A Game-Theoretic Approach for Elastic Distributed Data Stream Processing

Gabriele Mencagli, University of Pisa

Distributed data stream processing applications are structured as graphs of interconnected modules able to ingest high-speed data and to transform them in order to generate results of interest. Elasticity is one of the most appealing features of stream processing applications. It makes it possible to scale up/down the allocated computing resources on demand in response to fluctuations of the workload. On clouds this represents a necessary feature to keep the operating cost at affordable levels while accommodating user-defined QoS requirements. In this paper we study this problem from a game-theoretic perspective. The control logic driving elasticity is distributed among local control agents capable of choosing the right amount of resources to use by each module. In a first step, we model the problem as a non-cooperative game in which agents pursue their self-interest. We identify the Nash equilibria and we design a distributed procedure to reach the best equilibrium in the Pareto sense. As a second step, we extend the non-cooperative formulation with a decentralized incentive-based mechanism in order to promote cooperation by moving the agreement point closer to the system optimum. Simulations confirm the results of our theoretical analysis and the quality of our strategies.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Design, Theory, Performance

Additional Key Words and Phrases: Autonomic Computing, Data Stream Processing, Elasticity, Game Theory.

ACM Reference Format:
Mencagli, G. 2015. A Game-Theoretic Approach for Elastic Distributed Data Stream Processing. *ACM Trans. Auton. Adapt. Syst.* 9, 4, Article 39 (March 2010), 34 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Distributed data stream processing applications running on clusters, grids and clouds are composed of a plurality of active components (*processing modules* in our terminology) connected via *streams* [Cugola and Margara 2012]. The graph ingests input elements coming from external sources (e.g., cameras, sensors, financial tickers) and the application logic continuously transforms raw input data into final results by following different paths in the graph according to the computation semantics. This vision is representative of a variety of application domains like continuous queries [Andrade et al. 2014], wireless sensor networks [Lea 2007], high-frequency trading [Andrade et al. 2011] and social media [Reuter and Cimiano 2012].

Streaming applications necessitate taking advantage of multiple host machines to support high-throughput processing. To sustain the actual stream pressure, hotspot modules can be internally parallel [Andrade et al. 2014]. The problem of choosing

Author's address: Gabriele Mencagli, Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, I-56127, Pisa, Italy. Phone: +39-050-221-3132. Email: mencagli@di.unipi.it.

DOI: http://dx.doi.org/10.1145/0000000.0000000

the amount of resources to use by each module is exacerbated by the dynamic nature of the streaming context, often featuring time-varying arrival patterns, burstly data rates, with the modules' internal logic exhibiting variable processing times per stream element. Static peak-load configurations are not sustainable for resource consumption [Hummer et al. 2013; Gedik et al. 2014]. Moreover, on cloud platforms the resources are rented on a pay-as-you-go basis, thus choosing the right amount of resources is essential to keep the operating cost at affordable levels [Gohad et al. 2013].

Elasticity is the answer to deal with this variability, by adapting the application configuration to the actual workload at run-time [Hummer et al. 2013]. Most of the existing papers have focused on control strategies supporting elastic scaling for single streaming components or small aggregations of them [Hummer et al. 2013; Gedik et al. 2014; Castro Fernandez et al. 2013]. The problem becomes more complex if we consider arbitrary graph structures, since the decisions taken by the elastic strategy at the level of each single component influence the performance of the other directly or indirectly connected components (the so-called *backpressure* or remote congestion). The goal of this paper is to study formal control strategies dealing with this problem.

The distributed control strategies proposed in this paper are studied from a game-theoretic perspective. Game theory [Vorobjov 1994] captures the interplay between agents, where their decisions are interdependent. Accordingly, each module has its own control logic performed by a local control agent. Agents are interconnected and exchange information in order to reach an agreement in the amount of resources to use. As a first contribution, we propose a *non-cooperative* formulation modeling this problem. Since the agents are responsible for controlling modules of the same application, they are not in conflict. However, they work in a situation of *strategic interaction* in which the outcome/cost of an agent depends also on the decisions of other agents. We characterize the set of Nash equilibria, i.e. the resource allocation strategies where no single agent has an incentive to unilaterally deviate from his actual strategical choice. Furthermore, we propose a distributed procedure to allow the agents to reach the Pareto-optimal Nash equilibrium, which exists in our problem and is the best solution in the non-cooperative setting.

The non-cooperative approach is realistic in multi-tenant scenarios in which different parts of the same application are managed by multiple profit-making entities like in federated cloud systems [Gomes et al. 2012; Li et al. 2013]. In that case the strategy may be unaware of the fact that the received jobs are part of the same application, which usually has global QoS requirements and constraints in the total cost the user is willing to pay for the execution. Therefore, as a second contribution we design a *decentralized incentive-based mechanism* [Park and van der Schaar 2010; Rogerson 1994] where agents change the game parameters by maintaining the non-cooperative nature of their interaction scheme. The mechanism promotes cooperation by making the individual decisions compatible with social welfare optimization.

We conclude this work with experiments performed in a simulation environment, in order to confirm the results of the theoretical analysis in terms of performance, efficiency and cost optimality. Furthermore, we analyze the overhead of the strategies in terms of the amount of exchanged messages needed to reach the desired accuracy.

The outline of this paper is the following. Sect. 2 provides a brief description of some related works in this area. Sect. 3 shows an overview of the problem and the main solution concepts. Sect. 4 introduces a formal definition of our non-cooperative strategy, which will be extended in Sect. 5 with the incentive-based mechanism. Sect. 6 shows a set of simulations by taking into account a distributed streaming application operating in a mobile cloud environment. Finally, Sect. 7 provides the conclusions of this work and our future research directions.

2. RELATED WORK

The major contribution of this paper is the investigation of novel distributed control strategies formalized using a game-theoretic approach. Although game theory is a natural framework in which to formally study multi-agent systems, its application in the domain of stream processing is essentially new at the best of our knowledge. In this section we review the state-of-the-art in order to critically evaluate our contribution.

2.1. Data Stream Processing

Recently, elasticity has been investigated in the data stream processing domain. In [Gedik et al. 2014] the authors have proposed an elastic scaling strategy for stateless and stateful streaming operators (the equivalent of our module concept). The approach is based on heuristics without models to drive the decisions, e.g., the parallelism degree is increased until the congestion is resolved. In the case of a remote congestion due to other operators, the strategy stops to increase the parallelism degree for a certain time period. Therefore, this work lacks in providing a model-based coordination between operators, which is instead our main goal.

An approach for clouds has been described in [Gulisano et al. 2012]. It is based on a centralized manager that monitors all the elastic instances in the system. Centralized approaches are common in the literature. Another example has been described in [Lohrmann et al. 2015] based on queueing theory models to provide latency guarantees. This work is interesting because it takes into account paths of operators instead of single instances. The strategy is based on a centralized controller that receives the monitored data needed to instantiate the model. In [Heinze et al. 2014] the FUGU component is a centralized controller which assigns operators to hosts and derives scaling decisions. These works differ from our approach, where the emphasis is on decomposing the centralized strategy into multiple local communicating agents. We claim that our contribution is definitely important for large-scale distributed applications.

ChronoStream [Wu and Tan 2015] is a framework for stream processing on multitenant platforms. A centralized controller is responsible to scale up/down the system by migrating the state in a transparent fashion in the case of stateful computations. The execution model groups operators into computation slices assigned to the execution. Again, our approach has a different perspective, as the application graph is not only an abstraction to initially define the application, but it represents the execution model where modules are the concurrent activities further parallelizable if necessary.

2.2. Game Theory and Clouds

Game theory has attracted the interest of researchers to solve various resource allocation problems in cloud computing. In [Kwok et al. 2007] a non-cooperative strategy has been proposed for grid computing, in which machines are used selfishly by multiple applications. Other works have followed a similar rationale [Ye and Chen 2013] by proposing game-theoretic strategies to achieve load balancing in systems of parallel cloud servers hosting applications of different users. Similarly, in [Ardagna et al. 2013] a competitive setting has been utilized between multiple SaaS providers that share the same infrastructure (IaaS). The problem is modeled in a game-theoretic framework by providing the existence and the convergence to generalized Nash equilibria. Our work differs from this vision: we study the distributed control of single applications composed of several elastic jobs (modules) that can act selfishly or not.

Cooperative game theory has been applied to manage multi-organization cloud computing environments [Niyato et al. 2011], in which cloud providers offer their services by sharing resources in order to reduce the costs. Pricing mechanisms have been studied to enable cooperative behaviors. In [Zhang and Xiao 2014] cloud providers decide

the resource configuration while a broker entity is in charge of deciding the optimal resource price in a centralized fashion. In contrast, the incentive-based mechanism studied in this paper is fully decentralized, with each agent able to change its incentive parameter such that the agreement point is closer or better equal to the system optimum.

The approach studied in [Meilander et al. 2013] has interesting commonalities with our work. The authors propose a decision-making strategy supported by cost models for real-time online interactive applications (briefly, ROIA) executed on cloud environments (e.g., massively multi-player online games, real-time training and e-learning based on high-performance simulation). The approach models the effect of various load-balancing actions on the application scalability and analyzes the computational overhead of such reconfigurations. Similar approaches have been developed with the goal of introducing *cost models* of reconfiguration activities [Bertolli et al. 2010; 2009]. Typical actions consist in load redistributions among existing resources, or in adding/removing active servers to the actual application configuration. Such actions are usually triggered in response to wide variations in the number of application users. Our work distinguishes for at least two important aspects: first, it can be applied to generic distributed stream processing applications that cover a more general spectrum than ROIA; second, our work investigates decision-making strategies from a game-theoretic perspective, which has not been applied in [Meilander et al. 2013].

3. MOTIVATION AND SOLUTIONS

In this work we focus on elastic stream processing applications executed on distributed environments. Parallelism can be exploited between modules working on different input stream elements in parallel. A stream element is a data structure (e.g., financial ticks, video frames) whose reception triggers the internal processing logic of the module. In the sequel, we use the term *task* to refer to a stream element exchanged between modules. Furthermore, modules that are not able to sustain the actual input stream speed (hotspots) can be internally parallelized by having multiple internal *replicas* working on a subset of the input tasks distributed by a scheduling functionality.

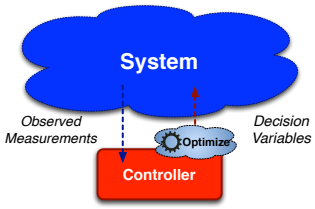
Elasticity consists in the possibility to change the configuration parameters of the processing modules at run-time, in order adapt them to the actual workload. In this paper we focus on the adaptation of the amount of replicas used by each processing module, i.e. its *parallelism degree*. Finding the right amount of resources to use is formulated as an optimization problem modeling the desired trade-off between performance (throughput) and the cost of the resource allocation.

3.1. Principles of Distributed Control

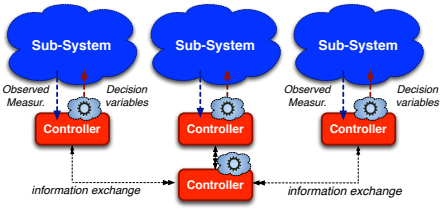
To choose the best values of the configuration parameters (*decision variables*), a self-optimizing system solves an optimization problem [Maggio et al. 2012]. Due to the time variability of the actual execution conditions, the optimization problem needs to be solved periodically, at each *control step* (a fixed time interval). In fact, the execution may be affected by several uncontrollable exogenous factors, such as the arrival rate from input streams and the processing time per stream element, that are modeled as *disturbance variables* outside of the the system control.

In the simplest control structure shown in Fig. 1a, a unique control agent (*controller*) receives monitoring data from the whole system and computes the optimal values of *all* the decision variables. This solution is viable for systems with few decision and monitored variables [Scattolini 2009]. Other solutions decompose the system into a set of subsystems controlled by properly organized *control agents*. *Multi-layer schemes* (Fig. 1b), based on controllers with different degrees of authority, are a very flexible solution which allows different strategies to be used at the different layers of the hier-

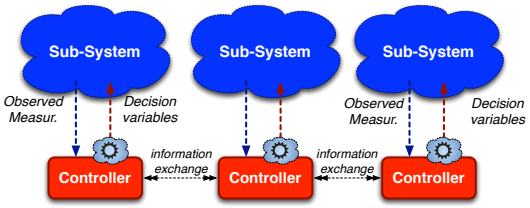
archy. However, higher level controllers need complex models to capture the behavior of lower level controllers, and may be critical for scalability and reliability.



(a) Centralized scheme.



(b) Multi-layer scheme.



(c) Single-layer scheme.

Fig. 1: Single- and multi-agent control structures for distributed systems.

Finally, *single-layer schemes* (Fig. 1c) have a flat organization. All the controllers are at the same level of authority and perform the same strategy in a distributed manner. Local controllers usually exchange data to find an agreement.

3.2. Distributed Control Strategies

In this paper we investigate distributed control strategies for stream processing applications based on single-layer schemes, as sketched in Fig. 2. In this example the control agents are interconnected following the same shape of the graph between processing modules (though their interconnections are bidirectional). Fundamental aspects of this vision are: *i)* controllers communicate only between neighbors; *ii)* controllers have a *partial knowledge* of the system structure and behavior, and at each step they need to exchange information several times until an agreement is reached.

In this work we examine two distributed control strategies. In the first one controllers act in a *non-cooperative fashion* [Vorobjov 1994]. They choose the best values of their decision variables without taking into account the effects their actions have on the other processing modules (the so-called *negative externalities*). This vision is realistic of real-world systems. As an example, in federated cloud environments [Gomes et al. 2012; Li et al. 2013] separated jobs are distributed amongst multiple virtual machines owned by different, independent profit-maximizing entities aimed at provisioning physical resources such that their individual cost is optimized. Here an agreement is a *Nash equilibrium* [Vorobjov 1994]: a configuration of the decision variables such that no controller can improve its cost by changing unilaterally its actions. The equilibria need to be reevaluated periodically by the strategy, because the disturbance variables change the location of the equilibria in the admissible solution set.

There are many cases in which control agents are motivated to act *cooperatively* [Chalkiadakis et al. 2012]. A cooperative controller takes into account the preferences of the other controllers while evaluating its strategy. To do that, a possible solution is to change the reward/cost levels associated with the controllers' actions in

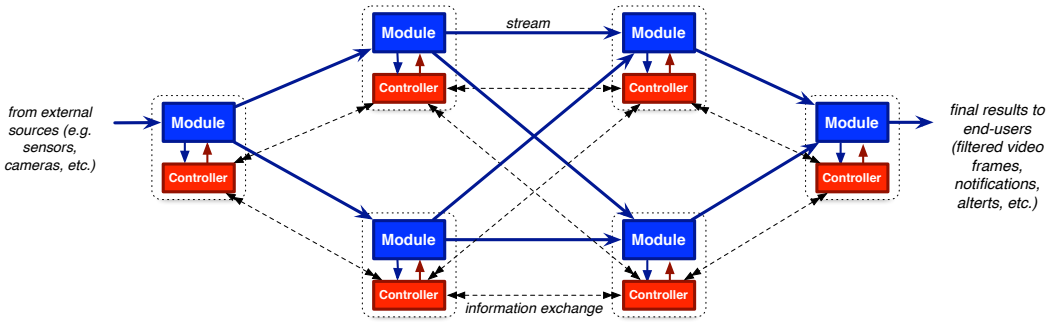


Fig. 2: Flow graph of modules. Each module is coupled with a control agent able to change the local configuration parameters according to the results of the decision-making process.

such a way that the desired cooperative behavior can be enforced. An approach consists in *incentive-based mechanisms*, e.g., pricing, taxation, rewards [Alpcan and Pavel 2009], that promote cooperation by moving the agreement point towards more socially desirable solutions. An example is to reach the solution that optimizes the joint cost of the controllers, i.e. the so-called *system optimum*. Incentive-based mechanisms can be applied by a single centralized authority having a complete vision of the system or in a fully decentralized fashion. In this second case the design of the mechanism is more challenging as controllers have only a partial knowledge.

