Analysis of Control-theoretic Predictive Strategies for the Adaptation of Distributed Parallel Computations

Gabriele Mencagli
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3, I-56127, Pisa, Italy
mencagli@di.unipi.it

Marco Vanneschi
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3, I-56127, Pisa, Italy
vannesch@di.unipi.it

ABSTRACT

In adaptive distributed parallel applications the adaptation process is based on the ability to change some characteristics of parallel components, such as the parallelism form and the parallelism degree, in response to unexpected execution conditions. Although existing research work has studied this problem, it is of increasing importance to investigate adaptation strategies able to reach important properties like the stability of control decisions, i.e. to guarantee that reconfigurations are effective and durable, and control optimality, expressed by means of cooperative and non-cooperative agreements between decisions of different controllers. These properties are crucial in distributed environments like Grids and Clouds, where reconfigurations imply a cost both in terms of a performance degradation as well as a monetary charge. In this paper we briefly introduce the basic ideas of our methodology and we introduce different adaptation strategies based on alternative formulations of the Model-based Predictive Control technique. First hints about the effectiveness of our approach are discussed through experiments developed in a simulation environment.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*

Keywords

Reconfigurations; Autonomic Computing; Parallel Computations; Model-based Predictive Control

1. INTRODUCTION

In dynamic execution contexts the achievement of desired levels of *Quality of Service* (QoS) requires to continuously adapt the application configuration in response to exogenous uncontrollable factors like workload variations and time-varying user requirements. For distributed parallel applications reconfiguration activities consist in run-time modifications of the parallelism degree and the parallelism form of specific application components. The development of adaptive applications implies to face with two central issues. The first one concerns how a distributed system can change its behavior at run-time, i.e. *which reconfigurations are defined and how they are implemented.* The second point investigates the logic and the methodological tools behind the definition of ***adaptation strategies***, i.e. *how reconfigurations are selected in response to critical and unexpected execution conditions.*

On Cloud environments the problem of dynamic reconfigurations introduces new issues peculiar to this computing paradigm. In elastic cloud infrastructures, computing resources are usually delivered on-demand to the users in terms of virtual machines deployed in remote provider data centers. When executing adaptive applications, the modification of the application configuration leads to significant changes in the used infrastructure by frequently creating and shutting down virtualized resources. This can induce different costs on the computation, both in terms of performance degradation (e.g. parallel modules could be blocked waiting for the reconfiguration process to complete) as well as in terms of a monetary charge due to the dynamic provisioning of resources. It is clear that, given the nature of these execution environments, reconfigurations do not come for free, but finding appropriate additional resources could be tricky, time-consuming and economically expensive.

In this paper we present an overview of our methodology [12]. In our approach, the adaptation strategy is decomposed among the application components/modules. Each module is composed of an Operating Part (a reconfigurable parallel computation performing structured parallelism patterns such as task-farm, data-parallel and divide-and- conquer schemes) and a Control Part interconnected in a closed-loop model. Each *local control problem* is composed of: (i) a cost function that describes the local QoS goals and reconfiguration costs; (ii) a model that formally describes the future evolution of local QoS variables. The module adaptation is driven by a control-theoretic strategy known as *Model-based Predictive Control* [5]. Proper definitions of the cost functions drive the adaptation strategy towards the achievement of properties like:

- the *stability* of the control decisions. We use this term in a different meaning w.r.t the classic concept of stability of dynamical systems used in Control Theory. A stable adaptation strategy avoids oscillating behaviors and minimizes the number of reconfigurations (avoid to modify the application configuration unnecessarily);

ORMACloud'13, June 17, 2013, New York, NY, USA.

- *control optimality*, i.e. reaching desired trade-offs between contrasting QoS goals (e.g. performance and resource consumption).

The presence of interconnected components leads to a critical problem: the reconfigurations selected by a module may influence QoS parameters of other modules of the application (control sub-problems can be coupled with each other). This means that Control Parts should communicate in order to take effective reconfiguration decisions. We face with this problem by adopting two diametrically different approaches:

- *controllers operate selfishly*: each controller optimizes its local cost without taking into account the effects of its actions on the other controllers;
- *controllers operate in a cooperative fashion*, in order to select reconfigurations that are optimal from a global viewpoint instead of pursuing their self-interest.

The outline of this paper is the following. In the next section we review existing research work on adaptiveness for distributed parallel computations. Section 3 presents the general description of our methodology. In Section 4 we discuss some preliminary results concerning the application of our methodology. Finally, Section 5 concludes this paper.

2. RELATED WORK

Providing computing systems with run-time supports to dynamic reconfigurations has been the subject of several researches in different fields like Mobile, Grid and Cloud Computing. On such environments, it is of great importance to dynamically provide computing resources to applications featuring variable QoS requirements and characterized by irregular workload conditions. Examples are described in [15, 11], in which provisioning mechanisms of virtual machines are developed to accelerate compute-intensive jobs submitted into Cloud platforms.

