CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2012; **00**:1–25
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Adaptive Model Predictive Control of Autonomic Distributed Parallel Computations with Variable Horizons and Switching Costs

G. Mencagli*†

Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, I-56127 Pisa, Italy

SUMMARY

Autonomic computing is a paradigm for building systems capable of adapting their operation when external changes occur, such as workload variations, load surges and changes in the resource availability. The optimal configuration in terms of the number of computing resources assigned to each component must be automatically adjusted to the new environmental conditions. To accomplish the execution goals with the desired Quality of Service, decision-making strategies should be in charge of selecting the best reconfigurations by taking into account metrics like performance, efficiency (avoiding wasting resources), number and frequency of reconfigurations and their amplitude (performing minimal modifications of the current configuration). This paper presents a decision-making strategy that merges the potential of Model Predictive Control with a cooperative optimization framework. After a description of our approach, we investigate the effect of different switching costs to model the resource allocation problem. We use a control method in which our proactive decision-making strategy (designed to use future prediction horizons) is made adaptive itself by dynamically changing the horizon length on the basis of the prediction errors. Simulations have been used to exemplify our approach and to discuss the effectiveness of the variable-horizon strategy in achieving the best trade-offs between reconfiguration metrics. Copyright © 2012 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Parallel Computing; Autonomic Computing; Model Predictive Control; Distributed Cooperative Optimization.

1. INTRODUCTION

The recent technological advancement in computing and networking technology has resulted in a rapid growth in the size and complexity of modern distributed parallel applications, usually assembled out of a set of interacting (possibly parallel) software components executed over distributed and heterogeneous resources. Examples are streaming applications for the execution of continuous queries [1], distributed surveillance systems [2], financial trading and emergency management systems [3, 4]. For these applications the resource requirement to maintain the desired level of *Quality of Service* (QoS) is often dynamic, not easily predictable, and dependent upon the future workload. For this reason, the optimal configuration in terms of component identification and mapping onto the physical resources must be continuously adapted to the new operating conditions.

The paradigm enabling this behavior is known as *Autonomic Computing* [5]. An autonomic application is able to automatically modify its configuration according to the settings of its computing environment and the properties of its workload, by reducing the management complexity to the users. The key aspect of this paradigm is the synergic interaction between: *i)* proper run-time

†E-mail: mencagli@di.unipi.it
*Correspondence to: Gabriele Mencagli, Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, I-56127 Pisa, Italy.

support mechanisms in charge of modifying the resource allocation to the different parts of the application, and *ii)* a decision-making strategy (also called adaptation strategy in the sequel) that triggers reconfigurations when specific execution conditions are met.

In recent years several studies have focused on the definition of software mechanisms to support the dynamic adaptation with minimum execution overhead. A survey is presented in [6], while a description of the most important performance and consistency issues has been provided in [7]. Although reconfiguration mechanisms are an important part of autonomic systems, decision-making strategies are essential to achieve the system goals [8]. Decision-making strategies should be qualitatively and quantitatively compared using specific metrics of the adaptation process: the performance (throughput, response time), number of QoS violations, number of used resources, frequency of reconfigurations and their complexity in terms of number of involved components, amount of allocated/deallocated resources and the reconfiguration overhead.

The definition of decision-making strategies can exploit a mature set of methodologies used in a variety of disciplines such as control theory, artificial intelligence, machine learning, distributed optimization and decision theory. A brief review of the most recent solutions is presented in Sect. 2. Along this line, this paper follows the research direction of investigating novel adaptation strategies for distributed parallel applications. The rationale of our approach is to combine formal control techniques inspired by control theory with advanced distributed optimization methods. Our work is based on the following characterizing aspects:

- the application of an optimal control method known as *Model Predictive Control* [9] (MPC), based on the knowledge of mathematical models of the target system and the on-line optimization of cost/utility functions over future time horizons with a fixed or variable length;
- the distributed cooperative solution of the optimization problems using an optimization technique known as *Distributed Subgradient Method* [10];
- the formulation of the optimization problems using different *switching cost* functions modeling the cost incurred in applying reconfigurations on the system, and their impact on the quantity and quality of reconfigurations.

This paper is organized as follows. Sect. 2 provides a comparison with the existing literature. Sect. 3 presents a general description of autonomic parallel applications, and outlines the properties and metrics to evaluate and compare adaptation strategies. Sect. 4 introduces our methodology. Sect. 5 presents a validation of our approach in a simulation environment by showing the concrete application of our technique on a video-streaming application. Finally, Sect. 6 gives the conclusion of this work and the future research directions.

2. ANALYSIS OF THE LITERATURE

In this section we present an overview of the literature about autonomic solutions for distributed/parallel environments. Then, we compare the autonomic features provided by our approach and by similar existing works. Finally, we discuss how this paper extends our prior works.

2.1. Overview of autonomic solutions for distributed/parallel systems

A systematic comparison of different methodologies to develop adaptation strategies has been made in [8]. An approach inspired by artificial intelligence methods consists in providing the mapping between execution conditions and corresponding reconfigurations through *heuristics* expressed by policy rules [11, 12]. As an example, *event-condition-action* rules [13] (ECA) indicate the system reaction to the presence of specific conditions whose occurrence is continuously monitored. Rule-based strategies have been applied in [14] for distributed emergency management systems, and in [15, 16] for parallel applications expressed by composition of algorithmic skeletons [17]. Although they are a very flexible solution, policy rules are not easy to be tuned: rules calibration may be hard and it is often difficult to prove the convergence to optimal reconfiguration choices. Furthermore, when different rules make contrasting decisions, *conflict resolution strategies* [18]

DOI: 10.1002/cpe

must be taken into account to avoid to take the system into meta-stable states. In more advanced works policy rules are not static, but they are dynamically modified by using the past knowledge of the system behavior (particularly, the use of *reinforcement learning* has been investigated in [19]).

Another branch of the research focuses on the definition of adaptation strategies using ideas and principles inspired by control theory and automation. Canonical approaches based on proportional, proportional-integrative and proportional-integrative-derivative controllers have been used to control software performance [20, 21]. Standard control techniques have been applied to web servers and enterprise applications [22, 23] with the goal of adapting the performance and power consumption. The control of jobs progress and CPU allocation of data centers has been studied in [24] by using a combination of admission control and standard feedback control laws.

One of the most critical issues of autonomic systems is the ability to determine the most effective reconfigurations to fulfill the system objectives, that is by trading-off the transient cost of reconfigurations and their benefit at steady state. This aspect is critical in environments like Grids and Clouds, where the resource (re)allocation process may incur significant performance and economic costs. This problem has been addressed in [25, 26, 27], in which algorithms for the dynamic resizing of data centers have been discussed by introducing switching costs to model the penalty of toggling a server back-and-forth between active and power saving modes. Adaptive cost prediction for Grid and Cloud computing applications has been studied in [28] by proposing a programming framework for Java based on the Bulk Synchronous Parallel model (BSP).

Recently, advanced control-based techniques have been studied as alternative solutions to improve the effectiveness of the reconfiguration selection. Model Predictive Control [9] (briefly MPC) is a popular approach that has inspired our work. MPC is based on the use of a system model as a prediction model to calculate the optimal reconfiguration trajectory over a limited future time horizon. Only the first element of the optimal trajectory is applied to the system and the procedure is repeated at the beginning of the next sampling interval (this principle is called *receding horizon*). MPC has been one of the most promising and successful advanced control-based strategies [9]. First works to control computing systems using MPC have been described in [29] to adapt the number of physical machines allocated to web servers. On Clouds these concepts have been studied for the dynamic allocation of virtual machines in [30, 31]. However, the potential of MPC-based strategies in the context of distributed/parallel systems, e.g. to control the quality and the quantity of reconfigurations and their impact on the system objectives, is not fully explored and deserves further research efforts.

2.2. *Comparison with previous research*

Autonomic frameworks for distributed/parallel systems can be classified according to four general characteristics: *i)* the goal of the adaptation process, *ii)* the classes of reconfigurations supported, *iii)* the strategy, e.g. policy-based or control-based, *iv)* the organization of the control scheme, i.e. a unique, centralized control entity, or multiple control entities organized in a decentralized, distributed or hierarchical manner. Tab. I shows the comparison of the works presented in Sect. 2.1. Although these works represent only a fraction of the ongoing research (which is quite large), they are a representative summary of the current status of the studies. The goal of this comparison is to fit our work into the literature and highlight the differences with the existing approaches.

The first four works are policy-based autonomic systems. In the most cases the control logic is executed by a centralized entity [11, 12, 19]. For large systems in terms of number of components and reconfiguration choices this solution is infeasible for scalability, reliability and expandability reasons. In [14] each autonomic entity has a local controller executing private policy rules. The approach is *decentralized*, in the sense that controllers do not communicate to achieve a global goal: rules are split into distinct parts assigned to local controllers that work separately. More sophisticated interactions have been studied in [15, 16], in which controllers are organized in a *hierarchy* exploiting the fact that applications are expressed as composition/nesting of well-known parallelism patterns (skeletons like farm, pipe, loop). Our approach proposes a different solution for distributed applications expressed as autonomic parallel components interconnected in generic graph structures. The adaptation strategy of the application is decomposed into a distributed set

DOI: 10.1002/cpe

of controllers organized in a *single layer* (without a hierarchy). Controllers exchange information between neighbors *iteratively* and *cooperatively*: at each iteration controllers obtain information about what the plans of the other controllers are. At the end of the iterations the controllers find reconfiguration choices that lead to the global optimization of the system behavior. To the best of our knowledge this is the first time that such distributed cooperative single-layer control scheme is used for controlling distributed parallel applications.

	Goal of Adaptation	Reconfigurations	Strategy	Scheme
Bahati et al. [11, 12]	[response time, throughput]	[buffer size, num. of threads]	static rules	centralized
Bahati et al. [19]	[response time, throughput]	[MaxClients, TCP keepalive]	dynamic rules	centralized
ACCORD [14]	[CPU load, memory]	[algorithms, data layout]	static rules	decentralized
Behavioural Skeleton [15, 16]	[throughput]	[parallelism degree]	static rules	hierarchical
Controlware [22]	[hit rate, response time]	[cache space, num. of processes]	feedback loop	centralized
Raghavendra et al. [23]	[power consumption]	[clock frequency, power budget, turn on/off nodes]	feedback loop	hierarchical
Park&Humphrey [24]	[job progress time]	[CPU allocation]	feedback loop admission control	centralized
LLC (Data Centers) [29]	[profit optimization with switching costs]	[nodes per service cluster, clock rate of nodes]	fixed-horizon MPC	centralized
LLC (Clouds) [31]	[response time, power consumption]	[fraction of CPU, node on/off, VM on/off]	fixed-horizon MPC	centralized
LLC (Distributed) [32]	[profit optimization]	[clock rate of nodes, node on/off]	fixed-horizon MPC	hierarchical

Table I. Comparison between autonomic frameworks for distributed/parallel environments.

The works in the second group [22, 23, 24] are based on *reactive* feedback controllers that use PI (proportional-integrative) or PID (proportional-integrative-derivative) control laws exploiting linearized system models. As it is known, such canonical control-theoretic solutions have inherent limitations as the need of linearized models with continuous inputs that are limiting factors for controlling distributed/parallel systems with discrete reconfiguration choices and switching behaviors. Moreover, when corrective actions have long dead times (time to complete a reconfiguration) as for resource (re)allocation mechanisms in Grids and Clouds, reconfiguration choices must be planned in anticipation to changes in the operating conditions. Also our work follows a control-theoretic perspective, with the adoption of a non-standard control method (*Model Predictive Control*) which will be used to devise effective resource (re)allocation policies acting proactively to workload variations.

