not assume that, designers turn to non-cooperative games and seek an outcome that corresponds to a rough consensus when no coordination is allowed. We argue that both assumptions are inapplicable in many real distributed systems, e.g., the Internet, and propose designing self-stabilizing and Byzantine fault-tolerant distributed game authorities. Once established, the game authority can secure the execution of any complete information game. As a result, we reduce costs that are due to the processes' freedom of choice. Namely, we reduce the price of malice.

"Capitalism is the best economic system in the world because it demands and rewards hard work. It challenges us to be excellent.

But like everything else in life, capitalism can be perverted and exploited. Bad people can find ways to cheat. That's why the federal government oversees the American economy to make sure there is some justice and honesty in pursuit of profit." [Bill O'Reilly, September 18, 2008 in *FOX NEWS*].

1. Introduction

Game theory analyzes social structures of agents that have freedom of choice within a moral code. Society allows freedom and selfishness within this moral code, which is enforced by existing social structures, i.e., legislative, executive, and judicial. Social rules encourage individual profit from which the entire society gains. Distributed computer systems can improve their scalability and robustness by using explicit social structures. We propose using a game authority middleware to enforce the rules of the game, which the honest majority decides upon.

The power of game theory is in predicting the game outcome for specific assumptions. The prediction holds as long as the players cannot tamper with the social structure or change the rules of the game, e.g., the prisoner cannot escape from prison in the classical prisoner dilemma. Therefore, we cannot predict the game outcome without suitable assumptions on failures and honest selfishness.

There are attempts to define various aspects of selfish-computer systems: the selfish MAC Layer that does not back off in [5], the Byzantine Nash equilibrium of a replicated state-machine in [2], and the selfish mechanism for virus inoculation in the presence of malicious agents in [21], to name a few. In fact, [21] discovers that the performance ratio between selfish

☆ Also appeared in Dolev et al. (2007)[9], Dolev et al. (2006) [12] and Dolev et al. (2006) [13].

* Corresponding author. Fax: +46 31 772 3663.
E-mail addresses: dolev@cs.bgu.ac.il (S. Dolev), elad@chalmers.se (E.M. Schiller), spirakis@cti.gr (P.G. Spirakis), tsigas@chalmers.se (P. Tsigas).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.02.014

mechanisms that do and do not have malicious agents (named the *price of malice*, PoM), are as important as the performance ratio between selfish mechanisms and centralistic mechanisms (named, respectively, the *price of anarchy* (PoA) [18,17] and the *price of stability* (PoS) [3]). We study these performance ratios in the presence of game authority implementation and discover significant improvements.

We argue that when designing distributed selfish-computer systems it is unsuitable to assume that all software layers and components act selfishly. Under this strong assumption, the designer has to consider a complex game among all selfish agents, which has many possible software actions and imprecise cost (utility) in the presence of failures. Moreover, not all games have a predictable outcome; many games have very long stabilization periods and incomplete information games[1] deteriorate the system efficiency. Consequently, designers cannot predict the outcome without a suitable perspective on the various system aspects.

This paper explains how to implement a game authority so that the middleware can recover after the occurrence of transient failures.

1.1. The middleware services

The game authority facilitates interaction among agents of the (higher) application layer, where users control programs. The middleware implements a social structure that relies on the common moral code of the majority of entities, and overcomes the non-moral Byzantine behavior of minorities.[2] Our social structure follows the principle of power separation. The key middleware services are as follows.

- **The legislative service**, which allows agents to set up the rules of the game in a democratic manner, e.g., robust voting [14], which can facilitate a democratic decision about a preferable outcome for the majority.
- **The judicial service**, which audits the agents' actions and orders the executive service to punish agents following their foul play.
- **The executive service**, which executes actions and manages their associated information: publishing utilities, collecting choice of actions, and announcing the play outcome. Moreover, by order of the judicial service, this service restricts the action of dishonest agents.

In this paper, we focus on the implementation of the judicial service. Our design of the game authority is presented in order to demonstrate the proof of existence, rather than the most efficient implementation. Further research can improve the design and allow better scalability (e.g., using auditing, rather than constant monitoring) and perhaps design to fit incomplete information games. Our main goal is to demonstrate the middleware's proof of existence and its potential benefits in the context of distributed systems of selfish computers.