Besides efficient run-time supports [14, 1], emerging computing environments raise critical problems related to when reconfigurations should be executed in order to optimize performance and economic aspects. Therefore, adaptation strategies have gained much attention. A common approach consists in providing the mapping between execution conditions and corresponding reconfigurations explicitly, through a set of *policy rules* expressed by a declarative language [8]. Examples of autonomic frameworks adopting this approach are described in [9] for distributed computing systems, and [1, 2] more oriented towards high-performance applications.

On the other hand, the adoption of Control Theory foundations to adapt computing systems [6] have moved beyond the preliminary stage. An example of a framework in which control-theoretic strategies have been applied to improve software performance is described in [16]. A comprehensive overview of decision-making strategies is presented in [10]. In this work emerges that Optimal Control approaches are suitable to optimize performance requirements and operating costs by avoiding unnecessary reconfigurations. Despite the existence of first activities in this area [7], this research direction is still open and requires further investigations.

3. OVERVIEW OF THE METHODOLOGY

The core element of our methodology is the concept of *adaptive parallel module* (i.e. shortly ***ParMod***), an active unit featuring a parallel computation and an adaptation strategy to respond to dynamic execution conditions. A ParMod is structured into two interconnected parts:

- the **Operating Part** performs a structured parallel computation [4], expressed parametrically w.r.t the parallelism degree. The computation is activated by receiving tasks from input data streams. Results are transmitted onto output data streams directed to other application components;
- the **Control Part** (controller) observes the Operating Part execution and performs reconfiguration activities (i.e. parallelism degree variations - number of threads/processes of the current implementation - and changes in the executed parallelism form).

The Operating Part collects and periodically exchanges measurements (*monitoring data*) representing the behavior of the parallel computation (e.g. memory usage, resource utilization, service time and computation latency). In order to take effective reconfiguration decisions, Control Parts of different ParMods exchange *control messages* in order to reach specific agreements between their reconfiguration decisions. Figure 1 outlines the internal structure of a ParMod and the existing interconnections between sub-systems.

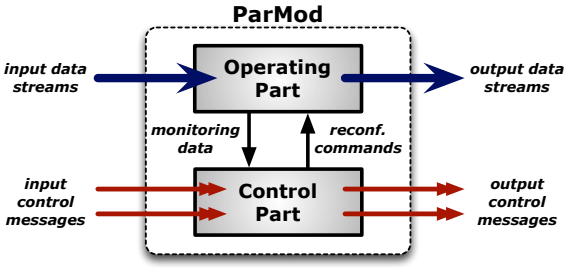


Figure 1: Internal structure of an adaptive parallel module.

In our model the control logic evaluation is performed at equally spaced time instants. We call *control step* the time interval between two successive evaluations of the adaptation strategy.

3.1 Distributed Model-based Predictive Control

For each application module M_i we identify a local model involving the following set of variables:

- ***local QoS variables*** ($\mathbf{x}_i(k) \in \mathbb{R}^n$) are metrics related to performance, memory usage and resource consumption for a given control step k;
- ***local control variables*** ($\mathbf{u}_i(k) \in \mathcal{U}_i$) are controllable parameters that identify the ParMod configuration used throughout the k-th control step;
- ***local disturbances*** ($\mathbf{d}_i(k) \in \mathbb{R}^m$) model exogenous uncontrollable factors influencing the module's QoS (e.g. the arrival rate from external sources).

Coupling relationships between sub-problems are modeled though a proper set of input/output ***interconnecting variables*** between Control Parts. We denote with $\mathbf{v}_{in\text{-}i}(k) \in \mathbb{R}^z$

the interconnection variables received by ParMod M_i from the other modules, and with $\mathbf{v}_{out\text{-}i}(k) \in \mathbb{R}^t$ the interconnecting variables transmitted by M_i to the other controllers.

A general formulation of the Operating Part model is given by the following discrete-time model:

$$\mathbf{x}_i(k+1) = \Phi_i\Big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k),\, \mathbf{v}_{in\text{-}i}(k)\Big) \qquad (1)$$

The model allows the Control Part to predict the values assumed by local QoS variables as a function of control inputs, local disturbances, input interconnecting variables from other controllers and present values of QoS variables. In this case we speak about a *dynamic model*, described by a set of difference equations. Otherwise, if future QoS values do not depend on the present values, we speak about a *static model* expressible through a set of algebraic equations.

The output interconnecting variables are calculated as a function of the actual QoS variables, disturbances and control inputs:

$$\mathbf{v}_{out\text{-}i}(k) = Z_i\Big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k)\Big) \qquad (2)$$

where Z_i is called the *output generation function*.

The basic strategy adopted by each Control Part follows the general principle of Model-based Predictive Control [5] (shortly MPC). MPC is an Optimal Control approach in which the current reconfiguration is taken by solving a finite-horizon optimization problem using the current value of QoS variables and statistical multiple-step ahead predictions of future disturbances. Only the first reconfiguration of the optimal sequence is applied to the Operating Part, and the procedure is repeated at the next control step. This strategy has the effect of making the prediction horizon slide in the future step-by-step, in order to continuously adapt the optimal reconfigurations to the update disturbance predictions.