MPC is a very popular controller design method in the process industry whose potential in the control of computing systems has not been explored fully. The existing work closest to our research is the *Limited Look-ahead Control* framework (LLC) published in several papers since 2003 ([29, 31, 32] represent a subset of the most recent, interesting publications). Our work has many distinguishable features with respect to this research. First of all, all the applications of LLC use predictive strategies working with fixed-length prediction horizons. Our work, for the first time in this domain, proposes a *variable-horizon* strategy in which the length of the horizon is adapted according to the current prediction errors. As it will be shown in this paper, this strategy is beneficial to improve the effectiveness of the adaptation process. Furthermore, our work discusses the use of different switching costs. Switching costs have been used also in [29] to model the cost incurred when powering up/down virtual machines. In this paper we study three popular definitions of switching costs, and for the first time we analyze their impact on the following set of metrics: reconfiguration *stability* (number and frequency of reconfigurations), *amplitude* (average "size" of reconfigurations), *efficiency* (use of resources) and *performance* (throughput). As far as we know, this is the first time that these metrics have been analyzed in an integrated manner. Finally, in the

DOI: 10.1002/cpe

existing publications LLC has been mainly used in a centralized fashion [29, 31]. A hierarchical scheme is proposed in [32], in which controllers have been organized in a three-layer hierarchy. Controllers at the different layers use different fixed horizon lengths and control a subset of the system parameters. In contrast, the strategy proposed in this paper organizes controllers at the same layer, without higher-layer controllers that may constitute centralization points or single points of failure. For these reasons our work differentiates itself from the existing literature by proposing an application of Model Predictive Control with novel features in terms of distribution and cooperation.

2.3. *How this paper extends our prior works*

This paper continues our research published in [33, 34, 35, 36, 37, 38]. Our predictive strategy has been presented for the first time in [34] for the centralized control of distributed parallel applications on clusters and clouds. The distributed control strategy and the cooperative formulation have been described in [33, 35, 36]. This paper extends these prior works in the following directions: *i)* we study the impact of different switching costs in the formulation of the control problem, and we analyze their effect on a meaningful set of adaptation metrics; *ii)* we propose a variable-horizon strategy in which the horizon length is adapted step-by-step achieving better trade-offs between reconfiguration metrics with respect to the ones achieved using fixed horizons.

3. AUTONOMIC APPLICATIONS AND DECISION-MAKING STRATEGIES

In this section we describe our vision of autonomic distributed parallel applications. The goal of the discussion is to state the fundamental properties that we want to achieve from the adaptation strategies studied in this paper.

3.1. *Overview of the problem*

Complex distributed applications can be structured as *computation graphs* (flow graphs, workflows as in Fig. 1) whose nodes are components/modules[†] operating on data streams and applying a computation on each stream element. Each module (briefly **ParMod**) is an autonomous computing agent: *i)* it receives stream elements (*tasks*) from preceding modules, *ii)* executes a computation on each received stream element, *iii)* sends results to a destination selected according to a deterministic or a probabilistic criterion. This vision is representative of many real-life stream-based applications like the ones in the domains of video streaming and image processing, scientific applications, continuous queries and emergency management systems.

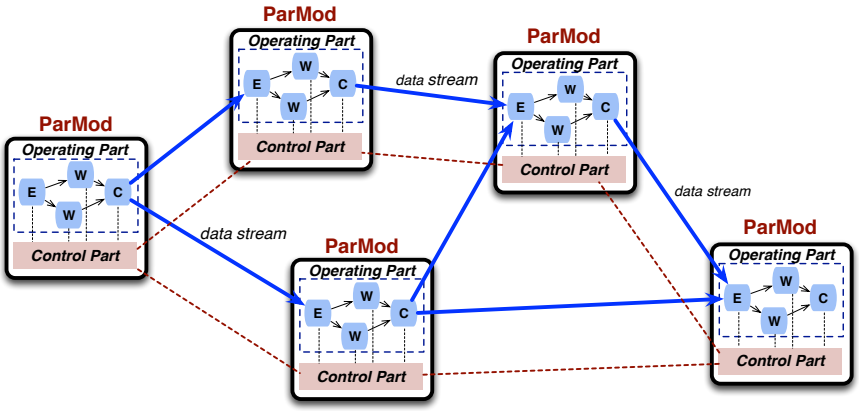


Figure 1. Example of workload graph of parallel modules interconnected by data streams.

[†]In the following we use the term processing module as a synonym of component.

DOI: 10.1002/cpe

We can identify two levels of parallelism. *Inter-component parallelism* consists in the decomposition of an application in distinct components. Components operate on different tasks in parallel. To remove or reduce the presence of bottlenecks, each component can be parallelized in order to sustain the current arrival rate of tasks. *Intra-component parallelism* consists in parallelizing each component by using proper parallelism patterns. Both *task parallel* and *data parallel* skeletons [17] are exploitable inside each component, the former by processing multiple tasks in parallel (e.g. task farming), the latter by partitioning each received task (typically consisting in a large data structure such as a multi-dimensional array) on multiple Workers applying the same computation in parallel on different partitions. This vision of two-level skeleton-based graphs is representative of our past experience in programming models for parallel computations (SkiE, P3L and ASSIST [39, 40, 41]) expressed according to the Structured Parallel Programming [17, 42].

Several component parameters can be modified at run-time in order to adapt to the current execution condition and workload. Reconfigurations are changes in the parallelism degree, the size of input buffers, or the dynamic redistribution of the input data to achieve better load balancing.

Fig. 1 shows a computation graph and the internal structure of the processing modules. Each module is composed of two parts named *Operating Part* and *Control Part* interconnected in a closed-loop fashion. QoS measurements from the Operating Part are periodically sampled by the Control Part that issues reconfiguration inputs to the Operating Part. The Operating Part performs a structured parallel computation executed by different functionalities, e.g. an Emitter (`E`) implementing distribution strategies of the input tasks to the Workers, a set of Workers (`{W}`) realizing the computation on the tasks (the cardinality of this set is called *parallelism degree*), and a Collector (`C`) responsible to collect results and transmit them to the output streams. Based on the parallelism form, Workers can be independent (e.g. *farm* or *map* skeletons) or they interact according to a specific communication pattern (e.g. *stencils*).

In this paper we focus on the general and important problem of adapting the parallelism degrees of the processing modules (i.e. the amount of resources they use) according to the workload variations, with the goal of maintaining acceptable levels of performance and avoiding wasting resources.

3.2. *Properties of adaptation strategies*

Performance and efficiency are two critical properties that drive the reconfiguration process. The allocation of new resources by increasing the parallelism degree is a necessary action to deal with rapid workload peaks. In this case a fast allocation of new resources allows the component to sustain higher arrival rates. On the contrary, when the workload features a lower intensity, idle resources can be released to improve the efficiency.

This problem is exacerbated by the cost of the resource (re)allocation. The first type of cost is a performance overhead. In fact, even in the case of resources immediately available to join the computation, the execution needs to be stopped and reach a consistent state before using the new parallelism degree. The task currently being executed and possibly the internal state of the component must be redistributed among the new set of Workers. To do that, reconfiguration protocols [3, 43] must be implemented to maintain the computation semantics. Furthermore, the reconfiguration overhead can be related to the manage of the physical platform. As an example changes in the parallelism degree imply to recruit and reserve a proper amount of resources with a time delay incurred when reassigning resources, e.g. by powering up a physical node or the delay in migrating connections/data when activating a server.

Other types of costs have a different nature. As an example on data centers the cost of toggling a server back-and-forth includes the energy used [23, 27] during the reconfiguration process. In elastic Cloud environments virtual resources are usually delivered on a pay-per-use basis. Therefore, during a switching between different configurations, it is important to take into account the monetary cost of the newly selected configuration. For instance this cost can depend on the type of provisioned resources [44], or it can be proportional to the amount of allocated/deallocated resources [45].

The role of decision-making strategies is to take into account all the meaningful properties of the resource allocation problem in order to choose the best reconfiguration decisions. We can identify four main properties:

DOI: 10.1002/cpe

Performance: an important aspect of the adaptation strategy is the ability to achieve the best performance as possible with the current execution conditions and workload. Adaptation strategies can be compared in terms of *throughput*, i.e. number of completed tasks per time unit, and/or by taking into account the *response time* to process a single task.

Efficiency: it is a metric telling us how close the effective performance of a component is from its ideal one. Efficiency closes to unity means that the assigned resources are used effectively; low efficiency (near to zero) means that we are wasting resources, i.e. the parallelism degree is oversized with respect to the operational needs.

Reconfiguration number and stability: due to the presence of reconfiguration costs, the goal of the adaptation strategy is to do the strictly necessary number of reconfigurations to achieve the desired QoS. Furthermore, if reconfigurations are applied too frequently, the system can spend more time in reconfiguring itself than processing incoming tasks. To capture this aspect, the *stability* of an adaptation strategy is expressed as the average time between consecutive reconfigurations of the same component, i.e. how frequently reconfigurations are issued by the Control Part (see Fig. 2).

Reconfiguration amplitude: for *amplitude* we intend the number of allocated/deallocated resources (see Fig. 2) during a reconfiguration of an application component. It is reasonable to bind the cost of a reconfiguration to its amplitude. On distributed environments performing a large reconfiguration incurs in searching and locating many resources suitable to join the execution. The delay and the monetary cost to find such resources is realistically proportional to the number of resources needed [46, 47], thus the minimization of the reconfiguration amplitude is often an important factor.

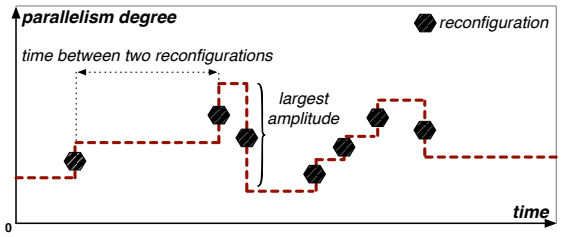


Figure 2. Time between successive reconfigurations and reconfiguration amplitude.

The general goal of adaptation strategies is to reach a good trade-off between these properties. In the next section we will describe our strategies and we will show how reconfiguration goals and costs are modeled in our approach.

4. DISTRIBUTED COOPERATIVE MODEL PREDICTIVE CONTROL

Distributed Model Predictive Control [48, 9] is an advanced control approach applied to a complex system composed of several interconnected sub-systems. Each sub-system is associated with a control problem composed of a local model and an objective function, as schematized in Fig. 3. Usually the passing of time is discretized in fixed time intervals called *control steps*. For the sake of simplicity we assume that all the sub-systems use the same control step definition.

The local model describes the relationship between local *QoS variables* representing quantitative and qualitative metrics of the sub-system execution, and local *control* and *disturbance variables*, which in turn represent the reconfiguration decisions and exogenous uncontrollable events affecting the relationship between reconfigurations and the resulting QoS.

For each sub-system the adaptation strategy consists in finding the best control sequence that optimizes the local objective function. The objective function takes into account the future values

DOI: 10.1002/cpe

of local QoS variables over a *prediction horizon* of $h \geq 1$ future control steps from the current time, where the predictions are computed by using the local model and future forecasted values of disturbances (usually obtained by time-series forecasting tools).

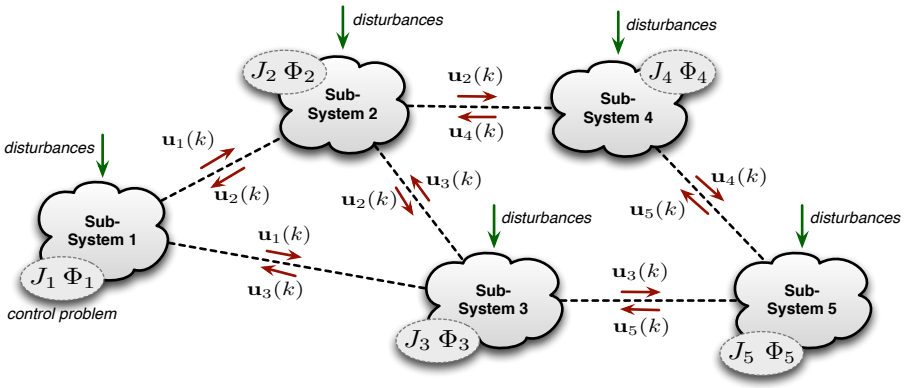


Figure 3. Interconnected sub-systems and control sub-problems. We denote by $\mathbf{u}_i(k)$, J_i and Φ_i the control inputs, the objective function and the model of the i-th sub-system.