One of the design enhancements that we consider is the recovery from transient faults. Self-stabilizing systems [7,8] can recover after the occurrence of transient faults. These systems are designed to automatically regain their consistency from any starting state of the automaton. The arbitrary state may be the result of violating the assumptions of the game model or the system settings. The correctness of a self-stabilizing system is demonstrated by considering every sequence of actions that follows the last transient fault and is, therefore, proved assuming an arbitrary starting state of the automaton.

1.2. Our contribution

In this paper, we suggest the game authority as a middleware, and demonstrate its existence and its potential benefits in the context of distributed systems of selfish computers.

- **Reducing the price of malice (PoM [21]).** We consider an example of performance degradation due to malicious manipulation. We explain how the game authority can detect manipulation by malicious agents and then punish them for their foul play. This example illustrates the importance of the manipulation resiliency property that the game authority provides.
- **Self-stabilization.** We explain how to implement a game authority that can recover from the occurrence of transient failures.

The new ability of the honest majority to vote for its preferable game requires a new cost criterion that better evaluates games in selfish-computer systems.

- **Multi-round anarchy cost.** We present a new cost criterion that better captures repeated games with selfish agents (that are restricted to playing by the rules). We consider an example of a resource allocation game and study our new criterion: *multi-round anarchy cost*. We demonstrate that our resource allocation game is asymptotic optimal with respect to the new criterion. Thus, the agents' society may choose to play this particular resource allocation game.

---

[1] Game theory uses this term to describe a game where individual agents may not be able to predict (precisely) the effect their actions will have on the other agents.

[2] Assuming standard requirements for a Byzantine agreement, i.e., more than two-thirds of the processes are (selfish but) honest, and authentication utilizes a Byzantine agreement that needs only a majority. Moreover, the communication graph is not partitioned; e.g., there are $2f + 1$ vertex disjoint paths between any 2 processes, in the presence of at most f Byzantine processes.

2. Preliminaries

In this paper, the game theory related definitions and notations follow those of [23]. The computational game model consists of a set, N, of computing and interacting entities, which we call *agents*. We associate every agent with a unique processor and allow the communication system to facilitate their actions.

A *game* $\Gamma = \langle N, (\Pi_i)_{i\in N}, (u_i)_{i\in N}\rangle$ in its strategic form is defined by a finite set of *agents* $N = \{1, \ldots, n\}$, a finite set of *strategies* Π_i (i.e., applicable actions) for each agent $i \in N$, and by a *cost function* (utility) $u_i : \Pi \to \mathbb{R}$ for each agent, where $\Pi \equiv \times_{i\in N}\Pi_i$ is the set of *pure strategy profiles* (PSPs).

The *social cost* of a PSP, $\pi \in \Pi$, is the sum of all *individual costs*, $u_i(\pi)$, of honest agents $i \in N$. The freedom of choice principle implies that a selfish agent $i \in N$ unilaterally deviates from PSP π to a different PSP, π', because its individual cost $u_i(\pi')$ is smaller than $u_i(\pi)$.

A *pure Nash equilibrium* (PNE) is a pure strategy profile $\pi = (\pi_1, \ldots \pi_n)$ such that, for each agent i, $u_i(\pi) \leq u_i(\pi_1, \ldots, \pi'_i, \ldots, \pi_n)$ for any $\pi'_i \in \Pi_i$. A game may not possess a PNE at all. However, if we extend the game to include *mixed strategy* by allowing each agent to choose her/his applicable actions with certain probabilities (and if we extend the cost functions u_i to capture expectation), then an equilibrium is guaranteed to exist [22].

Given a PSP $(\pi_i, \pi_{-i}) \in \Pi$ (pure or mixed), the *strategy* of an agent is denoted by π_i, where $i \in N$. Given π_{-i}, we assume that the best response, π_i, of agent i to π_{-i} can be efficiently computed (in polynomial time). Namely, given π_{-i}, the strategy π_i produces an outcome that is the most favorable to i.