Distributed MPC schemes can be applied to control large-scale systems such as distributed computations. In this case the global optimization problem is composed of a set of coupled sub-problems. ***Non-cooperative schemes*** consist in controllers that operate selfishly. At each control step each Control Part ($\mathrm{CP_i}$) selects the optimal control trajectory by solving the following local problem[1]:

$$\begin{aligned} &\underset{\overline{U}_i(k)}{\arg\min}\; J_i\Big(\overline{X}_i(k+1),\, \overline{U}_i(k),\, \overline{V}_{in\text{-}i}(k)\Big) \qquad (3)\\ &s.t.\\ &\mathbf{x}_i(k+1) = \Phi_i\big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k),\, \mathbf{v}_{in\text{-}i}(k)\big)\\ &\mathbf{u}_i(k) \in \mathcal{U}_i \end{aligned}$$

$\mathrm{CP_i}$ selects the control trajectory $\overline{U}_i^*(k)$ that optimizes its local objective (cost) function J_i given the current trajectory of received interconnecting variables and predicted local disturbances. Therefore, in this formulation, each controller optimizes its local cost function without taking into account the effects of its actions on the other controllers.

A selfish approach can be ineffective when we want to achieve the global optimality in a system-wise sense. This problem can be solved by resorting on ***cooperative control schemes*** in which controllers, instead of simply reacting to the decisions of the partners, account for the effects of their actions on the local objectives of the others. In this case, each Control Part $\mathrm{CP_i}$ should select the optimal control trajectory by solving a global optimization problem stated as follows:

$$\underset{\overline{U}_1(k),\ldots,\overline{U}_N(k)}{\arg\min}\; J_G = \sum_{i=1}^{N} w_i\, J_i\Big(\overline{X}_i(k+1),\, \overline{U}_i(k),\, \overline{V}_{in\text{-}i}(k)\Big) \qquad (4)$$

s.t.

$\mathbf{x}_i(k+1) = \Phi_i\big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k),\, \mathbf{v}_{in\text{-}i}(k)\big) \;\; i = 1, 2, \ldots, N$

$\mathbf{u}_i(k) \in \mathcal{U}_i \;\; i = 1, 2, \ldots, N$

where J_G is the *global cost function* of the whole system, defined as the weighted sum of local objectives.

4. EVALUATION OF THE METHODOLOGY

In order to provide first preliminary results, we have developed a ParMod simulation environment based on the OmNeT++ discrete event simulator[2]. A ParMod is composed of two simulation parts, the first one implements the Operating Part and the second is in charge of executing the adaptation logic (Control Part). The behavior of a simulation module can be programmed following an event-driven programming style. A module can receive different classes of messages from other modules. Each time a new message is received, the `handlemessage()` routine is called automatically. Inside the definition of this routine the programmer can specify different handlers based on the type of the received message, and can also generate new messages that will be transmitted to other modules. Communications are performed through the definition of *ports*: each port is binded with a port of another module in such a way that each message transmitted using a local port will be delivered to a well-identified destination (see Figure 2).

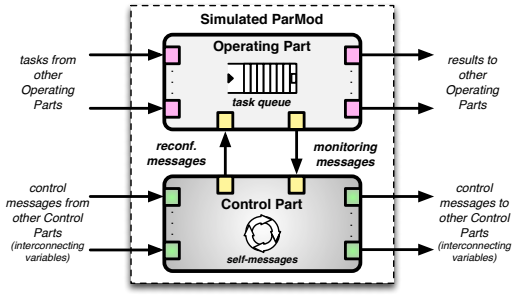


Figure 2: Simulation of an Adaptive Parallel Module.

In order to reproduce the behavior of a structured parallel computation operating on a stream of input tasks, the Operating Part implements a queue logic with a blocking semantics. Tasks are received and buffered into a queue data structure. The communication protocol is based on the transmission of `SEND` and `ACK` messages. If the received task can be buffered (i.e. there is a free position in the queue), the Operating Part transmits an `ACK` message to the sender. Otherwise, if there is no free space in the queue, the received task and other information (e.g. the sender identifier) are stored in a special data-structure and the `ACK` transmission

[1] an uppercase overlined letter denotes a sequence of values (trajectory) starting from a given control step.

[2] visit http://www.omnetpp.org/ for further details about this open-source simulator.

is delayed until a position in the queue is freed. The communication protocol makes the sender wait for an explicit `ACK` message before transmitting the next task to a destination.

The Operating Part can adopt two working logics reproducing structured parallelism schemes and their impact on performance parameters such as the service time and the computation latency: (i) a *task-farm semantics*, in which at most p tasks in parallel can be executed, where p is the current parallelism degree; (ii) a *data-parallel semantics*, in which only one task at a time can be processed with an execution time equal to the calculation time divided by the parallelism degree. The parallelism degree is simulated as an attribute of the Operating Part.