At the beginning of each control step MPC controllers (one for each sub-system) apply the following procedure in parallel:

1. future values of local disturbances are estimated over a prediction horizon of h control steps. Time-series analysis and statistical filters (e.g. ARMA, ARIMA, Holt-Winters, Kalman Filters and Artificial Neural Networks) are common tools used to make such predictions;
2. the local optimization problem is solved by finding the optimal trajectory of reconfigurations over the prediction horizon;
3. the controller applies the *receding horizon* principle, i.e. only the first control choice of the optimal trajectory is applied to the system and the procedure is repeated at the beginning of the next control step.

The rationale is intuitive: by applying only the first element of the optimal trajectory, disturbance forecasting errors and model inaccuracies do not accumulate, yielding an implicit feedback mechanism.

A complex issue of distributed control is the presence interrelated sub-problems. The values of the QoS variables of a sub-system may depend on the values of control variables adopted by other sub-systems, as in Fig. 3. The consequence is that controllers need to coordinate themselves in the decision process. This is possible by establishing an interaction pattern between controllers that exchange their reconfiguration proposals until an agreement is reached.

Several interaction methods have been studied in the literature [49] by distinguishing between the way in which controllers communicate (hierarchical, decentralized or distributed schemes) and the goal of the optimization (e.g. controllers working selfishly or cooperatively). Our approach is characterized by two important features:

- a *single-layer scheme* in which all the controllers are at the same level and perform the MPC strategy in a fully distributed manner. This approach makes it possible to develop reliable distributed control strategies without single points of failure and centralization points;
- a *cooperative interaction* between controllers that minimize the global cost of the application rather than pursuing their individual objectives.

4.1. Problem formulation

In this paper we are interested in studying MPC strategies for adapting the parallelism degrees of parallel modules in stream-based distributed computations. According to the classic MPC formulation, Control Parts work in a synchronous fashion by evaluating the adaptation strategy

DOI: 10.1002/cpe

at fixed time intervals (control steps). In contrast, the Operating Part execution proceeds asynchronously with respect to their control logic, i.e. parallel computations respond to the control decisions of the controllers by reconfiguring themselves. We identify for each ParMod $\mathcal{M}_i$ the following set of model variables used by the adaptation strategy:

- a QoS metric of a ParMod is its mean *inter-departure time* between two consecutive results onto the output streams, i.e. the average time between the completion of two successive tasks (inverse of the throughput). We denote by $T_{D_i}(k)$ the inter-departure time of $\mathcal{M}_i$ during control step k;
- the control variable of $\mathcal{M}_i$ is its parallelism degree $n_i(k) \in \mathcal{U}_i$, where $\mathcal{U}_i$ is the closed interval $[1,\, n_i^{max}]$ of integers;
- we model two disturbance variables affecting the performance: the mean *calculation time* per task $T_{calc\text{-}i}(k)$, i.e. the average granularity of tasks during control step k, and the output stream probabilities, i.e. $p_{i,j}(k)$ indicates the transmission probability from $\mathcal{M}_i$ to $\mathcal{M}_j$.

The performance of a module $\mathcal{M}_i$ can be measured *in isolation*, i.e. without considering the interaction with other application modules. This concept of *ideal* performance is captured by the *mean service time* of a module $T_{S_i}(k)$, i.e. the average time interval between the beginning of the executions on two consecutive input tasks. We assume that $T_{S_i}(k) = \mathcal{H}_i(n_i(k))$, where $\mathcal{H}_i$ is a model describing the relationship between service time and parallelism degree. As an example $\mathcal{H}$ can be an analytical model of the performance, as studied in the literature for structured parallel computations [42, 50], or an empirical model based on measured data.

The *effective* performance of a module can be different than the ideal one. The presence of bottlenecks (hot spots) in the graph causes a slow-down in the performance of all the application modules. This behavior is due to two main reasons: *i)* the finiteness of the input queues in front of modules, and *i)* the blocking semantics of communications in distributed environments, i.e. the transmission of a task to a full capacity destination queue is delayed until a free position is available. The effective performance is captured by the inter-departure time. To model the relationship between service times and inter-departure times, we use a model valid for *acyclic graphs with a single source module* (the proof of this result is provided in [51]):

Proposition 4.1 (Inter-departure times)**.** Given a single-source acyclic computation graph $\mathcal{G}$ of N components, the inter-departure time T_{D_i} from a generic component $\mathcal{M}_i$ can be expressed in the following way:

$$T_{D_i}(k) = \max_{j=1}^{N} \Big\{ f_{i,j}\big(T_{S_j}(k)\big) \Big\} \tag{1}$$

Each function $f_{i,j}$ models the inter-departure time of $\mathcal{M}_i$ if $\mathcal{M}_j$ is the bottleneck of the workflow graph. $f_{i,j}$ is defined as a function of the service time of $\mathcal{M}_j$:

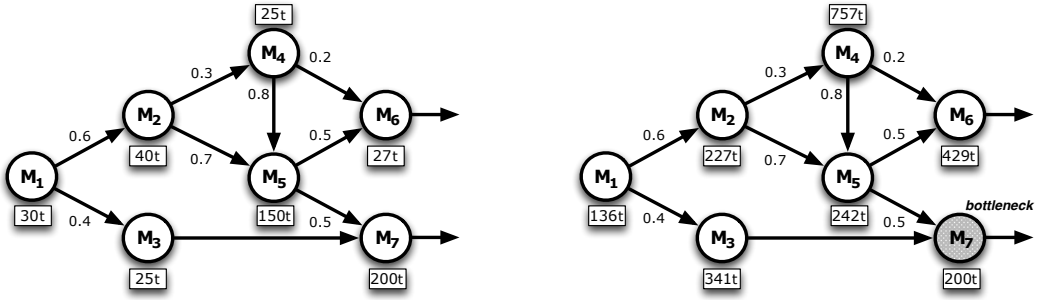
$$f_{i,j}\big(T_{S_j}(k)\big) = T_{S_j}(k) \cdot \frac{\displaystyle\sum_{\forall \pi \in \mathcal{P}(\mathcal{M}_s \to \mathcal{M}_j)} \left(\prod_{\forall (u,v) \in \pi} p_{u,v}(k) \right)}{\displaystyle\sum_{\forall \pi \in \mathcal{P}(\mathcal{M}_s \to \mathcal{M}_i)} \left(\prod_{\forall (u,v) \in \pi} p_{u,v}(k) \right)} \tag{2}$$

We denote by $\mathcal{M}_s$ the source of the graph, $\mathcal{P}(\mathcal{M}_s \to \mathcal{M}_i)$ the set of all the paths starting from $\mathcal{M}_s$ and reaching $\mathcal{M}_i$, and (u, v) an edge of the path π. Since we do not know which component is the bottleneck, the inter-departure time is calculated as the maximum between the functions $f_{i,j}$ for $j = 1, \ldots, N$.

Fig. 4a shows an example of a graph in which modules are labeled with their service times (t is a standardized time unit) and with the routing probability for each arc. Fig. 4b shows the steady-state behavior of the graph by applying Proposition 4.1. In this example $\mathcal{M}_7$ is the bottleneck.

As demonstrated in [51] this method provides sufficiently accurate results independently from the statistical distribution of service times and when the input queues are sufficiently sized. In the

DOI: 10.1002/cpe



(a) Ideal graph.

(b) Steady-state graph.

Figure 4. The ideal graph (labeled with service times) and the corresponding steady-state graph (labeled with inter-departure times).

example of Fig. 4 few tens of buffer positions are sufficient to get an error lower than 2% with exponential, normal and uniform distributions (see [51] chapter 3 page 70 for further details).

Each ParMod is characterized by a local objective function J_i modeled as a cost function taking into account the future performance results, resource consumption and reconfiguration costs:

Definition 4.1 (Local Cost Function)**.** We define the local cost function of a ParMod $\mathcal{M}_i$ as follows:

$$J_i(k) = \sum_{t=k}^{k+h(k)-1} \underbrace{\alpha_i \, T_{D_i}(t)}_{\textit{performance cost}} + \sum_{t=k}^{k+h(k)-1} \underbrace{\beta_i \, n_i(t)}_{\textit{resource cost}} + \sum_{t=k}^{k+h(k)-1} \underbrace{\Delta_i^{sw}(t)}_{\textit{switching cost}} \tag{3}$$

In the definition α_i and β_i are positive weights. The performance cost is proportional to the inter-departure time: the higher the inter-departure time (the slower the ParMod) the higher the cost. Analogously, the resource cost is proportional to the parallelism degree. The local cost function accounts for horizons of several steps in the future, where $h(k)$ is the horizon length used at control step k. For the sake of simplicity controllers adopt the same horizon length at each step.

The switching cost plays a critical role in the selection of the optimal control trajectory. Its intent is to bind control decisions between consecutive control steps by modeling the penalty for switching from a parallelism degree to another one. The goal of the switching cost is to capture different costs related to a reconfiguration, such as the setup time, power consumption and the monetary cost for changing the ParMod configuration. In the following we present three switching costs that have been proposed in the literature in similar works:

- The *Absolute Value Switching Cost* (`abs_sw`) models a cost proportional to the absolute value between parallelism degrees at consecutive control steps, i.e. the number of allocated/deallocated resources during a reconfiguration:

$$\Delta_i^{sw}(k) = \gamma_i \left| n_i(k) - n_i(k-1) \right| \tag{4}$$

  the parameter γ_i is a constant representing the weight of the switching amplitude on the total cost (e.g. by including all the penalties incurred in a reconfiguration). Similar switching costs have been applied to the dynamic control of data centers [25] (by regulating the number of active servers) and in video streaming computations [26].

- The *On-Off Switching Cost* (`on/off_sw`) is a slight modification of the previous definition characterized by a different cost for the acquisition and the release of resources:

$$\Delta_i^{sw}(k) = \gamma_i^{on} \left[n_i(k) - n_i(k-1) \right]^+ + \gamma_i^{off} \left[n_i(k-1) - n_i(k) \right]^+ \tag{5}$$

DOI: 10.1002/cpe

where γ_i^{on} and γ_i^{off} are positive constants and $(\cdot)^+ = \max\{0, \cdot\}$. This switching cost has been used in dynamic resizing algorithms of data centers [27, 52].

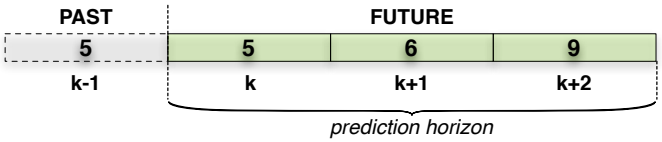
- The *Square Switching Cost* (`pow_sw`) defined as follows:

$$\Delta_i^{sw}(k) = \gamma_i \left[n_i(k) - n_i(k-1) \right]^2 \tag{6}$$

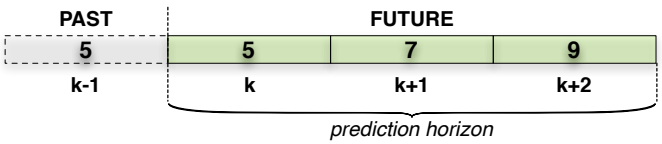
it is defined as the square of parallelism degree variations, where γ_i is a constant weight. Quadratic cost functions are widely used in control theory literature, as they penalize rapid reconfiguration of system states [53].

Switching costs are important in the case of disturbances with high variance or exhibiting trends and seasonal patterns. In those cases it is possible that reconfigurations made without switching costs would have been big in amplitude and prone to some kind of up and down fluctuations. In this scenario switching costs act as a brake by smoothing the reconfiguration trajectories.

Different definitions of switching costs can have different effects on the reconfiguration decisions. As an example Fig. 5 shows two trajectories of parallelism degrees of a ParMod with a horizon of three control steps starting from the current step k. The two trajectories differ in the value used at control step $k+1$ (6 in the first trajectory, 7 in the second one). According to Expr. 3 the local cost function takes into account the sum of the switching costs over the horizon. With the `abs_sw` definition the two trajectories have the same cost, since the sum of the reconfiguration amplitudes is the same for the two control trajectories. In contrast, the `pow_sw` definition returns a different cost, higher for the first trajectory where we have the highest reconfiguration amplitude equal to 3 from step $k+1$ to $k+2$.