3. The middleware

The task of the game authority is to verify that all agents play the game by its rules. This can be achieved by verifying that all agents follow the rules of the game in each play. Namely, before the start of every play, the agents make sure that there is a majority of (honest but selfish) agents that agree on their cost functions and on the result of the previous play (if there was such a play). The agents then play the game and the system publishes the outcome of the play to all agents. Once the outcome is published, the game authority can audit the actions of the agents and punish the agents that did not play by the rules. We organize the middleware of game authority by using the three services (legislative, judicial and executive).

3.1. The legislative service

A key decision that the legislative service makes is about the rules of the game. In more detail, the service is required to guarantee *coherent game settings*, i.e., all honest agents agree on the game $\Gamma = \langle N, (\Pi_i)_{i\in N}, (u_i)_{i\in N}\rangle$. In particular, the service defines the cost (utility) functions $(u_i)_{i\in N}$.

We note that existing manipulation resilience voting algorithms can facilitate these decisions (see [14]). For the sake of simplicity, we assume that the agents have fixed preferences throughout the system run and consider a predefined game Γ, which the society elects before starting the system. We note that a possible design extension can follow the agents' changing preferences and repeatedly reelect the system's game.

3.2. The judicial service

The task of the judicial service is to audit the actions that the agents take in every play. Moreover, the judicial service orders the executive service to punish agents that do not play honestly.

In more detail, the service is required to guarantee the following: (1) *Legitimate action choice.* Every honest agent $i \in N$ chooses actions π_i only from its set of applicable actions Π_i. (2) *Private and simultaneous action choice.* The choice of actions of all honest agents is taken simultaneously, i.e., the system does not reveal the choice of agent $i \in N$ before all have committed to their actions. (3) *Foul plays.* Action $\pi_i \in \Pi_i$ of agent $i \in N$ is *foul* if π_i is not i's best response to π_{-i}, where (π'_i, π_{-i}) is the PSP of the previous play. The judicial service should instruct the executive service to punish the agents that make foul plays.

3.3. The implementation

We start by considering the cases in which agents follow merely pure strategies before we turn to considering mixed strategies (in Section 5).

The algorithm relies on a Byzantine agreement protocol (BAP) and cryptographic primitives such as commitment schemes (see [4]). Moreover, we use a Byzantine common pulse generator (similar to the one of [11]) to synchronize the different services. Moreover, the Byzantine common pulse generator allows the system to repeat a sequence of activating the different instantiations of the Byzantine agreement protocol.

Requirements (1) to (3) can be guaranteed. Upon a pulse, all agents start a new play of the game that is carried out by a sequence of several activations of the Byzantine agreement protocol. The play starts by announcing the outcome, $\pi \in \Pi$, of the previous play (if there was such a play). Here the Byzantine agreement protocol is used to assure that all agents agree on π. Next, every agent chooses its best response, $\pi'_i \in \Pi_i$, to π_{-i} and uses a commitment scheme in order to announce

its commitment on π_i' without revealing π_i'. Again, we use the Byzantine agreement protocol in order to ensure that all agents agree on the set of commitments. Once all commitments are agreed upon, the agents reveal their PSP of the play, π'. Moreover, all agents audit the PSP of the play and use the Byzantine agreement protocol to agree on the set, $N' \subset N$, of agents that have made a foul play. Lastly, the judicial service orders the executive service to punish N' and to play according to π'.

3.4. The executive service

The task of the executive service is to carry out the agents' actions. The service manages the associated information of the actions: announcing the play outcome, publishing the utilities and collecting the choice of actions. Moreover, by order of the judicial service, this service restricts the action of dishonest agents according to the punishment scheme.

Punishment schemes. Effective punishment is an essential mechanism for reducing the price of malice (PoM [21]) when considering detectable manipulation. However, punishment is useful when there is a price that the dishonest agent is not willing to pay. In other words, a complete Byzantine agent bears any punishment while aiming at maximizing the social cost. Therefore, it seems that the only effective option is to disconnect Byzantine agents from the network (see [6]). We note that there are punishment schemes based on agent reputation or real money deposits. Other approaches consider more elaborate punishment schemes in which dishonest agents can be deterred (see [10]).

We assume that the executive service is trustworthy. This assumption is common in the game theory literature under the name of a *trusted third party*, e.g., the taxation services in mechanism design (see [15]). We note that [1] facilitates such assumptions.