4.1 Competition and Cooperation between Controllers

In multi-federated Cloud environments, users have the possibility to transparently access to a pool of virtualized computing resources without regard to where resources are physically located and what Cloud providers really provide them. In this situation, adaptation strategies can become extremely complex, since different parts of a distributed application can be executed on Cloud providers applying different costs and billing models. In this case the application controllers, which dynamically regulate the resource consumption of the application, need to exchange information in order to meet the desired QoS goals but, at the same time, try to reach the best execution cost.

We study two diametrically different ways to perform the interaction between controllers: (i) a **selfish scheme**, in which Control Parts interact in order to reach an agreement modeled by the concept of *Nash Equilibrium*; (ii) a **cooperative scheme**, in which controllers cooperate in order to take optimal reconfiguration decisions in a system-wise sense. To contextualize these strategies, we consider the problem of regulating the resource consumption (expressed in terms of provisioned computing nodes) of a distributed parallel application executed in a federated Cloud environment. In order to maintain the desired level of performance, the necessary amount of used resources can change consistently over the time, due to application-dependent reasons (e.g. a variable workload) or caused by the dynamic behavior of the execution platform (e.g. availability of computing and networking resources).

Each ParMod is aimed at optimizing two QoS objectives: its effective performance, modeled by its *inter-departure time* (the steady-state average time between two successive result departures), and its *resource consumption*, proportional to the number of used nodes (the parallelism degree of the computation). For each ParMod M_i we introduce a *local cost function* defined over a future horizon of one control step (*one-step ahead MPC*):

$$J_i(k) = \alpha_i \, T_{D_i}\big(k+1\big) + \beta_i \, n_i(k) \tag{5}$$

where $T_{D_i}(k)$ models the inter-departure time (QoS variable) assumed at the beginning of control step $k+1$ (it refers to the behavior measured at the end of the last control step k), and $n_i(k) \in \mathcal{U}_i$ is the parallelism degree (control input) adopted throughout the k-th control step (where $\mathcal{U}_i$ is the closed interval $[1, n_i^{max}]$). α_i and β_i are two positive coefficients representing the desired trade-off between performance and resource utilization.

The prediction of the steady-state performance of a module is a complex problem, which requires to represent the stochastic behavior of the entire computation graph (e.g. modeled as a network of queues) in order to identify the presence of bottlenecks. From the performance viewpoint, each module is characterized by a *mean service time* which depends on the internal structured parallelization. For the sake of simplicity, we assume that the service time of a ParMod scales perfectly with its parallelism degree, i.e. $T_{S_i}(k) = T_{calc\text{-}i}(k)/n_i(k)$, where $T_{calc\text{-}i}(k)$ represents the mean calculation time (modeled as a disturbance) per task during control step k. In this paper we adopt a simple yet powerful approach to estimate the steady-state performance. The method, originally presented in [12], is valid for acyclic graphs with a single source module and can be summarized by the following result:

THEOREM 1 (PERFORMANCE MODELING). *Given a single source acyclic graph G composed of N modules, the inter-departure time T_{D_i} from M_i is given by the following model (Φ_i as stated in expression 1):*

$$T_{D_i}(k{+}1) = \max\Big\{ f_{i,1}\big(T_{S_1}(k)\big),\, f_{i,2}\big(T_{S_2}(k)\big), \ldots, f_{i,N}\big(T_{S_N}(k)\big) \Big\} \tag{6}$$

Each term $f_{i,j}$ with $j = 1, 2, \ldots, N$ expresses the inter-departure time of M_i if module M_j is the bottleneck of the graph. $f_{i,j}$ is defined as a function of the service time of M_j:

$$f_{i,j}\big(T_{S_j}(k)\big) = T_{S_j}(k) \, \frac{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_1 \to M_j)} \left(\prod_{\forall (s,d) \in \pi} p_{s,d}(k) \right)}{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_1 \to M_i)} \left(\prod_{\forall (s,d) \in \pi} p_{s,d}(k) \right)} \tag{7}$$

where M_1 denotes the source of G, $\mathcal{P}(M_1 \to M_i)$ is the set of all the paths in the graph starting from M_1 and reaching M_i, and $p_{s,d}(k)$ is the probability to transmit tasks from M_s to M_d during control step k (it is modeled as a disturbance). Since we do not know which module will be the bottleneck, the inter-departure time of M_i is calculated by taking the maximum between the functions $f_{i,j}$ for $j = 1, \ldots, N$.

