A Cooperative Predictive Control Approach to Improve the Reconfiguration Stability of Adaptive Distributed Parallel Applications

Gabriele Mencagli, University of Pisa
Marco Vanneschi, University of Pisa
Emanuele Vespa, University of Pisa

Adaptiveness in distributed parallel applications is a key feature to provide satisfactory performance results in the face of unexpected events such as workload variations and time-varying user requirements. The adaptation process is based on the ability to change specific characteristics of parallel components (e.g. their parallelism degree) and to guarantee that such modifications of the application configuration are effective and durable. Reconfigurations often incur a cost on the execution (a performance overhead and/or an economic cost). For this reason advanced adaptation strategies have become of paramount importance. Effective strategies must achieve properties like control optimality (making decisions that optimize the global application QoS), reconfiguration stability expressed in terms of the average time between consecutive reconfigurations of the same component, and optimizing the reconfiguration amplitude (number of allocated/deallocated resources). To control such parameters, in this paper we propose a method based on a Cooperative Model-based Predictive Control approach in which application controllers cooperate to make optimal reconfigurations and taking account of the durability and amplitude of their control decisions. The effectiveness and the feasibility of the methodology is demonstrated through experiments performed in a simulation environment and by comparing it with other existing techniques.

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

General Terms: Design, Theory, Performance

Additional Key Words and Phrases: Parallel Computing, Autonomic Computing, Model-based Predictive Control, Distributed Cooperative Optimization, Reconfiguration Stability.

ACM Reference Format:
Mencagli, G., Vanneschi, M., Vespa, E. 2012. A Cooperative Predictive Control Approach to Improve the Reconfiguration Stability of Adaptive Distributed Parallel Applications. ACM Trans. on Auton. and Adapt. Syst. 9, 4, Article 39 (June 2013), 28 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Nowadays the scientific community widely recognizes the need for smart strategies and run-time supports to make distributed parallel applications adaptive. By *adaptiveness* we mean the general property of a system to react to statically unpredictable dynamics related to the computation semantics and the execution environment. The adaptation process involves the ability to perform *dynamic reconfigurations* (e.g. vary-

Author's addresses: G. Mencagli, M. Vanneschi and E. Vespa, Computer Science Department, University of Pisa, (Current address) Largo B. Pontecorvo, 3, I-56127, Pisa, Italy. Email: {mencagli, vannesch, vespa}@di.unipi.it

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

ing the parallelism degree and the parallel version adopted by software components) and the control logic that takes proper reconfigurations at certain time instants.

Reconfigurations often induce performance and economic costs. Before making a reconfiguration decision, it is important to consider the cost of the newly selected configuration (e.g. in a pay-per-use environment dependent on the class of dynamically provisioned computing resources [Yuan et al. 2011; Costa et al. 2013]). The reconfiguration cost can also be an economic cost or a performance overhead [Vanneschi and Veraldi 2007] proportional to the *reconfiguration amplitude*, i.e. number of computing resources allocated/deallocated during a change in the application configuration. Consequently, *reconfigurations should be executed only when they bring real benefits in achieving the desired Quality of Service*.

In this direction applications need to be equipped with effective *adaptation strategies*. Apart from the desired control optimality (e.g. trade-off between performance and resource consumption), strategies can be compared according to their reconfiguration *stability* and *amplitude*. In this paper we use the term stability with a different meaning w.r.t the classic concept of closed-loop stability used in Control Theory. In our vision the degree of stability of an adaptation strategy, hereinafter referred as *reconfiguration stability*, expresses how frequent reconfigurations are issued by the adaptation strategy. This concept can be quantitatively captured by the average time between consecutive reconfigurations of the same part of the controlled system. Due to the presence of reconfiguration costs, the goal of adaptation strategy is to achieve the desired QoS by performing the strictly necessary number of reconfigurations. If the effects of reconfigurations are robust and durable, the result is a performance improvement and a reduction of operating costs.

The literature is rich in work related to adaptiveness for distributed systems. Some approaches are inspired by Artificial Intelligence. Examples are strategies programmed using declarative languages [Kephart and Walsh 2004] (e.g. event-condition-action rules) that map undesirable situations onto corrective actions. Similar strategies have been adopted to control the parallelism degree of Algorithmic Skeletons in [Aldinucci et al. 2008; Weigold et al. 2012]. Another research direction is based on Control Theory foundations [Hellerstein et al. 2004]. The optimization of performance, power and resource consumption of physical and virtualized platforms such as HPC clusters and data-centers have been addressed using Admission Control [Park and Humphrey 2011] and Optimal Control [Kusic et al. 2011]. Although the results of that work are convincing, properties like reconfiguration stability and amplitude are not directly addressed. Targeting these aspects and providing more application-oriented methodologies is an important research problem still open to new contributions.

In our seminal work [Mencagli 2012; Mencagli and Vanneschi 2011; 2013] we have applied *Model-based Predictive Control* [Garcia et al. 1989] (shortly MPC) to distributed parallel applications modeled as general graphs of parallel components (e.g. streaming computations such as video surveillance, continuous query and digital signal processing systems). This approach is based on two fundamental assumptions:

— the capability to predict the future application QoS using a performance model of the most relevant metrics of interest;
— the possibility to express trade-offs between different QoS requirements using a set of objective functions associated with application components.

In this strategy the application behavior is predicted to find the optimal sequence of reconfigurations over a finite time horizon. By using statistical predictions of external factors influencing the application execution, it is possible to make reconfigurations in advance reducing QoS violations. In this paper we extend our prior work by providing formal adaptation strategies taking account of stability and amplitude aspects of

reconfigurations, and optimizing performance and efficiency goals. Our main contributions are:

— we introduce multiple-step ahead predictive strategies in which reconfigurations take into account not only what is going to happen in the immediate future, but also what is likely to happen further in the prediction horizon;
— we introduce a *Switching Cost* term that has significant effects on the number of reconfigurations and their amplitude;
— we compare our approach with adaptation strategies expressed by policy rules.

This paper is organized as follows. In the next section we compare our approach with other existing solutions. Section 3 introduces our methodology. In Section 4 we present predictive control strategies. Section 5 applies our approach to an example of optimization between performance and resource consumption objectives, and provides a first evaluation of our strategies by simulations. In Section 6 we give the conclusion of this work and the future directions of our research.

2. RELATED WORK

In the context of distributed parallel applications adaptiveness is a critical feature which requires a careful design of run-time supports and the definition of decision-making strategies. A first issue concerns how reconfigurations are defined and executed, since they represent intrusive actions often inducing performance degradations [Arshad et al. 2007; Gomes et al. 2007; Tsai et al. 2007]. The work presented in [Vanneschi and Veraldi 2007] provides an overview of reconfiguration issues for structured parallelism patterns (task-parallel and data-parallel programs).

The definition of adaptation strategies is a complex issue which leads to interdisciplinary researches in fields like Control Theory and Artificial Intelligence. A solution consists in using *policy logic rules* [Kephart and Walsh 2004], which specify a mapping between unexpected events and corresponding corrective actions on the system. Frameworks adopting this vision are described in [Liu and Parashar 2006] for general distributed systems, and [Aldinucci et al. 2008; Coppola et al. 2007; Aldinucci et al. 2006] for high-performance applications. In [Ghanbari et al. 2011] an accurate comparison between mathematical approaches and policy rules (called heuristic rules) is discussed in the domain of elasticity for Cloud environments.

Policy rules can pose serious problems of programmability and effectiveness when conflicts between rules arise [Reiff-Marganiec and Turner 2004]. In [Aldinucci et al. 2009] the control of large-scale systems has been approached by considering applications as composition and nesting of Algorithmic Skeletons [Cole 2004], and organizing a corresponding hierarchy of controllers featuring their adaptation rules. In our opinion this view may be unrealistic, especially because parallelization is often introduced when specific sub-parts of preexisting systems act as performance bottlenecks. Our approach starts from a different perspective: applications will be considered as general as possible while the parallelization inside each component will be performed by instantiating structured parallelism patterns.

Besides rule-based approaches, strategies investigating the applicability of Control Theory to computing systems have moved beyond the preliminary stage. A formal control of software performance is described in [Zhang et al. 2002]. Application of PID (Proportional-Integral-Derivative) controllers to web servers and enterprise applications is described in [Hellerstein et al. 2004; Horvath et al. 2007; Raghavendra et al. 2008] with encouraging results in controlling non-functional aspects like performance and power consumption. Admission Control of HPC servers has been recently described in [Park and Humphrey 2011] using black-box models to predict job progress and the CPU allocation.

A thorough overview of control-theoretic strategies is presented in [Maggio et al. 2012]. In this work it emerges that Optimal Control [Garcia et al. 1989] is a promising approach, not only in theory but in real applications. In [Abdelwahed et al. 2004] and [Kusic and Kandasamy 2007] a finite-horizon optimal strategy has been applied to the dynamic adaptation of the clock rate of embedded CPUs and to control the number of machines allocated to a web server.

An emerging field in which these concepts can be applied is the dynamic allocation of Cloud resources [Yuan et al. 2011; Costa et al. 2013]. An example is shown in [Kusic et al. 2011], in which optimal control is applied by adapting the number of active virtual machines. Our work extends the actual state-of-the-art by giving the following contributions:

— our approach can be applied to general distributed parallel applications whose structure is provided in terms of interacting components, their internal performance and the probability of performing communications among distributed entities;
— the methodology is applied directly to applications, i.e. coupling them with their control logic. This solution distinguishes from other approaches oriented towards the control of physical or virtualized computing platforms, e.g. autonomic control of data centers and HPC servers as in [Kusic et al. 2011; Loureiro et al. 2012; Park and Humphrey 2011; Wang et al. 2008; Khargharia et al. 2008];
— we use a *single-layer organization* in which controllers are at the same level of authority. We avoid to use hierarchical schemes that may introduce single points of failure and more complex system models (e.g. taking into account the behavior of lower-control layers too [Kandasamy et al. 2006; Aldinucci et al. 2009]);
— optimal control techniques are computationally expensive and must be solved using heuristics and relaxations to alleviate the combinatorial state-space explosion [Abdelwahed et al. 2009]. In this paper we apply a feasible cooperative method based on a continuous relaxation of the original problem. This makes it possible to consume a negligible part of resources for the control activity with a minor optimality loss;
— we introduce two meaningful metrics, the reconfiguration stability and the maximum reconfiguration amplitude, and we use them to compare adaptation strategies.

3. PREDICTIVE CONTROL OF DISTRIBUTED PARALLEL APPLICATIONS

In this section we describe the concept of adaptive parallel module (namely *ParMod*) and the adaptation strategy to control graphs of parallel components. In this paper we use the term parallel module (ParMod) as a synonym of application component.

3.1. Adaptive Parallel Modules

A distributed parallel application can be represented as a directed acyclic computation graph in which the vertices are parallel modules that process the data (representing tasks), and the edges are continuous data streams that connect two or more components. Each ParMod is composed of two parts organized in a closed-loop model:

— the **Operating Part** performs a parallel computation instantiating structured parallelism paradigms (e.g. task-farm and data-parallel schemes). The computation is activated by receiving tasks from input data streams from other application components;
— the **Control Part** (controller) observes the Operating Part and performs reconfiguration activities (e.g. run-time modifications of the parallelism degree - by allocating new threads/processes and connecting them to the actual computation structure).