4. Self(ish)-stabilizing

We now shift focus to show an implementation of the game authority that can recover from transient failures and periods during which the agents act upon short-lived myopic logic. We name the combination of these two properties *self(ish)-stabilization*.

The game authority can be coded in the form of a do forever loop that is supported by a self-stabilizing Byzantine pulse synchronization algorithm (that is similar to [11]). Thus, by showing that every service is self-stabilizing, we show that the entire middleware is self(ish)-stabilizing.

It is easy to see that the legislative service is stateless and therefore self-stabilizing. We note that not every judicial service is self-stabilizing, because the Byzantine agreement protocol (BAP) has an internal state (e.g., epoch numbers). We demonstrate that the judicial service can be self-stabilizing by showing the existence of a self-stabilizing Byzantine agreement algorithm. We remark that the executive service is application dependent, and therefore should be made self-stabilizing on a case basis.

The self-stabilizing Byzantine agreement algorithm is a composition of two distributed algorithms. We use the self-stabilizing Byzantine clock synchronization algorithm of [11]. Whenever the clock value reaches the value 1, the self-stabilizing Byzantine agreement algorithm invokes the Byzantine agreement protocol (BAP) of [16,19], for example. We take the clock size, $\log M$, to be large enough to allow exactly one Byzantine agreement, where M is the number of clock values.

Theorem 1. *The algorithm described above is a self-stabilizing Byzantine agreement protocol.*

We call the self-stabilizing Byzantine agreement described above SSBA.

4.1. The settings of a distributed system

We formally describe self-stabilizing distributed systems that stabilize in the presence of Byzantine faults before we turn to demonstrating the proof.

The system consists of a set of computing and communicating entities, which we call *processors*. We denote the set of processors by $\mathcal{P}$, where $|\mathcal{P}|$ is finite. The graph $G(\mathcal{V}, \mathcal{E})$ denotes the communication graph of the system, where $\mathcal{V}$ is the set of processors, and where there is an edge in $\mathcal{E}$ between every pair of processors p_i and p_j that can directly communicate. Every such p_i and p_j are called *neighbors*. We assume that every processor has a unique identifier.

In the proposed system, every processor emulates an agent, which is a program that encodes an agent in a strategic game. The program of a processor p_i consists of a sequence of *steps*. For ease of description, we assume a synchronous model in which all processors execute steps automatically and simultaneously. The *state* s_i of a processor p_i consists of the values of all processor variables (including its control variables such as the value of its program counter).

A *common pulse* triggers each step of p_i. The step starts sending messages to neighboring processors, receiving all messages sent by the neighbors and changing its state accordingly.

A processor is *Byzantine* if it does not follow its program, i.e., does not execute the *agent* or does not participate correctly in implementing the game authority middleware. We assume standard requirements for Byzantine protocols, i.e., more than two-thirds of the processes are (selfish but) honest, and authentication utilizes a Byzantine agreement that needs only

a majority. Moreover, the communication graph is not partitioned, i.e., there are $2f + 1$ vertex disjoint paths between any 2 processes, in the presence of at most f Byzantine processes.

We describe the global state of the system, the system *configuration*, by the vector of the state of the processors $(s_1, s_2, \ldots, s_n)$, where each s_i is the state of processor p_i. We describe the system configuration in the instance in which the pulse is triggered, when there are no messages in transit. We define an *execution* $E = c_0, c_1, \ldots$ as a sequence of system configurations c_i, such that each configuration c_{i+1} (except the initial configuration c_0) is reached from the preceding configuration c_i by an execution of steps by all the processors.

The *task* τ of a distributed system is defined by a set of executions *LE* called *legal* executions. For example, task τ may be defined by the correct game behavior of agents which are carried out according to the rules of the game, and in which the set of *enabled agents* present rational behavior. By enabled agents, we mean that rational agents may decide to punish mischievous agents and disable their foul plays.

Self-stabilizing distributed systems [7,8] can recover after the occurrence of transient faults. The system is designed to automatically regain its consistency from any starting configuration. The arbitrary configuration may be the result of unexpected faults caused by the environment and/or mischievous agents' behavior.