(a) First control trajectory.



(b) Second control trajectory.

Figure 5. Examples of control trajectories of the parallelism degree with a three-step horizon.

From this example we evince that the `pow_sw` switching cost penalizes control trajectories with a larger reconfiguration amplitude. This aspect plays a central role in this work and will be studied in Sect. 5 devoted to the experimental evaluation of our approach.

4.2. Enabling cooperation with the Distributed Subgradient Method

Two characterizing aspects of distributed MPC strategies are the way in which controllers are interconnected and the type of agreement that they achieve. In our approach we use the following principle to interconnect controllers in a single-layer scheme:

Assumption 4.1 (Interconnections between Control Parts)**.** Two Control Parts are interconnected (in both the directions) if and only if there exists a data stream between their Operating Parts.

The second point concerns the type of negotiation. At each control step the goal of the controllers is to cooperatively minimize the sum of their local cost functions. We implement this behavior by using the *Distributed Subgradient Method* originally proposed in [10, 54, 55] for multi-agent environments. To apply this method, we use a continuous relaxation of the problem in which

DOI: 10.1002/cpe

parallelism degrees are positive real variables. The optimality loss of this relaxation is acceptable if parallelism degrees assume large values, which is justified by the current tendency of today's distributed platforms featuring hundreds/thousands of processing elements as in data centers, Grids, and more recently in Clouds.

The Distributed Subgradient Method optimizes the sum $J_G(k) = \sum_{i=1}^{N} J_i(k)$ of cost functions in a distributed manner. This method has interesting properties that are particularly suitable to solve our control problem:

- *Local knowledge*: each controller knows only its local cost function and the model to predict the steady-state performance (inter-departure time) of the Operating Part;
- *Convexity*: the method is able to minimize the sum of convex cost functions. According to Expr. 3, our cost functions are convex in the parallelism degrees provided that the service time model $\mathcal{H}_i$ and the switching cost are convex functions (it can be easily verified that `abs_sw`, `on/off_sw` and `pow_sw` definitions preserve the convexity);
- *Convex constraints*: the admissible solutions of the problem must belong to a convex set (closed intervals $\mathcal{U}_i$ are convex sets by definition);
- *Non-differentiability*: the method works without any assumption on the differentiability of cost functions. This is important because our local cost functions are non-differentiable (due to the pointwise maximum in Expr. 1);
- *Partially interconnected controllers*: controllers communicate with a limited set of neighbors.

Each Control Part maintains an estimate of the optimal ***strategy profile matrix*** $\mathcal{S}(k) \in \mathbb{R}^{h(k)\times N}$, where the i-th column is the parallelism degree trajectory of ParMod $\mathcal{M}_i$ over a prediction horizon of $h(k)$ control steps. Neighboring controllers exchange their estimates iteratively within each control step, and compute the next estimate using the following update rule:

$$\mathcal{S}_i^{(q+1)}(k) = P_{\mathcal{U}}\left[\sum_{j=1}^{N}\left(w_{i,j}\,\mathcal{S}_j^{(q)}(k)\right) - a\,g_i\right] \tag{7}$$

where $w_{i,j}$ is a positive weight assigned by controller i to the estimate received by neighbor j (zero if j is not a neighbor), q is the current iteration, $a > 0$ is a real positive *step-size* and g_i is a *subgradient* of J_i at point $\mathcal{S}_i^{(q)}(k)$ which denotes the estimate of the i-th controller at the iteration q. P is the euclidean projection onto the convex set $\mathcal{U}$ of admissible strategy profile matrices.

To prove the convergence to the optimal solution [10, 54, 55] two important conditions must be satisfied: *i)* a rule on the weights that a controller uses when combining its estimate with the ones received from its neighbors; *ii)* a connectivity rule ensuring that the information of each controller influences any other controller after a finite number of information exchanges.

The first aspect is related to how the weights are assigned to the estimates. We denote by $\mathcal{W} \in \mathbb{R}^{N\times N}$ the weight matrix. As proved in [10] $\mathcal{W}$ must be *doubly stochastic*:

Definition 4.2 (Doubly Stochastic Matrix)**.** A doubly stochastic matrix is a matrix $\mathcal{W} = (w_{i,j})$ such that $w_{i,j} \geq 0$ and whose rows and columns sum to 1, i.e.:

$$\sum_i w_{i,j} = \sum_j w_{i,j} = 1$$

The second aspect is related to the presence of a path between each pair of controllers:

Assumption 4.2 (Connectivity)**.** The interconnection between Control Parts must form a *connected undirected graph*, i.e. for each pair of controllers there must exist at least one path connecting them.

If the previous conditions are satisfied, the Distributed Subgradient Method converges to the optimal solution [10, 54, 55], i.e.:

$$\lim_{q\to\infty} \mathcal{S}_i^{(q)}(k) = \mathcal{S}^{opt}(k) \;\text{ for } i = 1,\ldots,N$$

DOI: 10.1002/cpe

where $\mathcal{S}^{opt}(k)$ is the strategy profile matrix optimizing $J_G(k) = \sum J_i(k)$.

The cooperative MPC strategy based on the Distributed Subgradient Method consists in an iterative protocol in which local estimates are exchanged several times within each control step. The sequence of actions performed by a generic controller i is the following:

- the controller acquires past disturbance values from its Operating Part and calculates statistical predictions over the prediction horizon;
- the controller uses an initial estimate of the strategy profile matrix and begins the iterative protocol for a fixed number of iterations $\mathcal{I}$;
- at each iteration the controller receives the local estimates from the neighbors, applies the update rule described in Expr. 7 and transmits the next estimate to the neighbors;
- after the last iteration the controller applies the first element of its optimal reconfiguration trajectory (the i-th column of its final strategy profile matrix) as the new parallelism degree for step k. The element needs to be rounded to an integer value (e.g. by rounding it to the nearest integer).

4.3. Adaptive horizon and feasibility of the approach

In the following we discuss two important aspects of our distributed cooperative MPC strategy: *i)* the selection policy of the horizon length, and *ii)* the feasibility of the approach.

***Variable Horizon MPC*:** we have denoted by $h(k)$ the length of the prediction horizon at control step k. Long horizons increase the foresight of the controller with a potential advantage in improving the desired control properties. However, a too long horizon requires very accurate predictions to determine good reconfiguration trajectories. The prediction accuracy is a critical problem of MPC controllers. In this paper we propose an *adaptive horizon* strategy in which the controllers choose the best time horizon in an adaptive fashion for each step of the execution. Variable horizon strategies have been applied in autonomous navigation systems [56] and for general time-varying linear dynamical systems [57]. As stated in Sect. 2.2, its application in distributed/parallel systems is a novelty of this work. We will take into account two strategies:

- a *fixed-horizon MPC* strategy (`FH-MPC`) in which the length of the prediction horizon is fixed throughout the execution, i.e. $h(k) = h$;
- a *variable-horizon MPC* strategy (`VH-MPC`) in which the length of the prediction horizon can be modified at each control step by evaluating proper shortening and prolongation strategies. The length at step k is a value $h(k)$ such that $1 \leq h(k) \leq h^{max}$ where h^{max} is an upper bound.

In this paper we use a simple prolongation/shortening strategy in which the horizon length is adapted to the current accuracy of disturbance predictions. The idea is to change the horizon length based on how well we are predicting disturbances. This idea is based on the assumption that the immediate future is similar to the immediate past and the last prediction errors are likely near to the errors for the predictions that will be made at time instants close in the future.

We denote by $Err(k,h)$ the mean absolute percentage error (`MAPE`) between the predicted disturbance trajectory and the corresponding real measurements over a horizon of h steps starting from step k:

$$Err(k,h) = \frac{100}{h} \sum_{i=0}^{h-1} \frac{\left|\mathbf{d}\big(k+i\big) - \hat{\mathbf{d}}\big(k+i|k\big)\right|}{\mathbf{d}\big(k+i\big)}$$

where $\mathbf{d}$ is the vector of disturbance variables and $\mathbf{d}(k+i)$ is the real value of the disturbances for step $k+i$. We denote by $\hat{\mathbf{d}}(k+i|k)$ the prediction for step $k+i$ performed using the information available at time instant k.

The idea is to dynamically regulate the horizon length on the basis of the past prediction errors. In particular we are interested in the errors of the last h-step ahead predictions for $h = 1, 2, \ldots, h^{max}$. A graphical description is shown in Fig. 6 where we denote by k the present control step:

This scheme is characterized by two important aspects. Firstly, at each control step controllers perform several h-step ahead predictions (for $h = 1, \ldots, h^{max}$), since their accuracy will be used

DOI: 10.1002/cpe

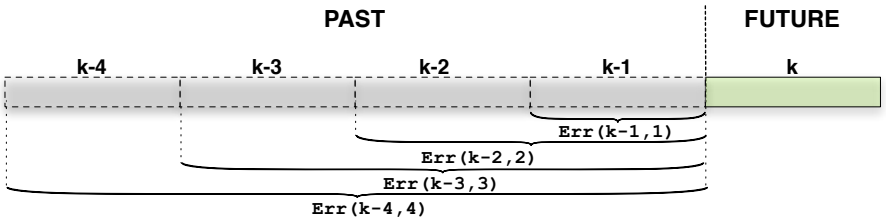


Figure 6. MAPE of the last predicted trajectories up to control step k with $h^{max} = 4$.

in the future to select the best horizon length. Secondly, the same horizon length should be used by *all* the controllers at each step. To do that, the controllers can exchange their local disturbances in order to have a global knowledge of the prediction errors. Alternatively, each controller can evaluate the prediction accuracy on its local disturbances only ($Err^i(k,n)$ denotes the prediction error of disturbances belonging to Control Part i), and the final horizon is selected by reaching an agreement among controllers, e.g. by choosing the minimum horizon length from the ones calculated by the controllers. This second solution is sketched in Alg. 1.

The value at the end of the for loop is the longest horizon such that the last l-step ahead prediction of local disturbances has an error smaller than a threshold θ. The final length of the horizon (row 6) is the minimum between the lengths found by the controllers.

Algorithm 1: Dynamic selection of the horizon length.

```
1 foreach control step k each Control Part i do
2     h_i(k) ← 1;
3     for l = 1 to h^max do
4         if Err^i(k − l, l) ≤ θ then
5             h_i(k) ← l;
6     h_i(k) = min_{j=1}^{N} {h_j(k)}
```

***On the approach feasibility*:** the Distributed Subgradient Method has some interesting features from the feasibility viewpoint. A critical parameter is the number of messages exchanged by controllers before reaching a sufficiently approximate estimation of the optimal solution. The total number of messages n^{msg} per step can be calculated as follows: $n^{msg} = \mathcal{I} \sum_{i=1}^{N} |\mathcal{N}_i|$ where $\mathcal{I}$ is the number of iterations and $\mathcal{N}_i$ is the set of neighbors of Control Part i. As proved in [10], if the step-size a in Expr. 7 is too small and the distance from the starting estimate to the optimal solution is large, the method may take thousands iterations to converge. However, in our application of the Distributed Subgradient Method two aspects can be considered:

- we can choose in a clever way the initial estimate $\mathcal{S}_{[i]}^{(0)}(k)$ in Expr. 7. An effective strategy consists in a *warm start*, i.e. the initial estimate is equal to the final one calculated at the previous control step. The rationale is the following: if the values of disturbances between consecutive steps are similar, the optimal strategy profile matrix determined at the previous step is a starting estimate likely near to the new optimal solution;
- few iterations are in general sufficient since controllers apply an integer rounding at the end of the method. Therefore, a high level of precision is not required actually.

Further details can be found in [33]. The next section exemplifies our approach by discussing the results in terms of the properties introduced in Sect. 3.2.