4.1.1 Finding Nash Equilibria

The first approach is based on controllers that pursue their self-interest, i.e. each Control Part selects the reconfiguration decision (*best response*) that optimizes its local cost function given the control information received by the other controllers. In this problem controllers exchange their service times, which represent input/output interconnecting variables. In order to make our analysis tractable, the parallelism degree $n_i(k)$ is treated as a positive real-valued number. For each ParMod we introduce the following concept:

DEFINITION 1 (IDEAL PARALLELISM DEGREE). *The **ideal parallelism degree** of M_i during control step k is a value $n_i^*(k)$ such that the local cost function J_i is minimized in isolation, i.e. assuming that $T_{D_i}(k+1) = f_{i,i}(T_{S_i}(k))$:*

$$n_i^*(k) = \sqrt{\frac{\alpha_i \, T_{calc\text{-}i}(k)}{\beta_i}} \tag{8}$$

It expresses the optimal trade-off between performance and resource cost when the ParMod has no knowledge about the behavior of the other modules.

Given the particular structure of the local cost functions, we can state the conditions under which a set of parallelism degrees represents an allocation satisfying the Nash optimality concept:

PROPOSITION 1 (NASH EQUILIBRIA). *Given a vector of parallelism degrees* $[n_1^{(e)}(k), n_2^{(e)}(k), \ldots, n_N^{(e)}(k)]^T$ *such that the following conditions are satisfied:*

$$\begin{cases} T_{S_i}(k) = T_{D_i}(k+1) \ \ \forall i = 1, 2, \ldots, N \\ \\ n_i^{(e)}(k) \leq n_i^*(k) \ \ \forall i = 1, 2, \ldots, N \end{cases}$$

This set of parallelism degrees represents a Nash equilibrium.

The previous conditions identify a set of control decisions such that: (i) for each ParMod its service time is equal to its inter-departure time; (ii) no ParMod uses a parallelism degree greater than its ideal parallelism degree. The first condition implies that each ParMod has not the unilateral incentive to increase its parallelism degree, since its effective performance can not be improved if the other modules do not change their control decisions. With the second condition each ParMod M_i has not the unilateral incentive to decrease the current parallelism degree (smaller parallelism degrees make the local cost function worse off).

At each control step, Control Parts interact in order to reach an agreement modeled by a Nash equilibrium. Starting from their ideal parallelism degree, the best response of ParMod M_i can be identified as follows:

- *a parallelism degree smaller than the ideal parallelism degree*: due to the presence of a module acting as a bottleneck, the best response of M_i is a parallelism degree $\tilde{n}_i(k)$ that allows the module to adapt to the bottleneck's performance;
- *the ideal parallelism degree*: in this case M_i is the bottleneck, and it has no incentive to deviate from the current control decision.

In the first case, $\tilde{n}_i(k)$ represents the parallelism degree that allows M_i to reach a service time equal to its steady-state inter-arrival time from the source modules (i.e. avoiding to be unnecessary fast). It can be calculated as: $\tilde{n}_i(k) = T_{calc\text{-}i}(k)/f_{i,b}(k)$ where M_b denotes the bottleneck of the graph.

It is worth noting that the same best responses are also valid when modeling non-ideal performance behaviors of parallel modules, e.g. when the service time stops to decrease or even increases after a specific parallelism degree. Intuitively, the service time should be modeled as a convex function of the parallelism degree to have the same best responses described before.

Control Parts exchange their service times (interconnecting variables) and apply their best response in order to adapt to the performance advertised by their neighbors. Assuming a connected graph between controllers, a Nash equilibrium can be reached after a fixed number of information exchanges, equal to the *diameter* of the network. Informally the rationale is that control decisions need to propagate reaching all the other controllers in a finite number of information exchanges. Since best responses are monotonically decreasing (adopting the ideal parallelism degrees as the starting point), the final result converges to an efficient Nash equilibrium in a Pareto sense. Finally, each Control Part selects an integer parallelism degree obtained by applying an integer rounding of the final result. Further details and the proofs about the correctness of this approach can be found in [12].

4.1.2 A Cooperative Approach

In the previous strategy the interactions between Control Parts have been described in terms of the best responses. In this section we present a different approach in which controllers solve their local control problems in a cooperative way: i.e. having in mind a global objective function of the whole system (e.g. defined as the sum of local cost functions).

Although a Nash equilibrium can be Pareto efficient, it can differ from the set of parallelism degrees that optimize the sum of the local cost functions (named *Social Optimum*). This is due to the selfish behavior of controllers: Control Parts can globally improve the total cost if they select control actions taking into account not only their local outcome, but also the effects of their actions on the other controllers.

We solve the cooperative problem using the ***Distributed Subgradient Method***, originally proposed in [13] for multi-agent environments. The method addresses the problem of optimizing the sum $J_G(k) = \sum J_i(k)$ of non-smooth convex functions known only by their agents. This method has the following set of interesting properties:

- each Control Part knows only its local cost function and the model to predict the steady-state performance of its Operating Part;
- each local cost is expressed by a *non-differentiable convex* function (we recall that the inter-departure time is defined as the point-wise maximum of a set of convex functions $f_{i,j}$);
- for scalability and reliability reasons, Control Parts are directly interconnected only between neighbors.