Figure 1 shows an abstract representation of two interconnected ParMods. The closed-loop interaction consists in the following information flows:

— ***monitoring data*** from Operating Part to Control Part are metrics that describe the current computation behavior (e.g. memory usage, resource utilization, service time and computation latency);
— ***reconfiguration commands***, issued by the Control Part, are messages that trigger the execution of reconfiguration activities;
— ***control messages***, exchanged between controllers, are aimed at making the local controller of a ParMod aware of the decisions taken by the other controllers.

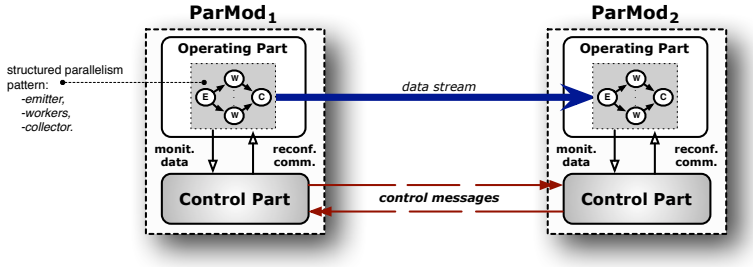


Fig. 1: Internal structure of adaptive parallel modules and their interconnections.

In our approach the passage of time is discretized in periods of fixed length, i.e. Control Parts are time-driven controllers. We call *control step* the time period of length τ between two successive evaluations of the adaptation strategy.

In the next section we will describe in more detail our control approach based on a well-known control-theoretic technique.

3.2. Distributed Model-based Predictive Control

The problem of controlling large-scale systems can be decomposed into a set of sub-problems corresponding to distinct parts of the system. Each sub-problem consists in: (i) a *local model*, which states how local QoS metrics change in response to specific reconfigurations; (ii) a *local objective*, e.g. formulated as a cost function taking into account future QoS levels and the cost of local control actions. Each model involves the following set of variables:

— ***QoS variables*** ($\mathbf{x}_i(k) \in \mathbb{R}^n$) of the sub-system i for a given control step k;
— ***control variables*** ($\mathbf{u}_i(k) \in \mathcal{U}_i$) which identify the sub-system configuration used during control step k;
— ***disturbances*** ($\mathbf{d}_i(k) \in \mathbb{R}^m$) model exogenous uncontrollable events that affect the relationship between control and QoS variables.

A general representation of the local model can be described by the following discrete-time expression:

$$\mathbf{x}_i(k+1) = \Phi_i\Big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k),\, \mathbf{u}_{j\neq i}(k)\Big) \tag{1}$$

QoS variables of sub-system i are expressed as a function of local control inputs, disturbances 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*. Furthermore, the next QoS of each sub-system is also related to the remaining control variables of the other

sub-systems (or a sub-set of them). Therefore, *the control problem of the whole system can be viewed as a set of coupled sub-problems* (as sketched in Figure 2).

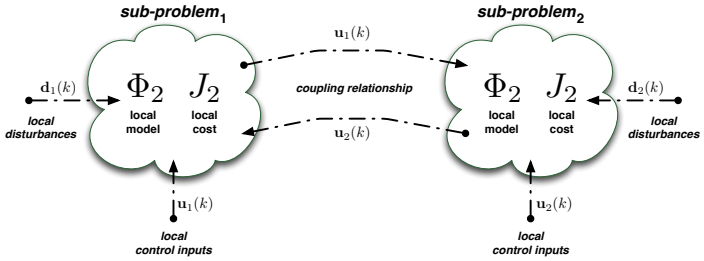


Fig. 2: Two interconnected sub-problems and their model variables.

In our approach distributed parallel applications are decomposed into a set of coupled sub-systems each one corresponding to a ParMod. Reconfigurations are taken by solving a finite-horizon distributed optimization problem. All Control Parts share the same notion of control step and prediction horizon (formed by a set of h consecutive control steps). At the beginning of each step, controllers predict the future values of disturbances over the horizon by applying history-based statistical filters. Each local cost is defined as a function of the future sequence of QoS variables achieved applying a trajectory of local control inputs (*reconfiguration plan*):

$$J_i(k) = \sum_{j=k}^{k+h-1} L\Big(\mathbf{x}_i(j+1),\, \mathbf{u}_i(j)\Big) \tag{2}$$

Due to the presence of coupling relationships between sub-problems, controllers exchange information (control messages) several times at each control step. The goal is to achieve the desired degree of coordination such that shared variables converge to the same value for all the controllers. We are interested in *cooperative* formulations in which controllers account for the effects of their actions on the objectives of the other controllers. Accordingly, the individual solutions of each sub-problem have to be equivalent to the optimal solution of the following plant-wide optimization problem:

$$\begin{aligned} &\underset{\overline{U}_1(k),\ldots,\overline{U}_N(k)}{\arg\min} \quad J_G = \sum_{i=1}^{N} w_i\, J_i(k) \\ &\textit{s.t.} \\ &\mathbf{x}_i(k+1) = \Phi_i\big(\mathbf{x}_i(k),\, \mathbf{d}_i(k),\, \mathbf{u}_i(k),\, \mathbf{u}_{j\neq i}(k)\big) \;\; i = 1, 2, \ldots, N \\ &\mathbf{u}_i(k) \in \mathcal{U}_i \;\; i = 1, 2, \ldots, N \end{aligned} \tag{3}$$

where $\overline{U}_i(k)$ is the reconfiguration plan of the i-th sub-system, J_G is a weighted sum of local cost functions (w_i is a positive weight) and N is the number of sub-systems.

Instead of applying the optimal reconfiguration plans in an open-loop fashion, it is more effective to adapt them in response to current disturbances. Model-based Predictive Control [Garcia et al. 1989] is a technique following this rationale. Of the optimal reconfiguration plan calculated at control step k, only the first control decision is effectively applied to the sub-system while the rest is discarded. Then, the procedure is repeated at the next control step using new measurements from the system. The

result is to move the prediction horizon towards the future step-by-step following the so-called receding horizon principle [Garcia et al. 1989].

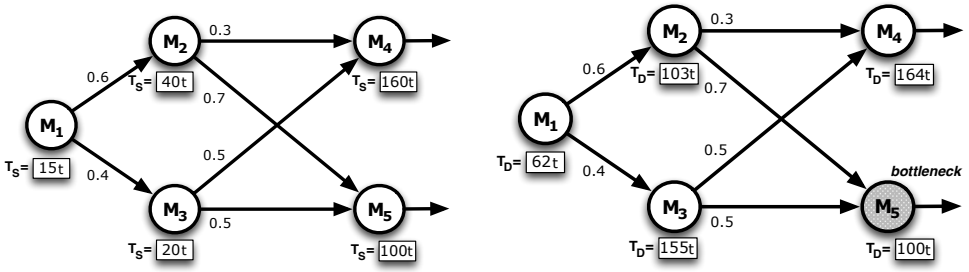
In the following section we will describe a concrete example and the method used to enact the coordination between controllers.

4. APPLICATION OF THE METHODOLOGY

In modern Grid and Cloud environments with seemingly unlimited resources, users are tempted to speed up their computations by continuously increasing their applications parallelism degree. However, if the wrong parts of a distributed application receive more computing resources, the additional resources may remain mostly idle and unnecessarily increase the processing cost. In this scenario the goal of the control problem is to reach desired trade-offs between performance and resource consumption, e.g. avoiding use of unnecessary resources without real benefits in terms of performance.

4.1. Formal Statement of the Control Problem

Each ParMod M_i of a distributed parallel application corresponds to a sub-system featuring its local strategy. The configuration parameters (control variables) are the parallelism degrees $n_i(k) \in \mathcal{U}_i$, where $\mathcal{U}_i$ is the closed interval $[1,\, n_i^{max}]$ of integer values (for the sake of simplicity computing resources are assumed to be homogeneous). Disturbances are parameters that may change unexpectedly and uncontrollably. Examples are the mean calculation time per task $T_{calc\text{-}i}(k)$, and the transmission probabilities between modules. We denote with $p_{i,j}(k)$ the probability of transmitting a task from M_i to M_j during control step k. Figure 3a shows an example of computation graph.



(a) Ideal graph. (b) Steady-state graph.

Fig. 3: The ideal graph is labeled with the service times (t is a standardized time unit), the steady-state graph expresses the effective inter-departure times.

Computation graphs can be modeled as *networks* of *queueing stations*, each one representing in an abstract form a parallel module with an input buffer, an inter-arrival time of tasks from a set of sources, and a *mean service time* $T_{S_i}(k)$.

As stated in Section 3, our methodology assumes that parallelizations inside ParMods follow structured parallelism paradigms. We initially consider a perfect scalability relation between parallelism degree and service time:

$$T_{S_i}(k) = \frac{T_{calc\text{-}i}(k)}{n_i(k)} \tag{4}$$

Even if this assumption could seem rather strong, we will see in the sequel that it is of little impact on our adaptation strategies, as possible non-ideal behaviors can be addressed with very few constraints.

In distributed computing the interaction between components usually follows the message-passing paradigm. Communications rely on *blocking mechanisms* to address the finiteness of input buffers. If a message attempts to enter a full capacity destination queue upon the completion of a service at node M_i, it is forced to wait in that node until the destination has a free position[1]. Therefore, we need a model to predict the effective behavior of ParMods based on the knowledge of the graph structure and the mean service times. We call the *mean inter-departure time* from a module the steady-state average time between two successive result departures from that module. We denote with $T_{D_i}(k)$ the QoS variable representing the inter-departure time of M_i at the beginning of control step k (it refers to the behavior measured at the end of the last control step $k-1$).

Modeling the steady-state performance of a queueing network is a complex problem [Balsamo 2011]. The existing results take important assumptions on the probability distributions [Lee et al. ; Lee and Pollock 1991] which are often restrictive hypotheses in real-world scenarios. In this paper we adopt a simplified modeling approach already discussed in [Mencagli 2012]. This method, valid for a large class of computation graphs - i.e. *acyclic graphs with a single source module* - makes it possible to evaluate the steady-state behavior without any assumption on the service times distribution and in the case of large enough buffers. The main result is summarized by the following theorem (the proof can be found in [Mencagli 2012] at page 213):

THEOREM 4.1 (STEADY-STATE ANALYSIS). *Given a single-source acyclic graph G composed of N modules, the inter-departure time T_{D_i} from each module M_i is given by:*

$$T_{D_i}(k+1) = \max\Big\{ f_{i,1}\big(T_{S_1}(k)\big),\ f_{i,2}\big(T_{S_2}(k)\big), \ldots, f_{i,N}\big(T_{S_N}(k)\big) \Big\} \tag{5}$$

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 M_j service time:

$$f_{i,j}\big(T_{S_j}(k)\big) = T_{S_j}(k)\, \frac{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_s \to M_j)} \left(\prod_{\forall (u,v) \in \pi} p_{u,v}(k) \right)}{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_s \to M_i)} \left(\prod_{\forall (u,v) \in \pi} p_{u,v}(k) \right)} \tag{6}$$

where M_s denotes the source of G, $\mathcal{P}(M_s \to M_i)$ is the set of all the paths in the graph starting from M_s and reaching M_i, and (u,v) is a directed edge of the given path π. 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 functions $f_{i,j}$.

Figure 3b shows the application of the theorem to the graph in Figure 3a. Module M_5 is detected as the *bottleneck*, i.e. the inter-departure times of all the other modules are influenced by the service time of M_5. This concept is formalized as follows:

Definition 4.1 (*Bottleneck*). ParMod M_b is the bottleneck of the graph *iff* for each module M_i its inter-departure time at steady-state is equal to $f_{i,b}(T_{S_b}(k))$.