4. NON-COOPERATIVE CONTROL OF STREAM PROCESSING APPLICATIONS

In this section we analyze the problem of finding the optimal allocation of computing resources to the modules of a stream processing application. We study the problem from a non-cooperative perspective, by characterizing the equilibria and the main limitations of this approach.

4.1. Performance Analysis of Flow Graphs

Let $\mathcal{M}_i$ be a generic module of a flow graph. We consider two performance metrics:

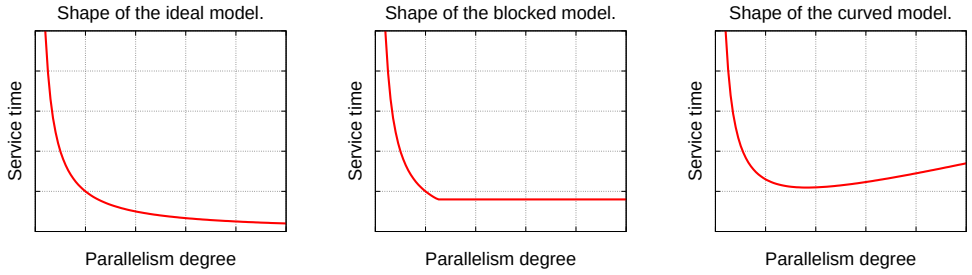
— the *ideal service time* (denoted by $T_{S_{i,k}}$) is the inverse of the average number of stream elements that $\mathcal{M}_i$ is able to serve in a time unit during control step k;
— the *inter-departure time* (denoted by $T_{D_{i,k}}$) is the inverse of the effective output rate (*throughput*) of results transmitted onto the output streams of $\mathcal{M}_i$ during step k, i.e. the average time interval between the generation of two consecutive results*.

The first metric is a measure of the $\mathcal{M}_i$'s performance *in isolation*, i.e. without taking into account the interaction with other modules. Let $T_{i,k}$ be the mean (sequential) *running time* per task during control step k. We model $T_{i,k}$ as a disturbance variable whose value may change over the execution and is monitored by the controller.

We express the $\mathcal{M}_i$'s service time as a function of its parallelism degree $n_{i,k}$ and the mean running time per task $T_{i,k}$. Fig. 3 shows some service time models. The ideal behavior with perfect scalability is represented in Fig. 3a. Possible non-ideal behaviors are depicted in Figs. 3b and 3c. Fig. 3b shows the case in which the service time remains constant for parallelism degrees greater than n_0. This model roughly approximates the behavior of memory-bound computations whose scalability is limited by the memory bandwidth. Fig. 3c shows the case in which the service time has a minimum

*We assume for simplicity that each module produces one result for each consumed input task.

point at n_0. This is possible if some functionalities (e.g., a task scheduler) become a bottleneck with high parallelism degrees. In all the cases the best model is the one that better approximates the profiling measurements. Unless otherwise noted, in the rest of the description we will suppose the ideal behavior, i.e. $T_{S_{i,k}} = T_{i,k}/n_{i,k}$, although our results can be easily generalized to any convex model as the ones shown in Fig. 3.



(a) Ideal model. (b) Blocked model. (c) Curved model.

Fig. 3: Different convex models of the service time of a processing module.

The most used cooperation model for distributed applications is the *message-passing* one [Leopold 2001]: modules interact by transmitting messages on destination buffers with finite capacity. If a module attempts to deliver a message to a full capacity destination buffer, it is forced to wait until the destination has enough space to enqueue the new message. Therefore, a bottleneck throttles down the other processing modules due to *backpressure*. Backpressure is an effect that starts at the bottleneck and propagates backwards through the flow graph. As a result, the inter-departure time of a module can be greater than its service time, i.e. $T_{D_{i,k}} \geq T_{S_{i,k}}$. The backpressure phenomenon originates the strategic interdependence between control subproblems. The effective performance achieved by a module depends on the parallelism degrees chosen by *all* the other modules. The utilization degree of the resources assigned to a module is capture by the *relative efficiency* metric $\varepsilon_{i,k} \in [0,1]$, defined as $\varepsilon_{i,k} = T_{S_{i,k}}/T_{D_{i,k}}$.

The relationship between inter-departure time and service time can be formalized by the model proposed in [Mencagli 2012], used in our prior works [Mencagli et al. 2013b; 2013a; 2014; Mencagli and Vanneschi 2011] and valid for a significant and large class of graphs: *acyclic computations with a single source*. The single-source and the acyclicity assumptions are not so restrictive in practice and allow a simple yet precise modeling. Flow graphs of continuous queries [Babu and Widom 2001] for examples, and the ones from the nesting of algorithmic skeletons [González-Vélez and Leyton 2010], have by construction one single entry point. In the most general case, it is possible to re-design the applications such that the streams generated by external sources are multiplexed and demultiplexed by a single entry point of the graph. Instead, cycles are not taken into account in this work, as they render the performance modeling more complex. Extensions covering this feature will be studied in our future works.

Let $\mathcal{G} = (\mathcal{M}, \mathcal{A})$ be a single-source acyclic directed graph, where $\mathcal{M}$ is the set of the indices of the modules (vertices) $\mathcal{M} = \{1, 2, \ldots, m\}$ ($s \in \mathcal{M}$ is the identifier of the source) and $\mathcal{A}$ is the set of streams (arcs). Each arc $(i,j) \in \mathcal{A}$ is labeled with the probability $p_{i,j}^k \in [0,1]$ that a message is routed to that arc during step k. We denote by $\mathcal{P}_k^{i \to j}$ the probability during step k of traversing any path starting from module $\mathcal{M}_i$ and ending in $\mathcal{M}_j$, where $\Lambda(i,j)$ is the set of the paths from $\mathcal{M}_i$ to $\mathcal{M}_j$. $\mathcal{P}_k^{i \to j}$ is defined

as follows:

$$\mathcal{P}_k^{i \to j} = \sum_{\forall \pi \in \Lambda(i,j)} \left(\prod_{\forall (u,v) \in \pi} p_{u,v}^k \right) \tag{1}$$

We use the following result whose proof can be found in [Mencagli 2012] (chapter 8, page 213):

PROPOSITION 4.1 (INTER-DEPARTURE TIME). *The inter-departure time of a module $\mathcal{M}_i$ with $i \in \mathcal{M}$ during control step k is expressed as follows:*

$$T_{D_{i,k}} = \max\Big\{ f_{i,1}^k\big(T_{S_{1,k}}\big),\ f_{i,2}^k\big(T_{S_{2,k}}\big), \ldots, f_{i,m}^k\big(T_{S_{m,k}}\big) \Big\} \tag{2}$$

Each function $f_{i,j}^k$ returns the lower bound of the inter-departure time of $\mathcal{M}_i$ by supposing $\mathcal{M}_j$ the bottleneck. The functions $f_{i,j}^k$ for $j = 1, 2, \ldots, m$ are defined as follows:

$$f_{i,j}^k(T_{S_{j,k}}) = T_{S_{j,k}} \frac{\mathcal{P}_k^{s \to j}}{\mathcal{P}_k^{s \to i}} \tag{3}$$

Since we do not know which module is the bottleneck, we take the highest lower bound, i.e. the maximum of $f_{i,j}^k$ for $j = 1, 2, \ldots, m$.

It is worth noting that $f_{i,i}^k$ returns the $\mathcal{M}_i$'s service time, as the inter-departure time of a module can never be smaller than its service time. The concept of bottleneck can be formalized as follows:

PROPOSITION 4.2 (BOTTLENECK). *A processing module $\mathcal{M}_b$ with $b \in \mathcal{M}$ is the bottleneck of graph $\mathcal{G}$ if and only if:*

$$T_{D_{i,k}} = f_{i,b}^k\big(T_{S_{b,k}}\big), \quad \textit{for all} \quad i = 1, 2, \ldots, m \tag{4}$$

PROOF. The inter-departure times of all the modules are limited by the bottleneck $\mathcal{M}_b$. According to Prop. 4.1, for all $\mathcal{M}_i$ for $i = 1, 2, \ldots, m$ we have:

$$f_{i,b}^k\big(T_{S_{b,k}}\big) \geq f_{i,j}^k\big(T_{S_{j,k}}\big), \quad \textit{for all} \quad j = 1, 2, \ldots, m$$

Therefore, according to Expr. 2 $T_{D_{i,k}} = f_{i,b}^k\big(T_{S_{b,k}}\big)$ for all $i = 1, 2, \ldots, m$. □

By applying Prop. 4.1 and 4.2 it is easy to prove that the bottleneck always exists. It is also possible that more than one module is a bottleneck. In that case, if $\mathcal{M}_i$ and $\mathcal{M}_j$ are both bottleneck we have that $f_{r,i}^k\big(T_{S_{i,k}}\big) = f_{r,j}^k\big(T_{S_{j,k}}\big)$ for any $r \in \mathcal{M}$.

4.2. Non-cooperative Game and Nash Equilibria

In the description we use a notation similar to the one used in [Saraydar et al. 2002]. Let $\mathbf{NPG}^{\mathcal{K}} = [\mathcal{M}, \{\mathcal{N}_i\}_{i \in \mathcal{M}}, \{\mathcal{J}_i(\cdot)\}_{i \in \mathcal{M}}]$ be the non-cooperative game played at control step k, where $\mathcal{M} = \{1, 2, \ldots, m\}$ is the index set for the cost-minimizing control agents, $\mathcal{N}_i$ is the *strategy set* and $\mathcal{J}_i(\cdot)$ is the cost function of the i-th controller. Each controller selects a parallelism degree $n_{i,k} \in \mathcal{N}_i$, where $\mathcal{N}_i$ is the interval of integers $[1, n_i^{max}]$. Let $\mathbf{n}_k = (n_{1,k}, \ldots, n_{m,k}) \in \mathcal{N}$ be a *strategy profile* where $\mathcal{N}$ is the set of all the admissible vectors. The cost of the i-th controller is expressed as a function of all the decision variables of the players, i.e. $\mathcal{J}_i(\mathbf{n}_k)$. Occasionally, we use the alternative notation $\mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k})$ to highlight that the i-th controller has control only on its decision variable $n_{i,k}$ whereas $\mathbf{n}_{-i,k} \in \mathcal{N}_{-i}$ is the vector of the parallelism degrees of the other players ($\mathcal{N}_{-i}$ is the set of strategy spaces of all the controllers except the i-th one).

In order to make our analysis tractable, we assume that each decision variable $n_{i,k}$ takes positive real-valued numbers in the interval $\mathcal{N}_i = (0, n_i^{max}]$ rather than integer values. $\mathbf{NPG}^{\mathcal{K}}$ becomes a *continuous strategic game* and we can obtain an admissible solution by rounding up the real values to the nearest positive integers. This simplification is reasonable for large-scale applications requiring tens/hundreds of processing nodes, which is a realistic in distributed environments like grids and clouds.

The local cost function of a control agent is designed to take into account the throughput level of the module and the cost for its actual resource allocation. More formally:

Definition 4.3 (*Local cost function*). The i-th controller is aimed at minimizing a cost function defined as follows:

$$\mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k}) = \alpha_i \, T_{D_{i,k}} + \mathcal{C}_i^{op}(n_{i,k}) \tag{5}$$

The function is composed of two parts. The first (*performance cost*) is a cost proportional to the inter-departure time: the higher the inter-departure time (the slower the module) the greater the cost. This part is properly weighted by a fixed price per time unit $\alpha_i > 0$. The second part (*resource cost*) expresses the operational cost related to resource consumption. We model it by a convex function $\mathcal{C}_i^{op} : \mathcal{N}_i \to \mathbb{R}_+$. The convexity assumption is quite general and captures many typical cost models applied in data centers and clouds [Lin et al. 2013; Yuan et al. 2011; Chaisiri et al. 2012]. We refer to the following cost definition:

$$\mathcal{C}_i^{op}(n_{i,k}) = C_{var} + C_{fix} = \beta_i \, \tau \, n_{i,k} \; + C_{fix} \tag{6}$$

This function describes a very general and representative model in which the monetary cost for using computing resources is composed of a fixed and a variable cost. The fixed cost is independent from the type and number of resources used. It may include occupancy, administration and power consumption. The variable cost is determined by a fixed per-usage cost model proportional to the number of rented resources, e.g., the number of cores specified for the application's virtual machines, and the time units τ of a control step. An example is the hourly-based pricing system of Amazon EC2 [Inc 2008]. We denote by $\beta_i > 0$ the price rate in monetary units per unit resource and unit time.

In the non-cooperative game each control agent optimizes its local cost function without taking care of the implications of its actions on the other controllers. The game is formally expressed as follows:

$$(\mathbf{NPG}^{\mathcal{K}}) \quad \min_{n_{i,k} \in \mathcal{N}_i} \mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k}), \;\; \textit{for all} \;\; i \in \mathcal{M} \tag{7}$$

Important strategy profiles are the ones where all the controllers are satisfied with the cost they pay given the parallelism degrees adopted by the others. Such operating points are called *equilibria*. The notion of *Nash Equilibrium* (briefly NE) is the fundamental solution concept of non-cooperative games [Vorobjov 1994; Nash 1951]. At a NE no controller can reduce its cost by making individual changes. This condition is stated as follows:

Definition 4.4 (*Nash equilibrium*). A strategy profile $\mathbf{n}_k = (n_{1,k}, \ldots, n_{m,k}) \in \mathcal{N}$ is a *Nash equilibrium* if for every $i \in \mathcal{M}$ the following conditions are satisfied:

$$\mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k}) \leq \mathcal{J}_i(n'_i, \mathbf{n}_{-i,k}), \;\; \textit{for all} \;\; n'_i \in \mathcal{N}_i$$

The parallelism degree chosen by a controller is its *best response* to the decisions of the others. The best response of the i-th controller is a mapping $\mathcal{B}_i : \mathcal{N}_{-i} \to \mathcal{N}_i$ that

assigns to each $\mathbf{n}_{-i,k} \in \mathcal{N}_{-i}$ a set of parallelism degrees defined as:

$$\mathcal{B}_i(\mathbf{n}_{-i,k}) = \left\{ n_{i,k} \in \mathcal{N}_i : \mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k}) \leq \mathcal{J}_i(n'_i, \mathbf{n}_{-i,k}), \ \textit{for all} \ \ n'_i \in \mathcal{N}_i \right\} \tag{8}$$

Therefore, a vector $\mathbf{n}_k = (n_{1,k}, \ldots, n_{m,k}) \in \mathcal{N}$ is a NE if and only if $n_{i,k} \in \mathcal{B}_i(\mathbf{n}_{-i,k})$ for all $i \in \mathcal{M}$.

To identify the NEs we need to formalize the best responses of the agents. In isolation the inter-departure time of a module always coincides with its service time. The cost function in isolation (denoted by $\overline{\mathcal{J}}_i$) is defined as follows:

$$\overline{\mathcal{J}}_i(n_{i,k}) = \alpha_i \frac{T_{i,k}}{n_{i,k}} + \beta_i \tau n_{i,k} + C_{fix} \tag{9}$$

In the definition we assume the ideal service time model shown in Fig. 3a. The "by-isolation" condition removes the strategic interdependence between controllers and the cost function depends on the parallelism degree $n_{i,k}$ only. The parallelism degree that optimizes the local cost function in isolation is defined as follows:

Definition 4.5 (*Ideal parallelism degree*). The ***ideal parallelism degree*** is a value $n^{\star}_{i,k}$ that minimizes $\overline{\mathcal{J}}_i$. It is determined by nullifying the first-order derivative:

$$n^{\star}_{i,k} = \sqrt{\frac{\alpha_i T_{i,k}}{\beta_i \tau}} \tag{10}$$

If $n^{\star}_{i,k} \notin \mathcal{N}_i$ the function $\overline{\mathcal{J}}_i$ is monotonically decreasing in $\mathcal{N}_i$ and we set $n^{\star}_{i,k} = n_i^{max}$. Therefore, we can denote the ideal parallelism degree vector by $\mathbf{n}^{\star}_k = (n^{\star}_{1,k}, n^{\star}_{2,k}, \ldots, n^{\star}_{m,k}) \in \mathcal{N}$.