DOI: 10.1002/cpe

5. SIMULATION RESULTS

In this section we provide an evaluation of our approach through experiments performed in the OmNeT++ discrete event simulator[‡]. OmNeT is an open-source simulation environment offering several features aimed at facilitating computer network modeling. Over the last years the simulator has been extended to reproduce the behavior of Grids and Clouds [58]. The ParMod logic has been simulated by two interacting *simulation objects* implementing the Operating Part and the Control Part. Simulation objects can be programmed using an event-driven programming style and they exchange messages through communication ports.

The Operating Part implements a queue logic. Parallelism inside the Operating Part can be controlled using the parallelism degree attribute. The Operating Part implements different parallel working logics: *i)* the *task-farm* semantics, in which the Operating Part is able to execute more tasks in parallel (up to the value of the parallelism degree); *ii)* the *data-parallel* semantics, in which the execution of each single task is parallelized by reducing its calculation time. The value of the calculation time of a task is modeled by a random variable (with exponential, normal or uniform distribution) with a given mean and variance. Result messages are transmitted onto one of the output ports selected according to a discrete probability distribution.

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. The simulator can reproduce a reconfiguration overhead implemented by a variable waiting time before re-starting the execution with a different parallelism degree. The goal of this feature is to model the delay in adding/removing servers in data centers or the time-to-deploy of virtual machines in Cloud environments [59].

The workflow graph used in the experiments is depicted in Fig. 7. The computation graph describes a synthesized video-streaming application for video surveillance in which an initial stream of video frames (each frame is considered a *task*) is produced by a sequential source component (an interface with a set of properly localized cameras). The sets $\{\mathcal{M}_2\}$ and $\{\mathcal{M}_1, \mathcal{M}_3\}$ represent two sub-parts of the application responsible to apply different denoising filters on the received frames. Based on a pre-processing phase performed by the source, frames are grouped together according to their features (e.g. presence of quiet zones, direction of the image gradient, pixel brightness as described in [60]) and transmitted either to the first or to the second filter.

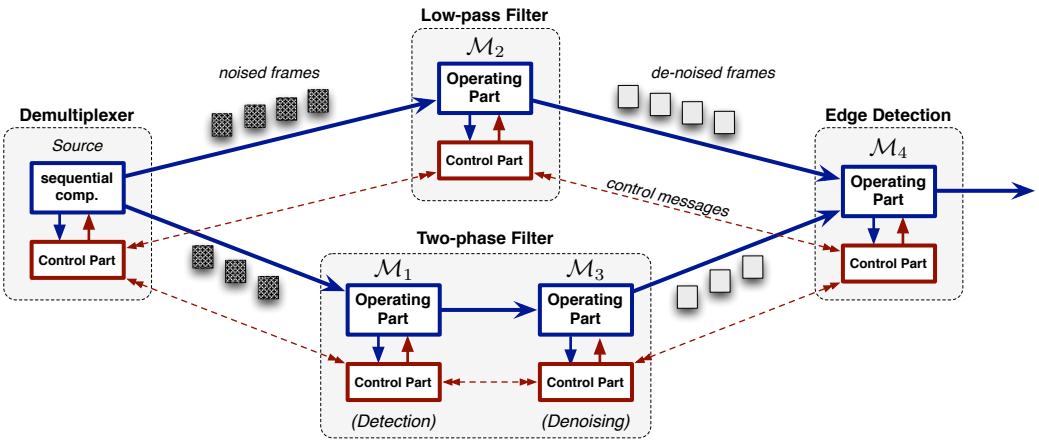


Figure 7. Computation graph of the synthesized video-streaming application.

Both the filters implement computationally demanding functions that need to be parallelized to meet real-time constraints. The first one is a simple *low-pass* filter applied to different frames independently, i.e. $\mathcal{M}_2$ is parallelized as a task-farm. The second one is more complex and simulates

[‡]Visit http://www.omnetpp.org/ for further details about the OmNeT++ simulator.

DOI: 10.1002/cpe

the two-phase filter described in [61]. The *detection* phase is performed by component $\mathcal{M}_1$ and consists in a *map* data-parallel parallelization. The second one (*denoising*), performed by $\mathcal{M}_3$, is a *map-reduce* parallelization operating on a sub-set of the image pixels identified by the first phase. The two phases operate in a pipeline fashion on each received frame. Filtered frames are finally collected by component $\mathcal{M}_4$ which performs a data-parallel computation for the last phase of edge detection. Final results are transmitted to further phases of the application (motion detection and object recognition) not analyzed in this benchmark. For the sake of simplicity we assume that the service time of the components decreases ideally with the parallelism degree, i.e. $T_{S_i}(k) = T_{calc_i}(k)/n_i(k)$ which maintains the convexity required by the Distributed Subgradient Method.

Tab. II summarizes the main configuration parameters by showing the mean calculation time per task and the maximum parallelism degree that can be assigned at each ParMod. The sequential calculation time per tasks (in seconds) is simulated by a normal random variable with a fixed mean and a standard deviation equal to 30% of the mean. The parameters represent a realistic configuration by assuming a frame size of 352x240 pixels and an average range of noise levels of 10% (see [61] for further details).

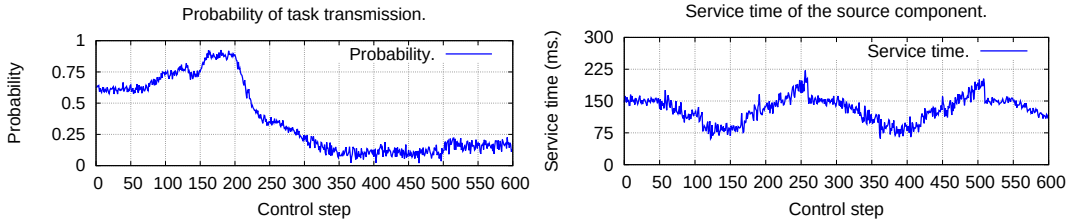
	Low-pass Filter	*Two-phase Filter*		*Edge Detection*
	$\mathcal{M}_2$	$\mathcal{M}_1$ (*Detection*)	$\mathcal{M}_3$ (*Denoising*)	$\mathcal{M}_4$
Calc. Time (s)	2.48s	1.90s	10.31s	4.30s
Max Par. Degree.	32	48	128	64

Table II. Sequential calculation times and maximum number of nodes per ParMod.

The ParMods adapt their parallelism degrees in order to sustain the current workload intensity. We analyze a scenario characterized by the presence (and the combined effect) of two disturbances:

- the source generates frames with a variable intensity (e.g. the frame rate is higher when motion is detected). In other words the mean service time of the source may change significantly during the execution;
- the source transmits frames to the first or to the second filter with a time-varying probability. We denote by $p(k)$ the probability to transmit a frame to $\mathcal{M}_2$. Therefore $1 - p(k)$ is the probability to transmit a frame to the two-phase filter implemented by $\{\mathcal{M}_1, \mathcal{M}_3\}$. Higher values of $p(k)$ correspond to a higher arrival rate to $\mathcal{M}_2$ and vice-versa.

The workload characterization of video surveillance systems has been studied in some past research works [62, 63, 64] by analyzing the behavioral patterns and workload models of some existing applications. As stated in these works, cyclic (seasonal) patterns and trends are representative patterns that characterize the workload of such systems (e.g. dynamic frame rate). In our benchmark we use two synthesized workload traces depicted in Fig. 8a, for the probability p, and in Fig. 8b for the service time of the source module. The execution consists of 600 control steps each one of 60s for a total simulation time of 10 hours.



(a) Probability.

(b) Source service time.

Figure 8. Multi-disturbance scenario of the application.

DOI: 10.1002/cpe

We use time-series forecasting methods to predict the future behavior of the workload. For the time-series in Fig. 8a we adopt a *Holt-Winters* forecasting model [65] able to estimate trends by using two exponentially weighted moving average filters (EWMA) for the level and the trend parts of the prediction. For the service time time-series, the predictions are generated by a seasonal Holt-Winters model with a third EWMA filter for the cyclic part. The filter parameters have been trained using an initial fitting period of observations, by minimizing the sum of the squared one-step ahead forecast errors.

5.1. Results with fixed horizons

We evaluate our MPC strategies on the benchmark application. Local control problems are solved in a cooperative fashion with the Distributed Subgradient Method. Each Control Part assigns the same weight to its estimate and the estimates received by its neighbors (*Equal Weight Rule* [10, 54]). The weight matrix for the graph of Fig. 7 is the following:

$$\mathcal{W} = \begin{array}{c} \begin{array}{ccccc} source & \mathcal{M}_1 & \mathcal{M}_2 & \mathcal{M}_3 & \mathcal{M}_4 \end{array} \\ \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 \\ 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \end{pmatrix} \end{array} \begin{array}{c} source \\ \mathcal{M}_1 \\ \mathcal{M}_2 \\ \mathcal{M}_3 \\ \mathcal{M}_4 \end{array}$$

The graph between controllers (red dashed lines connecting Control Parts in Fig. 7) is connected, so we can prove that the Distributed Subgradient Method converges to the optimal set of reconfiguration trajectories at each control step. Thanks to the use of the integer rounding and the warm start approach (see Sect. 4.3), only $\mathcal{I} = 100$ iterations are sufficient to achieve an average error of 2% between the optimal solutions and the final estimates calculated by the controllers.

In the first series of experiments we compare the results of `FH-MPC` strategies with different switching costs and horizon lengths. According to the concepts introduced in Sect. 3.2 we use the following metrics:

- the *stability* (frequency) of reconfigurations is measured by the total number of parallelism degree variations performed by the application components, and the average time between reconfigurations. We denote by `MSI` (*Mean Stability Index*) the average number of control steps between two successive reconfigurations of a ParMod;
- we use three metrics for the *reconfiguration amplitude*: *i)* the mean amplitude (the average number of nodes allocated/deallocated during a parallelism degree variation), *ii)* the standard deviation of the reconfiguration amplitude, *iii)* the "largest" reconfiguration occurred during the execution (peak amplitude);
- we define the *relative efficiency* as the ratio between the ideal service time of a ParMod and its inter-departure time i.e. $\mathcal{E}_i(k) = T_{S_i}(k)/T_{Di}(k)$. We measure the average efficiency as follows:
$$\overline{\mathcal{E}} = \sum_{i=1}^{N} \sum_{k=1}^{T} \mathcal{E}_i(k)$$
where N is the number of ParMods and T is the number of control steps ($N = 5$ and $T = 600$ in the experiments);
- for the *performance* we measure the total number of frames (tasks) leaving the systems, i.e. departing from ParMod $\mathcal{M}_4$.

Tab. III shows the values of the metrics for all the application components. For the sake of brevity the amplitude results are shown only for the third ParMod ($\mathcal{M}_3$), which is the one subject to more reconfigurations and with the highest amplitude among the other components ($\mathcal{M}_3$ executes the most compute-intensive phase of the application and needs more resources).

In the table we denote by `no_sw` the strategy without any switching cost ($\Delta_i^{sw}(k) = 0$ in Expr. 3). This strategy is applied with a horizon of one step since longer horizons would not change the

DOI: 10.1002/cpe

Strategy	*Reconf. Stability*		*Reconf. Amplitude* ($\mathcal{M}_3$)			*Efficiency*	*Tasks*
	Num. reconf.	MSI	*mean*	*stddev*	*max*		
no_sw ($h = 1$)	1069	2.64	1.21	0.47	3	0.94	192,467
abs_sw ($h = 1$)	612	4.16	1.19	0.50	4	0.86	200,987
abs_sw ($h = 2$)	897	3.42	1.15	0.39	3	0.92	198,258
abs_sw ($h = 3$)	999	2.78	1.17	0.44	3	0.92	193,857
pow_sw ($h = 1$)	530	4.87	1.05	0.28	2	0.84	202,043
pow_sw ($h = 2$)	690	3.77	1.09	0.37	3	0.88	209,498
pow_sw ($h = 3$)	716	3.73	1.07	0.33	3	0.91	201,569
max_degree	-	-	-	-	-	0.52	289,086

Table III. Results of FH-MPC strategies with different types of switching costs (h is the horizon length).

reconfiguration trajectories without a switching cost. The other strategies take into account different switching costs: the absolute value switching cost (abs_sw) and the square switching cost (pow_sw) introduced in Sect. 4.1, and three different lengths h of the prediction horizon of one, two and three control steps. Furthermore, in order to compare our adaptation strategies with a static configuration, we analyze the results of the execution with all the ParMods using their maximum parallelism degree throughout the execution (strategy denoted by max_degree).