The accuracy of the method has been evaluated in [Mencagli 2012] (Chapter 3 from page 70) using a queue network simulator. The results (not described in this paper for

[1] in the Queueing Networks literature this kind of blocking is called Blocking-After-Service (BAS).

space reasons) confirm that it is valid for any service time distribution (exponential, uniform and normal) and when the buffer queues are sufficiently sized (few tens of elements are sufficient to get an error smaller than 2%). For multiple-source graphs the analysis is much more complicated, and can not be performed without assuming the service time distribution (usually exponential [Lee et al. ; Lee and Pollock 1991]). However, we can redesign multiple-source computation graphs as graphs with a single fictitious source, in order to apply the previous method without limitations.

In addition to the performance model, the distributed MPC strategy requires the definition of local objective functions for each sub-problem. In this paper we study two different formulations of the MPC strategy. In the first one we do not model any abstract term related to the reconfiguration cost (we refer to this case as *Non-Switching Cost formulation* for brevity):

Definition 4.2 (*Non-Switching Cost Formulation*). Each parallel module has a local cost function defined over a horizon of *one future step*:

$$J_i(k) = \underbrace{\alpha_i \, T_{D_i}\big(k+1\big)}_{\textit{performance cost}} + \underbrace{\beta_i \, n_i(k)}_{\textit{resource cost}} \tag{7}$$

The first part is related to the effective performance achieved by ParMod M_i (i.e. its mean inter-departure time of results): the higher the inter-departure time (the slower M_i) the greater the values assumed by this part of the cost function. The second part expresses a cost proportional to the number of used nodes. α_i and β_i are user-defined coefficients establishing the desired trade-off between the two contrasting aspects of the cost function.

In order not to compromise the consistency and the correctness of computations, reconfigurations must be implemented in a careful way, e.g. avoiding loss of tasks or to provide result duplication (see [Bertolli et al. 2011; Vanneschi and Veraldi 2007] for further considerations of this problem). The consequence is that reconfigurations may have significant effects on the computation performance. For instance the Operating Part can be interrupted for an amount of time representing a performance degradation from the user's viewpoint [Vanneschi and Veraldi 2007; Arshad et al. 2007; Wang et al. 2008]. Given this consideration, in the second formulation of the MPC strategy we account for an abstract switching cost term in the definition of local cost functions:

Definition 4.3 (*Switching Cost Formulation*). The local cost function of each ParMod M_i is defined over a prediction horizon of h control steps (with $h \geq 1$):

$$J_i(k) = \underbrace{\sum_{q=k+1}^{k+h} \alpha_i \cdot T_{D_i}(q)}_{\textit{performance cost}} + \underbrace{\sum_{q=k}^{k+h-1} \beta_i \cdot n_i(q)}_{\textit{resource cost}} + \underbrace{\sum_{q=k}^{k+h-1} \gamma_i \cdot \Delta n_i(q)^2}_{\textit{switching cost}} \tag{8}$$

where α_i, β_i and γ_i are positive weights and $\Delta n_i(k)$ is defined as the difference between consecutive control decisions, i.e. $\Delta n_i(k) = n_i(k) - n_i(k-1)$. Compared to the cost function (7), performance and resource costs are spread over a horizon of h future steps by taking the sum of the inter-departure times and the parallelism degrees for each step of the horizon. The intent of the Switching Cost is to bind control decisions of consecutive steps. In this way the reconfiguration plan calculated at step k is not independent from what happened at step $k-1$.

The formulation with the Switching Cost can be useful in scenarios characterized by disturbances having high variance. In that case it is possible that parallelism degree

variations calculated without using the Switching Cost would have been big in amplitude and prone to some kind of up and down fluctuations. In that case the Switching Cost acts as a brake, mitigating this undesired effect by improving the reconfiguration stability. On the other hand, if disturbances exhibit nonstationarities such as trends and seasonal patterns, sufficiently long horizons can help in modifying the ParMod configuration in advance. This leads to the possibility to reduce the number of reconfigurations and adapt the parallelism degrees directly to the optimal values.

To complete the description, we introduce the following definitions:

Definition 4.4 (*Strategy Profile Matrix*). We denote with $\mathcal{S}(k) \in \mathbb{R}^{h \times N}$ a matrix in which the i-th column represents the reconfiguration plan of ParMod M_i and row j consists of the parallel degrees chosen by ParMods for the j-th step of the horizon:

$$\mathcal{S}(k) = \begin{bmatrix} n_1(k) & n_2(k) & \dots & n_N(k) \\ n_1(k+1) & n_2(k+1) & \dots & n_N(k+1) \\ \vdots & \vdots & & \vdots \\ n_1(k+h-1) & n_2(k+h-1) & \dots & n_N(k+h-1) \end{bmatrix}$$

The goal of the cooperative MPC approach is to solve the plant-wide optimization problem defined in (3). To do that, controllers need to cooperate in order to reach the optimal set of reconfiguration plans:

Definition 4.5 (*Social Optimum*). The social optimum is the strategy profile matrix $\mathcal{S}^{(s)}(k)$ such that the weighted sum J_G of local objectives is optimized.

Once the social optimum has been calculated, each Control Part applies the first control input (parallelism degree) of its optimal trajectory for the current control step k while the rest is discarded. Then, the adaptation strategy is re-evaluated at the next step $k+1$, using the updated measurements to find the new social optimum.

4.2. Cooperative MPC based on the Distributed Subgradient Method

The cooperative control is enforced using the *Distributed Subgradient Method* [Nedic and Ozdaglar 2009; Ram et al. 2009] and by making a continuous relaxation of the problem (i.e. parallelism degrees are considered real-valued variables). This assumption is justified by the current tendency of distributed parallel platforms, equipped with hundreds of computing entities, that allows us to sacrifice a little in the optimality for the sake of feasibility. This method has the following important features:

— each Control Part knows its local cost function and the model to predict the steady-state performance of its Operating Part;
— each local cost function of a controller must be convex;
— the method works without any assumption on the differentiability of local cost functions. This is an important property since our formulations (Definition 4.2 and 4.3) are characterized by *non-differentiable convex functions* (the inter-departure time is modeled as the point-wise maximum of a set of convex functions $f_{i,j}$);
— Control Parts can be completely or partially interconnected with each other.

This method is used to minimize the sum of individual cost functions $J_G = \sum J_i$ and it is based on the following principle: each controller maintains a local estimate of the social optimum. By exchanging their estimates controllers reach consensus on a value that approximates the global optimum point (see Figure 4).

The method proceeds iteratively from a starting local estimate. The estimate[2] at iteration $q+1$ is calculated by applying the following update rule:

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

$\mathcal{N}_i$ is the set of neighbors of i-th controllers, $a^{(q)} > 0$ is the *step-size* and $\mathcal{G}_i$ is a subgradient of J_i at the current estimate. $\mathcal{P}_{\mathcal{U}_f}$ is the Euclidean projection onto the convex set of admissible strategy profile matrices defined as: $\mathcal{U}_f = \mathcal{U}_1^h \times \mathcal{U}_2^h \times \ldots \times \mathcal{U}_N^h$, where each $\mathcal{U}_i$ is the interval of reals $[1,\, n_i^{max}]$. The update rule calculates the next estimate as a combination of the actual value and the received estimates (averaged according to properly assigned weights) and a movement on the direction of a subgradient of J_i. The first part serves to align the local estimate with the decisions of the neighbors, the second part is taken to minimize the local cost J_i.

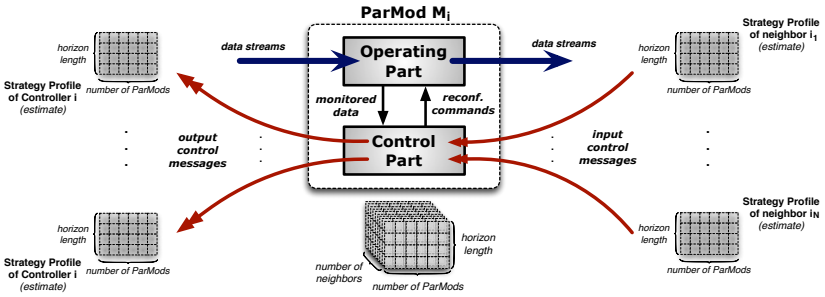


Fig. 4: Control messages between Control Parts of different ParMods.

To reach consensus this method requires a minimal connectivity assumption. We can distinguish between the directed graph formed by Operating Parts and the undirected graph of Control Parts. In principle the two graphs can have a completely different structure. However, a practical approach adopts the following rule:

PROPOSITION 4.6 (NEIGHBORING CONTROL PARTS). *Two Control Parts are interconnected iff there exists a data stream that interconnects their Operating Parts.*

As demonstrated in [Nedic and Ozdaglar 2009; Lobel and Ozdaglar 2011], the minimal connectivity assumption is related to the structure of the controller graph:

ASSUMPTION 4.1 (CONNECTIVITY). *To reach consensus the controller graph must be connected, i.e. there must exist a path (a sequence of Control Parts interconnected in the sense of Proposition 4.6) connecting any pair of controllers.*

To prove convergence to the social optimum, in [Nedic and Ozdaglar 2009; Lobel and Ozdaglar 2011] the authors state a further condition about how weights are assigned to the neighbors: the matrix $\mathcal{W} \in \mathbb{R}^{N\times N}$ must be *doubly stochastic*, i.e. all the columns and rows sum to 1. An admissible weight assignment is given by the following rule:

[2] $\mathcal{S}_{[i]}^{(q)}(k)$ is the strategy profile matrix estimate of controller i after q iterations.

Definition 4.7 (*Symmetric Weights Rule*). Each controller CP_i (Control Part of M_i) applies the following weights:

$$\begin{cases} \mathcal{W}[i,j] = \min\Big\{\dfrac{1}{|\mathcal{N}_i|+1}, \dfrac{1}{|\mathcal{N}_j|+1}\Big\} \ \forall\, j \in \mathcal{N}_i \\ \mathcal{W}[i,i] = 1 - \sum\limits_{j \in \mathcal{N}_i} W[i,j] \end{cases} \tag{10}$$

The local estimates converge to an approximation of the social optimum if: (i) the connectivity assumption holds; (ii) the weight matrix is doubly stochastic; (iii) each local cost function J_i is convex. Our formulations satisfy the convexity property, since:

— each $\mathcal{U}_i$ is a closed interval of reals, therefore also $\mathcal{U}_f$ is convex by definition;
— each $f_{i,j}$ is a convex function of the parallelism degree $n_j(k)$;
— the piece-wise maximum of convex functions is also a convex function.

It is worth noting that this approach is also capable of modeling non-ideal behaviors of ParMods (e.g. when the service time stops to decrease or even increases after a specific parallelism degree), provided that the service time of each ParMod is expressed as any convex function of the corresponding parallelism degree.