In the real execution, a module interacts with other modules of the application and its inter-departure time is lower bounded by a value that depends on the choices of the other agents. Given a strategy profile $\mathbf{n}_k = (n_{1,k}, n_{2,k}, \ldots, n_{m,k}) \in \mathcal{N}$ we denote by $\overline{n}_{i,k}$ the value of the parallelism degree defined as follows:

$$\overline{n}_{i,k} = \frac{T_{i,k}}{T^{min}_{D_{i,k}}} \qquad \text{with} \quad T^{min}_{D_{i,k}} = \max_{j \neq i} f^k_{i,j}(T_{S_{j,k}}) \tag{11}$$

where $T^{min}_{D_{i,k}}$ is the minimum inter-departure time that $\mathcal{M}_i$ can assume due to the other controllers' strategies. The local cost function $\mathcal{J}_i$ is a piecewise-defined function of $n_{i,k}$. The strategies of the other controllers determine the value $\overline{n}_{i,k}$ in which the domain $\mathcal{N}_i$ splits into two regions:

$$\mathcal{J}_i(n_{i,k}, \mathbf{n}_{-i,k}) = \begin{cases} \alpha_i \dfrac{T_{i,k}}{n_{i,k}} + \beta_i \tau n_{i,k} + C_{fix} & \text{if } 0 < n_{i,k} \leq \overline{n}_{i,k} \\ \\ \alpha_i T^{min}_{D_{i,k}} + \beta_i \tau n_{i,k} + C_{fix} & \text{if } \overline{n}_{i,k} < n_{i,k} \leq n_i^{max} \end{cases} \tag{12}$$

In the interval $(0, \overline{n}_{i,k}]$ the module $\mathcal{M}_i$ is the bottleneck and the values of the local cost function are the same of the one in isolation. In the second interval $(\overline{n}_{i,k}, n_i^{max}]$ the cost function grows linearly with the parallelism degree as the inter-departure time does not change. We need to determine the value of $n_{i,k}$ such that $\mathcal{J}_i$ is minimized with a given $\mathbf{n}_{-i,k} \in \mathcal{N}_{-i}$. In the first subdomain the first-order derivative is:

$$\frac{d\,\overline{\mathcal{J}}_i}{d\,n_{i,k}} = -\frac{\alpha_i T_{i,k}}{n_{i,k}} + \beta_i \tau \tag{13}$$

which is negative if $\overline{n}_{i,k} \leq n^{\star}_{i,k}$. Therefore, in the first subdomain the function $\mathcal{J}_i$ is monotonically decreasing if $\overline{n}_{i,k} \leq n^{\star}_{i,k}$. Otherwise the function has a minimum at $n^{\star}_{i,k}$. In the second interval the first-oder derivative is $\beta_i \tau > 0$, thus the function is always strictly monotonically increasing. We can derive the best-response correspondence of a controller in $\mathbf{NPG}^{\mathcal{K}}$:

PROPOSITION 4.6 (BEST RESPONSE). *The controller i's best response is a function in the usual sense defined as follows:*

$$\mathcal{B}_i(\mathbf{n}_{-i,k}) = \min\left\{n^{\star}_{i,k}, \overline{n}_{i,k}\right\} \tag{14}$$

PROOF. The proof derives directly from Expr. 12. Fixed $\mathbf{n}_{-i,k} \in \mathcal{N}_{-i}$ the parallelism degree that minimizes $\mathcal{J}_i$ is given by:

— if $n^{\star}_{i,k} \geq \overline{n}_{i,k}$ the function $\mathcal{J}_i$ is monotonically decreasing in the interval $(0,\, \overline{n}_{i,k}]$ and the minimum is the non-differentiability point $\overline{n}_{i,k}$;
— if $n^{\star}_{i,k} < \overline{n}_{i,k}$ the function has a minimum in $(0,\, \overline{n}_{i,k}]$ which is the point $n^{\star}_{i,k}$.

In conclusion, the *best response* of $\mathcal{M}_i$ is the minimum between $n^{\star}_{i,k}$ and $\overline{n}_{i,k}$. □

In the rest of this section we will analyze the properties of the NEs in $\mathbf{NPG}^{\mathcal{K}}$. We start by defining a very special class of strategy profiles defined as follows:

Definition 4.7 ($\mathcal{E}$*-vectors*). A strategy profile $\mathbf{n}_k \in \mathcal{N}$ is a $\mathcal{E}$-vector if and only if $\varepsilon_{i,k} = 1$ *for all* $i \in \mathcal{M}$, i.e. each module has the maximum relative efficiency.

LEMMA 4.8. *All the Nash equilibria of* $\mathbf{NPG}^{\mathcal{K}}$ *are* $\mathcal{E}$*-vectors.*

PROOF. The proof can be done by contradiction. Suppose that the vector $\mathbf{n}^e_k = (n^e_{1,k}, n^e_{2,k}, \ldots, n^e_{m,k}) \in \mathcal{N}$ is a NE and it is not a $\mathcal{E}$-vector. This means that there exists a module $\mathcal{M}_i$ with an inter-departure time $T_{D_{i,k}}$ higher than the service time $T_{S_{i,k}}$. Therefore, the control agent i can use a value $n'_{i,k} < n^e_{i,k}$ such that its new service time is now equal to its inter-departure time $T'_{S_{i,k}} = T_{D_{i,k}}$. In this way $\mathcal{M}_i$ reduces its resource cost by maintaining the same value of the inter-departure time, thus we have $\mathcal{J}_i(n'_{i,k}, \mathbf{n}_{-i,k}) < \mathcal{J}_i(n^e_{i,k}, \mathbf{n}_{-i,k})$ and according to Def. 4.4 $\mathbf{n}^e_k$ cannot be a NE. □

PROPOSITION 4.9 (IDENTIFICATION OF NES). *A $\mathcal{E}$-vector $\mathbf{n}^e_k \in \mathcal{N}$ is a Nash equilibrium of* $\mathbf{NPG}^{\mathcal{K}}$ *if* $\mathbf{n}^e_k \leq \mathbf{n}^{\star}_k$[†].

PROOF. No controller has the unilateral incentive to *increase* its parallelism degree. At $\mathbf{n}^e_k$ all the modules have a service time equal to the inter-departure time. If the i-th controller deviates unilaterally by using a parallelism degree greater than $n^e_{i,k}$, its new service time becomes smaller without changing the inter-departure time. Therefore, its resouce cost increases without improving the performance, and the local cost function assumes worse values.

To be a NE no controller must also have the unilateral incentive to *decrease* its parallelism degree. If a $\mathcal{M}_i$ deviates unilaterally by using values smaller than $n^e_{i,k}$, its new service time is equal to its new inter-departure time and the module becomes the bottleneck. Therefore, in the interval $(0, n^e_{i,k}]$ the local cost function $\mathcal{J}_i$ coincides with the one in isolation. To not have the unilateral incentive to use smaller parallelism

[†]$\mathbf{x} \leq \mathbf{y}$ means that each component of vector $\mathbf{x}$ is less than or equal to each corresponding component of vector $\mathbf{y}$.

degrees, the function must be monotonically decreasing for values smaller than $n^e_{i,k}$. According to Expr. 12 this is true if and only if $n^e_{i,k} \leq n^\star_{i,k}$. □

PROPOSITION 4.10. *Prop. 4.9 identifies all the NEs of* $\mathbf{NPG}^{\mathcal{K}}$.

PROOF. The proof derives directly from Lemma 4.8 and Prop. 4.9. □

To compare the outcome achieved by different strategy profiles we introduce the following well-known partial ordering relation:

Definition 4.11 (*Pareto dominance*). A vector $\mathbf{n}_k \in \mathcal{N}$ dominates a vector $\hat{\mathbf{n}}_k \in \mathcal{N}$ if, for all $i \in \mathcal{M}$, $\mathcal{J}_i(\mathbf{n}_k) \leq \mathcal{J}_i(\hat{\mathbf{n}}_k)$ and for some $i \in \mathcal{M}$ $\mathcal{J}_i(\mathbf{n}_k) < \mathcal{J}_i(\hat{\mathbf{n}}_k)$. A vector $\mathbf{n}_k$ is Pareto optimal if no other vector dominates it.

A resource allocation is more efficient than another if it is possible to decrease the cost of some of the control agents without hurting any other agent. As it is well known, NEs can be inefficient [Dubey 1986]. This concept is illustrated in Fig. 4 by assuming to have five controllers with their local cost functions. The figure shows the values of the cost functions with three strategy profiles: two inefficient NEs and a Pareto-optimal NE. The two inefficient NEs are not comparable: the second equilibrium is better than the first one expect for the third cost function. The third equilibrium is Pareto-optimal, thus it is no worse (better in this case) than the first two Nash equilibria in all the cost functions.

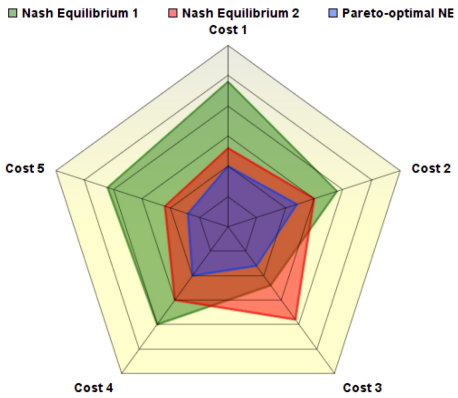
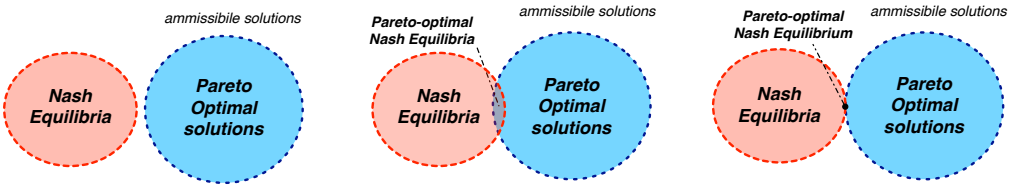


Fig. 4: Illustrative example of Nash Equilibria and Pareto-optimal Nash Equilibria.

In terms of existence of Pareto-optimal Nash equilibria, we can distinguish between three possible scenarios: no NE is Pareto optimal (Fig. 5a), or there exist many Pareto-optimal NEs (Fig. 5b) or exactly one unique Pareto-optimal NE (Fig. 5c).

LEMMA 4.12. *All the Pareto-optimal strategy profiles of* $\mathbf{NPG}^{\mathcal{K}}$ *are* $\mathcal{E}$*-vectors.*

PROOF. Suppose that the vector $\mathbf{n}_k = (n_{1,k}, n_{2,k}, \ldots, n_{m,k}) \in \mathcal{N}$ is Pareto optimal and it is not a $\mathcal{E}$-vector. Therefore, there exists a module $i \in \mathcal{M}$ such that $T_{D_{i,k}} > T_{S_{i,k}}$. Control agent i can use a new value $n'_{i,k} < n_{i,k}$ such that $T'_{S_{i,k}} = T_{D_{i,k}}$. In this way it reduces its resource cost without changing the inter-departure time, thus improving $\mathcal{J}_i$ without hurting the costs of the other controllers. The new vector $\mathbf{n}'_k$ equal to $\mathbf{n}_k$ except that agent i uses $n'_{i,k}$ dominates $\mathbf{n}_k$ which is absurd. □



(a) Scenario 1. (b) Scenario 2. (c) Scenario 3.

Fig. 5: Locations of Nash equilibria and Pareto-optimal solutions into the admissible set.

Therefore, both the set of NEs and the set of Pareto-optimal strategy profiles are subsets of the set of $\mathcal{E}$-vectors. To identify the Pareto-optimal strategy profiles that are also NEs, we introduce the following lemma and propositions:

LEMMA 4.13. *A total order relation exists among the $\mathcal{E}$-vectors.*

PROOF. Let $\mathbf{n}_k \in \mathcal{N}$ be a $\mathcal{E}$-vector. By definition we have $T_{S_{i,k}} = T_{D_{i,k}}$ for all $i \in \mathcal{M}$. According to Prop. 4.1, for any pair $i, j \in \mathcal{M}$ we have:

$$T_{S_{i,k}} = T_{D_{i,k}} \geq f_{i,j}^k(T_{S_{j,k}}) = T_{S_{j,k}} \frac{\mathcal{P}_k^{s \to j}}{\mathcal{P}_k^{s \to i}}$$

That can be rewritten as follows:

$$f_{j,i}^k(T_{S_{i,k}}) = T_{S_{i,k}} \frac{\mathcal{P}_k^{s \to i}}{\mathcal{P}_k^{s \to j}} \geq T_{S_{j,k}}$$

We also have $T_{S_{j,k}} = T_{D_{j,k}}$, thus we can apply the same reasoning:

$$T_{S_{j,k}} = T_{D_{j,k}} \geq f_{j,i}^k(T_{S_{i,k}}) = T_{S_{i,k}} \frac{\mathcal{P}_k^{s \to i}}{\mathcal{P}_k^{s \to j}}$$

Therefore we can conclude that $T_{S_{j,k}} = T_{D_{j,k}} = f_{j,i}^k(T_{S_{i,k}})$ for any pair $i, j \in \mathcal{M}$.

Let $\mathbf{n}_k \in \mathcal{N}$ and $\mathbf{n}'_k \in \mathcal{N}$ be two $\mathcal{E}$-vectors. We want to show that if for some $i \in \mathcal{M}$ $n_{i,k} \leq n'_{i,k}$ then $n_{j,k} \leq n'_{j,k}$ for all $j \in \mathcal{M}$ (the case $n_{i,k} > n'_{i,k}$ is symmetrical). We know that for any $j \in \mathcal{M}$:

$$T_{D_{i,k}} = T_{S_{i,k}} = f_{i,j}^k(T_{S_{j,k}})$$
$$T'_{D_{i,k}} = T'_{S_{i,k}} = f_{i,j}^k(T'_{S_{j,k}})$$

By hypothesis $n_{i,k} \leq n'_{i,k}$, thus we have:

$$T_{S_{i,k}} = \frac{T_{i,k}}{n_{i,k}} \geq T'_{S_{i,k}} = \frac{T_{i,k}}{n'_{i,k}}$$

and thus $T_{D_{i,k}} \geq T'_{D_{i,k}}$. We can write:

$$T_{D_{i,k}} = f_{i,j}^k(T_{S_{j,k}}) = \frac{T_{j,k}}{n_{j,k}} \frac{\mathcal{P}_k^{s \to j}}{\mathcal{P}_k^{s \to i}} \geq \frac{T_{j,k}}{n'_{j,k}} \frac{\mathcal{P}_k^{s \to j}}{\mathcal{P}_k^{s \to i}} = f_{i,j}^k(T'_{S_{j,k}}) = T'_{D_{i,k}}$$

which is true if and only if $n_{j,k} \leq n'_{j,k}$. As a particular case, we derive that if there exists an index $i \in \mathcal{M}$ such that $n_{i,k} = n'_{i,k}$ then $n_{j,k} = n'_{j,k}$ for all $j \in \mathcal{M}$, i.e. the two vectors are identical $\mathbf{n}_k = \mathbf{n}'_k$. □

PROPOSITION 4.14 (PARETO-OPTIMAL STRATEGY PROFILES). *A $\mathcal{E}$-vector $\mathbf{n}_k \in \mathcal{N}$ is Pareto optimal if $\mathbf{n}_k \not\prec \mathbf{n}_k^\star$ and $\mathbf{n}_k \not\succ \mathbf{n}_k^\star$.*

PROOF. For Lemma 4.12 it is sufficient to show that any $\mathcal{E}$-vector $\mathbf{n}'_k \in \mathcal{N}$ cannot dominate $\mathbf{n}_k$. According to the condition $\mathbf{n}_k \not\prec \mathbf{n}_k^\star$ there exists a $i \in \mathcal{M}$ such that $n_{i,k} \geq n^\star_{i,k}$. We distinguish three possible cases:

— if $n_{i,k} = n'_{i,k}$ for Lemma 4.13 the two vectors are identical, i.e. $\mathbf{n}_k = \mathbf{n}'_k$;
— if $n_{i,k} < n'_{i,k}$ then $n'_{i,k} > n^\star_{i,k}$. Since $\mathbf{n}_k$ is a $\mathcal{E}$-vector we have that $T_{D_{i,k}} = T_{S_{i,k}} = T^{min}_{D_{i,k}}$ (see Expr. 11). Therefore $\overline{n}_{i,k} = n_{i,k}$ which is greater or equal than $n^\star_{i,k}$. The same reasoning can be applied to $\mathbf{n}'_k$ and we have $\overline{n}'_{i,k} = n'_{i,k} > n_{i,k} = \overline{n}_{i,k}$. According to Expr. 12 in the interval $[\overline{n}_{i,k}, n_i^{max}]$ the local cost function $\mathcal{J}_i$ is *monotonically increasing*. Since $n_{i,k} < n'_{i,k}$ we have $\mathcal{J}_i(\mathbf{n}_k) < \mathcal{J}_i(\mathbf{n}'_k)$ thus $\mathbf{n}'_k$ cannot dominate $\mathbf{n}_k$;
— the last case is $n_{i,k} > n'_{i,k}$. For the hypothesis $\mathbf{n}_k \not\succ \mathbf{n}_k^\star$ there exists a $j \in \mathcal{M}$ such that $n_{j,k} \leq n^\star_{j,k}$. From Lemma 4.13 if $n_{i,k} > n'_{i,k}$ then $n_{j,k} > n'_{j,k}$ and thus $n^\star_{j,k} > n'_{j,k}$. Since $\mathbf{n}_k$ and $\mathbf{n}'_k$ are both $\mathcal{E}$-vectors, we have $\overline{n}_{j,k} = n_{j,k} > n'_{j,k} = \overline{n}'_{j,k}$. According to Expr. 12 in the interval $(0, \overline{n}_{j,k}]$ the local cost function is the one in isolation, which is *monotonically decreasing* in that interval if $\overline{n}_{j,k} \leq n^\star_{j,k}$ (see Expr. 13). Since $n^\star_{j,k} \geq \overline{n}_{j,k} = n_{j,k} > n'_{j,k}$, we have $\mathcal{J}_j(\mathbf{n}_k) < \mathcal{J}_j(\mathbf{n}'_k)$ thus $\mathbf{n}'_k$ cannot dominate $\mathbf{n}_k$.