The correctness of a self-stabilizing system is demonstrated by considering every execution that follows the last transient fault and is, therefore, proved by assuming an arbitrary starting configuration. The system should exhibit the desired behavior (of task τ) in an infinitely long period after a finite convergence period.

A configuration c is *safe* with regard to a task τ and to the distributed system if every nice execution that starts from c belongs to *LE*. We say that the algorithm satisfies its task τ when its execution is in *LE* (the property of closure). An algorithm is self-stabilizing if, starting from an arbitrary configuration, it reaches a safe configuration within a finite convergence period (the property of convergence).

4.2. The proof of Theorem 1

We define the set of legal executions, with respect to the task of the self-stabilizing Byzantine agreement protocol (BAP), as the set of executions, *LE*, in which the Byzantine agreement protocol (BAP) properties hold (i.e., termination, validity, and agreement). Let E be an execution, and $c \in E$ be a configuration, such that (1) all the clock values are 1, and (2) the Byzantine agreement protocol (BAP) is at its starting configuration (e.g., identical epoch and round numbers).

Lemma 2 (*Convergence*). *Starting from an arbitrary configuration, we reach a safe configuration within* $O(n^{(n-f)})$ *of clock pulses.*

Proof. Starting from an arbitrary configuration, within an expected $O(n^{(n-f)})$ clock pulses, a configuration c is reached in which all clock values are 1 (see [11]). In the atomic step that immediately follows c, all processes invoke the Byzantine agreement protocol (BAP) before changing their clock value. Hence, c is safe. □

Lemma 3 (*Closure*). *Let E be an execution that starts in a safe configuration. Then, $E \in LE$.*

Proof. Starting from a safe configuration, there is a period of M pulses, in which no process assigns 1 to its clock. During this period, the Byzantine agreement protocol (BAP) is executed for a long enough time that allows exactly one Byzantine agreement. Moreover, at the end of this period, all processes assign 1 to their clocks. Thus, there is an infinite sequence of such periods in which the Byzantine agreement protocol (BAP) reaches Byzantine agreements. □

Lemmas 2 and 3 imply Theorem 1.

5. Auditing mixed strategies

So far, we have explained that the game authority can audit agents that use only pure strategies. In this section we consider mixed strategies. We start by exploring scenarios in which mixed strategies raise troubling questions and then we propose a solution. We explore these scenarios by looking into the well-known game of *matching pennies*.

Matching pennies is a game for two agents, A and B, where each agent has two strategies: heads and tails. The agents choose their actions secretly and then reveal their choices simultaneously. If the pennies match (both heads or both tails), agent A receives 1 from agent B. If the pennies do not match (one heads and one tails), agent B receives 1 from agent A. This game has no PNE, but there is a unique Nash equilibrium in mixed strategies: each agent chooses heads or tails with equal probability. This way each agent makes the other indifferent between choosing heads or tails, so neither agent has an incentive to try a different strategy.

5.1. Hidden manipulative strategies

Suppose that agent B has a hidden manipulation for the heads strategy; the manipulation has no effect on the game whenever the pennies match or when B plays tails. However, whenever the pennies do not match and B chooses the heads strategy with manipulation, then A pays 9 to B. The new game is presented in Fig. 1. Clearly, since agent B knows that agent A plays each of the two strategies with probability $1/2$, then B plays the manipulated heads strategy with probability 1. The manipulation by B is successful, because B is able to increase its expected profit from 0 to 4, while A has decreased its expected profit from 0 to -4.

A/B	Heads	Tails	Manipulate
Heads	$(+1, -1)$	$(-1, +1)$	$(+1, -1)$
Tails	$(-1, +1)$	$(+1, -1)$	$(-9, +9)$

Fig. 1. Matching pennies with a hidden manipulation strategy.

5.2. Validating random choices

The above hidden manipulative strategies can be extended to a more general form. Namely, we consider dishonest agents that deviate from an equilibrium by selecting actions that, according to the game model, should decrease their benefit, i.e., are not the best response. The challenge is in verifying that a sequence of random choices follow a distribution of a credible mixed strategy.