4.2.1. Subgradient calculus and cooperative protocol. The distributed subgradient method requires to find a subgradient at each iteration. Since J_i is not differentiable, there can exist more than one subgradient at a given point. The set of all subgradients at a point x is called *subdifferential* denoted by $\partial J_i(x)$. To calculate a subgradient of a convex function F, we exploit the following rules:

— if F is differentiable at x, the unique subgradient is the gradient at that point, i.e. $\partial F(x) = \{\nabla F(x)\}$;
— for $\delta \geq 0$, $\partial\big(\delta\, F(x)\big) = \delta\, \big(\partial F(x)\big)$;
— let $F(x) = F_1(x) + \ldots + F_n(x)$ where each F_i is convex, then $\partial F(x) = \partial F_1(x) + \ldots + \partial F_n(x)$ where $+$ is the Minkowski addition between two sets;
— let $F(x) = \max_{i=1,\ldots,n} F_i(x)$ where each F_i is convex, then $\partial F(x) = \text{Co}\{\partial F_i(x) \,|\, F_i(x) = F(x)\}$, i.e. the subdifferential is the convex hull of the subdifferentials at x of each active function (a function that gives the maximum at that point).

Figure 5 outlines the pseudo-code of the cooperative protocol. The procedure consists of two phases. In the first one, Control Parts disseminate predicted values of local disturbances to the other controllers (this can be easily done using the controller interconnections and a fixed number of information exchanges). In the second phase the distributed subgradient method is executed for $\mathcal{I}$ iterations. At the end, each controller CP_i uses its final control trajectory, i.e. the i-th column of $\mathcal{S}^{(q)}_{[i]}(k)$. Next, only the first element of that trajectory is applied and the rest of the sequence is discarded.

At each iteration controllers receive estimates from their neighbors, update their local estimates and transmit them. The number of exchanged messages is given by the following expression (N is the number of modules):

$$N_{msg} = \mathcal{I} \sum_{i=1}^{N} \big|\mathcal{N}_i\big| = \mathcal{I}\, 2|E|$$

Since the sum of the vertex degrees of an undirected graph is two times the number of arcs $|E|$, the total number of messages is proportional to the square of the number of controllers for dense graphs or linear with the number of controllers for sparse graphs.

Fig. 5: *Cooperative interaction protocol.*

foreach *control step* k *each* CP_i **do**
  acquisition of last measured disturbances from OP_i;
  statistical prediction of disturbances for control step k;
  disturbances dissemination between controllers;
  $\mathcal{S}_{[i]}^{(0)}(k) =$ initial_point;
  for $q = 0$ **to** $\mathcal{I} - 1$ *do* **do**
    send_to_neighbors$\left(\mathcal{S}_{[i]}^{(q)}(k)\right)$;
    receive_from_neighbors$\left(\mathcal{S}_{[j]}^{(q)}(k)\ \forall j \in \mathcal{N}_i\right)$;
    calculate $\mathcal{S}_{[i]}^{(q+1)}(k)$ using the subgradient rule;
  end
  use an integer rounding of the component $(1, i)$ of $\mathcal{S}_{[i]}^{(q)}(k)$ as the new parallelism degree for step k;
end

5. SIMULATION ENVIRONMENT AND EXPERIMENTS

To provide a first evaluation of our approach, we have developed a simulation environment based on the OmNeT++ discrete event simulator[3]. The simulation goal is to analyze from a qualitative and quantitative viewpoint the reconfiguration decisions taken by parallel components involved in general graph computations.

5.1. Simulation Environment

The ParMod logic has been simulated by two OmNeT components implementing the Operating and Control Parts. Components can be programmed using an event-driven programming style. The `handlemessage()` routine is called every time a new message is received. Components exchange messages through communication ports (Figure 6).

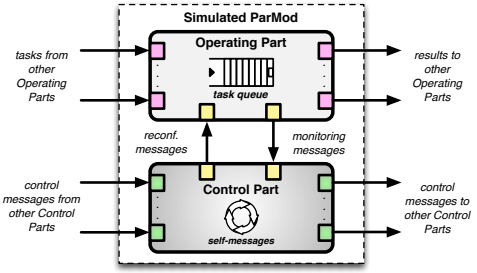


Fig. 6: OmNeT++ simulation of a ParMod.

The Operating Part implements a queue logic. To reproduce a blocking semantics, we have implemented a communication protocol based on the transmission of `send` and `ack` messages. If there is a free position in the queue, the Operating Part transmits an `ack` message to the sender. Otherwise, the received task and the sender identifier are stored in a special data-structure and the `ack` transmission is delayed until a position in the queue is freed. Result messages are transmitted onto output ports selected according to a discrete probability distribution.

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

When a new task is extracted from the queue, the Operating Part simulates the execution of structured parallel computation with a parametric parallelism degree. We have implemented two different working logics:

— a *task-farm* scheme, in which the Operating Part can start the execution of multiple tasks in parallel, with a maximum number given by the current parallelism degree. Each task has a calculation time given by the value of a random variable following a given probability distribution (e.g. exponential, uniform or normal);
— a *data-parallel* scheme, in which the parallelism degree influences both the service time and the computation latency (perfect scalability or non-ideal behaviors can be modeled). In this case the Operating Part works on a single task at a time.

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 simple modifications of the parallelism degree attribute of the Operating Part.

As stated in Section 4.1, reconfigurations can cause performance overhead. To provide a simulation of this fact, we introduce a reconfiguration delay. At every change of parallelism degree, the Operating Part suspends receipt of tasks for an amount of time modeled by a random variable with a given distribution, mean and variance.

5.2. A Distributed Streaming Application

Distributed streaming applications have become increasingly important and spread from small to large-scale systems (e.g. surveillance, traffic control and manufacturing processes) potentially executable on Cloud infrastructures applying complex billing models. To exemplify our methodology, we instantiate our technique to the benchmark graph depicted in Figure 7. In this application an initial stream of data produced by the first module is split into two sub-streams received by modules M_2 and M_3. This structure is representative of a large class of applications:

— video streaming computations, in which the source stream is demultiplexed by a preprocessing phase to generate separated video sequences [Lisani et al. 2005]. Each sub-stream can be processed independently, applying algorithms (e.g. object recognition and motion detection) with different execution times;
— real-time signal processing applications (e.g. speech recognition, radars, audio compression), in which heterogeneous input samples from different sources are retrieved and split into homogeneous sub-streams (e.g. as in [Valente 2010]) which are processed and results combined by a post-processing phase.

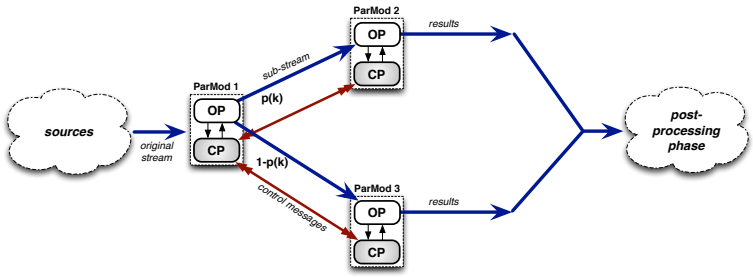


Fig. 7: Benchmark computation graph.

The application adjusts itself to the current splitting probability $p(k)$, by adding or removing nodes in order to optimize throughput and avoid wasting resources. A similar

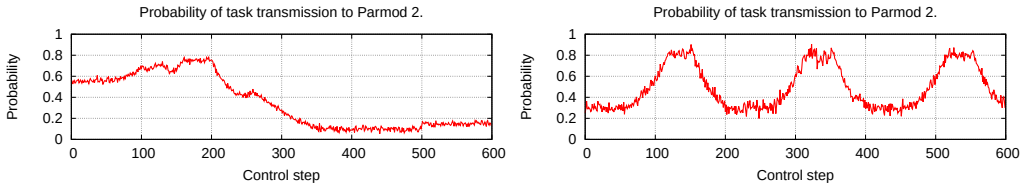
problem has been studied in [Loesing et al. 2012] for the Cloud domain, by changing the number of nodes assigned to application components executing continuous query processing (this technique is called "cloud bursting"). Also in this case we highlight that such work does not address the properties which we are interested in: reconfiguration stability and amplitude.

In the experiment we reproduce the behavior of a cloud-based image processing application able to recover noisy images. The source component produces a stream of noisy images retrieved from cameras and performs a pre-processing phase (parallelized as a task-farm) aimed at finding correlations with a data-set of images. Each task, composed of a noisy image and a proper feature description, is transmitted to either component M_2 or M_3 able to apply data-parallel versions of two different filters (M_2 computation is 20% slower than the one of the M_3, see Table I).

	ParMod 1	**ParMod 2**	**ParMod 3**
T_{calc}	15 sec.	48 sec.	38 sec.
n_i^{max}	20	128	32

Table I: Mean calculation times and maximum number of nodes per ParMod.

5.2.1. Dynamic workload scenario. In order to analyze different scenarios, we take two different probability time-series simulating an execution composed of 600 steps of 120 seconds. Figure 8a (hereinafter referred to as Irregular-Workload) shows a time-series exhibiting an irregular behavior alternating trend phases (e.g. from step 200 to 350) and level-shifts (e.g. from step 100 to 200 and near step 500). Figure 8b (Seasonal-Workload) depicts a periodic behavior, with phases during which M_2 receives tasks more frequently and phases in which most of the tasks are transmitted to M_3. These time-series are meaningful to our evaluation, since seasonal patterns and trends are important non-stationarities characterizing the workload of Cloud and Grid systems [Khan et al. 2012; Quiroz et al. 2009].



(a) Irregular-Workload. (b) Seasonal-Workload.

Fig. 8: Time-series of probability $p(k)$.

To predict future values of $p(k)$, we use the *Holt-Winters* techniques [Gelper et al. 2010] already applied in [Kusic et al. 2011] to control the allocation/deallocation of virtual resources in Cloud environments. For the first time-series we apply a filter able to capture trend component. h-step ahead predictions are calculated as follows:

$$\hat{p}(k+h) = p^s(k) + h\,p^t(k) \tag{11a}$$

$$p^s(k) = a\,p(k) + (1-a)\,\big(p^s(k-1) + p^t(k-1)\big) \tag{11b}$$

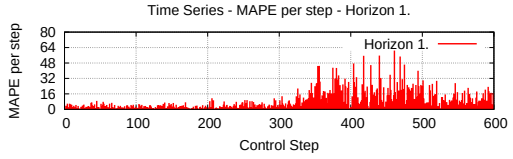
$$p^t(k) = b\,\big(p^s(k) - p^s(k-1)\big) + (1-b)\,p^t(k-1) \tag{11c}$$

where 11b expresses the smoothing component (mean level) and 11c the trend component. Expression 11a makes h-step ahead predictions by extending the time-series into the future w.r.t the trend. Parameters a and b (ranging between 0 and 1) have been estimated using a fitting initial period of observations, by minimizing the sum of the squared one-step ahead forecast errors. For the second time-series we use a variant which exploits an additional exponential (EWMA) filter for estimating an additive seasonal component (further details can be found in [Gelper et al. 2010]).