In conclusion, $\mathbf{n}_k$ cannot be dominated by any other $\mathcal{E}$-vector. Since all the Pareto-optimal vectors are $\mathcal{E}$-vectors, we can conclude that $\mathbf{n}_k$ is Pareto optimal. □

PROPOSITION 4.15 (PARETO-OPTIMAL NES). *A NE $\mathbf{n}^e_k \in \mathcal{N}$ of $\mathbf{NPG}^{\mathcal{K}}$ is Pareto optimal if there exists at least one control agent $i \in \mathcal{M}$ such that $n^e_{i,k} = n^\star_{i,k}$.*

PROOF. The strategy profile $\mathbf{n}^e_k$ satisfies the condition $\mathbf{n}^e_k \leq \mathbf{n}^\star_k$ (see Prop. 4.9). To be further a Pareto-optimal strategy profile, the vector needs to satisfy the conditions $\mathbf{n}^e_k \not\prec \mathbf{n}^\star_k$ and $\mathbf{n}^e_k \not\succ \mathbf{n}^\star_k$. All these conditions are satisfied if and only if $\exists i \in \mathcal{M} : n^e_{i,k} = n^\star_{i,k}$. □

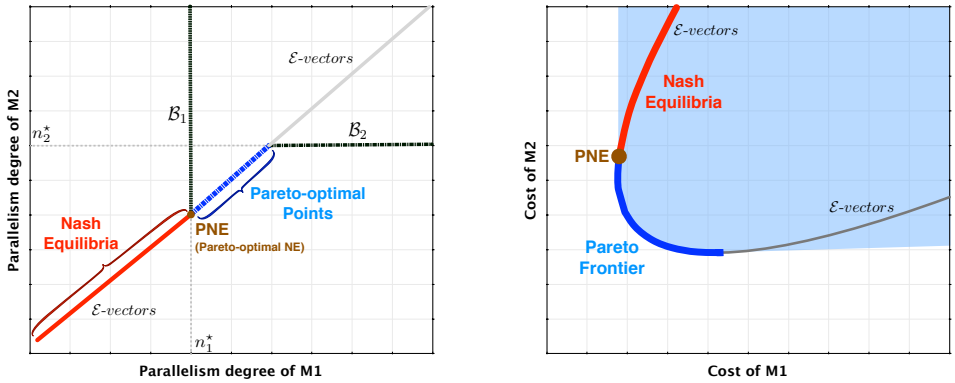
PROPOSITION 4.16 (UNIQUENESS OF THE PNE). *In $\mathbf{NPG}^{\mathcal{K}}$ there exists exactly one NE which is Pareto optimal (denoted by PNE).*

PROOF. Let $\mathbf{n}_k \in \mathcal{N}$ and $\mathbf{n}'_k \in \mathcal{N}$ be two distinct Pareto-optimal NEs of $\mathbf{NPG}^{\mathcal{K}}$. For Prop. 4.15, let $i, j \in \mathcal{M}$ be two indexes such that $n_{i,k} = n^\star_{i,k}$ and $n'_{j,k} = n^\star_{j,k}$. Since the two vectors are distinct, for Lemma 4.13 we have $i \neq j$. Owing to the fact that $\mathbf{n}_k$ and $\mathbf{n}'_k$ are NEs and $\mathcal{E}$-vectors (Prop. 4.9) we can derive:

$$n'_{i,k} < n_{i,k} = n^\star_{i,k}$$
$$n^\star_{j,k} = n'_{j,k} > n_{j,k}$$

which is absurd for Lemma 4.13. Therefore i must be equal to j and the two vectors are identical, i.e. the PNE is unique. □

In conclusion, $\mathbf{NPG}^{\mathcal{K}}$ admits an infinite set of NEs with exactly one NE which is Pareto optimal. Fig. 6a illustrates the best response functions in a game with two control agents that control two processing modules interconnected in a tandem graph, i.e. $\mathcal{M}_1 \rightarrow \mathcal{M}_2$. Both the best response functions are shown in the figure: $\mathcal{B}_1$ is a function of the parallelism degree of the second module while $\mathcal{B}_2$ of the first one. The intersection points of the two plots satisfy the conditions $n_{1,k} = \mathcal{B}_1(n_{2,k})$ and $n_{2,k} = \mathcal{B}_2(n_{1,k})$, and by definition they are the NEs of the game. Inspection of Fig. 6a shows that the intersection is the red line segment marked in bold in the figure. All the points on that segment



(a) Best response functions.

(b) Pareto frontier, NEs and PNE.

Fig. 6: Best responses and location of Nash equilibria, Pareto-optimal points and PNE: example of the $\mathcal{M}_1 \rightarrow \mathcal{M}_2$ graph.

are the NEs of $\mathbf{NPG}^{\mathcal{K}}$, i.e. we have an infinity (in fact a *continuum*) of equilibria, and the unique Pareto-optimal NE is denoted by PNE. The blue segment identifies all the Pareto-optimal points of the game. The union of the red, the blue segments and the gray one represent all the $\mathcal{E}$-vectors of the game.

Fig. 6b shows the outcome in terms of the values of the two functions $\mathcal{J}_1(n_{1,k}, n_{2,k})$ and $\mathcal{J}_2(n_{1,k}, n_{2,k})$ for each admissible point $(n_{1,k}, n_{2,k})$. The figure has been generated using a specific configuration of the cost parameters and the running times per task. Its goal is to exemplify where NEs, the PNE and the $\mathcal{E}$-vectors are located in the space of admissible solutions. In particular, the blue area identifies all the possible outcomes. The curved line corresponds to the outcomes achieved by the set of the $\mathcal{E}$-vectors, with the red part representing the outcomes of the NEs and the blue part the ones of the Pareto-optimal points (the so-called *Pareto frontier*).

4.3. Reaching the Pareto-optimal Nash Equilibrium

In this section we design a distributed procedure which allows the agents to reach the PNE as their agreement point. In fact, the Pareto-optimal NE is the best stable point given the non-cooperative structure of the game. The procedure is an example of *negotiation scheme* [Espinasse et al. 1997; Scattolini 2009], where agents have different goals and need to determine their positions and criteria for an agreement. The algorithm is based on the following characterizing points:

— each controller knows only its own cost function and not those of the other agents;
— decisions are taken *independently* by the agents without any central authority. However, no agent can reach an equilibrium on its own, but some information (*control messages*) needs to be transmitted among the agents;
— control messages are transmitted (and received) from any control agent to a given subset of the others called *neighbors* (agents are *partially interconnected*);
— the procedure is *iterative*. Control messages are transmitted and received by the agents many times within the control step.

As exemplified in Fig. 2, we assume the following interconnection scheme between control agents:

ASSUMPTION 1. *Control agent i is directly interconnected with agent j and vice versa if and only if the two corresponding processing modules $\mathcal{M}_i$ and $\mathcal{M}_j$ are interconnected by a stream in the flow graph.*

More formally, let $\mathcal{G} = (\mathcal{M}, \mathcal{A})$ be a flow graph. For each module $\mathcal{M}_i$ with $i \in \mathcal{M}$ we denote by $\mathcal{H}_i \subseteq \mathcal{M}$ the set of the indices of its neighbors defined as follows:

$$\mathcal{H}_i = \Big\{ j \in \mathcal{M} | \; \exists (j,i) \in \mathcal{A} \vee \exists (i,j) \in \mathcal{A} \Big\} \tag{15}$$

The graph between control agents has the same structure of the flow graph except that arcs are undirected (represent bidirectional communications). It is easy to show that the graph between agents is *connected* (i.e. it has a single connected component).

Since each agent receives control messages only from a subset of the other agents, its inter-departure time is expressed in terms of the service times of the neighbors:

$$T_{D_{i,k}} = \max_{j \in \mathcal{H}_i} \Big\{ f_{i,j}^{k}(T_{S_{j,k}}) \Big\}$$

This model does not capture the presence of all the possible bottlenecks. However, the negotiation will be able to discover and take into account the global bottleneck through a proper propagation of information between control agents, so that all the agents are aware of the presence of a bottleneck in any part of the application after a finite number of information exchanges (*iterations*).

The procedure to locate the PNE is described in Function 1 (`Find-PNE`). Agents do not exchange their parallelism degrees directly, but they advertise the service time resulting from their last parallelism degree proposal.

Function 1: Find-PNE(k, i, $\mathcal{J}_i$)

input : the index of the current control step, the index of an agent and its local cost function.
output : the parallelism degree adopted by agent i at the Pareto-optimal Nash equilibrium (PNE).

1 $n_{i,k}^{(0)} = n_{i,k}^{\star}$; `// ◁ using Def. 4.5.`
2 $T_{S_{i,k}}^{(0)} = T_{i,k}/n_{i,k}^{\star}$; `// ◁ using the service time ideal model.`
3 **for** *q=1* **to** $\mathcal{D}$ **do**
4 **send_to_neighbors**$\Big(T_{S_{i,k}}^{(q-1)}$ to any $j \in \mathcal{H}_i\Big)$;
5 **receive_from_neighbors**$\Big(T_{S_{j,k}}^{(q-1)}$ from all $j \in \mathcal{H}_i\Big)$;
6 $n_{i,k}^{(q)} = \text{optimize}(\mathcal{J}_i)$; `// ◁ Best response.`
7 $T_{S_{i,k}}^{(q)} = T_{i,k}/n_{i,k}^{(q)}$;
8 **return** $n_{i,k}^{(\mathcal{D})}$;

The procedure executed by a generic agent $i \in \mathcal{M}$ proceeds as follows. In the initialization part the agent computes its ideal parallelism degree (row 1) and its initial service time at row 2 (we assume a perfect scalability). The agent sends its service time to the neighbors and receives their service times at rows 4 and 5. Then, the agent computes its best response at row 6. Although data are exchanged only between neighbors, the effect of a local decision must propagate in the graph reaching all the other agents. To this end, the procedure is executed for a number of iterations equal to the *diameter* $\mathcal{D}$ of the graph, i.e. the longest shortest path between any pair of agents.

Alg. 1 shows the pseudocode of the non-cooperative strategy. At the end of the algorithm each agent applies an integer rounding (e.g., to the nearest integer) of the parallelism degree value obtained by `Find-PNE`.

Algorithm 1: NonCoop-Strategy(k, $\{\mathcal{J}_i(\cdot)\}_{i=1}^{m}$)

input : the index of the current control step k, the set of the local cost functions of the agents.
output : every agent determines the new integer parallelism degree for control step k.

1 **foreach** *control agent* $i \in \mathcal{M}$ **do**
2 $n_{i,k} =$ `Find-PNE`$(k, i, \mathcal{J}_i)$;
3 agent i applies an integer rounding of $n_{i,k}$;

PROPOSITION 4.17. *The set of the parallelism degrees returned by the* `Find-PNE()` *procedure executed by all agents simultaneously is the PNE of* $\mathbf{NPG}^{\mathcal{K}}$.

PROOF (SKETCH). In the initialization phase (row 1) all the agents set their initial decision to their ideal parallelism degree. Let $\mathcal{M}_b$ with $b \in \mathcal{M}$ be the bottleneck according to Prop. 4.2. The idea of the proof is based on the following property: at each iteration $q \geq 1$ each agent $i \in \mathcal{M}$ adapts its parallelism degree to the modules at distance at most q from it. In the first iteration $q = 1$ we have:

$$T_{S_{i,k}}^{(1)} = \max_{j \in \mathcal{H}_i \cup \{i\}} \left\{ f_{i,j}^k(T_{S_{j,k}}^{(0)}) \right\}$$

If $b \in \mathcal{H}_i$, i.e. $\mathcal{M}_b$ is in the neighborhood of $\mathcal{M}_i$, we have:

$$T_{S_{i,k}}^{(1)} = f_{i,b}^k(T_{S_{b,k}}^{(0)})$$

After just one iteration the service time (and thus the parallelism degree) of $\mathcal{M}_i$ adapts to the bottleneck of the flow graph. If $b \notin \mathcal{H}_i$, we repeat the reasoning for $q = 2$. In this case we can write:

$$T_{S_{i,k}}^{(2)} = \max_{j \in \mathcal{H}_i \cup \{i\}} \left\{ f_{i,j}^k(T_{S_{j,k}}^{(1)}) \right\} = \max_{j \in \mathcal{H}_i \cup \{i\}} \left\{ f_{i,j}^k \left(\max_{p \in \mathcal{H}_j \cup \{j\}} \left\{ f_{j,p}^k(T_{S_{p,k}}^{(0)}) \right\} \right) \right\}$$

If $\mathcal{M}_b$ is at distance 2 from $\mathcal{M}_i$ we have:

$$T_{S_{i,k}}^{(2)} = \max_{j \in \mathcal{H}_i \cup \{i\}} \left\{ f_{i,j}^k \left(f_{j,b}^k(T_{S_{b,k}}^{(0)}) \right) \right\}$$

For each $j \in \mathcal{H}_i$ we can write:

$$T_{S_{i,k}}^{(2)} = T_{S_{b,k}}^{(0)} \frac{\mathcal{P}_k^{s \to b}}{\mathcal{P}_k^{s \to j}} \frac{\mathcal{P}_k^{s \to j}}{\mathcal{P}_k^{s \to i}} = T_{S_{b,k}}^{(0)} \frac{\mathcal{P}_k^{s \to b}}{\mathcal{P}_k^{s \to i}} = f_{i,b}^k(T_{S_{b,k}}^{(0)})$$

The reasoning can be easily repeated for any $q > 0$. After $\mathcal{D}$ iterations all the control agents adapt to the bottleneck at any distance. The final set of parallelism degrees is a $\mathcal{E}$-vector with one agent using its ideal parallelism degree (agent b) and the others choosing a parallelism degree smaller than or equal to their ideal one. According to Prop. 4.14 this vector is the PNE. □

The completion time and the total number of exchanged messages can be determined as follows:

$$Time = \mathcal{D} \cdot \overline{\mathcal{H}} \qquad \text{and} \quad N_{msg} = \mathcal{D} \cdot \sum_{i=1}^{m} \left|\mathcal{H}_i\right| = \mathcal{D} \cdot 2\left|\mathcal{A}\right|$$

The average completion time is proportional to the graph diameter multiplied by the average number of neighbors per agent (i.e. the degree of a vertex) denoted by $\overline{\mathcal{H}} = \sum |\mathcal{H}_i|/m$. The number of messages is equal to the graph diameter multiplied by the total messages transmitted per iteration of the for loop (the sum of the degree of all the vertices is $2\left|\mathcal{A}\right|$ for handshaking lemma). Tab. I shows the time and message complexity of the non-cooperative strategy by distinguishing between *dense flow graphs*, with typical diameter $\mathcal{D} = \mathcal{O}(\log m)$ and average degree $\overline{\mathcal{H}} = \mathcal{O}(m)$, and *sparse flow graphs* with $\mathcal{D} = \mathcal{O}(m)$ and $\overline{\mathcal{H}} = \mathcal{O}(\log m)$.