5.3. The solution

In every round of the game, the agents use a private pseudo-random generator for privately selecting actions according to the PSP of the round. We ensure that an action is indeed random by taking Blum's approach [4]. Namely, the agents commit to their PSP using a cryptographic commitment scheme. Before the play and after all agents have received all commitments, the agents publicly reveal their private action selection. Therefore, just before the next play starts, the honest majority can detect any foul play using the Byzantine agreement protocol (BAP).

We note that in our implementation of the judicial service, we take the simplest auditing approach; the agents audit each other's actions in every round of the game. A possible extension can consider any bounded number of rounds. Here, for the sake of efficiency, the agents commit to the private seed that they use for their pseudo-random generator; they reveal their seed at the end of the sequence of rounds and then audit each other's actions. In practice, one may consider several auditing techniques (see [20]) and decide to verify the honest selfishness of agents that raise suspicion among the honest majority.

5.4. Benefit: Reduced price of malice

By auditing the choices of the agents the game authority clearly reduces the ability of dishonest agents to manipulate.

6. Multi-round anarchy cost

The agent society is composed of individuals with different goals and wishes regarding the preferable outcome. The opportunity to select a game that the honest majority prefers shifts the perspective of the distributed system designer. Transitionally, the designer should aim at modeling the system precisely and consider all possible failures. Using the proposed middleware of game authority, the distributed system designer can virtually set the rules of the game, because the game authority can guarantee that these rules are followed.

In this new situation, there is a need to estimate the eventual performance criteria of repeated games, rather than the cost in a particular play, e.g., the price of anarchy (PoA) [18,17] that considers the worst Nash equilibrium and the price of stability (PoS [3]) that considers the best Nash equilibrium.

We consider an example of a *repeated resource allocation* (RRA) scenario in which a consortium of Internet companies shares licenses for advertisement clips on video Web sites. We note that the unpredictable loads on the hosts cause service availability issues. There are many complex ways to model this scenario. One simple way is as follows. In every play, each agent places a (single unit) demand for a resource. We assume that at the end of every play all agents know the load that exists on the resources. The load of a resource determines the time it takes to service the demands for this resource. Every agent wishes to minimize the time it takes to service its demands for the resources that it chooses. We assume that the number of plays is *unknown*, i.e., every play could be the last one. Thus, selfish agents choose resources in an ad hoc manner. In other words, the choices are according to a repeated Nash equilibrium; independent in every round.

Corollary 4 claims that the simplest game of RRA is optimal. Therefore, it could be that the consortium majority prefers backlog size as the host's only selection criterion (and rejects criteria such as video content and attempts of synchronized advertisement). In this case, the game authority can support the agent society's preferences, whereas in the case of more complex selection criteria, the game outcome may be hard to predict, or the multi-round anarchy cost might be higher. The multi-round anarchy cost is defined as the (eventually) expected ratio between the cost of the worst-case equilibrium and of the optimal (centralistic) solution.

Corollary 4 (*Supervised RRA*). *A game authority that supervises the RRA game can guarantee an* $O(1)$ *multi-round anarchy cost.*

Thus, there is a clear motivation for the distributed system designer to use the proposed game authority. By virtually setting these rules, the designer can simplify the protocols and perhaps improve the efficiency of the system.

We now turn to demonstrate Corollary 4 and show that the repeated resource allocation (RRA) game has an asymptotically optimal cost whenever the game authority assures that all agents are honestly selfish. Let B (bins) be a set of resources ($|B| = b > 1$), $\ell_a(k)$ the load of $a \in B$ (after k rounds), $M(k) = \max\{\ell_a(k)\}_{a\in B}$, $m(k) = \min\{\ell_a(k)\}_{a\in B}$, and $EM(k)$

the expectation of $M(k)$ after a sequence $S = \pi(0), \pi(1), \ldots$, where $\pi(k) \in \Pi$ is a result of a Nash equilibrium way to select resources on round k. The k-round cost of anarchy is the ratio $R(k) = SC(k)/OPT(k)$, where $SC(k)$ is the worst-case $EM(k)$ over all possible sequences S, and $OPT(k)$ is the optimal solution. As for the repeated resource allocation, $\Sigma_{a\in B}\ell_a(k) = nk$, $OPT(k) = \lfloor nk \rfloor/b + 1$, and $R(k) \leq SC(k)b/nk$. Lastly, let $R = \lim_{k\to\infty} R(k)$ be the *asymptotic cost of anarchy* (if it exists, $R = \limsup_{k\to\infty} R(k)$).