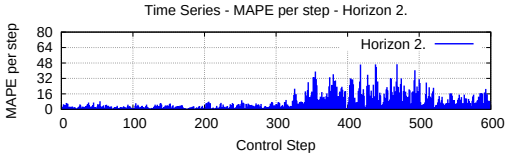
Since we consider horizons of length $h \geq 1$, we are interested in evaluating the percentage error between the real and the predicted trajectories at each control step. The *Mean Absolute Percentage Error* (denoted by `MAPE`) is defined as follows:

$$\mathcal{E}(k) = \frac{100}{h} \cdot \sum_{i=k}^{k+h-1} \left| \frac{p(i) - \hat{p}(i)}{p(i)} \right| \tag{12}$$

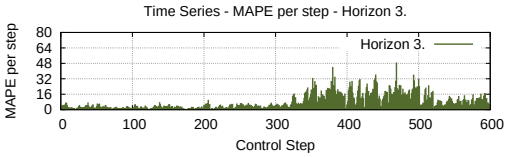
Figure 9 shows the histograms relative to the errors for the two time-series using up to 4-step ahead predictions. As we can note the largest errors occur at the end of the trend phase in the first time-series, and after each decreasing and increasing phase in the seasonal workload. This is due to two main factors. Firstly, the Holt-Winter filter needs some time to react to sudden changes. Secondly, the two time-series expose a certain amount of variance that affects the predictions. We can observe that as we exploit longer horizons, the number of error peaks decreases. This is an effect of the



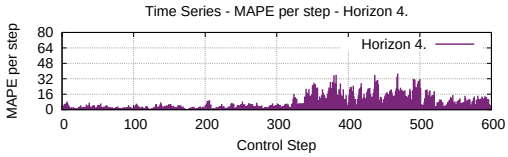
(a) Irregular-Workload h=1.



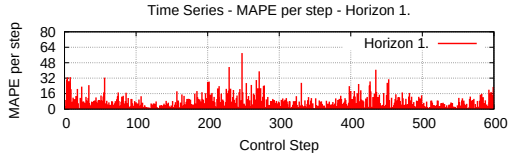
(b) Irregular-Workload h=2.



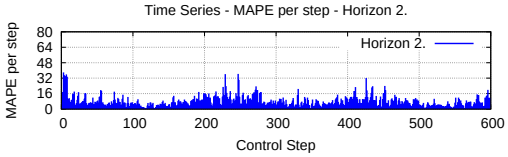
(c) Irregular-Workload h=3.



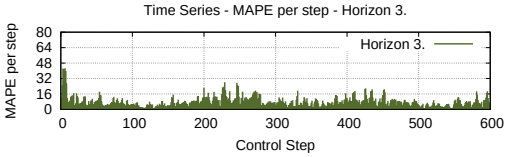
(d) Irregular-Workload h=4.



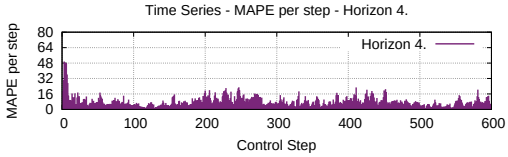
(e) Seasonal-Workload h=1.



(f) Seasonal-Workload h=2.



(g) Seasonal-Workload h=3.



(h) Seasonal-Workload h=4.

Fig. 9: Mean-Absolute Percentage Errors.

averaging, since even if peaks are still in that phase they are smoothed by the average of the errors over the entire horizon. Table II shows that the global MAPE slightly grows as we increase the horizon. This behavior can be graphically observed in Figure 9, where the error lines become denser using longer horizons.

Of course this section does not exhaustively cover the problem of time-series forecasting, which is in general complex. However for our examples the Holt-Winters filters will be sufficient to provide a first validation of our predictive control strategies.

	Horizon 1	**Horizon 2**	**Horizon 3**	**Horizon 4**
Irregular-Workload	7.88%	8.11%	8.26%	8.38%
Seasonal-Workload	7.71%	7.82%	8.05%	8.32%

Table II: Global MAPE over the entire execution.

5.2.2. Comparison between control strategies. In order to show the potential of our approach, we compare the Non-Switching Cost formulation with a rule-based strategy inspired to the work presented in [Aldinucci et al. 2008; Weigold et al. 2012]. To control performance and resource utilization using event-condition-action rules, a solution is to observe the ParMod *utilization factor* calculated as the ratio between its ideal service time and the last measured inter-departure time. The utilization factor gives an idea of the efficiency of resource utilization: the smaller the factor is the more the parallelism degree is over-sized, resulting in a waste of computing resources. Values closed to 1 indicate an optimal resource utilization (we can note that the inter-departure time can not be smaller than the ideal service time).

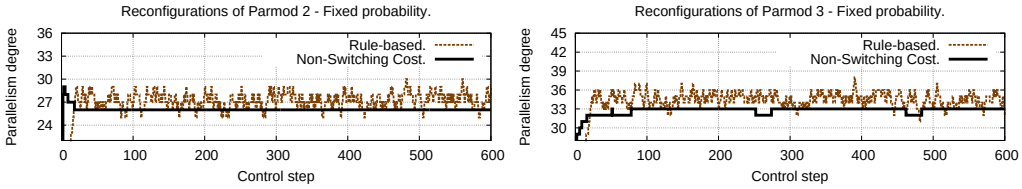
In rule-based frameworks the control logic is usually designed in a decentralized manner by equipping each control entity with its own set of logic rules. Figure 10 shows an example. If the actual utilization factor is lower than a minimum threshold T_{min}, the logic decreases the parallelism degree since resources are currently under-utilized. On the contrary, if utilization factor is greater than a upper threshold T_{max}, ParMod might likely be the actual application bottleneck. In this case the strategy tries to speculatively increase the parallelism degree to improve the application performance.

```
if (utilization_factor > Tmax) then parDegree := min (MAX_DEGREE, parDegree + 1);

if (utilization_factor < Tmin) then parDegree := max (1, parDegree - 1);
```

Fig. 10: Rule-based strategy for parallelism degree adaptation.

When the thresholds are too close, ParMods tend to produce oscillating reconfigurations. Contrarily, if they are too much spaced the strategy performs fewer reconfigurations at the expense of the performance goal (if T_{max} is too high) or the resource consumption (if T_{min} is too low). To show that behavior, we have executed a simulation with a fixed probability $p = 0.5$. Figure 11 depicts the reconfigurations of M_2 and M_3 using the rule-based strategy and our Non-Switching Cost formulation with $\alpha = 100$ and $\beta = 0.5$ (performance cost is dominant).



(a) ParMod 2. (b) ParMod 3.

Fig. 11: Rule-based (T_{max}=0.95 - T_{min}=0.90) and Non-Switching Cost strategy.

The Non-Switching Cost strategy provides more stable reconfigurations: parallel modules stabilize to their optimal parallelism degrees. The rule-based strategy performs fluctuating reconfigurations near the optimal values. This behavior can also be observed by considering a more dynamic scenario as the one of the Irregular-Workload depicted in Figure 8a. The series of parallelism degrees (Figure 12) exhibits a specular behavior between M_2 and M_3: phases where the second ParMod acquires computing resources correspond to time periods in which M_3 releases resources (and vice-versa). From step 100 to 200 the source ParMod moves its parallelism degree from the maximum one (20 nodes). This is because M_2 becomes the application bottleneck (it uses its maximum number of nodes) and the source reduces its parallelism degree to avoid wasting resources. We can note that this phase is delayed in the rule-based case. This is due to the reactive behavior of policy rules: no disturbance prediction is performed and reconfigurations are issued only in response to the measured utilization factor.

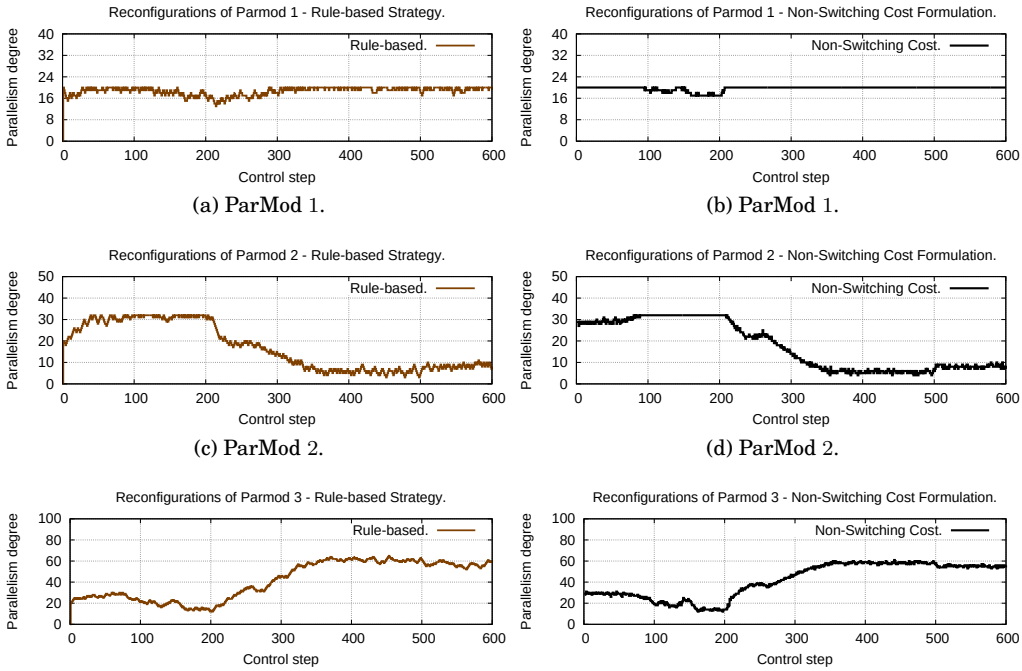
The rule-based strategy performs more reconfigurations. This is evident in the last part of the execution (from step 400 to 600) where the reconfigurations of M_2 and M_3 are much more variable compared with the Non-Switching Cost case.

Table III reports the performance results, the number of reconfigurations and the average efficiency (the mean of utilization factors over the entire execution). We consider a variable reconfiguration overhead implemented as a waiting time after each parallelism degree variation. It is modeled as a normal random variable with mean 25 seconds (almost 10% of the control step length). A similar delay has been considered in [Wang et al. 2008] to model the reconfiguration delay in Cloud environments (time-to-deploy on a virtual machine using a different number of computing entities).

	Reconf.	**Completed Tasks**	**Efficiency**
Rule-based (0.95 - 0.85)	734	165,308	0.893
Rule-based (0.95 - 0.88)	1,043	163,642	0.911
Rule-based (0.95 - 0.90)	1,293	156,584	0.934
Rule-based (0.95 - 0.92)	1,486	152,261	0.959
Non-Switching Cost	745	167,939	0.985

Table III: Reconfigurations, completed tasks and efficiency (Irregular-Workload).

We evaluate different configurations of the rule-based strategy. By using smaller values of T_{min} we make the resource release less aggressive resulting in a smaller number of reconfigurations at the expense of a lower efficiency. With a higher T_{min} more reconfigurations are performed and the efficiency improves. T_{max} has been fixed to 0.95, which is the best value to achieve highest number of completed tasks. *The Non-Switching Cost strategy performs fewer reconfigurations in general, reaching a near optimal efficiency*. The efficiency is 2.6% greater than the best rule-based strategy, with



(a) ParMod 1. (b) ParMod 1.

(c) ParMod 2. (d) ParMod 2.

(e) ParMod 3. (f) ParMod 3.

Fig. 12: Reconfigurations with Irregular-Workload.

a reduction in reconfigurations of 49%. The configuration $T_{min} = 0.95$ and $T_{max} = 0.85$ performs a similar number of reconfigurations but losing 9% in efficiency.

Due to reconfiguration overhead, more reconfigurations do not always correspond to a higher number of completed tasks. By increasing T_{min} we are able to improve the resource utilization but with many more reconfigurations (passing from 0.85 to 0.92 the number for reconfigurations doubles). The Non-Switching Cost strategy directly adapts the parallelism degrees to the optimal values using a steady-state performance model instead of unitary reconfigurations as in the rule-based approach.

A similar behavior has been obtained using the Seasonal-Workload. Table IV outlines the main results that qualitatively follows what we have seen before.