	Dense graphs	***Sparse graphs***
Time	$\mathcal{O}(m \log m)$	$\mathcal{O}(m \log m)$
Messages	$\mathcal{O}(m^2 \log m)$	$\mathcal{O}(m^2)$

Table I: Time and message complexity of the non-cooperative strategy.

Sparse graphs are realistic of many real-life streaming applications. Examples are the Vwap, Lois and LinearRoad applications studied in [Tang and Gedik 2013], composed of tens of modules with relatively few interconnections among them. Dense graphs represent a worst case.

5. DISTRIBUTED CONTROL WITH COOPERATION

In every instance of $\mathbf{NPG}^{\mathcal{K}}$ each control agent is aimed at minimizing its own cost function and choosing its optimal parallelism degree. So doing, each agent ignores the harm it imposes on the other agents. To address this issue, *pricing-based* and *incentive-based mechanisms* [Alpcan and Pavel 2009] have been applied in a wide number of applications of Game Theory in domains ranging from peer-to-peer networks [Park and van der Schaar 2010] to economics [Rogerson 1994]. Such mechanisms try to improve the quality of the equilibria in the Pareto sense. In the previous section we have designed a procedure to converge to an agreement point which is the PNE strategy profile. The goal of the incentive-based mechanism is to move such agreement point closer to the *system optimum*, i.e. the strategy profile optimizing the so-called *social welfare* defined as the sum of the costs of the agents, i.e. $\mathcal{J}_G(\mathbf{n}_k) = \sum_{i=1}^{m} \mathcal{J}_i(\mathbf{n}_k)$.

Clearly, the system optimum is a Pareto optimal strategy profile because no other point yields to a strictly lower joint cost (the proof is straightforward and we omit it for brevity). The so-called "price of stability" ($\mathcal{P}o\mathcal{S}$) is the metric used to quantify how NEs are socially inefficient. It is the ratio between the social welfare with the best equilibrium (the unique PNE in our case) and the one at the system optimum.

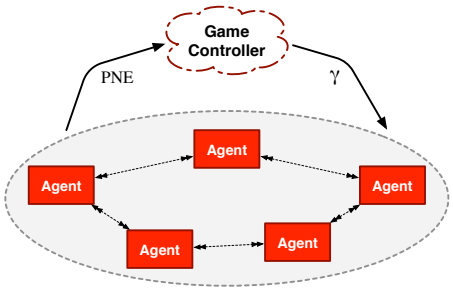
Instead of being an inherent feature of the game, the $\mathcal{P}o\mathcal{S}$ stems from the arbitrary choice of the game parameters [Alpcan and Pavel 2009]. This is also true in our $\mathbf{NPG}^{\mathcal{K}}$. The identification of $\mathcal{E}$-vectors does not depend on the parameters $\{\alpha_i, \beta_i\}_{i=1}^{m}$ of the cost functions. Instead, these parameters change the location of NEs within the $\mathcal{E}$-vectors set. The idea of incentive-based mechanisms follows this intuition in spirit, by changing some of the game parameters in order to locate NEs to a more desired region.

5.1. Decentralized Incentive-based Mechanism

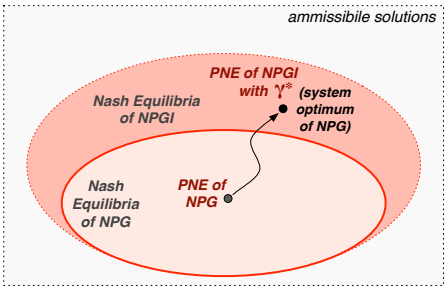
We develop a non-cooperative game with incentives ($\mathbf{NPGI}^{\mathcal{K}}$). With respect to $\mathbf{NPG}^{\mathcal{K}}$ each control agent is associated with a local cost function $\mathcal{J}_i^c : \mathcal{N} \times \mathbb{R} \to \mathbb{R}$, $i = 1, 2, \ldots, m$, parameterized by a scalar *incentive parameter* $\gamma_i \in \mathbb{R}$. For the moment being we suppose no restriction in the domain of this parameter. The incentive-based mechanism is *asymmetric* since each agent has a own incentive parameter different from the others. Let be $\gamma \in \mathbb{R}^m$ the *incentive vector*. By assuming that the new game for any $\gamma \in \mathbb{R}^m$ has a unique Pareto-optimal NE (as $\mathbf{NPG}^{\mathcal{K}}$), we define the *game mapping* $\mathcal{Z}$ as a function that maps incentive vectors onto PNE points:

$$\mathcal{Z} : \mathbb{R}^m \to \mathcal{N} \qquad \textit{and} \qquad \mathcal{Z}^{-1} : \mathcal{N} \to \mathbb{R}^m \tag{16}$$

This function is not explicitly expressible in general. However, the parameterized game can be used in a dynamic incentive scheme as illustrated in Fig. 7a. Ideally, a conceptually centralized logic executed by a *game controller* is in charge of changing the values of the incentive parameters in order to move the location of the PNE point to a better $\mathcal{E}$-vector. The game controller and the agents constitute a sort of "outer feedback loop" in which the first one decides the values of the incentive parameters (it does not have direct access to all the system parameters like running times per task, arcs probability, and so forth) and receives from the agents the new PNE point and the value of the social welfare at that point. The loop is repeated until the social welfare cannot be further bettered, i.e. we have found a vector $\gamma^* \in \mathbb{R}^m$ s.t. the PNE of $\mathbf{NPGI}^{\mathcal{K}}$ coincides with the system optimum of the original game, as sketched in Fig. 7b.



(a) Hypotetical game controller and control agents.



(b) Moving the PNE to a different location.

Fig. 7: Exemplification of the incentive-based mechanism.

However, such centralized locus of control cannot physically exist, because it represents a centralization point that contradicts our fully distributed approach. The idea is to decentralize the central logic of the game controller among the control agents in such a way that each agent is able to change its incentive parameter in such a way that the cooperative behavior can be enforced in a distributed way.

In $\mathbf{NPGI}^{\mathcal{K}}$ we assume that the i-th agent minimizes a cost function with the following structure that includes a reward given by an incentive function $r_i : \mathcal{N} \times \mathbb{R}_+ \to \mathbb{R}_+$:

$$\mathcal{J}_i^c(\mathbf{n}_k, \gamma_i) = \mathcal{J}_i(\mathbf{n}_k) - r_i(\mathbf{n}_k, \gamma_i) \tag{17}$$

Our goal is to choose an incentive function such that the mechanism is *incentive compatible*, i.e. the PNE point of $\mathbf{NPGI}^{\mathcal{K}}$ with a proper vector γ should be able to achieve a better social welfare than the PNE point of the original game without incentives. It

is worth noting that the social welfare is always calculated as the sum of the local cost functions of the original game $\mathbf{NPG}^{\mathcal{K}}$, i.e. $\{\mathcal{J}_i(\cdot)\}_{i=1}^{m}$ that is the ones that the user want to optimize.

The specific incentive scheme must be tailored for the problem at hand. According to the formulation described in Sect. 4.2, we know that the best response of any agent can never be greater than its ideal parallelism degree (see Prop. 4.6). To move the agreement closer to the system optimum, some agents should be able to choose a parallelism degree greater than their ideal one, thus adopting a control action not optimal from a selfish perspective. Motivated by the cost structure shown in Expr. 5 and 6, it is sufficient to use a *linear* incentive function of the form:

$$r_i(n_{i,k}, \gamma_i) = \gamma_i \, \tau \, n_{i,k} \tag{18}$$

That is a *usage-based incentive function* in which the received reward is proportional to the amount of resources used by the agent. Furthermore, as a result of this choice, we can note that all the propositions discussed in Sect. 4.2 (e.g., best responses, uniqueness of the PNE) are still valid for the new game with incentives, which has essentially the same structure of $\mathbf{NPG}^{\mathcal{K}}$ with different cost parameters. In addition we have:

$$\lim_{\gamma_i \to \beta_i} \frac{d\,\overline{\mathcal{J}_i^c}}{d\,n_{i,k}} = \frac{d\,(\overline{\mathcal{J}_i} - r_i)}{d\,n_{i,k}} = \sqrt{\frac{\alpha_i \, T_{i,k}}{(\beta_i - \gamma_i)\,\tau}} = +\infty$$

Therefore, with a proper choice of $\gamma_i \in \mathbb{R}$ the i-th agent is incentivated to use a parallelism degree arbitrarily greater than the ideal one. In conclusion, a proper selection of the vector $\gamma \in \mathbb{R}^m$ allows to move the PNE of $\mathbf{NPGI}^{\mathcal{K}}$ to any Pareto-optimal strategy profile of $\mathbf{NPG}^{\mathcal{K}}$, thus also to the system optimum of the original problem.

5.2. Reaching the System Optimum

At the beginning of each step the agents play many times (*rounds*) the game $\mathbf{NPGI}^{\mathcal{K}}$ with different values of the incentive vector. Fig. 8 exemplifies the mechanism in a case of four agents. At the first round every agent i executes the `Find-PNE` procedure with $\gamma_i = 0$ to determine the PNE of the original game $\mathbf{NPG}^{\mathcal{K}}$. At the end of the procedure there exists one agent choosing its ideal parallelism degree (see Prop. 4.15). According to Prop. 4.2 this module is the bottleneck of the flow graph, e.g., the third module in the example of Fig. 8. A better social welfare might be obtained if this module would choose a parallelism degree greater than its ideal one and the other agents would properly adapt to it. However, this is not possible in a purely non-cooperative setting, because for Prop. 4.6 any parallelism degree greater than the ideal one does not pursue the self-interest of the agent. Therefore, that agent increments by a positive stepsize Δ_3^c the value of its incentive parameter γ_3 in order to increase the value of its ideal parallelism degree. Then, the $\mathbf{NPGI}^{\mathcal{K}}$ is played again (`Find-PNE`) by locating the new PNE at a different (greater) $\mathcal{E}$-vector. If the system achieves a better social welfare than the one obtained at the end of the previous round, the agent of the new bottleneck (which can be a different module than the previous round) tries to further improve the social welfare by incrementing its incentive parameter. Otherwise the procedure stops and each agent applies an integer-rounding of the final parallelism degree obtained.

The procedure does not have any centralization point as each agent is in charge of autonomously checking whether it is the bottleneck and in that case incrementing its γ_i. However, a global feedback is still necessary to check if the social welfare has been improved in the last round or not. The social welfare is an aggregate information that needs to be determined by the agents according to a proper protocol. Centralized protocols assign responsibility to a specific agent in charge of collecting the values

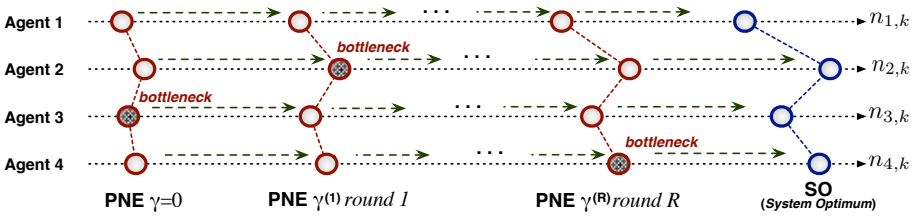


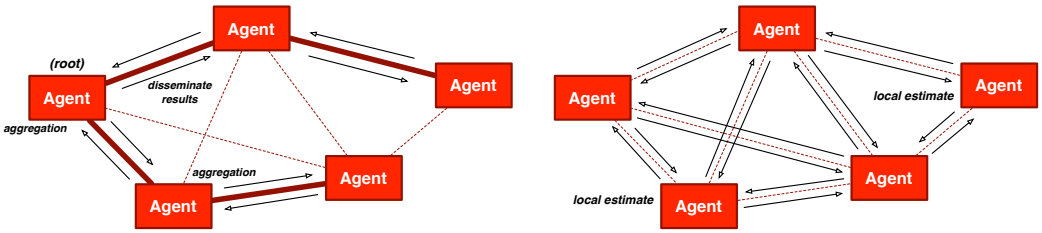
Fig. 8: Incentive logic: starting from the PNE with $\gamma = 0$ the agents play repeatedly $\mathbf{NPGI}^{\mathcal{K}}$ with a different $\gamma \in \mathbb{R}^m_+$.

of the local costs, computes the sum and spreads it over the graph. Such approaches are hampered by scalability and reliability issues due to the presence of a leader entity. Moreover, they require multi-hop communications. We tackle the problem in a different way by using approaches inspired by two prevailing solutions studied in the literature:

— *tree-based protocols* [Makhloufi et al. 2014; Prieto and Stadler 2007] use spanning trees to collect aggregate information. The computation is performed hierarchically in a bottom-up fashion. Agents collect measurements from their children, compute a partial aggregate and transmit it to their parent. The same spanning tree is used to broadcast the global result from the root to all the agents;
— *gossip-based protocols* [Makhloufi et al. 2009; Akdere et al. 2006; Kempe et al. 2003] do not require any particular structure. Each agent maintains an estimate of the global aggregate and exchanges/combines this value with other partners several times until convergence. The protocols may require several iterations to converge and the final approximate result is available directly on all the agents.

Tree-based protocols are particularly suitable for our purpose, because flow graphs have usually a fixed topology and the spanning tree can be computed once when the application starts (Fig. 9a). The latency is linear in the diameter of the graph (for minimum height spanning trees), while the messages are linear in the number of agents.

We also study gossip-based protocols for their structureless nature and for the symmetricity of the agents role. In most of the existing protocols [Makhloufi et al. 2009; Akdere et al. 2006; Kempe et al. 2003] agents exchange their estimates with a set of randomly chosen partners. When the partner is not an immediate neighbor, multi-hop communications are needed. In Sect. 6 we will use the *Local Push-Sum* protocol [Geibig and Bradler 2010] studied for locality-aware sparse networks (Fig. 9b). The local estimates are exchanged only between immediate neighbors by avoiding multi-hop communications. The protocol converges after a number of iterations $\mathcal{I} > 0$ proportional to the graph diameter. Further details can be found in [Geibig and Bradler 2010].



(a) Tree-based aggregation.

(b) Gossip-based aggregation.

Fig. 9: Tree-based and gossip-based protocols for decentralized aggregation.

Alg. 2 shows the incentive-based strategy. At each round the agents execute `Find-PNE` to determine the PNE with the actual incentive vector, compute the sum of the original local costs $\{\mathcal{J}_i(\cdot)\}_{i=1}^m$ using `Aggregate-Protocol()` and check whether the social welfare has been bettered or not. The `idealDegree`(γ_i) formula is defined as follows:

$$\texttt{idealDegree}(\gamma_i) = \min\left\{\sqrt{\frac{\alpha_i\, T_{i,k}}{(\beta_i - \gamma_i)\,\tau}}, n_i^{max}\right\}$$

The algorithm stops whether the current incentive vector does not improve the value of the social welfare or we have reached the maximum number of rounds $\mathcal{R} > 0$.