In this strategy controllers exchange their estimate of the globally optimal strategy profile $\mathcal{S}(k) \in \mathbb{R}^N$, i.e. a set of control actions (i.e. parallelism degrees) for each ParMod. Controllers iteratively exchange their local estimates and compute the next estimate using the following rule:

$$\mathcal{S}_{[i]}^{(q+1)}(k) = \mathcal{P}_{\mathcal{U}_f} \left[\sum_{j=1}^{N} \Big(\mathcal{W}[i,j]\, \mathcal{S}_{[j]}^{(q)}(k) \Big) \; - \; a^{(q)}\, \mathcal{G}_i \right] \qquad (9)$$

where q is the current iteration, $a^{(q)} > 0$ is the *step-size* and $\mathcal{G}_i$ is a *subgradient* of J_i at point $\mathcal{S}_{[i]}^{(q)}(k)$[3]. $\mathcal{P}_{\mathcal{U}_f}$ is the Euclidean projection onto the convex set of admissible strategy profiles defined by: $\mathcal{U}_f = \mathcal{U}_1 \times \mathcal{U}_2 \times \ldots \times \mathcal{U}_N$.

Each controller maintains a set of weights representing the importance given to the estimates received by the controllers (zero is assigned to non-neighbor controllers). To prove the convergence to the social optimum, in [13] the authors state a condition about how the weights should be assigned: the weight matrices $\mathcal{W} \in \mathbb{R}^{N \times N}$ should be *doubly stochastic*, i.e. all the columns and rows sum to 1.

The cooperative MPC strategy consists in a sequence of actions performed by the controllers at each control step k:

[3] $\mathcal{S}_{[i]}^{(q)}(k)$ is the estimate of the i-th controller at iteration q.

- each controller acquires monitoring information from its Operating Part and calculates one-step ahead predictions of disturbances;
- each controller uses a specific initial estimate of the social optimum and applies the iterative protocol for a fixed number of iterations;
- after the last iteration, each controller knows its optimal parallelism degree and applies it (properly rounded to the nearest integer) as the new parallelism degree for control step k.

In this formulation, since we adopt a horizon of 1 step in the future, a reconfiguration trajectory corresponds to a parallelism degree for the current control step.

In terms of number of exchanged messages, this strategy is outwardly similar to the selfish approach described in Section 4.1.1. It consists in a sequence of iterations in which each controller receives local estimates from the neighbors, updates its local estimate and transmits it the neighboring controllers. W.r.t the selfish strategy, in which the number of information exchanges depends on the topological properties of the computation graph (diameter), here the number of iterations should be sufficient enough to guarantee a desired approximation of the social optimum. However, the feasibility can be drastically improved by considering two aspects:

- each controller applies an integer rounding of the final parallelism degree;
- to reduce the number of iterations, we can use as the starting estimate the social optimum calculated at the previous control step (*warm start*), which is likely close to the new social optimum.

4.1.3 Comparison and Optimality

The two adaptation strategies have been applied to a simple computation graph composed of three parallel modules (Figure 3). Tasks are generated by the first module, which dispatches them to the other ParMods according to a discrete probability distribution ($p(k)$ is the probability to transmit tasks to the second ParMod). We have simulated a scenario in which $p(k)$ periodically fluctuates, alternating execution phases in which the second module is much more stressed and phases in which tasks are more often transmitted to the third module. $p(k)$ variates for uncontrollable reasons related to the application semantics, thus it has been modeled as a disturbance variables.

Figure 4 depicts the mean values of $p(k)$ at each sampling interval of an execution of 600 control steps (each one of 240 seconds). This behavior is typical of a *seasonal workload*. For this reason we have performed statistical predictions (the solid red line in the figure) using a seasonal Holt-Winters smoothing technique [3] (composed of three EWMA filters for the smooth, trend and the seasonal components). The results show quite accurate predictions, with a relative error for one-step predictions less than 10%.

Due to the variations of $p(k)$, the optimal parallelism degrees change during the execution. In the non-cooperative approach, at each control step the controllers select an integer approximation of the efficient Nash equilibrium. In the cooperative approach controllers reach an approximation of the social optimum at each control step, by applying the Distributed Subgradient Method.

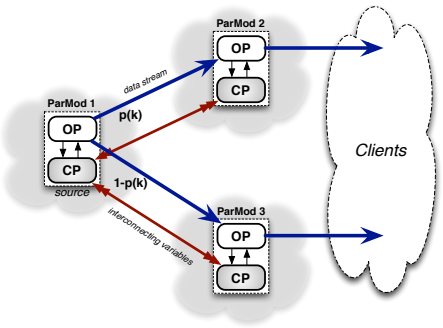


Figure 3: Computation graph of the first experiment.