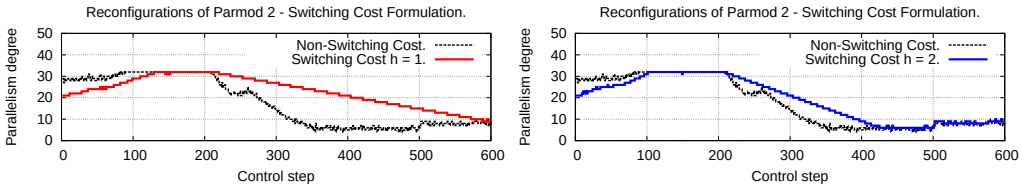
	Reconf.	**Completed Tasks**	**Efficiency**
Rule-based (0.95 - 0.85)	1,032	143,612	0.904
Rule-based (0.95 - 0.88)	1,244	140,046	0.923
Rule-based (0.95 - 0.90)	1,411	133,162	0.934
Rule-based (0.95 - 0.92)	1,558	116,677	0.948
Non-Switching Cost	932	158,650	0.960

Table IV: Reconfigurations, completed tasks and efficiency (Seasonal-Workload).

In conclusion our strategy reaches near optimal performance and efficiency by performing fewer reconfigurations than the rule-based approach. We note that Figure 10 shows only a reasonable example of strategy based on policy rules, and optimizations

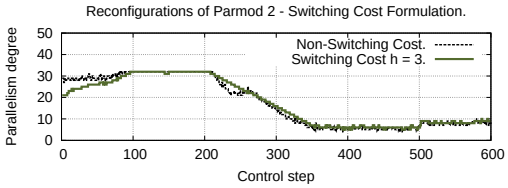
and heuristics could have been introduced to improve it. Nevertheless, it has been sufficient to provide a meaningful comparison between the two approaches.

5.2.3. Using the switching cost to improve the reconfiguration stability. An important topic is the definition of strategies able to improve the stability of control decisions with a negligible impact on efficiency and performance. In this section we discuss the effects on stability of the Switching Cost formulation presented in Definition 4.3. Figure 13 and 14 show the sequence of parallelism degrees using the Irregular-Workload pattern.

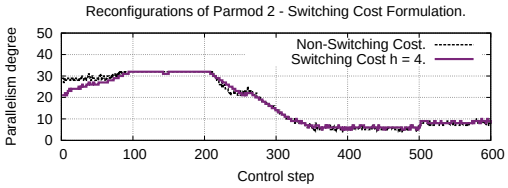


(a) Horizon $h = 1$.

(b) Horizon $h = 2$.



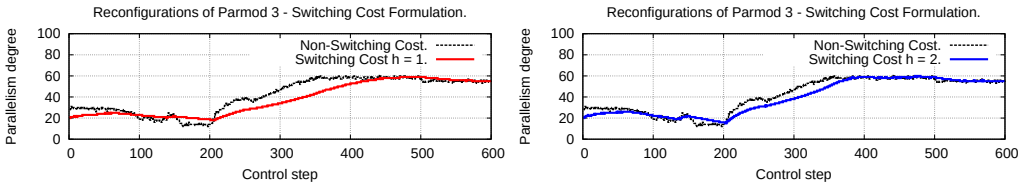
(c) Horizon $h = 3$.



(d) Horizon $h = 4$.

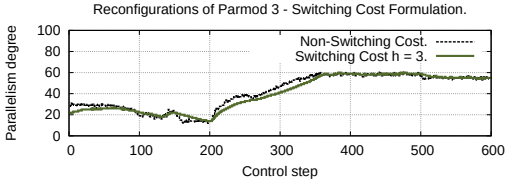
Fig. 13: Switching Cost Formulation: reconfigurations of M_2 ($\alpha = 10$, $\beta = 0.5$, $\gamma = 4.2$).

We consider four horizon lengths from 1 to 4 control steps. Reconfigurations with the Switching Cost are smoother compared to the Non-Switching Cost case. In other words, *the Switching Cost acts as a disincentive to parallelism degree variations*. During phases in which the workload is lighter, it slows down the release of computing

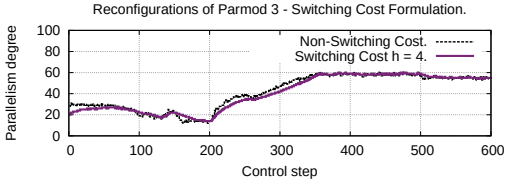


(a) Horizon $h = 1$.

(b) Horizon $h = 2$.



(c) Horizon $h = 3$.



(d) Horizon $h = 4$.

Fig. 14: Switching Cost Formulation: reconfigurations of M_3 ($\alpha = 10$, $\beta = 0.5$, $\gamma = 4.2$).

resources, while in phases of heavy workload it slows down the allocation of new resources. As we consider longer horizons, the sequence of parallelism degrees is nearer to the Non-Switching Cost case (reconfigurations increases using longer horizons).

The reason of this behavior is that our multiple-step ahead forecasts are able to capture future trends in the observed time-series. To understand better the effect of trend predictions let us consider an example. Figure 15 shows the control trajectories of ParMod M_2 when an increasing trend is detected. Since the probability is expected to increase, the parallelism degree of M_2 will assume greater values in the future.

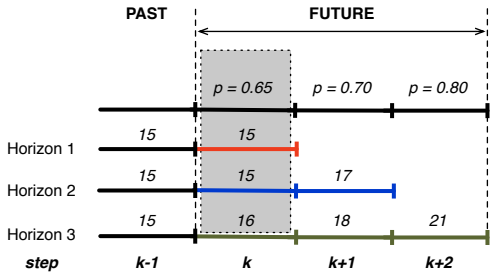


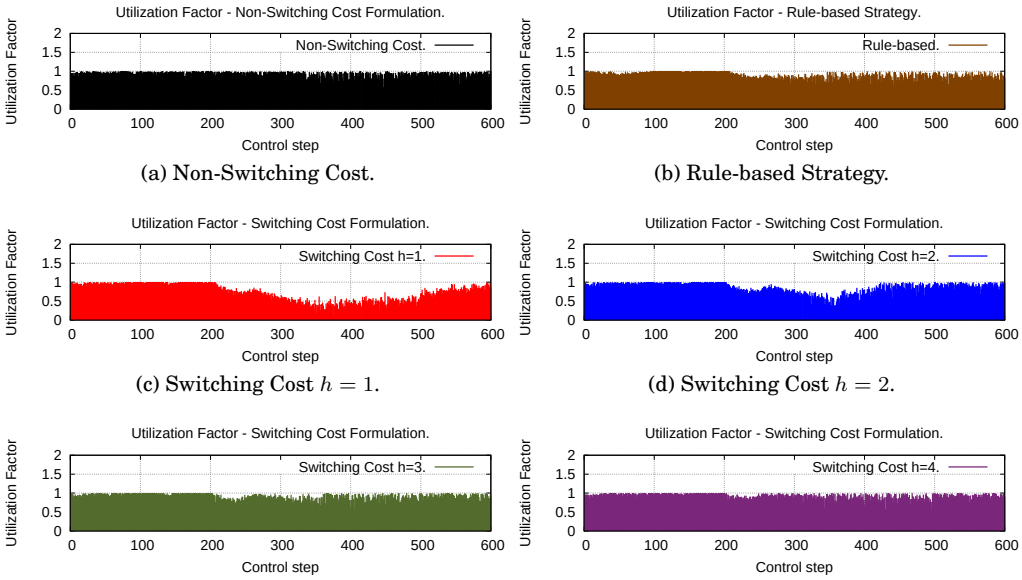
Fig. 15: Example of control trajectories of M_2 using different horizon lengths.

Up to a horizon of two steps the parallelism degree remains stable even if the probability is expected to have a higher value in the future (from 0.65 to 0.80 in three steps). Using a three step horizon, the parallelism degree changes passing from 15 to 16. This happens because the Control Part is able to recognize that in three steps the required resources will augment considerably. Thanks to the Switching Cost, the resource acquisition is distributed among the steps of the horizon and reconfigurations are taken in advance. The opposite behavior is observed during decreasing trends.

The horizon length has important consequences in terms of efficiency of resource utilization. Figure 16 shows the utilization factor of ParMod M_2 over the execution. This ParMod is the most interesting one since it performs a severe release of resources from step 220 to step 350 (Irregular-Workload). As we have seen in Section 5.2.2, the Non-Switching Cost strategy optimizes the efficiency reaching nearer optimal results compared with the the rule-based approach. Figures 16c, 16d, 16e and 16f show the efficiency using the Switching Cost formulation. Using a one-step horizon the release of resource is extremely slow, producing a severe efficiency degradation after control step 200. As we consider longer horizons, controllers have a better degree of foresight and can more precisely evaluate if the release of a certain set of resources is effectively useful (e.g. we enter into a trend phase). This fact leads to an evident benefit: *using sufficiently long horizons the utilization factor envelope tends to the optimal one*.

The utilization factor is at most close to 1. An ideally optimal efficiency is not possible due to: (i) disturbance forecasting errors, that produce non-exact performance predictions; (ii) we apply an integer rounding of continuous parallelism degrees. Nevertheless *the results show a minor efficiency loss in favor of the approach feasibility* (1.5% and 3.6% with the Non-Switching Cost and Switching Cost h=4 strategy).

Another metric is related to the reconfiguration amplitude. There are several cases in which keeping the reconfiguration amplitude as small as possible could be important. A notable example is on Cloud environments, in which finding a large amount of additional resources can be a time-consuming and costly operation [Warneke and Kao 2011; Meiländer et al. 2012]. This concept is stated by the following definition:



(a) Non-Switching Cost. (b) Rule-based Strategy.

(c) Switching Cost $h = 1$. (d) Switching Cost $h = 2$.

(e) Switching Cost $h = 3$. (f) Switching Cost $h = 4$.

Fig. 16: Utilization factor of ParMod M_2.