Algorithm 2: Incentive-based-Strategy(k, $\{\mathcal{J}_i(\cdot)\}_{i=1}^m$)

input : the index of the current control step k, the set of the local cost functions of the agents.
output : every agent determines the new integer parallelism degree for control step k.

foreach *control agent* $i \in \mathcal{M}$ **do**
  `round` $= 0$, $\gamma_i = 0$, $n_{i,k} = 0$;
  `globalCost` $= +\infty$;
  do
    `round` $=$ `round` $+ 1$;
    $n_{i,k}^{old} = n_{i,k}$;
    `oldGlobalCost` $=$ `globalCost`;
    if $n_{i,k} ==$ `idealDegree`(γ_i) **then**
      $\gamma_i = \gamma_i + \Delta_i^c$; // $\triangleleft$ agent i is the bottleneck.
    $n_{i,k} =$ `Find-PNE`$(k, i, \mathcal{J}_i^c)$;
    `globalCost` $=$ `Aggregate-Protocol`$(\mathcal{J}_i, n_{i,k})$;
  while *((*`globalCost` $<$ `oldGlobalCost`*)* and *(*`round` $< \mathcal{R}$*))*;
  agent i applies an integer rounding of $n_{i,k}^{old}$;

Tab. II shows the complexity in the worst case in which the algorithm stops after reaching the maximum number of rounds. The complexity is the same in terms of orders of magnitude between using the tree-based and the gossip-based protocol for the aggregation, since in both the cases it is linear in the graph diameter. As we will see in the experiments, the incentive-based strategy with gossiping requires more rounds and messages due to a higher multiplicative factor in its complexity.

	Dense graphs		***Sparse graphs***	
	Tree-based	Gossiping	Tree-based	Gossiping
Time	$\mathcal{O}(\mathcal{R}\, m \log m)$	$\mathcal{O}(\mathcal{R}\, m \log m)$	$\mathcal{O}(\mathcal{R}\, m \log m)$	$\mathcal{O}(\mathcal{R}\, m \log m)$
Messages	$\mathcal{O}(\mathcal{R}\, m^2 \log m)$	$\mathcal{O}(\mathcal{R}\, m^2 \log m)$	$\mathcal{O}(\mathcal{R}\, m^2)$	$\mathcal{O}(\mathcal{R}\, m^2)$

Table II: Time and message complexity of the incentive-based strategy.

6. SIMULATION RESULTS

We evaluate our decision-making strategies on the OmNeT++ discrete event simulator[‡]. OmNeT is an open-source simulation environment offering several features to

[‡]Visit http://www.omnetpp.org/ for further details about the OmNeT++ simulator. The source code of the experiments will be made available at http://www.di.unipi.it/∼mencagli/omnet-sim.zip.

facilitate computer network modeling. Over the last years the simulator has been extended to reproduce the behavior of grids and clouds [Núñez et al. 2012] and it has been used in some of our past publications [Mencagli et al. 2014; 2013b; 2013a].

We use OmNeT to simulate flow graphs of streaming modules. The idea is to reproduce the backpressure due to the presence of hotspots. In the simulator a module consists of two interconnected simulation objects: *i)* an operating part implements the internal functional logic of the module; *ii)* a control part implements the agent executing the local decision strategy. OmNeT objects are programmed using an event-driven programming style and they exchange messages through communication ports.

In the current version of the simulator the operating part reproduces the behavior of internally parallel data stream operators [Hummer et al. 2013; Gedik et al. 2014]. The operating part mimics the behavior of multiple operator replicas inside the module, interfaced with the input and the output streams through a distributor and a collector entity. The distributor manages a *queue* where input tasks received from the sender modules are temporarily stored before being distributed to the replicas according to some policies that depend on the operator semantics (e.g., round-robin or hash-based distributions). Accordingly, each input task is distributed to one of the replicas. The simulator allows the control part to change the actual number of replicas within the operating part of the module, i.e. the current parallelism degree.

The control part has an internal notion of control step emulated by the reception of a self-message generated by an internal timer. Reconfigurations are implemented as modifications of the parallelism degree attribute of the operating part.

6.1. Application Description

We study a streaming application targeting mobile cloud computing platforms [Yang et al. 2013] (briefly, MCC). The MCC paradigm is based on the integration between mobile devices and cloud resources used to offload resource-intensive tasks. Examples are mobile stream processing applications which retrieve data from cameras, sensors and perform perception related tasks like object and face recognition to augment the original scene in the display of the final user. As studied in [Yang et al. 2013], these applications need elastic use of resources for two main reasons: *i)* they require to sustain the actual input rate to avoid losing important details; *ii)* computer vision algorithms are usually time-consuming and require parallel implementations.

The test-bed application is an object recognition service designed to be hosted on cloud resources and interfaced with mobile users. The application is inspired to the one described in [Yang et al. 2013] and consists in a flow graph of five main modules interconnected as depicted in Fig. 10.

The `Dispatcher` module multiplexes several input streams of digital images (tasks) received from outside mobile sources. It detects the amount and the type of noise affecting the received images and dispatches them either to the second module or the third one. The second module (`Denoiser-1`) applies a variation of the well-known *NL-mean* filter [Buades et al. 2005] working well with imaged affected by Gaussian noise. The third module (`Denoiser-2`) applies a weighted median filter [Gayathri and Sabeenian 2013] suitable for images mainly affected by impulse noise. The probability to dispatch images to one of the two denoisers is supposed to be equiprobable. The `Edge-Detector` module applies a suite of edge detection algorithms (Candy, Sobel, Marr-Hildreth) based on the image characteristics. The `Recognizer` compares the shapes obtained by the previous phase with a database of possible candidates.

The simulator abstracts some of the details of the real application while focusing on the relevant aspects for the evaluation. In the simulation the running time per image of each module is modeled by a normally distributed random variable. According to the default setting of the OmNeT simulator, the pseudo-random number generator used in

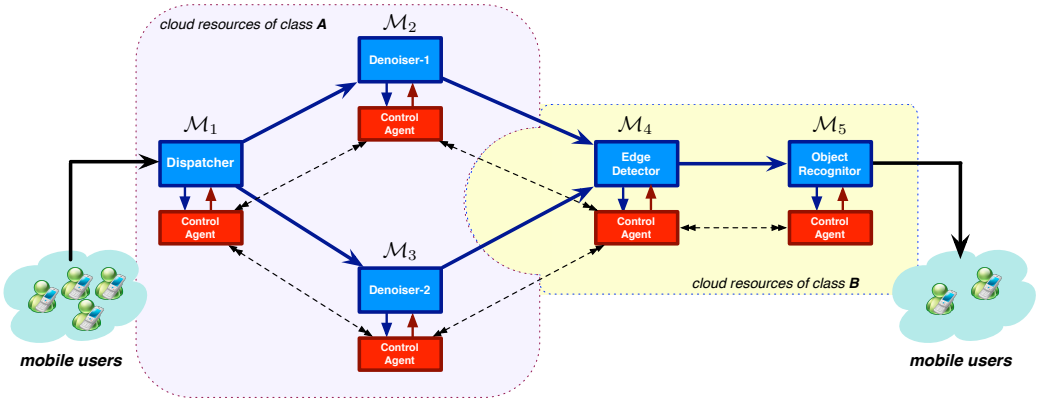


Fig. 10: Flow graph of the object recognition service on a MCC environment.

our simulations is based on the Mersenne-Twister algorithm, which is by far the most widely used high-quality pseudo-random number generator [Law and Kelton 1999]. Tab. III shows several configuration parameters of the modules and, in particular, the mean values of the random variables that have been profiled by executing the algorithms on an Intel Xeon Sandy-Bridge E5-2650 CPU (we observe a standard deviation about 30% of the mean value).

	Dispatcher	***Denoiser-1***	***Denoiser-2***	***Edge-Detector***	***Recognitor***
Module no.	①	②	③	④	⑤
Avg. Runn. Time (sec)	0.1	2.48	3.66	7.80	14.44
Cost α_i ($/sec)	0.5	0.5	0.5	0.5	0.5
Cost β_i ($/$\tau$)	4.83×10^{-3}	4.83×10^{-3}	4.83×10^{-3}	1.77×10^{-2}	1.77×10^{-2}
Max no. cores	1	8	8	32	32

Table III: Configuration parameters of the simulation.

For simplicity we assume that all the service times of the parallel modules follow the ideal model (see Fig. 3a). In fact, although non-ideal models could also be taken into account, they do not have impact on the way in which the agents interact to converge to a common decision. Tab. III shows the cost parameters used. Providers apply the common *pay-as-you-go* billing model [Gohad et al. 2013], in which users pay a fixed price per unit of use and unit of time. Providers usually offer different classes of resources, with different CPU frequencies, memory and storage capabilities. Fig. 11 shows the on-demand instance prices applied by Amazon EC2 in June 2015. They distinguish between *"Compute Optimized"*, *"Memory Optimized"* and *"Storage Optimized"* resources.

The cost per hour is proportional to the number of cores, thus the model described in Expr. 6 is representative of real providers. We use this billing model to instantiate our cost parameters shown in Tab. III. The $\{\beta_i\}_{i=1,\ldots,5}$ parameters report the cost per core (dollars) per control step of 5 minutes. We make the following simplifications:

— cloud providers usually offer fixed configurations in terms of number of cores (e.g., 2, 4, 8). Therefore, not all the configurations are available to the user. In the simulation we suppose to use a flexible cloud provider which allows the user to instantiate its virtual machines with any number of cores smaller or equal to a maximum value;

— in real clouds each partial instance-hour consumed is billed as a full hour. We neglect this aspect and we assume that the user pays exactly for what he consumes.

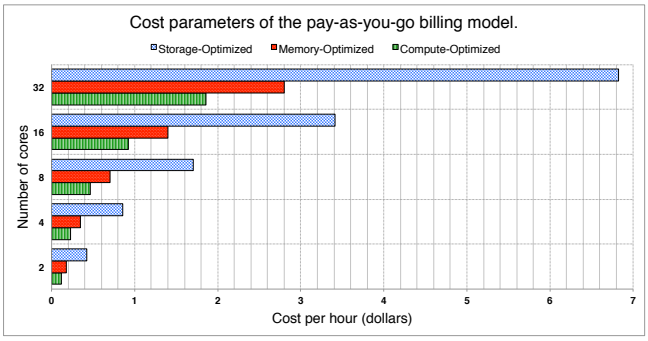


Fig. 11: Cost per hour of different resources provided by Amazon EC2.

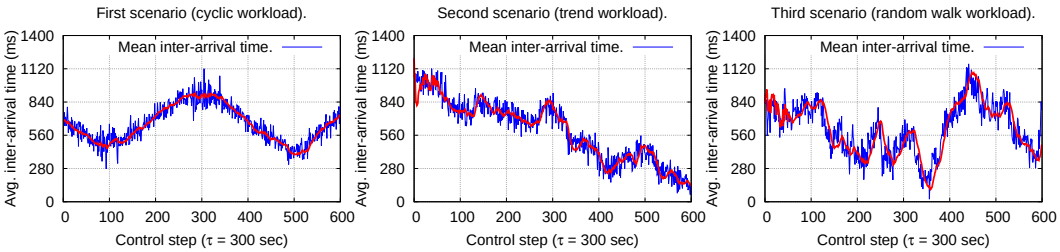
The cost parameters $\{\alpha_i\}_{i=1,\ldots,5}$ shown in Tab. III are chosen in order to properly weight the performance-oriented part of the cost function by capturing the user-degree of satisfaction related to the achieved throughput. The table shows the maximum number of cores for each module, which depends on the class of resources rented for the execution. The first module is executed on a single-core instance since it performs a fine-grained computation and it is never a bottleneck. Finally, the execution consists of 600 control steps of 300 seconds, for a total of 50 simulation hours. The measurements taken during the simulations are mainly based on the steady-state phase of the execution: after each reconfiguration the transient period lasts for few seconds, therefore its length is negligible with respect to the control step length.

6.2. Strategies Evaluation

In this section we present the results of the simulations. We will show the workload scenarios, the quality and quantity of reconfigurations executed and the relative efficiency achieved. Finally, we will analyze the throughput, cost efficiency and the efficacy of the incentive-based mechanism. Each simulation has been repeated 100 times. The variance of the measurements and the confidence intervals of the mean values are small. Therefore, we omit to show them in the plots for the sake of clarity.

Workload scenarios. Fig. 12 shows three scenarios that describe a different behavior of the mean *inter-arrival time* from external sources, i.e. the average time interval between two consecutive arrivals of requests to the `Dispatcher` module. The service time of the Dispatcher is equal to the maximum between its running time per task and the inter-arrival time, i.e. $T_{S_{1,k}} = \max\{T_{1,k}, T_{a,k}\}$ where $T_{a,k}$ is the mean inter-arrival time. The inter-arrival time is a disturbance subject to possible high variability, and its behavior can be described as a non-stationary process. The three scenarios are: *1)* `Cyclic` workload (Fig. 12a) represents a time-series with cyclic variations; *2)* `Trend` workload (Fig. 12b) is a time-series following a decreasing trend; *3)* `Rwalk` workload (Fig. 12c) is a sequence of values generated according to a random walk model.

At the beginning of each control step the Dispatcher control part estimates the mean inter-arrival time during the next step using an exponentially weighted moving average filter (EWMA), in order to smooth the noise of the time-series. The EWMA estimations are depicted with a red solid line in Fig. 12. In the simulation the inter-



(a) Cyclic workload. (b) Trend workload. (c) Rwalk workload.

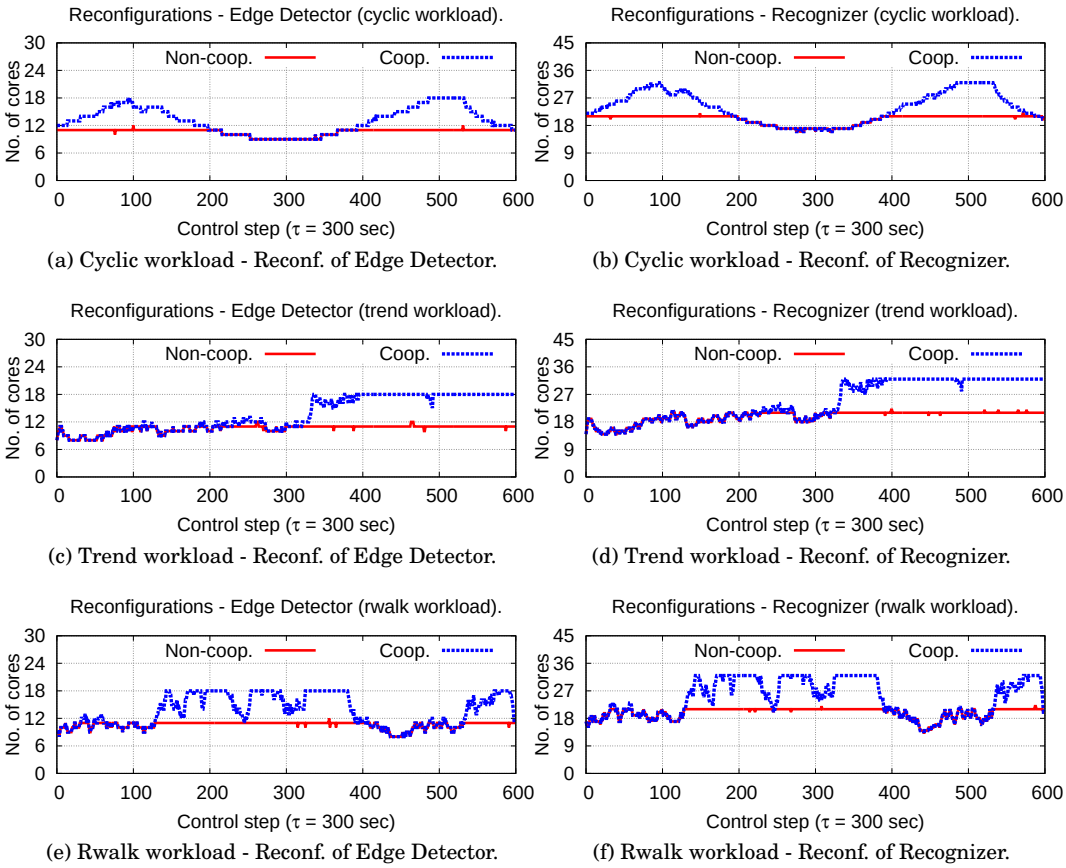
Fig. 12: Workload traces: mean value of the inter-arrival time for each step of the simulation.

arrival time is modeled as a normally distributed random variable whose mean value is changed at each step in order to follow the used workload trace.