The initial zero demand for all resources is assumed (when considering the asymptotic behavior of the repeated resource allocation service). Therefore, by information completeness, the loads on every resource are known after k plays and repeated Nash equilibrium is be formed throughout the play.

Theorem 5 (*Replaces Corollary 4*). *When the game authority supervises the repeated resource allocation service, it holds that* $\forall k : R(k) \leq 1 + 2b/k$, *and* $R = 1$.

Proof. For a particular play k, define x_i^a to be the probability that agent i places its demand on resource $a \in B$ ($\Sigma_{a\in B} x_i^a = 1$). Suppose that agent i places its demand on resource a. The expected load on resource $a \in B$ is $\lambda_i^a = 1 + \Sigma_{i\neq j} x_j^a + \ell_a(k)$. (Since agents make independent choices, we use the subscript notation to represent agent i's perspective.)

Lemma 6. $\Delta(k) = M(k) - \ell_a(k) \leq 2n - 1$ ($\forall a \in B$).

Proof. Suppose, in contradiction, that the assertion of the lemma does not hold in round k. Let $k' \leq k$ be the first round at which $\Delta(k) > 2n - 1$, and without loss of generality, assume that $\Delta(k) > 2n - 1$ in any round between k' and k. At round $k + 1$, we denote $a' \in B$ to be a resource with maximal load, and $i_0 \in N$ to be an agent with $x_{i_0}^a, x_{i_0}^{a'} > 0$ ($a, a' \in B$). The Nash equilibrium selection requires that $\lambda_{i_0}^a = \lambda_{i_0}^{a'}$, which implies that $1 + \Sigma_{i\neq i_0} x_i^a + \ell_a(k) = 1 + \Sigma_{i\neq i_0} x_i^{a'} + \ell_{a'}(k)$; $1 + \Sigma_{i\neq i_0} x_i^a = 1 + \Sigma_{i\neq i_0} x_i^{a'} + \Delta(k)$, and $\Delta(k) = \Sigma_{i\neq i_0}(x_i^a - x_i^{a'})$. The lemma is established because $\Sigma_{i\neq i_0}(x_i^a - x_i^{a'}) \leq n - 1$ contradicts $n < \Delta(k)$. □

Thus, no agent supports both a and a' in her/his play, i.e., if a has any support, then all agents place their demand solely on a. By Lemma 6, $(b-1)\Delta(k) \leq (b-1)(2n-1)$, and by the definition of $\Delta(k)$, we get $(b-1)M(k) - \Sigma_{a\neq a'}\ell_a(k) \leq (b-1)(2n-1)$ (denoted as Eq_1). We also know that $\Sigma_{a\in_B}\ell_a(k) = M(k) + \Sigma_{a\neq a'}\ell_a(k)$ and $M(k) + \Sigma_{a\neq a'}\ell_a(k) = nk$ (denoted as Eq_2). By adding equations Eq_1 and Eq_2, we get $bM(k) \leq nk + (b-1)(2n-1)$, which implies that $M(k) \leq (nk + (b-1)(2n-1))/b$. Since $OPT(k) \leq nk/b$, then $R(k) = EM(K)/OPT(k) \leq (nk + (b-1)(2n-1))/nk = 1 + (b-1)(2n-1)/nk \leq 1 + 2b/k$. □

7. Conclusions

Distributed algorithm designers often assume that processes execute identical software. Alternatively, when they do not assume this, designers turn to non-cooperative games. The game authority middleware places itself between these extremes, enabling the majority of the system processes to vote for and enforce the rules of the game.

Interestingly, the experience gained in structuring human society proves that scalability and advancement are gained by promoting honest selfishness and freedom of choice for individuals. The individual participates in forming the infrastructures that establish social rules and in their enforcement. The chosen rules promote competitiveness and individual gain from individual creativity and effort. Creativity and effort are imperative for the success of both individuals and society. Therefore, an honest majority with a beneficial attitude designs the rules in a way that individual success is driven by the individual actions whose outcome advances society.