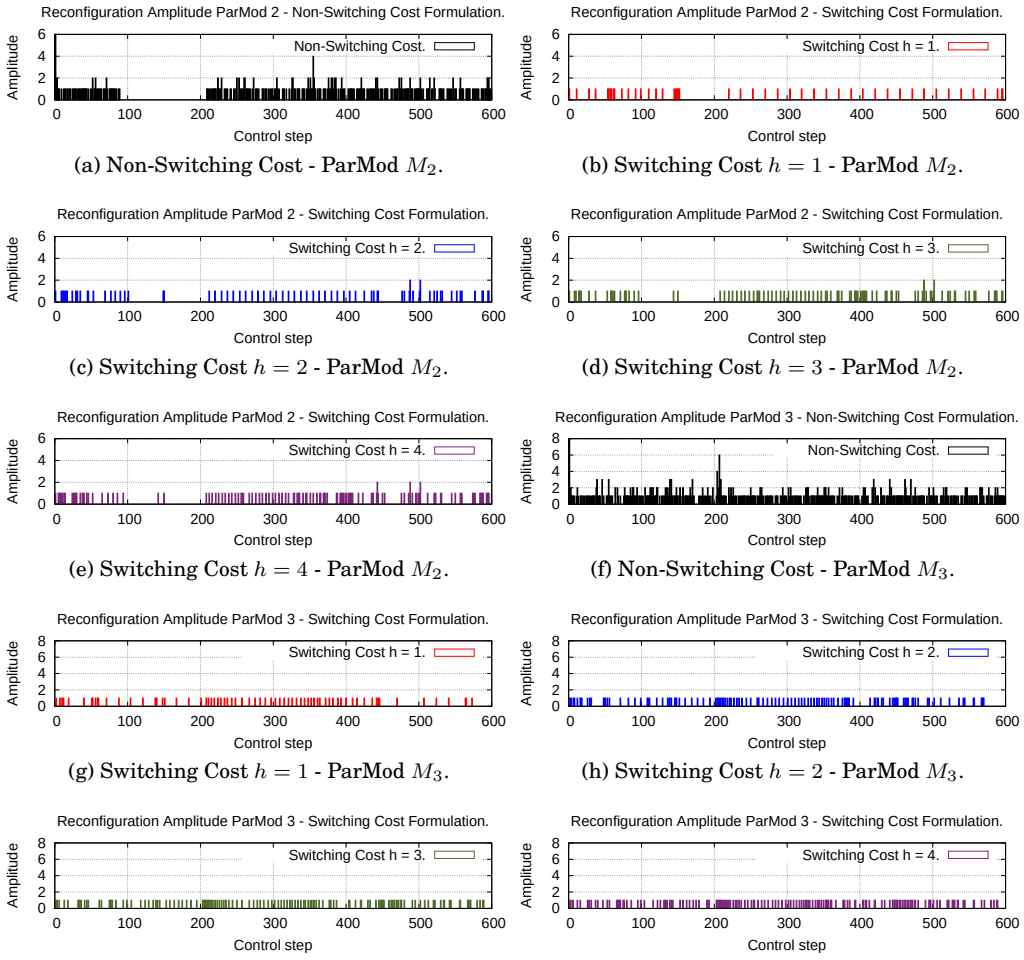
Definition 5.1 (*MRA*). The *Maximum Reconfiguration Amplitude* (shortly `MRA`) is the largest parallelism degree variation performed by ParMods during the execution.

Figure 17 shows the histograms of the amplitude for the second and the third ParMod in the Irregular-Workload case. The Non-Switching Cost strategy performs large reconfigurations reaching peaks of amplitude 4 and 6 for the second and the third module. The amplitude can be reduced using the Switching Cost. With a one step horizon all reconfigurations become unitary variations of parallelism degree. As we use longer horizons the `MRA` slightly increases, reaching a `MRA` of 2 for the second ParMod.

Table V shows the simulation results. Using the Switching Cost and a prediction horizon of one step, ParMods lose 3.50% of completed tasks and save 80% of reconfigurations. Using a three-step horizon we reach a better number of tasks but keeping the reconfiguration number significantly small. With a horizon $h = 4$ we achieve the best performance and good efficiency by decreasing the reconfigurations of 55% compared with the Non-Switching Cost strategy. In conclusion, *longer prediction horizons (until predictions are sufficiently accurate) mitigate the smoothing effect of Switching Cost by achieving better performance and still saving a great number of reconfigurations*.

	Reconf.	**Com. Tasks**	**Efficiency**	**MRA**	**MSI**
Rule-based (0.95 - 0.85)	734	165,308	0.893	1	1.42
Non-Switch. Cost	745	167,939	0.985	6	2.69
Switch. Cost $h = 1$	149	162,038	0.894	1	10.66
Switch. Cost $h = 2$	251	167,178	0.939	2	6.62
Switch. Cost $h = 3$	297	169,599	0.959	2	5.47
Switch. Cost $h = 4$	335	170,827	0.964	2	4.4

Table V: Simulation results using the Irregular-Workload time-series.



(a) Non-Switching Cost - ParMod M_2.
(b) Switching Cost $h = 1$ - ParMod M_2.
(c) Switching Cost $h = 2$ - ParMod M_2.
(d) Switching Cost $h = 3$ - ParMod M_2.
(e) Switching Cost $h = 4$ - ParMod M_2.
(f) Non-Switching Cost - ParMod M_3.
(g) Switching Cost $h = 1$ - ParMod M_3.
(h) Switching Cost $h = 2$ - ParMod M_3.
(i) Switching Cost $h = 3$ - ParMod M_3.
(j) Switching Cost $h = 4$ - ParMod M_3.

Fig. 17: Reconfiguration amplitude of ParMods M_2 and M_3.

To quantitatively compare different strategies in terms of reconfiguration stability, we introduce the following metric:

Definition 5.2 (*MSI*). The *Mean Stability Index* (shortly `MSI`) is the average number of control steps for which ParMod configuration is not modified.

Table V shows the global `MSI` (mean of the `MSI`s of each module). The rule-based strategy (with the thresholds achieving the best performance) has the lowest `MSI`: in the average case ParMods apply a reconfiguration less than every two control steps. Using the Switching Cost the `MSI` improves significantly. As we have seen before, the `MSI` is slightly worse using longer horizons, because reconfigurations are more frequent.

The results using the Seasonal-Workload are briefly summarized in Table VI. Qualitatively we can observe results similar as the previous case. Using long horizons we reduce the reconfiguration numbers and achieve better efficiency w.r.t the best rule-based configuration. Differently from the Irregular-Workload scenario, the Switching

Cost strategy with $h = 3$ completes a slightly smaller number of tasks compared with the Non-Switching Cost formulation, but still saving 63% of reconfigurations.

	Reconf.	Com. Tasks	Efficiency	MRA	MSI
Rule-based $(0.95$ - $0.85)$	1,032	143,612	0.904	1	1.66
Non-Switch. Cost	932	158,650	0.960	10	3.08
Switch. Cost $h = 1$	147	154,524	0.874	1	12.30
Switch. Cost $h = 2$	264	157,433	0.894	1	6.90
Switch. Cost $h = 3$	339	158,196	0.907	1	5.17
Switch. Cost $h = 4$	407	157,789	0.914	2	4.38

Table VI: Simulation results using the Seasonal-Workload time-series.

5.2.4. Feasibility of the approach. As stated in the literature [Nedic and Ozdaglar 2009], the subgradient method can be rather slow, i.e. a high number of iterations could be needed to reach a sufficiently accurate approximation of the social optimum. Two important aspects justify the exploitation of this method in our approach:

— each Control Part applies an integer rounding of the final real-valued parallelism degree. Therefore, a high level of precision is not actually required;
— instead of starting the cooperative algorithm from a fixed initial point, we use a ***warm-start*** approach: *the starting point at each step is equal to the social optimum calculated at the previous step*.

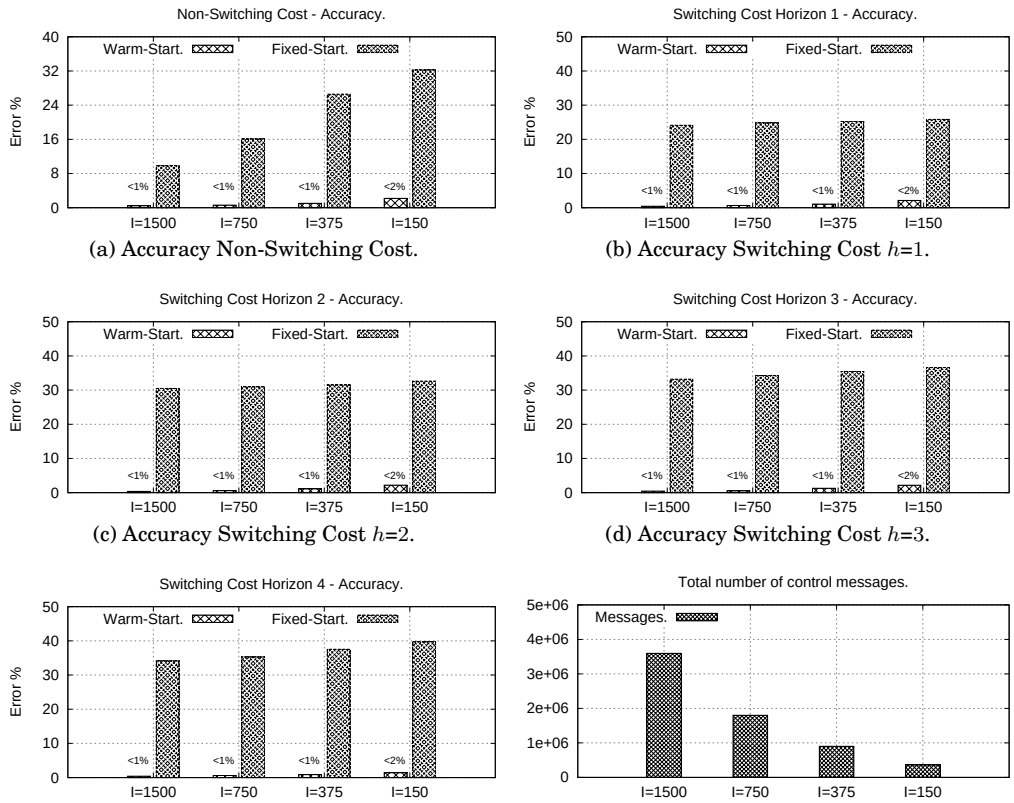
The warm-start technique makes it possible to greatly reduce the number of iterations, since it is highly likely that between consecutive control steps social optima are close. To demonstrate this, we have performed simulations using a different number of iterations $\mathcal{I}$. A smaller number of iterations needs to be balanced by a suitable selection of a bigger step-size a, in order to increase the convergence speed at the expense of lower precision. Figure 18 shows the relative error compared with the optimal reconfiguration sequences calculated using a mathematics software tool[4]. We consider the warm-start approach and the sequences obtained using a fixed initial point.

Using a fixed starting point the accuracy rapidly decreases as it becomes far from the next social optimum. In this case few iterations are not sufficient to reach a good approximation. With the warm start we achieve an important result: *the accuracy remains good (lower than 2%) also using few iterations*. Qualitatively we can appreciate the same behavior for the Switching Cost formulation (the errors increase more slowly since the number of reconfigurations is smaller than the Non-Switching Cost case).

Furthermore, Figure 18f outlines the total number of control messages exchanged during the entire execution. Since this number is proportional to $\mathcal{I}$, we can observe a fast decrease reaching $360,000$ messages using 150 iterations, equal to 600 messages per step.

In conclusion the feasibility is enacted by: (i) reducing the number of iterations with a negligible optimality loss using the warm-start approach; (ii) using partially interconnected controllers (keeping the number of neighbors of each Control Part small). In this way it is possible to greatly reduce the number of exchanged messages which has a direct relationship with the real applicability of the method in real distributed environments.

[4]Mathematica, see more details at http://http://www.wolfram.com/mathematica



(a) Accuracy Non-Switching Cost.
(b) Accuracy Switching Cost h=1.
(c) Accuracy Switching Cost h=2.
(d) Accuracy Switching Cost h=3.
(e) Accuracy Switching Cost h=4.
(f) Control messages.

Fig. 18: Feasibility of the Distributed Subgradient Method.

6. CONCLUSIONS AND FUTURE WORK

The aim of this paper is to apply an optimal control methodology able to achieve trade-offs between reconfiguration stability and amplitude, and the optimality of the decision process. We used the Model-based Predictive Control technique, in which multiple-step ahead predictions are exploited to calculate optimal long-term reconfiguration plans. In Section 4 we proposed two different formulations of the MPC strategy. In the first one, the adaptation is driven by the performance and resource utilization of application components. In the second formulation we introduced a term that discourages reconfigurations of large amplitude. Cooperation between controllers was enabled using the distributed subgradient method. Finally, we tested the effectiveness of the approach by simulations, and we compared our approach with rule-based strategies already applied for similar problems in the literature. According to our results, the Switching Cost formulation and a careful selection of the horizon length makes it possible to:

— achieve the desired control optimality, in terms of performance (accounting reconfiguration overhead) and efficiency (avoiding oversized/undersized parallelism degrees);
— regulate the reconfiguration amplitude (nodes involved in a reconfiguration);
— improve the reconfiguration stability, in terms of number of reconfigurations and the average time between subsequent modifications of the same component.