The static max_degree strategy is the one optimizing the number of completed tasks at the expense of a very low efficiency. In the average case we waste half of the assigned resources (the efficiency is near to 0.5).

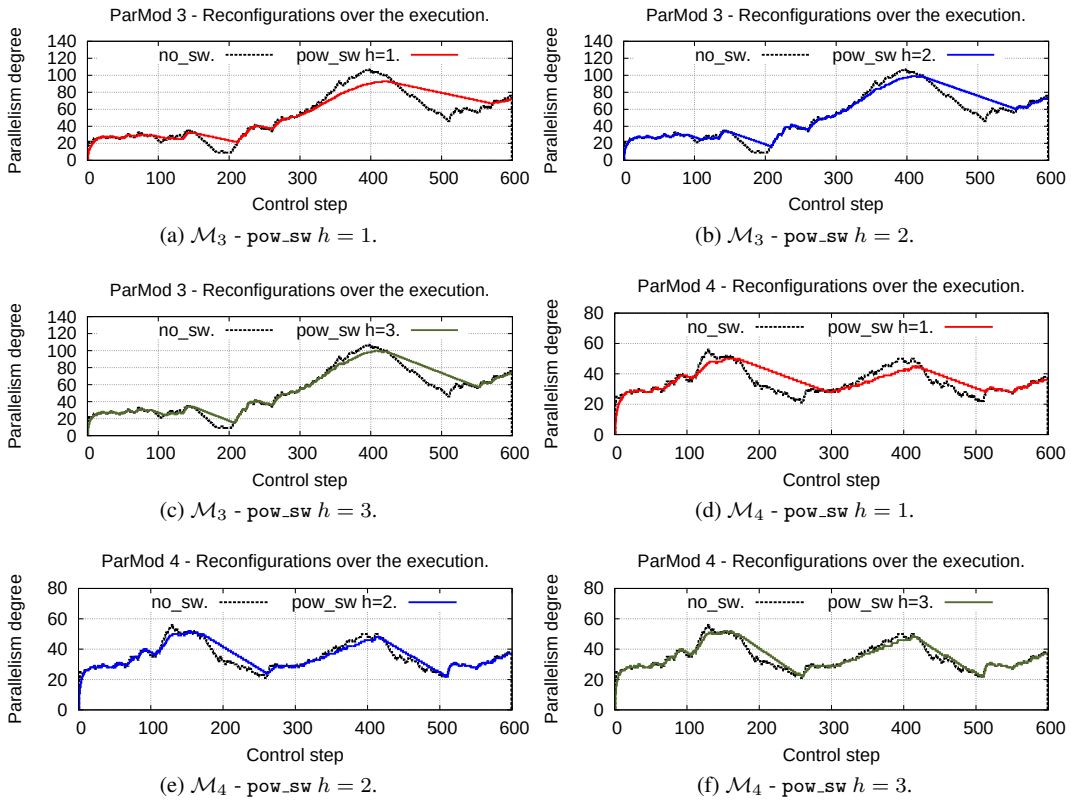
The no_sw strategy is effective to improve the efficiency. At each control step the Control Parts select the optimal parallelism degrees to achieve the best trade-off between performance and resource consumption. If the workload can be accurately predicted, the use of oversized parallelism degrees is undesirable since it increases the resource cost without a real performance improvement. With this strategy the efficiency becomes near to the ideal one but we lose 33% in completed tasks than the max_degree strategy. The reason for this loss in performance has two folds. Firstly, the use of parallelism in excess in the max_degree strategy makes it possible to complete more frames when the future workload is underestimated. Secondly, a high number of reconfigurations can be counter-productive because reconfigurations may bring to a long-term improvement in performance, but this positive aspect must be traded-off with the transient cost to apply them. To model this aspect in the simulator we use an average reconfiguration delay of 15s.

An effective adaptation strategy should avoid to perform unnecessarily reconfigurations. This is the intent of switching cost strategies. In Tab. III we can see that by using a switching cost and a horizon of one step, the reduction in reconfigurations is remarkable (of 42.75% and 50.42% with abs_sw and pow_sw). To understand the effect of switching costs Fig. 9 shows the sequences of reconfigurations of $\mathcal{M}_3$ and $\mathcal{M}_4$ with the no_sw and the pow_sw strategies (qualitatively the same behavior has been observed with abs_sw). The reconfiguration sequence of $\mathcal{M}_3$ depends on the combined effect of the source service time variability and the probability $p(k)$. In fact, both of them influence the arrival rate (workload) to the sub-system $\{\mathcal{M}_1, \mathcal{M}_3\}$. During phases in which the arrival rate is higher the parallelism degree of $\mathcal{M}_3$ increases. The contrary happens during phases characterized by a lower pressure of the input stream.

The reconfiguration sequence of $\mathcal{M}_4$ is not influenced by the probability $p(k)$ but follows the variability of the source service time only. This is an expected phenomenon. In fact, $\mathcal{M}_4$ receives frames from $\mathcal{M}_2$ and $\mathcal{M}_3$ and processes them in a FIFO fashion. Thus, its arrival rate does not depend on how the source split the frames between the low-pass filter ($\mathcal{M}_2$) and the two-phase filter ($\{\mathcal{M}_1, \mathcal{M}_3\}$).

By observing Fig. 9 we can point out that the switching cost is a disincentive to reconfigurations. During phases in which the arrival rate becomes lower it slows down the release of computing resources, while during phases with a growing rate it slows down the allocation of new resources. As we consider longer horizons the number of reconfigurations increases (this is also true for the abs_sw switching cost). The reason is that our multiple-step ahead forecasts are able to capture

DOI: 10.1002/cpe



(a) $\mathcal{M}_3$ - `pow_sw` $h = 1$. (b) $\mathcal{M}_3$ - `pow_sw` $h = 2$.

(c) $\mathcal{M}_3$ - `pow_sw` $h = 3$. (d) $\mathcal{M}_4$ - `pow_sw` $h = 1$.

(e) $\mathcal{M}_4$ - `pow_sw` $h = 2$. (f) $\mathcal{M}_4$ - `pow_sw` $h = 3$.

Figure 9. Reconfiguration sequences of the third and the fourth ParMod.

future trends in the workload. During increasing trend phases, if the horizon is sufficiently long controllers can anticipate the acquisition of new resources that will be necessary in the near future by reducing the number of reconfigurations and their amplitude. The opposite behavior characterizes decreasing trend phases, where resources are released more quickly by using longer horizons. The reduction in reconfigurations is more marked with the `pow_sw` switching cost because the square of parallelism degree variations has a higher weight than the absolute value.

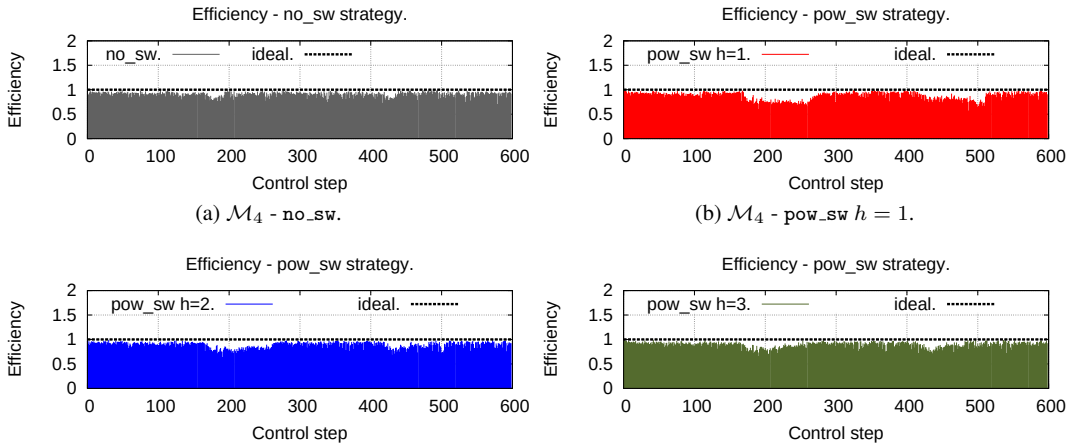
In conclusion, by increasing the horizon length more reconfigurations are performed and the sequence tends to the one without the switching cost. However, many reconfigurations due to the high variance of disturbances can be avoided by optimizing the reconfiguration stability. With `pow_sw` and $h = 1$ there is a reconfiguration every 4.87 control steps in the average case, that is a 46% improvement compared with the `MSI` without using the switching cost (2.64).

A sufficiently long horizon has also a positive effect on the relative efficiency, since the reconfiguration sequence follows more quickly the disturbance variability. In the case of a horizon of three steps the average efficiency is similar to the one achieved by the `no_sw` strategy, as depicted in Tab. III and graphically represented in Fig 10 for ParMod $\mathcal{M}_4$. It is worth noting that the phases in which the efficiency is not optimal are the ones in which the ParMod releases resources during decreasing trends of its workload (see Figs. 9d, 9e and 9f).

In terms of completed tasks the best strategy is `pow_sw` with a prediction horizon of two control steps. In this benchmark this strategy achieves the best trade-off between the number of reconfigurations and their amplitude by optimizing the number of completed tasks, which is 8.13% greater than in the `no_sw` case (losing 6% in efficiency only).

The last aspect is the analysis of the reconfiguration amplitude. Tab. IV shows the percentage of δ-reconfigurations (reconfigurations of amplitude δ) using different horizon lengths and switching costs. As discussed in Sect. 3.2, the `pow_sw` definition, besides being able to reduce the number

DOI: 10.1002/cpe



(a) $\mathcal{M}_4$ - `no_sw`. (b) $\mathcal{M}_4$ - `pow_sw` $h = 1$.

(c) $\mathcal{M}_4$ - `pow_sw` $h = 2$. (d) $\mathcal{M}_4$ - `pow_sw` $h = 3$.

Figure 10. Efficiency of $\mathcal{M}_4$ at each control step of the execution.

of reconfigurations, is further capable of improving the reconfiguration amplitude. The baseline is the case without switching costs, where there is no brake in the release/acquisition of resources. In this case almost 20% of reconfigurations involve more than one added/removed node. As we can observe, only a slight improvement in the average amplitude can be achieved with `abs_sw`. With the `pow_sw` strategy and a horizon of one step the number of 1-reconfigurations increases of 14% and we are able to remove completely reconfigurations with an amplitude greater than 2.

Strategy	*1-reconf.*	*2-reconf.*	*3-reconf.*	*4-reconf.*
`no_sw` ($h = 1$)	82.2%	15.6%	2.2%	-
`abs_sw` ($h = 1$)	84.3%	11.8%	2.92%	0.98%
`abs_sw` ($h = 2$)	85.6%	13.3%	1.1%	-
`abs_sw` ($h = 3$)	85.2%	12.9%	1.9%	-
`pow_sw` ($h = 1$)	96.2%	3.8%	-	-
`pow_sw` ($h = 2$)	92.7%	6.9%	0.4%	-
`pow_sw` ($h = 3$)	93.8%	5.2%	1%	-

Table IV. Analysis of the reconfiguration amplitude of ParMod $\mathcal{M}_4$ with different switching costs.

To conclude this first part of the experiments, Tab. V shows a qualitative analysis of the adaptation strategies designed with fixed horizon lengths. The table highlights an important fact: there does not exist a strategy optimizing all the adaptation properties. The `no_sw` strategy optimizes the efficiency by properly sizing the parallelism degrees at each control step of the execution. The consequence is a lower stability and a larger mean amplitude due to the disturbance variance. This also induces a lower performance due to the overhead to apply reconfigurations. The stability and amplitude can be optimized by using the `pow_sw` strategies. In this case the horizon length plays a decisive role. As we use longer horizons the stability and amplitude slightly worsen with the advantage of improving the efficiency. In this example the strategy completing the highest number of tasks is achieved by the selection of a two step ahead horizon.