Reconfigurations and efficiency. Fig. 13 shows the number of cores used by the modules with the non-cooperative and the incentive-based (cooperative) strategy. We expect that the sequences of the parallelism degrees track the external workload variability. As expected, the phases in which the inter-arrival time is lower (the stream pressure is more intensive) correspond to time periods in which the modules are configured to use more cores. The opposite occurs in phases of higher inter-arrival time. For the sake of brevity, we limit our analysis to the last two modules, Edge Detector and Recognizer, which perform coarse-grained computations on the stream elements and require a higher number of reconfigurations than the others.

It is important to observe that there are execution phases in which the two strategies take the same decisions at each step, and phases in which the incentive-based strategy uses higher parallelism degrees than the pure non-cooperative approach. The first situation happens when the modules use a number of cores lower than their ideal parallelism degree. This occurs during the execution intervals between $[200, 400]$ steps in the cyclic workload, between $[0, 300]$ steps in the trend workload and between $[0, 100]$ and $[400, 550]$ steps in the rwalk workload. As depicted in Fig. 12, these intervals correspond to phases in which the workload is less intensive. The other phases of the execution are characterized by a higher pressure of the input stream and the modules need more cores to sustain the input rate. In the non-cooperative strategy each module is forced to use a number of cores lower or at most equal to its ideal parallelism degree (Prop. 4.6). This is the reason because the reconfiguration sequences performed with the non-cooperative strategy are flattened when a certain parallelism degree value is reached, e.g., around 21 cores for the Recognizer. The ideal parallelism degree of a module changes if its mean running time per image changes (Def. 4.5). In this experiment the values of the running times listed in Tab. III are generated by using stationary time-series with a fixed mean and variance, thus the ideal parallelism degrees do not change or change very marginally during the execution. In fact, some ± 1 fluctuations are sporadically observed.

With the incentive-based strategy the Recognizer and Edge Detector modules are able to choose a number of cores greater than their integer-rounded ideal parallelism degrees, i.e. 21 and 16 according to Expr. 10. This is possible owing to the mechanism described in Sect. 5.2: the two modules increase their local incentive parameters as long as they are able to get closer to the system optimum. The effect is that the reconfigurations track more accurately the workload, although they are still flat during the time periods in which the Recognizer module uses its maximum number of cores (32).



(a) Cyclic workload - Reconf. of Edge Detector.
(b) Cyclic workload - Reconf. of Recognizer.
(c) Trend workload - Reconf. of Edge Detector.
(d) Trend workload - Reconf. of Recognizer.
(e) Rwalk workload - Reconf. of Edge Detector.
(f) Rwalk workload - Reconf. of Recognizer.

Fig. 13: Reconfiguration sequences of the Edge Detector and Recognizer modules with the non-cooperative and the incentive-based (cooperative) strategies.

Tab. IV shows the number of reconfigurations performed by the two strategies and the mean relative efficiency per module over the entire execution (average of the efficiency values for each step).

As expected and confirmed by Fig. 13, the incentive-based strategy performs a greater number of reconfigurations, as the parallelism degree values are not bounded by the ideal parallelism degrees. The relative efficiency is high and in general near to 1. This is an expected result of the theoretical analysis described in Sects. 4 and 5. The non-cooperative strategy selects an integer approximation of the new PNE strategy profile at each step, which is always a $\mathcal{E}$-*vector*. Due to the integer rounding the efficiency values are not exactly 1 but slightly lower. In the case of the two denoiser modules the efficiency is slightly lower than the others, as they need fewer cores to sustain the input pressure and the integer rounding penalizes the efficiency more than modules needing more cores (Edge Detector and Recognizer). The same considerations can be applied for the incentive-based strategy, where the system optimum at each step always belongs to the $\mathcal{E}$-vector set.

Performance and cost. We study the cost and the total number of completed tasks over the entire execution. Tab. V lists the total cost, defined as the sum of the cost paid

	Reconf.		Efficiency	
	Non-coop.	*Coop.*	*Non-coop.*	*Coop.*
`Denoiser-1`	4	8	0.87	0.88
`Denoiser-2`	5	14	0.86	0.90
`Edge-Det.`	12	45	0.99	0.97
`Recognizer`	23	96	0.97	0.98

(a) Cyclic workload.

	Reconf.		Efficiency	
	Non-coop.	*Coop.*	*Non-coop.*	*Coop.*
`Denoiser-1`	5	11	0.87	0.89
`Denoiser-2`	7	21	0.87	0.88
`Edge-Det.`	51	125	0.98	0.96
`Recognizer`	121	199	0.97	0.99

(b) Trend workload.

	Reconf.		Efficiency	
	Non-coop.	*Coop.*	*Non-coop.*	*Coop.*
`Denoiser-1`	6	12	0.86	0.88
`Denoiser-2`	8	44	0.86	0.87
`Edge-Det.`	68	205	0.99	0.97
`Recognizer`	154	297	0.98	0.99

(c) Rwalk workload.

Table IV: Number of reconfigurations and relative efficiency of the modules.

by each module for all the execution steps. In general, the incentive-based strategy is able to provide lower costs than the non-cooperative one. The cost reduction depends on the workload characteristics. In the trend workload we observe the largest benefit of using the cooperative approach, with an overall cost reduction of about 18%. The cost reduction in the cyclic workload scenario is modest, only of about 5%. Tab. V also reports the mean price of stability (Sect. 5) over the 600 simulation steps.

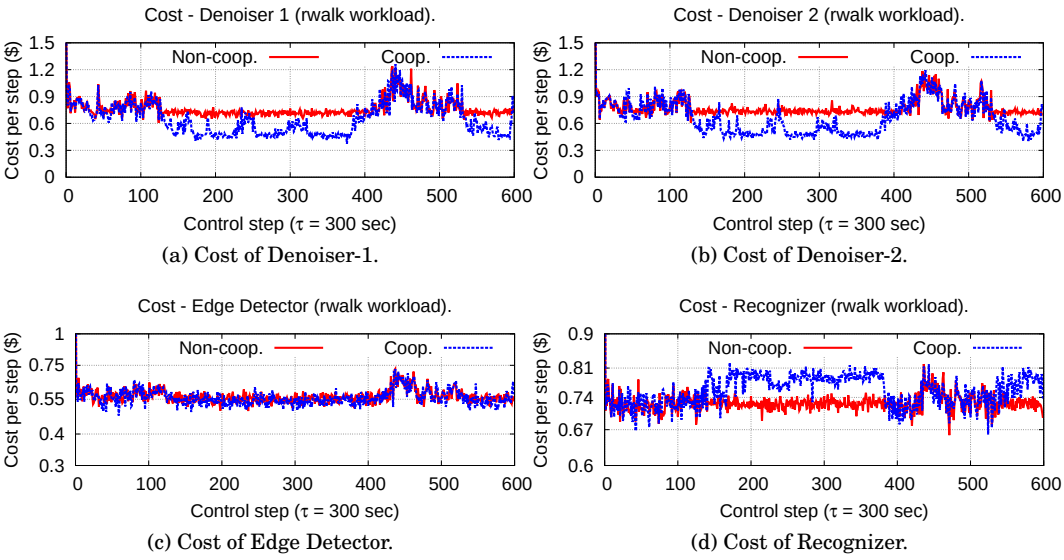
	Cyclic		Trend		Rwalk	
	Non-coop.	*Coop.*	*Non-coop.*	*Coop.*	*Non-coop.*	*Coop.*
Total cost	1,792	1,698	2,123	1,812	1,941	1,786
Price of stability	0.95		0.82		0.92	
Com. Tasks	241,736	289,885	248,377	311,026	240,851	300,447
`Avg. throughput` (img per step)	403	483	414	518	401	501

Table V: Total operating cost and number of completed tasks over the entire execution.

To provide a better understanding of how the incentive-based strategy works, Fig. 14 shows the cost per step of each parallel module under the rwalk workload. The cost of the non-cooperative and the cooperative strategies are reported in the same plot to simplify the comparison. Owing to the incentive-based mechanism, the cost of the two denoiser modules is reduced during the time interval between step 130 and 400 (see the blue line) where the cooperative strategy uses higher parallelism degrees.

The Recognizer deserves a special attention (Fig. 14d). During the same execution phase, the cost achieved by the incentive-based strategy is higher than the one achieved by behaving selfishly. This is the essence of cooperation: the incentive scheme forces the Recognizer module to use parallelism degrees that slightly worsen its individual cost; this marginal lose is counterbalanced by a decrease in the cost of some of the other modules such that the social welfare can be globally improved.

Furthermore, Tab. V shows the total number of tasks completed over the entire execution. As expected, the cooperative strategy allows the application to complete more tasks and to reach higher throughput than the non-cooperative strategy. The reason is quite intuitive. The incentive-based strategy uses higher parallelism degrees. Since



(a) Cost of Denoiser-1.

(b) Cost of Denoiser-2.

(c) Cost of Edge Detector.

(d) Cost of Recognizer.

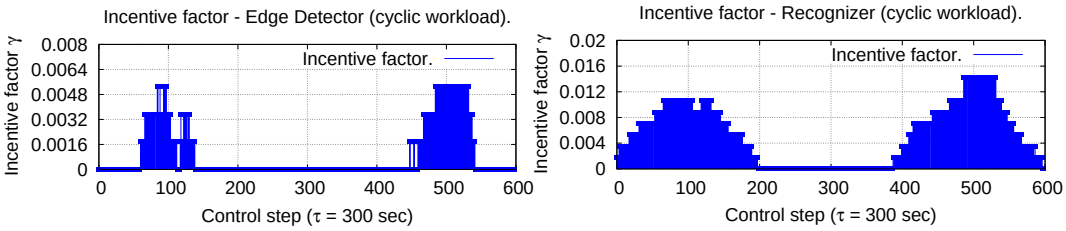
Fig. 14: Cost per step paid by each control agent during the execution: rwalk workload.

resources are never wasted (the PNE and system optimum are both $\mathcal{E}$-vectors), this yields to achieve higher throughput levels on average.

Tuning of the incentive-based mechanism. As stated in Sect. 5, the incentive-based mechanism allows to locate the agreement point closer to the system optimum. Fig. 15 shows the values of the incentive parameters γ_4 and γ_5 applied by the fourth and the fifth module. The other modules do not need to apply any incentive parameter throughout the execution ($\gamma_2 = \gamma_3 = 0$). In the experiments each agent i uses an increment stepsize $\Delta_i^c = \beta_i/10$ (the same used in the previous experiments).

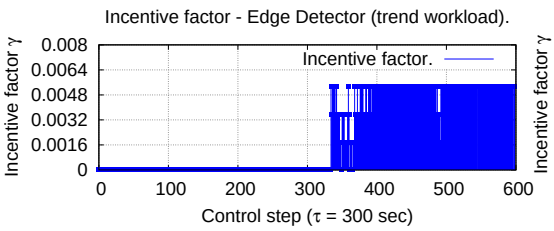
The values of the incentive parameters at each step depend on how far the PNE is from the system optimum. As we can note, no incentive is used during the steps in which the system optimum of $\mathbf{NPG}_\mathcal{K}$ is also coincidentally the PNE point. In those cases the non-cooperative strategy is directly able to find an integer rounding of the system optimum (Alg. 1). This behavior happens during the time periods in which the non-cooperative and the cooperative strategies produce identical reconfigurations, see Fig. 13.

Fig. 16a shows the average number of rounds in the rwalk workload as a function of the increment stepsize Δ_i^c (which is the same both for Edge Detector and Recognizer). As expected, the greater the stepsize the smaller the number of rounds performed by the agents. The number of rounds also affects the cost optimality. By using a smaller stepsize the strategy is able to reach a better approximation of the system optimum at each step. To this end, Fig. 16b shows the average price of stability over the entire execution. Smaller stepsizes make it possible to achieve better PoS values and consequently a lower cost of the application. As it is evident from Fig. 16b, too small stepsizes are not useful as the PoS stops to grow for stepsizes greater than a threshold. The reason is that at the end of Alg. 2 the control strategy applies an integer rounding. Therefore, a very high degree of precision in not really needed, and few rounds per step are often sufficient to achieve a near optimal cost reduction. The results with the other workload scenarios are qualitatively similar. We omit them for brevity.

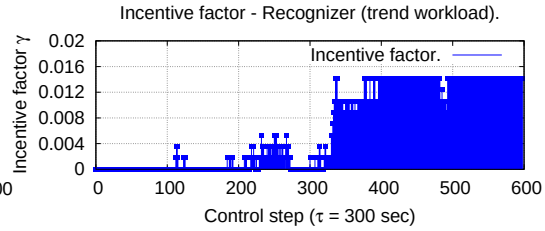


(a) Cyclic workload - γ_4 of Edge Detector.

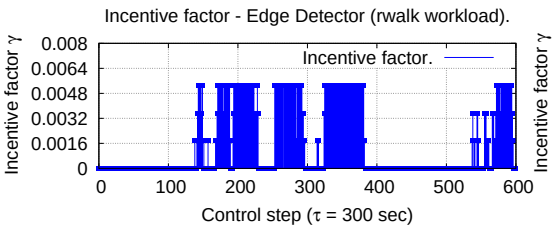
(b) Cyclic workload - γ_5 of Recognizer.



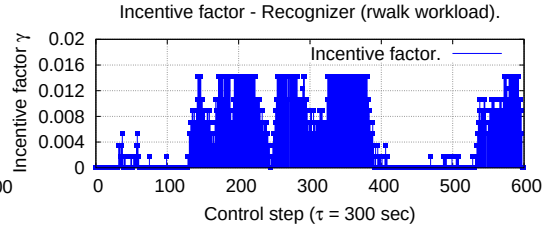
(c) Trend workload - γ_4 of Edge Detector.



(d) Trend workload - γ_5 of Recognizer.

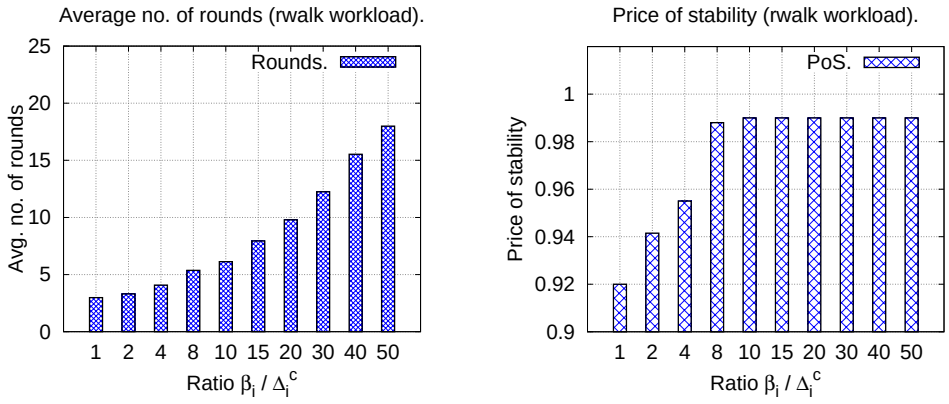


(e) Rwalk workload - γ_4 of Edge Detector.



(f) Rwalk workload - γ_5 of Recognizer.

Fig. 15: Incentive parameters (factors) applied by Edge Detector and Recognizer control agents at each step of the execution.



(a) Avg. no. of rounds.