In contrast to other research work [Abdelwahed et al. 2009], the computational cost of our approach does not depend on the horizon length, and feasibility has been enacted using a proper configuration of the distributed subgradient method and by supporting partial interconnections between controllers.

Our research leads to potential extensions in the future. Adaptive Control is one of the possible directions, by using steps of variable lengths or alternating different formulations of the MPC strategy during the execution. The effects of multiple-step ahead predictions using non-linear filters can be investigated in order to study the behavior of our strategies under more complex situations featuring the strong variability of multiple disturbances. Kalman filters and Neural Networks are notable candidates. This work can be improved by extending the concept of reconfiguration. As an example it could be interesting to consider the switching between alternative parallel versions of the same module, based on different parallelism paradigms (e.g. task-farm and data parallelism are notable examples) characterized by a different speed-up and service time using the same parallelism degree. This extends the space of admissible solutions and can be modeled using convex functions that still admit to be solved cooperatively using the method discussed in this paper. Furthermore, choosing the right parallel version and parallelism degree is critical for other non-functional properties of parallel computations, notably power consumption and memory utilization. Therefore, future extensions of this work can take into account these parameters in addition to the performance metrics studied in this paper.

At present our approach still needs to be validated in real-world applications. How to concretely integrate our methodology in Cloud/Grid environments and how to provide programming constructs to express predictive control strategies are critical and open issues that deserve further investigations in the future.

Part of this work has been recently accepted to presentation in two international conferences [Mencagli et al. 2013a; 2013b].

REFERENCES

ABDELWAHED, S., BAI, J., SU, R., AND KANDASAMY, N. 2009. On the application of predictive control techniques for adaptive performance management of computing systems. *Network and Service Management, IEEE Transactions on 6,* 4, 212–225.

ABDELWAHED, S., KANDASAMY, N., AND NEEMA, S. 2004. Online control for self-management in computing systems. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*. 368 – 375.

ALDINUCCI, M., CAMPA, S., DANELUTTO, M., AND VANNESCHI, M. 2008. Behavioural skeletons in gcm: Autonomic management of grid components. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008.* 54–63.

ALDINUCCI, M., DANELUTTO, M., AND KILPATRICK, P. 2009. Towards hierarchical management of autonomic components: A case study. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*. 3–10.

ALDINUCCI, M., DANELUTTO, M., AND VANNESCHI, M. 2006. Autonomic qos in assist grid-aware components. In *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE Computer Society, Washington, DC, USA, 221–230.

ARSHAD, N., HEIMBIGNER, D., AND WOLF, A. L. 2007. Deployment and dynamic reconfiguration planning for distributed software systems. *Software Quality Control 15,* 3, 265–281.

BALSAMO, S. 2011. Network performance engineering. Springer-Verlag, Berlin, Heidelberg, Chapter Queueing networks with blocking: analysis, solution algorithms and properties, 233–257.

BERTOLLI, C., MENCAGLI, G., AND VANNESCHI, M. 2011. Consistent reconfiguration protocols for adaptive high-performance applications. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. 2121 –2126.

COLE, M. 2004. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Comput. 30,* 3, 389–406.

COPPOLA, M., DANELUTTO, M., TONELLOTTO, N., VANNESCHI, M., AND ZOCCOLO, C. 2007. Execution support of high performance heterogeneous component-based applications on the grid. In *Euro-Par'06: Proceedings of the CoreGRID 2006, UNICORE Summit 2006, Petascale Computational Biology and Bioinformatics conference on Parallel processing*. Springer-Verlag, Berlin, Heidelberg, 171–185.

COSTA, R., BRASILEIRO, F., LEMOS, G., AND SOUSA, D. 2013. Analyzing the impact of elasticity on the profit of cloud computing providers. *Future Generation Computer Systems* 0.

GARCIA, C. E., PRETT, D. M., AND MORARI, M. 1989. Model predictive control: theory and practice a survey. *Automatica 25*, 335–348.

GELPER, S., FRIED, R., AND CROUX, C. 2010. Robust forecasting with exponential and holtwinters smoothing. *Journal of Forecasting 29,* 3, 285–300.

GHANBARI, H., SIMMONS, B., LITOIU, M., AND ISZLAI, G. 2011. Exploring alternative approaches to implement an elasticity policy. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 716–723.

GOMES, A. T. A., BATISTA, T. V., JOOLIA, A., AND COULSON, G. 2007. Architecting dynamic reconfiguration in dependable systems. 237–261.

HELLERSTEIN, J. L., DIAO, Y., PAREKH, S., AND TILBURY, D. M. 2004. *Feedback Control of Computing Systems*. John Wiley & Sons.

HORVATH, T., ABDELZAHER, T., SKADRON, K., AND LIU, X. 2007. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput. 56,* 4, 444–458.

KANDASAMY, N., ABDELWAHED, S., AND KHANDEKAR, M. 2006. A hierarchical optimization framework for autonomic performance management of distributed computing systems. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*. 9.

KEPHART, J. AND WALSH, W. 2004. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*. 3–12.

KHAN, A., YAN, X., TAO, S., AND ANEROUSIS, N. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*. 1287–1294.

KHARGHARIA, B., HARIRI, S., AND YOUSIF, M. S. 2008. Autonomic power and performance management for computing systems. *Cluster Computing 11,* 2, 167–181.

KUSIC, D. AND KANDASAMY, N. 2007. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. *Cluster Computing 10,* 4, 395–408.

KUSIC, D., KANDASAMY, N., AND JIANG, G. 2011. Combined power and performance management of virtualized computing environments serving session-based workloads. *Network and Service Management, IEEE Transactions on 8,* 3, 245–258.

LEE, H., BOUHCHOUCH, A., DALLERY, Y., AND FREIN, Y. Performance evaluation of open queueing networks with arbitrary configuration and finite buffers. *Annals of Operations Research 79*, 181–206.

LEE, H.-S. AND POLLOCK, S. M. 1991. Approximation analysis of open acyclic exponential queueing networks with blocking. *Oper. Res. 38,* 6, 1123–1134.

LISANI, J.-L., RUDIN, L., MONASSE, P., MOREL, J. M., AND YU, P. 2005. Meaningful automatic video demultiplexing with unknown number of cameras, contrast changes, and motion. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*. 604–608.

LIU, H. AND PARASHAR, M. 2006. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 36,* 3, 341 – 352.

LOBEL, I. AND OZDAGLAR, A. 2011. Distributed subgradient methods for convex optimization over random networks. *Automatic Control, IEEE Transactions on 56,* 6, 1291 –1306.

LOESING, S., HENTSCHEL, M., KRASKA, T., AND KOSSMANN, D. 2012. Stormy: an elastic and highly available streaming service in the cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. EDBT-ICDT '12. ACM, New York, NY, USA, 55–60.

LOUREIRO, E., NIXON, P., AND DOBSON, S. 2012. Decentralized and optimal control of shared resource pools. *ACM Trans. Auton. Adapt. Syst. 7,* 1, 14:1–14:31.

MAGGIO, M., HOFFMANN, H., PAPADOPOULOS, A. V., PANERATI, J., SANTAMBROGIO, M. D., AGARWAL, A., AND LEVA, A. 2012. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans. Auton. Adapt. Syst. 7,* 4, 36:1–36:32.

MEILÄNDER, D., PLOSS, A., GLINKA, F., AND GORLATCH, S. 2012. 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. Springer-Verlag, Berlin, Heidelberg, 149–158.

MENCAGLI, G. 2012. *A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations*. Ph.D Thesis, Department of Computer Science, University of Pisa, Italy.

MENCAGLI, G. AND VANNESCHI, M. 2011. Qos-control of structured parallel computations: A predictive control approach. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*. CLOUDCOM '11. IEEE Computer Society, Washington, DC, USA, 296–303.

MENCAGLI, G. AND VANNESCHI, M. 2013. Analysis of control-theoretic predictive strategies for the adaptation of distributed parallel computations. In *Proceedings of the First ACM Workshop on Optimization Techniques for Resources Management in Clouds*. ORMaCloud '13. ACM, New York, NY, USA, 33–40.

MENCAGLI, G., VANNESCHI, M., AND VESPA, E. 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. Won the outstanding paper award.

MENCAGLI, G., VANNESCHI, M., AND VESPA, E. 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.

NEDIC, A. AND OZDAGLAR, A. 2009. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on 54,* 1, 48 –61.

PARK, S.-M. AND HUMPHREY, M. 2011. Predictable high-performance computing using feedback control and admission control. *Parallel and Distributed Systems, IEEE Transactions on 22,* 3, 396–411.

QUIROZ, A., KIM, H., PARASHAR, M., GNANASAMBANDAM, N., AND SHARMA, N. 2009. Towards autonomic workload provisioning for enterprise grids and clouds. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*. 50–57.

RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. 2008. No "power" struggles: coordinated multi-level power management for the data center. *SIGOPS Oper. Syst. Rev. 42,* 2, 48–59.

RAM, S. S., NEDIC, A., AND VEERAVALLI, V. V. 2009. Distributed subgradient projection algorithm for convex optimization. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. ICASSP '09. IEEE Computer Society, Washington, DC, USA, 3653–3656.

REIFF-MARGANIEC, S. AND TURNER, K. J. 2004. Feature interaction in policies. *Comput. Netw. 45*, 569–584.

TSAI, W. T., SONG, W., CHEN, Y., AND PAUL, R. 2007. Dynamic system reconfiguration via service composition for dependable computing. In *Proceedings of the 12th Monterey conference on Reliable systems on unreliable networked platforms*. Springer-Verlag, Berlin, Heidelberg, 203–224.

VALENTE, F. 2010. Multi-stream speech recognition based on dempstershafer combination rule. *Speech Communication 52,* 3, 213 – 222.

VANNESCHI, M. AND VERALDI, L. 2007. Dynamicity in distributed applications: issues, problems and the assist approach. *Parallel Comput. 33,* 12, 822–845.

WANG, X., DU, Z., CHEN, Y., LI, S., LAN, D., WANG, G., AND CHEN, Y. 2008. An autonomic provisioning framework for outsourcing data center based on virtual appliances. *Cluster Computing 11,* 3, 229–245.

WARNEKE, D. AND KAO, O. 2011. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE Trans. Parallel Distrib. Syst. 22,* 6.

WEIGOLD, T., ALDINUCCI, M., DANELUTTO, M., AND GETOV, V. 2012. Process-driven biometric identification by means of autonomic grid components. *Int. J. Auton. Adapt. Commun. Syst. 5,* 3, 274–291.

YUAN, Q., LIU, Z., PENG, J., WU, X., LI, J., HAN, F., LI, Q., ZHANG, W., FAN, X., AND KONG, S. 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.

ZHANG, R., LU, C., ABDELZAHER, T. F., AND STANKOVIC, J. A. 2002. 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. IEEE Computer Society, Washington, DC, USA, 301–.

Received June 2013; revised June 2013; accepted June 2013