In the last part of this paper we will extend our discussion by showing the application of variable-horizon MPC strategies according to the formulation described in Sect. 4.3.

5.2. Results with variable horizons

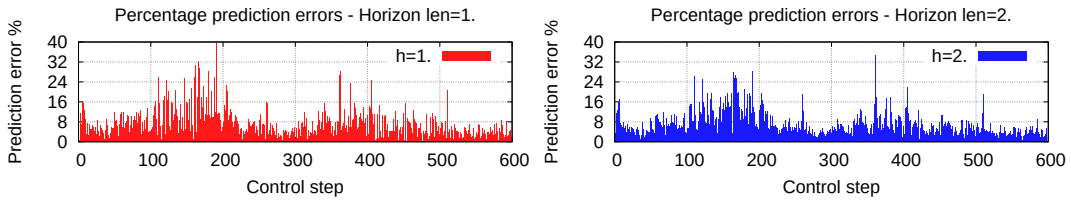
Our `VH-MPC` approach consists in adapting the horizon length according to the current disturbance prediction errors. Control Parts will prolong the prediction horizon during phases of the execution characterized by sufficiently accurate predictions. On the opposite a shortening of the horizon

DOI: 10.1002/cpe

Strategy	*Efficiency*	*Stability*	*Amplitude*	*Performance*
`no_sw` ($h = 1$)	$\star\star\star$	$\star$	$\star$	$\star$
`pow_sw` ($h = 1$)	$\star$	$\star\star\star$	$\star\star\star$	$\star\star$
`pow_sw` ($h = 2$)	$\star\star$	$\star\star$	$\star\star$	$\star\star\star$
`pow_sw` ($h = 3$)	$\star\star\star$	$\star\star$	$\star\star$	$\star\star$

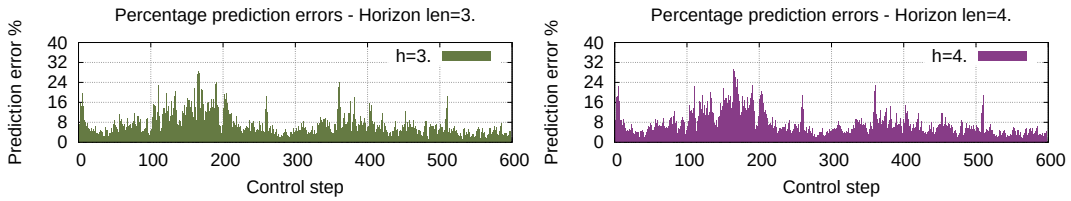
Table V. Qualitative analysis: $\star\star\star$ denotes the best results, $\star$ the worst ones.

will be performed when the current accuracy is not satisfactory. The rationale is intuitive: the prediction accuracy decreases by using too long horizons, thus longer horizons will be chosen provided that a sufficient accuracy can be expected by the next multiple-step ahead predictions. Fig. 11 shows the percentage errors between the real disturbance trajectory and the h-step ahead prediction (with $h = 1, \ldots, 4$) performed at each control step. We measure the average errors of both the disturbances, i.e. the probability $p(k)$ and the source service time.



(a) 1-step ahead prediction errors.

(b) 2-step ahead prediction errors.



(c) 3-step ahead prediction errors.

(d) 4-step ahead prediction errors.

Figure 11. Percentage error between real measurements and predicted trajectories at each control step.

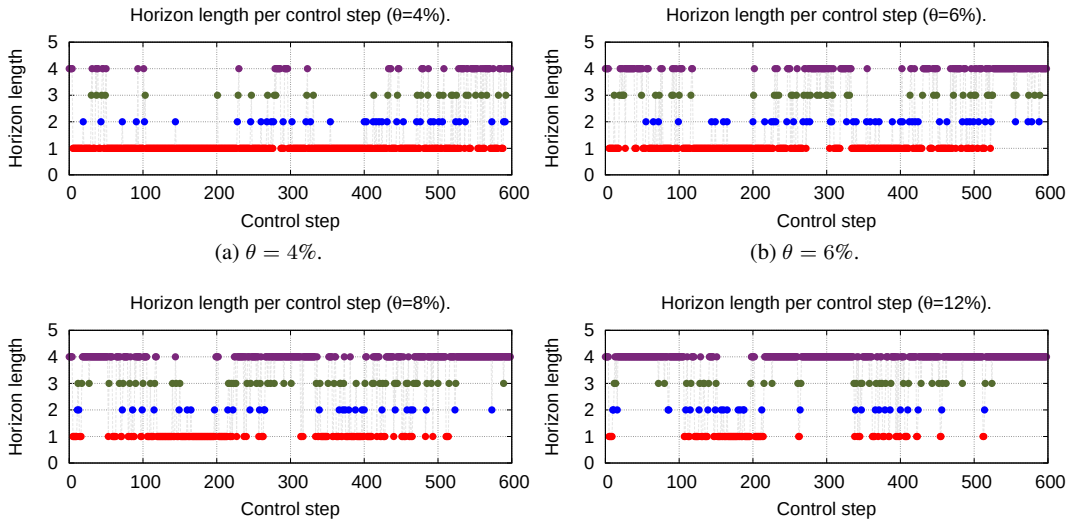
We can note that with longer horizons we have lower error peaks. This is an effect of averaging, since even if peaks are still present they are smoothed by the average of the errors over the horizon. The total average error is of 6.7%, 6.9%, 7.2% and 7.4% for $h = 1, 2, 3, 4$ steps ($h^{max} = 4$ is the maximum horizon length considered in this experiment). This behavior can be graphically observed in Fig. 11, where the error lines are denser using longer horizons.

We have performed simulations using the `VH-MPC` strategy with different thresholds θ for the prediction accuracy (see Sect. 4.3). Fig. 12 shows the horizon length used at each control step of the execution with different thresholds of 4%, 6%, 8% and 12%.

It is interesting to compare the results of Fig. 12 with the prediction errors shown in Fig. 11. With a low tolerance of 4% for most of the control steps we use the minimum horizon of just one step. This can be clearly observed in Fig. 12a, where the average prediction horizon during the entire execution is of 1.53 control steps. The horizon is prolonged up to the maximum length $h^{max} = 4$ mainly in the last part of the execution (from step 530 to 600), where the accuracy is good also with 4-step ahead predictions (see Fig. 11d).

With a high tolerance threshold the controllers use longer horizons on average. For higher values of θ the number of steps in which we use the maximum horizon length increases. However, we can recognize two execution phases characterized by less accurate predictions: the phase from step 100 to 220 and from 380 to 420. During these phases the ParMod controllers favor the use of the shortest horizon also with a high threshold of 12%. A complete analysis of the *average horizon length* with different threshold values is depicted in Fig. 13.

DOI: 10.1002/cpe



(a) $\theta = 4\%$. (b) $\theta = 6\%$.

(c) $\theta = 8\%$. (d) $\theta = 12\%$.

Figure 12. Horizon length selected by the VH-MPC strategy at each control step.

The dynamic selection of the horizon length makes it possible to achieve better trade-offs between performance, efficiency, number of reconfigurations, their stability and amplitude than by using a fixed horizon length. Tab. VI shows the results in detail. The VH-MPC strategy with $\theta = 4\%$ is very effective. Compared with FH-MPC(h = 1), which is the best one in terms of reconfiguration stability and amplitude, we reach a similar number of reconfigurations (we have only a small increase of 5.19%) and we are still able to remove all the reconfigurations with amplitude greater than 2. In terms of efficiency the result is similar to the one of the FH-MPC(h = 2) strategy and 3.6% better than FH-MPC(h = 1). Finally, in terms of completed tasks we are able to complete $207,596$ tasks during the entire execution, slightly lower than the best fixed-horizon strategy. The difference with FH-MPC(h = 2) is of 1% only.

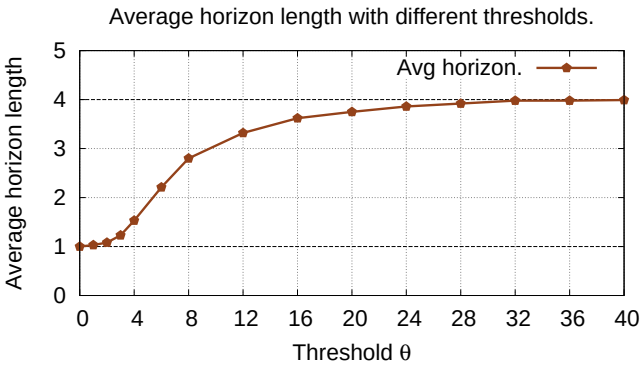


Figure 13. Average horizon length with different thresholds θ.

In conclusion the results are consistent with our initial intuitions. The variable-horizon strategy with a good threshold achieves better trade-offs compared with strategies using a fixed horizon length. The proposed scheme is effective and with low complexity since the horizon selection policy only needs to collect the last prediction errors.

DOI: 10.1002/cpe

Strategy	*Num. reconf.*	*Avg. hor.*	*Reconf. Amplitude* ($\mathcal{M}_3$)			*Efficiency*	*Tasks*
			mean	*1-reconf*	*max*		
VH-MPC $\theta = 4\%$	559	1.53	1.056	94.95%	2	0.896	207,596
VH-MPC $\theta = 6\%$	614	2.21	1.058	94.62%	2	0.898	203,708
VH-MPC $\theta = 8\%$	652	2.80	1.061	94.35%	2	0.900	204,616
VH-MPC $\theta = 12\%$	680	3.32	1.066	93.85%	2	0.900	203,717

Table VI. Results of the VH-MPC strategies with different thresholds.

6. CONCLUSIONS AND FUTURE WORK

The tendency of the modern distributed computing platforms is to provide autonomic features enabling the users to dynamically manage their time-varying resource requirements through (re)allocation policies. The optimal application configuration on these environments is a critical problem that needs to take into account the optimization of several metrics. The reconfiguration sequence can be evaluated according to metrics related to the reconfiguration stability and amplitude (their frequency and "size"), while the result of the adaptation process can be analyzed from the performance (e.g. number of completed tasks) and efficiency (utilization factor of the assigned resources) viewpoint.

In this paper we propose an application of Distributed Model Predictive Control, an adaptation strategy in which the control problem of the entire system is decomposed in coupled sub-problems solved by a set of controllers. In our method control problems are solved in a cooperative fashion according to the Distributed Subgradient Method, a feasible technique applicable to non-differentiable convex cost functions. The first goal of this paper is to study different switching costs in the definition of the optimization problem, and their impact on the performance, efficiency, reconfiguration stability and amplitude. Qualitatively, all the switching cost functions behave similarly, with the pow_sw definition able to optimize the reconfiguration amplitude.

The second goal is the analysis of a variable-horizon MPC strategy, in which the horizon length is adapted during the execution instead of being a fixed value. We propose an approach in which controllers choose the horizon length according to the last prediction errors. This paper presents a validation on a simulation environment able to reproduce the interaction among distributed parallel components. We evaluate fixed- and variable-horizon strategies and their qualitative/quantitative effects on the interesting metrics of the adaptation process. The results show that with fixed horizons there does not exist an optimal strategy achieving the best results for all the metrics, and the variable-horizon MPC is effective in improving the trade-off.

On the future we plan to extend our work. A first direction is to explore the use of a discounting factor in the formulation of optimization problems. Furthermore, the control step length, instead of being a fixed sampling interval as in the examples considered in this paper, can be a manipulable parameter of the controllers. This may be an interesting solution to quickly respond to the time-varying nature of the workload without a too large control overhead, e.g. by intensifying the sampling frequency when we detect highly variable disturbances. Finally, it is our goal to evaluate our ideas in a real implementation for Grid and Cloud applications.