The configuration parameters and the mean calculation times are summarized in Table 1. Cost parameters (α and β) are user-defined and state the desired trade-off between performance and resource consumption. Although the α parameter is the same for all the modules (i.e. they give the same importance to the performance), each ParMod applies a different resource utilization cost since we suppose that parallel modules are executed on Cloud platforms applying different cost parameters. The calculation times are generated according to normally distributed random variables.

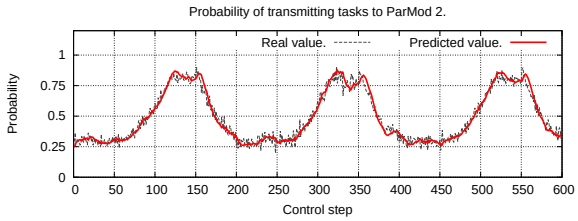


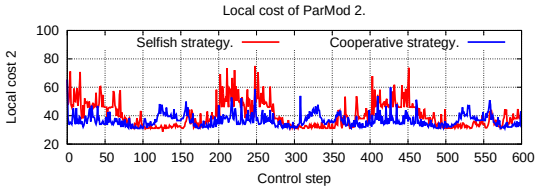
Figure 4: Probability time-series.

Figure 5 shows the actual value of local cost functions of the second ParMod and of the global cost for each control step of the execution. The red lines correspond to the selfish strategy while the blue lines are the values of the local costs with the cooperative approach. Although there are time intervals in which the local cost of the second module is better using the selfish strategy (Figure 5a), in the average case the cooperative approach provides a better global cost throughout the execution (Figure 5b). *This behavior is the essence of cooperation: in order to optimize the sum of the local costs, there are execution phases in which the second ParMod selects non-locally optimal control decisions.*

Table 2 reports the total values of the global and the local

	ParMod 1	**ParMod 2**	**ParMod 3**
T_{calc-i}	15 sec.	30 sec.	40 sec.
α_i	4	4	4
β_i	0.6	2.5	1.5

Table 1: Cost parameters and mean calculation times.



(a) Local cost of ParMod 2.

(b) Global Cost.

Figure 5: Comparison between selfish and cooperative strategies.

costs over the entire execution. In terms of total cost, the cooperative approach achieves a 20% reduction compared to the selfish strategy. This means that the average *price of stability* (ratio between the cost of the best Nash equilibrium and that of the global optimal outcome) is equal to 0.80 in this example. This is the optimality loss due to the selfish interaction between controllers.

	Non-cooperative	**Cooperative**
Total J_1	7,778	4,939
Total J_2	24,375	21,990
Total J_3	25,994	19,490
Total J_G	58,147	46,419

Table 2: Sum of local and global costs with the non-cooperative and cooperative strategies.

In terms of exchanged messages, the selfish strategy can be completed after two information exchanges (which is the diameter of the graph shown in Figure 3). For the cooperative strategy the number of iterations is one order of magnitude higher (experiments have been performed using 125 iterations per step).

4.2 Improving the Reconfiguration Stability

In this section we describe the potential of our control strategies by introducing different formulations of local cost functions. Besides modeling a simple cost proportional to the resource utilization, more complex strategies can account for the cost of switching from a configuration to another one. This can be a key factor in heterogeneous environments like Clouds, in which the cost of applying a reconfiguration (which requires the dynamic provisioning of computing resources) is of paramount importance in addition to the reaching of desired QoS levels. We introduce a second formulation characterized by a local cost function defined as follows:

Definition 2 (Switching Cost Formulation). *The local cost function of each ParMod M_i is defined over a prediction horizon of h control steps (with $h \geq 1$):*

$$J_i(k) = \underbrace{\sum_{q=k}^{k+h-1} \alpha_i \, T_{D_i}(q+1)}_{performance\ cost} + \underbrace{\sum_{q=k}^{k+h-1} \beta_i \cdot n_i(q)}_{resource\ cost} + \underbrace{\sum_{q=k}^{k+h-1} \gamma_i \cdot \Delta_i(q)^2}_{switching\ cost} \quad (10)$$

where $\Delta_i(k) = n_i(k) - n_i(k-1)$. The switching cost term binds control decisions between consecutive steps allowing to express formulations with a parametric horizon length.

The main goal of this formulation is to discourage reconfiguration with large amplitude (i.e. involving a great number of released/provisioned computing nodes), that we can assume to be more expensive (both economically as well as from a performance viewpoint). The switching cost acts as a break to reconfigurations: it avoids fluctuating behaviors due to disturbance with high variance and, unlike the formulation described in (5), makes it possible to consider a MPC strategy with an arbitrary horizon length.

In order to understand the importance of this alternative formulation, we extend the example described in Section 4.1.3 by applying a cooperative formulation using the local costs introduced in Definition 2. The convergence to the social optimum is proven by the fact that the local cost functions with the switching cost maintain the convexity property.

To highlight the importance of the switching cost, we consider a different time-series of probability $p(k)$ which exhibits several small phases characterized by trend components (Figure 6). Also for this disturbance series, we apply a multiple step-ahead Holt-Winters forecasting technique, which gives a relative error of 8.92%, 9.22%, 9.49% and 9.69% using horizons of 1, 2, 3 and 4 control steps.