Our game authority design is a step towards forming computer system structures that are inspired by a successful democratic society. We believe that such a middleware infrastructure is essential for the advancement and scalability of Internet-wide societies.

References

[1] I. Abraham, D. Dolev, J.Y. Halpern, Lower bounds on implementing robust and resilient mediators, in: R. Canetti (Ed.), TCC, in: Lecture Notes in Computer Science, vol. 4948, Springer, 2008, pp. 302–319.
[2] A.S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, C. Porth, Bar fault tolerance for cooperative services, SIGOPS Oper. Syst. Rev. 39 (5) (2005) 45–58.
[3] E. Anshelevich, A. Dasgupta, J.M. Kleinberg, É. Tardos, T. Wexler, T. Roughgarden, The price of stability for network design with fair cost allocation, in: FOCS, IEEE Computer Society, 2004, pp. 295–304.
[4] M. Blum, Coin flipping by telephone a protocol for solving impossible problems, SIGACT News 15 (1) (1983) 23–27.
[5] M. Cagalj, S. Ganeriwal, I. Aad, J.-P. Hubaux, On selfish behavior in csma/ca networks, in: INFOCOM, IEEE, 2005, pp. 2513–2524.
[6] A. Clement, J. Napper, H.C. Li, J.-P. Martin, L. Alvisi, M. Dahlin, Theory of bar games, in: I. Gupta, R. Wattenhofer (Eds.), PODC, ACM, 2007, pp. 358–359.
[7] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (11) (1974) 643–644.
[8] S. Dolev, Self-stabilization, MIT Press, Cambridge, MA, USA, 2000.
[9] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Brief announcement: Game authority for robust and scalable distributed selfish-computer systems, in: PODC'07, 2007, p. 356.
[10] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Strategies for repeated games with subsystem takeovers implantable by deterministic and self-stabilizing automata. Tech. Rep. 2008:11, Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, April 2008.
[11] Shlomi Dolev, Jennifer L. Welch, Self-stabilizing clock synchronization in the presence of byzantine faults, J. ACM 51 (5) (2004) 780–799.
[12] Shlomi Dolev, Elad M. Schiller, Paul G. Spirakis, Game authority: For robust distributed selfish-computer systems, Tech. rep., DELIS, 2006. Accessible via http://delis.upb.de/docs/.

[13] Shlomi Dolev, Elad M. Schiller, Paul G. Spirakis, Game authority: For robust distributed selfish-computer systems, Tech. rep., Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, March 2006.
[14] E. Elkind, H. Lipmaa, Hybrid voting protocols and hardness of manipulation., in: X. Deng, D.-Z. Du (Eds.), ISAAC, in: LNCS, vol. 3827, Springer, 2005, pp. 206–215.
[15] J. Feigenbaum, S. Shenker, Distributed algorithmic mechanism design: Recent results and future directions, in: DIALM '02: Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 2002, pp. 1–13.
[16] J.A. Garay, Y. Moses, Fully polynomial byzantine agreement for processors in rounds, SIAM J. Comput. 27 (1) (1998) 247–290.
[17] E. Koutsoupias, C.H. Papadimitriou, Worst-case equilibria, in: C. Meinel, S. Tison (Eds.), STACS, in: Lecture Notes in Computer Science, vol. 1563, Springer, 1999, pp. 404–413.
[18] E. Koutsoupias, C.H. Papadimitriou, Worst-case equilibria, Comput. Sci. Rev. 3 (2) (2009) 65–69.
[19] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401.
[20] T.F. Lunt, Automated audit trail analysis and intrusion detection: A survey, in: Proceedings of the 11th National Computer Security Conference, 1988, pp. 65–73. URL: http://www.csl.sri.com/papers/survey88/.
[21] T. Moscibroda, S. Schmid, R. Wattenhofer, When selfish meets evil: Byzantine players in a virus inoculation game, in: E. Ruppert, D. Malkhi (Eds.), PODC, ACM, 2006, pp. 35–44.
[22] J. Nash, Equilibrium points in *n*-person games, Proc. National Academy of Sciences of USA 36 (1950) 48–49.
[23] M. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, 1994.