REFERENCES

1. Babu S, Widom J. Continuous queries over data streams. *SIGMOD Record* September 2001; **30**(3).
2. Räty TD. Survey on contemporary remote surveillance systems for public safety. *Trans. Sys. Man Cyber Part C* Sep 2010; **40**(5):493–515.
3. Bertolli C, Mencagli G, Vanneschi M. A cost model for autonomic reconfigurations in high-performance pervasive applications. *Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems*, CASEMANS '10, ACM: New York, NY, USA, 2010; 3:20–3:29.
4. Fantacci R, Vanneschi M, Bertolli C, Mencagli G, Tarchi D. Next generation grids and wireless communication networks: towards a novel integrated approach. *Wirel. Commun. Mob. Comput.* 2009; **9**(4):445–467.
5. Huebscher MC, McCann JA. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.* 2008; **40**(3):1–28.

DOI: 10.1002/cpe

6. McKinley P, Sadjadi S, Kasten E, Cheng BHC. Composing adaptive software. *Computer* 2004; **37**(7):56–64.
7. Vanneschi M, Veraldi L. Dynamicity in distributed applications: issues, problems and the assist approach. *Parallel Comput.* 2007; **33**(12):822–845.
8. Maggio M, Hoffmann H, Papadopoulos AV, Panerati J, Santambrogio MD, Agarwal A, Leva A. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans. Auton. Adapt. Syst.* Dec 2012; **7**(4):36:1–36:32.
9. Garcia CE, Prett DM, Morari M. Model predictive control: theory and practice a survey. *Automatica* May 1989; **25**:335–348.
10. Nedic A, Ozdaglar A. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on* jan 2009; **54**(1):48 –61.
11. Bahati RM, Bauer MA, Vieira EM. Policy-driven autonomic management of multi-component systems. *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, CASCON '07, IBM Corp.: Riverton, NJ, USA, 2007; 137–151.
12. Weng D, Bahati RM, Bauer MA. Policy-based autonomic management in virtual machine systems. *GCA*, Arabnia HR, Gravvanis GA (eds.), CSREA Press, 2009; 120–125.
13. Kephart J, Walsh W. An artificial intelligence perspective on autonomic computing policies. *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, 2004; 3–12.
14. Liu H, Parashar M. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* may 2006; **36**(3):341 – 352.
15. Aldinucci M, Campa S, Danelutto M, Vanneschi M. Behavioural skeletons in gcm: Autonomic management of grid components. *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008.*, 2008; 54–63.
16. Weigold T, Aldinucci M, Danelutto M, Getov V. Process-driven biometric identification by means of autonomic grid components. *Int. J. Auton. Adapt. Commun. Syst.* Jul 2012; **5**(3):274–291.
17. González-Vélez H, Leyton M. A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers. *Softw. Pract. Exper.* Nov 2010; **40**(12):1135–1160.
18. Reiff-Marganiec S, Turner KJ. Feature interaction in policies. *Comput. Netw.* August 2004; **45**:569–584.
19. Bahati R, Bauer M. Towards adaptive policy-based management. *Network Operations and Management Symposium (NOMS), 2010 IEEE*, 2010; 511–518.
20. Hellerstein JL, Diao Y, Parekh S, Tilbury DM. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.
21. Diao Y, Hellerstein J, Parekh S, Griffith R, Kaiser G, Phung D. A control theory foundation for self-managing computing systems. *Selected Areas in Communications, IEEE Journal on* dec 2005; **23**(12):2213 – 2222.
22. Zhang R, Lu C, Abdelzaher TF, Stankovic JA. Controlware: A middleware architecture for feedback control of software performance. *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, IEEE Computer Society: Washington, DC, USA, 2002; 301–.
23. Raghavendra R, Ranganathan P, Talwar V, Wang Z, Zhu X. No "power" struggles: coordinated multi-level power management for the data center. *SIGOPS Oper. Syst. Rev.* Mar 2008; **42**(2):48–59.
24. Park SM, Humphrey M. Predictable high-performance computing using feedback control and admission control. *Parallel and Distributed Systems, IEEE Transactions on* 2011; **22**(3):396–411.
25. Lin M, Wierman A, Roytman A, Meyerson A, Andrew LL. Online optimization with switching cost. *SIGMETRICS Perform. Eval. Rev.* Jan 2012; **40**(3):98–100.
26. Yu H, Chang EC, Tsang Ooi W, Chan MC, Cheng W. Integrated optimization of video server resource and streaming quality over best-effort network. *Circuits and Systems for Video Technology, IEEE Transactions on* 2009; **19**(3):374–385.
27. Wang K, Lin M, Ciucu F, Wierman A, Lin C. Characterizing the impact of the workload on the value of dynamic resizing in data centers. *INFOCOM, 2013 Proceedings IEEE*, 2013; 515–519.
28. Bonorden O, Gehweiler J, Heide F. A web computing environment for parallel algorithms in java. *Parallel Processing and Applied Mathematics*, *Lecture Notes in Computer Science*, vol. 3911, Wyrzykowski R, Dongarra J, Meyer N, Waśniewski J (eds.). Springer Berlin Heidelberg, 2006; 801–808.
29. Kusic D, Kandasamy N. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. *Cluster Computing* 2007; **10**(4):395–408.
30. Yuan Q, Liu Z, Peng J, Wu X, Li J, Han F, Li Q, Zhang W, Fan X, Kong S. A leasing instances based billing model for cloud computing. *Proceedings of the 6th international conference on Advances in grid and pervasive computing*, GPC'11, Springer-Verlag: Berlin, Heidelberg, 2011; 33–41.
31. Kusic D, Kandasamy N, Jiang G. Combined power and performance management of virtualized computing environments serving session-based workloads. *Network and Service Management, IEEE Transactions on* 2011; **8**(3):245–258.
32. Kusic D, Kandasamy N, Jiang G. Approximation modeling for the online performance management of distributed computing systems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* Oct 2008; **38**(5):1221–1233.
33. Mencagli G, Vanneschi M, Vespa E. A cooperative predictive control approach to improve the reconfiguration stability of adaptive distributed parallel applications. *ACM Trans. Auton. Adapt. Syst.* Mar 2014; **9**(1):2:1–2:27.
34. Mencagli G, Vanneschi M. Towards a systematic approach to the dynamic adaptation of structured parallel computations using model predictive control. *Cluster Computing* 2014; :1–21.
35. Mencagli G, Vanneschi M, Vespa E. Reconfiguration stability of adaptive distributed parallel applications through a cooperative predictive control approach. *Proceedings of the 19th international conference on Parallel Processing*, Euro-Par'13, Springer-Verlag: Berlin, Heidelberg, 2013; 329–340.
36. Mencagli G, Vanneschi M, Vespa E. Control-theoretic adaptation strategies for autonomic reconfigurable parallel applications on cloud environments. *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, 2013; 11–18. Won the outstanding paper award.
37. Mencagli G, Vanneschi M. Qos-control of structured parallel computations: A predictive control approach. *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*,

CLOUDCOM '11, IEEE Computer Society: Washington, DC, USA, 2011; 296–303.
38. Bertolli C, Mencagli G, Vanneschi M. Analyzing memory requirements for pervasive grid applications. *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010; 297–301, doi:10.1109/PDP.2010.71.
39. Vanneschi M. The programming model of assist, an environment for parallel and distributed portable applications. *Parallel Comput.* Dec 2002; **28**(12):1709–1732.
40. Bacci B, Danelutto M, Pelagatti S, Vanneschi M. A heterogeneous environment for hpc applications. *Parallel Computing* 1999; **25**(13-14):1827–1852.
41. Bacci B, Danelutto M, Orlando S, Pelagatti S, Vanneschi M. P3l: A structured high-level parallel language, and its structured support. *Concurrency: Practice and Experience* 1995; **7**(3):225–255.
42. Cole M. Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. *Parallel Comput.* Mar 2004; **30**(3):389–406.
43. Bertolli C, Mencagli G, Vanneschi M. Consistent reconfiguration protocols for adaptive high-performance applications. *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, 2011; 2121–2126.
44. Costa R, Brasileiro F, Lemos G, Sousa D. Analyzing the impact of elasticity on the profit of cloud computing providers. *Future Generation Computer Systems* 2013; .
45. Han R, Ghanem MM, Guo L, Guo Y, Osmond M. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems* 2012; .
46. Hameurlain A, Cokuslu D, Erciyes K. Resource discovery in grid systems: a survey. *Int. J. Metadata Semant. Ontologies* Jul 2010; **5**(3):251–263.
47. Bertolli C, Buono D, Mencagli G, Torquati M, Vanneschi M, Mordacchini M, Nardini FM. Resource discovery support for time-critical adaptive applications. *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, ACM: New York, NY, USA, 2010; 504–508.
48. Camponogara E, Jia D, Krogh B, Talukdar S. Distributed model predictive control. *Control Systems, IEEE* feb 2002; **22**(1):44 –52.
49. Scattolini R. Architectures for distributed and hierarchical model predictive control - a review. *Journal of Process Control* 2009; **19**(5):723–731.
50. Yaikhom G, Cole M, Gilmore S, Hillston J. A structural approach for modelling performance of systems using skeletons. *Electronic Notes in Theoretical Computer Science* 2007; **190**(3):167 – 183. ¡ce:title¿Proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL 2007)¡/ce:title¿.
51. Mencagli G. *A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations*. Ph.D Thesis, Department of Computer Science, University of Pisa, Italy, 2012. URL http://etd.adm.unipi.it/t/etd-05242012-212444/.
52. Lin M, Liu Z, Wierman A, Andrew LLH. Online algorithms for geographical load balancing. *Proceedings of the 2012 International Green Computing Conference (IGCC)*, IGCC '12, IEEE Computer Society: Washington, DC, USA, 2012; 1–10.
53. Zhang Q, Zhu Q, Zhani M, Boutaba R. Dynamic service placement in geographically distributed clouds. *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, 2012; 526–535.
54. Lobel I, Ozdaglar A. Distributed subgradient methods for convex optimization over random networks. *Automatic Control, IEEE Transactions on* june 2011; **56**(6):1291 –1306.
55. Ram SS, Nedic A, Veeravalli VV. Distributed subgradient projection algorithm for convex optimization. *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '09, IEEE Computer Society: Washington, DC, USA, 2009; 3653–3656.
56. Frew E, Langelaan J, Joo S. Adaptive receding horizon control for vision-based navigation of small unmanned aircraft. *American Control Conference, 2006*, 2006; 6 pp.–.
57. Barlow J. Data-based predictive control with multirate prediction step. *American Control Conference (ACC), 2010*, 2010; 5513–5519.
58. Núñez A, Vázquez-Poletti JL, Caminero AC, Castañé GG, Carretero J, Llorente IM. icancloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.* Mar 2012; **10**(1):185–209.
59. Wang X, Du Z, Chen Y, Li S, Lan D, Wang G, Chen Y. An autonomic provisioning framework for outsourcing data center based on virtual appliances. *Cluster Computing* Sep 2008; **11**(3):229–245.
60. Lisani JL, Rudin L, Monasse P, Morel JM, Yu P. Meaningful automatic video demultiplexing with unknown number of cameras, contrast changes, and motion. *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, 2005; 604–608.
61. Aldinucci M, Spampinato C, Drocco M, Torquati M, Palazzo S. A parallel edge preserving algorithm for salt and pepper image denoising. *Image Processing Theory, Tools and Applications (IPTA), 2012 3rd International Conference on*, 2012; 97–104.
62. Noceti N, Santoro M, Odone F. Learning behavioral patterns of time series for video-surveillance. *Machine Learning for Vision-Based Motion Analysis*, Wang L, Zhao G, Cheng L, Pietikäinen M (eds.). Advances in Pattern Recognition, Springer London, 2011; 275–304.
63. Saini M, Atrey P, Kankanhalli M. Workload modeling for multimedia surveillance systems. *Emerging Paradigms in Machine Learning*, *Smart Innovation, Systems and Technologies*, vol. 13, Ramanna S, Jain LC, Howlett RJ (eds.). Springer Berlin Heidelberg, 2013; 419–440.
64. Saini M, Xiangyu W, Atrey P, Kankanhalli M. Dynamic workload assignment in video surveillance systems. *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, 2011; 1–6.
65. Chatfield C, Yar M. Holt-winters forecasting: Some practical issues. *Journal of the Royal Statistical Society. Series D (The Statistician)* 1988; **37**(2):pp. 129–140.