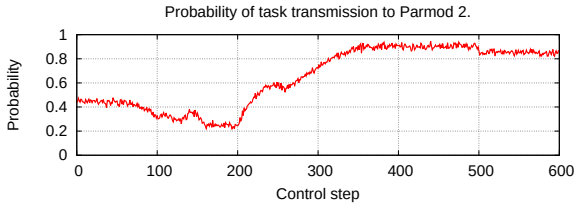


Figure 6: Second experiment: probability $p(k)$.

As we use longer horizons, controllers can more precisely evaluate if the release/acquisition of computing resources is effectively useful (e.g. avoiding to re-acquire/release them nearly in the future). This has important implications for the reconfiguration stability of the adaptation strategy, more formally defined as follows:

Definition 3 (Mean Stability Index). ***Mean Stability Index*** *(shortly* `MSI`*) is the average number of control steps for which no change in the parallelism degree occurs.*

In this example we apply the same cost parameters defined in Table 1 (we apply the same switching cost coefficient for the three modules, i.e. $\gamma_i = 3.5$). Table 3 outlines the global simulation results. The completed tasks is the number of tasks that leave the system. The formulation using the local cost functions defined in (5) (referred as *Non-Switching Cost* for brevity) produces 434 reconfigurations and completes 80,063 tasks. Such a high number of

reconfigurations is justified by the fact that this strategy precisely adapts to disturbance predictions without any break that slows down the allocation/release of resources. With the switching cost and a horizon of one step, we are able to drastically reduce the number of reconfigurations (43% less) with a small performance loss of 9.5%. If we consider longer prediction horizons we reach even more satisfying trade-offs between performance and reconfiguration stability (longer-horizon strategies respond more quickly to a pronounced change in the disturbance trend). With a horizon of four steps we loose only 0.69% of tasks but decreasing the reconfigurations of 21% w.r.t the Non-Switching Cost strategy.

	Reconf.	MSI	Tasks
Non Switching Cost	434	3.93	80,063
Switch. Hor=1	247	8.46	72,451
Switch. Hor=2	328	6.51	75,074
Switch. Hor=3	331	7.12	77,788
Switch. Hor=4	341	7.32	79,503

Table 3: Completed tasks and reconfiguration stability.

The `MSI` gives a quantitative measure of the reconfiguration stability. Without the switching cost, ParMods apply a parallelism degree variation every 4 control steps on average. By introducing the switching cost the reconfiguration stability increases significantly.

5. CONCLUSION

This paper presents an overview of our methodology [12]. Performance models of structured parallelizations can be used for applying finite-horizon MPC procedures based on statistical predictions of disturbances. The methodology is amenable to model complex situations in which controllers communicate pursuing their self-interest or cooperate in order to reach optimal reconfigurations in a system-wide sense. Specific formulations can be properly tuned in order to improve significant aspects of adaptation strategies. We claim that our methodology is sufficiently flexible to model adaptation strategies with different goals (e.g. control optimality, reconfiguration stability, little reconfiguration amplitude). The impact on real Cloud platforms is an important argument that still requires further investigations in the future.

6. REFERENCES

[1] M. Aldinucci, S. Campa, M. Danelutto, and M. Vanneschi. Behavioural skeletons in gcm: Autonomic management of grid components. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008.*, pages 54–63, Feb. 2008.

[2] M. Aldinucci, M. Danelutto, and M. Vanneschi. Autonomic qos in assist grid-aware components. In *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 221–230, Washington, DC, USA, 2006. IEEE Computer Society.

[3] C. Chatfield and M. Yar. Holt-winters forecasting: Some practical issues. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 37(2):pp. 129–140, 1988.

[4] M. Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Comput.*, 30(3):389–406, 2004.

[5] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice a survey. *Automatica*, 25:335–348, May 1989.

[6] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[7] S. Islam, J. Keung, K. Lee, and A. Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155 – 162, 2012.

[8] J. Kephart and W. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 3–12, June 2004.

[9] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):341 – 352, may 2006.

[10] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 7(4):36:1–36:32, Dec. 2012.

[11] D. Meiländer, A. Ploss, F. Glinka, and S. Gorlatch. A dynamic resource management system for real-time online applications on clouds. In *Proceedings of the 2011 international conference on Parallel Processing*, Euro-Par'11, pages 149–158, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] G. Mencagli. *A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations*. Ph.D Thesis, University of Pisa, Italy, 2012.

[13] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48 –61, jan. 2009.

[14] M. Vanneschi and L. Veraldi. Dynamicity in distributed applications: issues, problems and the assist approach. *Parallel Comput.*, 33(12):822–845, 2007.

[15] D. Warneke and O. Kao. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Trans. Parallel Distrib. Syst.*, 22(6), 2011.

[16] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 301–, Washington, DC, USA, 2002. IEEE Computer Society.