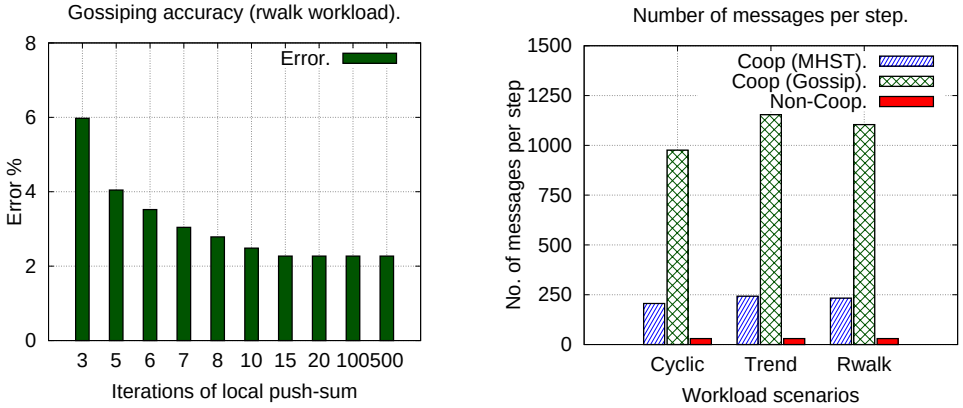
(b) Price of stability.

Fig. 16: Average number of rounds per step and average price of stability with different increment stepsizes of the cooperative strategy (rwalk workload).

We conclude this section by analyzing two final aspects. According to the description of Sect. 5.2, the cooperative strategy executes a distributed algorithm to compute the value of the social welfare at each round in a fully distributed manner. In the previous experiments we have used a tree-based protocol for distributed aggregation based on a minimum height spanning tree (briefly, MHST), while the local push-sum protocol is our gossip-based alternative to solve the same problem.

The accuracy of the gossip protocol depends on the number of iterations ($\mathcal{I} > 0$) performed. Fig. 17a shows the accuracy compared with the exact global cost determined by the MHST-based procedure. As we can see, few iterations are sufficient to reach a good accuracy: with 15 iterations the error is about 2%. As stated in [Geibig and Bradler 2010], the convergence time of local push-sum is proportional to the longest shortest path in the graph, i.e. the diameter.

Fig. 17b shows the total number of messages generated by the control agents at each step. We use $\Delta_i^c = \beta_i/10$ for the incentive-based strategy and $\mathcal{I} = 15$ for the gossiping aggregation. The incentive-based mechanism with gossiping aggregation produces the higher number of messages per step, slightly different in the three workload scenarios. When the aggregation is performed with the MHST-based approach, we obtain exact results with fewer messages at the expense of a non-homogeneous protocol. In contrast, the gossip-based protocol is fully distributed, with agents with the same behavior at each iteration. The non-cooperative strategy is reported in Fig. 17b to have a comparison baseline. The number of messages is much smaller than the cooperative strategy with MHST-based/gossiping aggregation, as it requires one single round per step to converge to the actual PNE strategy profile.



(a) Gossiping accuracy.

(b) Messages per step.

Fig. 17: Accuracy of the gossip-based protocol (local push-sum) and total number of messages per step.

7. CONCLUSIONS

This paper proposes new control strategies for distributed stream processing applications. The application modules are provided with elastic scaling capabilities in order to change their internal parallelism degree. As a first step, we formalize the performance modeling of flow graphs in order to identify bottlenecks and quantify the backpressure effect. Then, we distribute the control logic of the application among its modules, each one equipped with an agent performing the local control strategy. The first solution

consists in modeling the agents as rationale entities pursuing their self-interest. Accordingly, the agreement is modeled with the concept of Nash equilibrium. We identify the Nash equilibria of our problem and we design a distributed procedure to reach the best equilibrium. Furthermore, to promote cooperation, we introduce a decentralized incentive-based mechanism which maintains the non-cooperative scheme by changing the rewards of the agents such that better agreements can be reached in the cooperative sense. The simulation confirms the results of the theoretical analysis in terms of performance, efficiency and cost optimality.

In the future we plan to extend this work in different directions. We are currently developing some of the strategies proposed in this work in the `FastFlow` [Fas 2015] stream processing framework. Furthermore, we are aimed at extending our theoretical analysis by taking into account other relationships between modules, not only coupled according to the performance model presented in Sect. 4.1, but also in the use of resources, e.g., the available nodes are shared and can be used by any module of the application. In this way the problem could be studied in terms of generalized Nash equilibria, with some theoretical issues that we are currently studying.

REFERENCES

2007. *Learning from Data Streams: Processing Techniques in Sensor Networks* (1 ed.). Springer.

2015. FastFlow (FF). (2015). http://http://calvados.di.unipi.it/fastflow/

Mert Akdere, Cemal Çagatay Bilgin, Ozan Gerdaneri, Ibrahim Korpeoglu, Özgür Ulusoy, and Ugur Çetintemel. 2006. A Comparison of Epidemic Algorithms in Wireless Sensor Networks. *Comput. Commun.* 29, 13-14 (Aug. 2006), 2450–2457.

Tansu Alpcan and L. Pavel. 2009. Nash equilibrium design and optimization. In *Game Theory for Networks, 2009. GameNets '09. International Conference on*. 164–170.

Henrique Andrade, Buğra Gedik, and Deepak Turaga. 2014. *Fundamentals of Stream Processing*. Cambridge University Press. Cambridge Books Online.

H. Andrade, B. Gedik, K. L. Wu, and P. S. Yu. 2011. Processing High Data Rate Streams in System S. *J. Parallel Distrib. Comput.* 71, 2 (Feb. 2011), 145–156.

D. Ardagna, B. Panicucci, and M. Passacantando. 2013. Generalized Nash Equilibria for the Service Provisioning Problem in Cloud Systems. *Services Computing, IEEE Transactions on* 6, 4 (Oct 2013), 429–442.

Shivnath Babu and Jennifer Widom. 2001. Continuous Queries over Data Streams. *SIGMOD Rec.* 30, 3 (Sept. 2001), 109–120.

C. Bertolli, G. Mencagli, and M. Vanneschi. 2009. Adaptivity in Risk and Emergency Management Applications on Pervasive Grids. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*. 550–555. DOI:http://dx.doi.org/10.1109/I-SPAN.2009.92

Carlo Bertolli, Gabriele Mencagli, and Marco Vanneschi. 2010. A Cost Model for Autonomic Reconfigurations in High-performance Pervasive Applications. In *Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems (CASEMANS '10)*. ACM, New York, NY, USA, Article 3, 10 pages. DOI:http://dx.doi.org/10.1145/1858367.1858370

Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. 2005. A Non-Local Algorithm for Image Denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02 (CVPR '05)*. IEEE Computer Society, Washington, DC, USA, 60–65.

Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. 2013. Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 725–736.

S. Chaisiri, Bu-Sung Lee, and D. Niyato. 2012. Optimization of Resource Provisioning Cost in Cloud Computing. *Services Computing, IEEE Transactions on* 5, 2 (April 2012), 164–177.

Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. 2012. Cooperative Game Theory: Basic Concepts and Computational Challenges. *IEEE Intelligent Systems* 27, 3 (2012), 86–90.

Gianpaolo Cugola and Alessandro Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.* 44, 3, Article 15 (June 2012), 62 pages.

Pradeep Dubey. 1986. Inefficiency of Nash Equilibria. *Mathematics of Operations Research* 11, 1 (1986), pp. 1–8.

Bernard Espinasse, Guy Picolet, and Eugène Chouraqui. 1997. Negotiation support systems: A multi-criteria and multi-agent approach. *European Journal of Operational Research* 103, 2 (1997), 389 – 409.

R. Gayathri and R.S. Sabeenian. 2013. A performance analysis of efficient schemes and algorithms in image denoising procedures. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*. 1–5.

B. Gedik, S. Schneider, M. Hirzel, and Kun-Lung Wu. 2014. Elastic Scaling for Data Stream Processing. *Parallel and Distributed Systems, IEEE Transactions on* 25, 6 (June 2014), 1447–1463.

J. Geibig and D. Bradler. 2010. Self-organized aggregation in irregular wireless networks. In *Wireless Days (WD), 2010 IFIP*. 1–7.

A. Gohad, N.C. Narendra, and P. Ramachandran. 2013. Cloud Pricing Models: A Survey and Position Paper.. In *Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on*. 1–8.

Eduardo R. Gomes, Quoc Bao Vo, and Ryszard Kowalczyk. 2012. Pure exchange markets for resource sharing in federated clouds. *Concurrency and Computation: Practice and Experience* 24, 9 (2012), 977–991.

Horacio González-Vélez and Mario Leyton. 2010. A Survey of Algorithmic Skeleton Frameworks: High-level Structured Parallel Programming Enablers. *Softw. Pract. Exper.* 40, 12 (Nov. 2010), 1135–1160.

Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. 2012. StreamCloud: An Elastic and Scalable Data Streaming System. *IEEE Trans. Parallel Distrib. Syst.* 23, 12 (Dec. 2012), 2351–2365.

Thomas Heinze, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. 2014. Latency-aware Elastic Scaling for Distributed Data Stream Processing Systems. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS '14)*. ACM, New York, NY, USA, 13–22.

Waldemar Hummer, Benjamin Satzger, and Schahram Dustdar. 2013. Elastic stream processing in the Cloud. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3, 5 (2013), 333–345.

Amazon Inc. 2008. *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Inc., http://aws.amazon.com/ec2/#pricing. http://aws.amazon.com/ec2/\#pricing

David Kempe, Alin Dobra, and Johannes Gehrke. 2003. Gossip-Based Computation of Aggregate Information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*. IEEE Computer Society, Washington, DC, USA, 482–.

Yu-Kwong Kwok, Kai Hwang, and S. Song. 2007. Selfish Grids: Game-Theoretic Modeling and NAS/PSA Benchmark Evaluation. *Parallel and Distributed Systems, IEEE Transactions on* 18, 5 (May 2007), 621–636.

Averill M. Law and David M. Kelton. 1999. *Simulation Modeling and Analysis* (3rd ed.). McGraw-Hill Higher Education.

Claudia Leopold. 2001. *Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches*. John Wiley & Sons, Inc., New York, NY, USA.

Hongxing Li, Chuan Wu, Zongpeng Li, and F.C.M. Lau. 2013. Profit-maximizing virtual machine trading in a federation of selfish clouds. In *INFOCOM, 2013 Proceedings IEEE*. 25–29.

Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. 2013. Dynamic Right-sizing for Power-proportional Data Centers. *IEEE/ACM Trans. Netw.* 21, 5 (Oct. 2013), 1378–1391.

B. Lohrmann, P. Janacik, and O. Kao. 2015. Elastic Stream Processing with Latency Guarantees. In *The 35th International Conference on Distributed Computing Systems (ICDCS 2015)*. to appear.

Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. 2012. Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems. *ACM Trans. Auton. Adapt. Syst.* 7, 4, Article 36 (Dec. 2012), 32 pages.

Rafik Makhloufi, Grégory Bonnet, Guillaume Doyen, and Dominique Gaïti. 2009. Decentralized Aggregation Protocols in Peer-to-Peer Networks: A Survey. In *Modelling Autonomic Communications Environments*, JohnC. Strassner and YacineM. Ghamri-Doudane (Eds.). Lecture Notes in Computer Science, Vol. 5844. Springer Berlin Heidelberg, 111–116.

Rafik Makhloufi, Guillaume Doyen, Grégory Bonnet, and Dominique Gaïti. 2014. A survey and performance evaluation of decentralized aggregation schemes for autonomic management. *International Journal of Network Management* 24, 6 (2014), 469–498.

D. Meilander, S. Kottinger, and S. Gorlatch. 2013. A Scalability Model for Distributed Resource Management in Real-Time Online Applications. In *Parallel Processing (ICPP), 2013 42nd International Conference on*. 763–772. DOI:http://dx.doi.org/10.1109/ICPP.2013.90

Gabriele Mencagli. 2012. *A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations*. Ph.D Thesis, Department of Computer Science, University of Pisa, Italy. http://etd.adm.unipi.it/t/etd-05242012-212444/

G. Mencagli and M. Vanneschi. 2011. QoS-control of Structured Parallel Computations: A Predictive Control Approach. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. 296–303.

G. Mencagli, M. Vanneschi, and E. Vespa. 2013a. Control-theoretic adaptation strategies for autonomic reconfigurable parallel applications on cloud environments. In *High Performance Computing and Simulation (HPCS), 2013 International Conference on*. 11–18.

Gabriele Mencagli, Marco Vanneschi, and Emanuele Vespa. 2013b. Reconfiguration Stability of Adaptive Distributed Parallel Applications Through a Cooperative Predictive Control Approach. In *Proceedings of the 19th International Conference on Parallel Processing (Euro-Par'13)*. Springer-Verlag, Berlin, Heidelberg, 329–340.

Gabriele Mencagli, Marco Vanneschi, and Emanuele Vespa. 2014. A Cooperative Predictive Control Approach to Improve the Reconfiguration Stability of Adaptive Distributed Parallel Applications. *ACM Trans. Auton. Adapt. Syst.* 9, 1, Article 2 (March 2014), 27 pages.

John Nash. 1951. Non-Cooperative Games. *The Annals of Mathematics* 54, 2 (Sept. 1951), 286–295.

Dusit Niyato, Kun Zhu, and Ping Wang. 2011. Cooperative Virtual Machine Management for Multi-organization Cloud Computing Environment. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '11)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 528–537.

Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. 2012. iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. *J. Grid Comput.* 10, 1 (March 2012), 185–209.

Jaeok Park and M. van der Schaar. 2010. A Game Theoretic Analysis of Incentives in Content Production and Sharing Over Peer-to-Peer Networks. *Selected Topics in Signal Processing, IEEE Journal of* 4, 4 (Aug 2010), 704–717.

A. G. Prieto and R. Stadler. 2007. A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives. *IEEE Trans. on Netw. and Serv. Manag.* 4, 1 (June 2007), 2–12.

Timo Reuter and Philipp Cimiano. 2012. Event-based Classification of Social Media Streams. In *Proceedings of the 2Nd ACM International Conference on Multimedia Retrieval (ICMR '12)*. ACM, New York, NY, USA, Article 22, 8 pages.

William Rogerson. 1994. A Theory of Incentives in Procurement and Regulation by Jean-Jacques Laffont; Jean Tirole. *Journal of Political Economy* 102, 2 (1994), pp. 397–402.

C.U. Saraydar, Narayan B. Mandayam, and D. Goodman. 2002. Efficient power control via pricing in wireless data networks. *Communications, IEEE Transactions on* 50, 2 (Feb 2002), 291–303.

Riccardo Scattolini. 2009. Architectures for distributed and hierarchical Model Predictive Control - A review. *Journal of Process Control* 19, 5 (2009), 723–731.

Yuzhe Tang and B. Gedik. 2013. Autopipelining for Data Stream Processing. *Parallel and Distributed Systems, IEEE Transactions on* 24, 12 (Dec 2013), 2344–2354.

Nicolai Vorobjov. 1994. *Foundations of game theory - noncooperative games.* Birkhäuser. I–VI, 1–496 pages.

Yingjun Wu and Kian-Lee Tan. 2015. ChronoStream: Elastic stateful stream computation in the cloud. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. 723–734.

Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. 2013. A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing. *SIGMETRICS Perform. Eval. Rev.* 40, 4 (April 2013), 23–32.

Deshi Ye and Jianhai Chen. 2013. Non-cooperative Games on Multidimensional Resource Allocation. *Future Gener. Comput. Syst.* 29, 6 (Aug. 2013), 1345–1352.

Qin Yuan, Zhixiang Liu, Junjie Peng, Xing Wu, Jiandun Li, Fangfang Han, Qing Li, Wu Zhang, Xinjin Fan, and Shengyuan Kong. 2011. A Leasing Instances Based Billing Model for Cloud Computing. In *Proceedings of the 6th International Conference on Advances in Grid and Pervasive Computing (GPC'11)*. Springer-Verlag, Berlin, Heidelberg, 33–41.

Tienan Zhang and Peng Xiao. 2014. A Novel Resource Pricing Mechanism based on Multi-Player Gaming Model in Cloud Environments. *Journal of Software* 9, 6 (2014).