Noname manuscript No.
(will be inserted by the editor)

Towards a Systematic Approach to the Dynamic Adaptation of Structured Parallel Computations using Model Predictive Control

Gabriele Mencagli · **Marco Vanneschi**

Received: date / Accepted: date

Abstract Adaptiveness is an essential feature for distributed parallel applications executed on dynamic environments like Grids and Clouds. Being adaptive means that parallel components can change their configuration at run-time (by modifying their parallelism degree or switching to a different parallel variant) to face irregular workload or to react to uncontrollable changes of the execution platform. A critical problem consists in the definition of adaptation strategies able to select optimal reconfigurations (minimizing operating costs and reconfiguration overhead) and achieve the stability of control decisions (avoiding unnecessarily reconfigurations). This paper presents an approach to apply Model Predictive Control (a form of optimal control studied in Control Theory) to adaptive parallel computations expressed according to the Structured Parallel Programming methodology. We show that predictive control is amenable to achieve stability and optimality by relying on the predictability of structured parallelism patterns and the possibility to express analytical cost models of their QoS metrics. The approach has been exemplified on two case-studies, providing a first assessment of its potential and feasibility.

Keywords Adaptiveness · Model Predictive Control · Structured Parallel Programming · Performance Modeling

1 Introduction

For distributed applications executed on dynamic environments like Grids and Clouds, *adaptiveness* is a key property to maintain an acceptable quality for the services provided to users [28,30]. The presence of dynamic events affecting the execution platforms, such as caused by the time-varying availability of networking and computing resources, or related to the application semantics, featuring statically unpredictable load distributions, may prevent any static application configuration to be effective in reaching the desired *Quality of Service* (QoS). Therefore, optimized *reconfiguration mechanisms* and smart *adaptation strategies* need to be defined to enhance such systems with supports to deal with changing environments and dynamic computations.

In the case of distributed parallel applications adaptiveness assumes interesting characterizations. Workflow graphs of arbitrarily interconnected parallel components can be dynamically modified by different classes of reconfigurations. Reconfigurations can affect the *parallelism degree*: by modifying its parallelism degree a component can release underutilized resources or acquire resources necessary to reach the desired performance. Reconfigurations can also change the parallel implementation adopted by a component (parallel *variant* or *version*). As an example a computation defined on an input stream of elements can be parallelized by applying the calculation independently on each element in parallel (*task farming*) or, if the elements are complex data structures like arrays or matrices, by applying the calculation in parallel on partitions of the same data structure (*data parallelism*). The different parallel variants solve the same problem, i.e. they preserve the component input/output interfaces, but are characterized by a different service time, latency, load balancing and memory occupancy.

The decision-making process is a critical part of any adaptive system. Reconfigurations must be carefully selected by taking into account their positive effects on the computation and the cost for concretely applying them. Several properties and goals can drive the reconfiguration selection:

– reconfigurations can be aimed at reaching desired tradeoffs between different and potentially contrasting QoS

Gabriele Mencagli and Marco Vanneschi
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3
I-56127, Pisa, Italy
E-mail: {mencagli, vannesch}@di.unipi.it

goals (*control optimality*). Examples are the optimization of aspects like performance, memory usage, and number and types of used resources;
- when comparing strategies reaching similar QoS levels, we could prefer the one improving the *stability* of control decisions. Informally a "stable" strategy avoids oscillating behaviors and minimizes the number of reconfigurations. This may be of great importance when reconfigurations cause significant economic costs or transient reconfiguration overhead on the system.

Accordingly, adaptation strategies can be evaluated using both *quantitative* and *qualitative* metrics. For instance, the number of *QoS violations* - e.g. number of times observed measurements exceed desired thresholds - average number of nodes used by the computation, number of performed reconfigurations, average time between successive reconfigurations of the same component, complexity of reconfigurations (e.g. number of involved nodes) and so on.

A survey of adaptation strategies has been recently presented in [37] by comparing standard and advanced control-based solutions and heuristics. This work has demonstrated that *Model Predictive Control* [47] (MPC) is a powerful strategy able to achieve good optimality and stability in uncertain environments. MPC relies on two pillars: *(i)* the presence of a mathematical *model* capturing the essential features of the system; *(ii)* the definition of an *optimization problem* respecting the model dynamics evaluated using predictions over a future time *horizon*.

Existing works have discussed the application of MPC to power and performance control of data centers, web servers and embedded CPUs [33,1], and for the dynamic allocation of virtual machines on Clouds [34]. These works present strategies to control physical or virtualized computing environments without exploiting any knowledge about the logic of applications executed on top of them. In this paper we adopt a different perspective: MPC strategies can be defined by reasoning directly on the target application, by coupling the strategy with the application itself. To face the complexity of parallel programming, we adopt a vision in which parallel programs are instances of a limited set of parallelism patterns. This method, known as *Structured Parallel Programming* [17,23] (SPP), permits to exploit the knowledge of the computation/communication scheme to define parametric *cost models* [23,58] of the meaningful QoS metrics. In this paper we show how to define MPC strategies for structured parallel computations using their cost models to choose the best application configuration. The result is an approach with a high degree of systematicity and confirming the properties of MPC in terms of optimality and stability. The research contributions are summarized as follows:

- we provide the basis for a systematic approach to the definition of MPC strategies for parallel computations. We discuss the contributions of SPP in each phase of the approach;
- we show the flexibility of SPP in defining implementation variants of the same component, based on different parallelism paradigms. This is a powerful way to extend the concept of adaptiveness for parallel components;
- we exemplify the approach with two benchmarks. Although such case-studies do not cover the entire space of parallelism patterns, they are representative and widely used in everyday life. Furthermore, they show the application of MPC when different performance measures are concerned, i.e. *throughput* and *response time*. The goal of such case-studies is twofold: first they exemplify the approach in a very tutorial-like manner, second they give an insight into how much cost models of SPP are suitable (and sufficiently precise) to drive an effective adaptation, i.e. our MPC strategies are evaluated according to optimality and stability criteria;
- to enact the MPC feasibility, which often represents a critical issue, we show how traditional techniques can be applied to the control of structured parallel computations. *Branch & Bound strategies* based on a combinatorial representation of the problem are discussed by showing their effectiveness in the studied examples.

In the next section we provide a literature overview about adaptive systems. Section 3 recalls the features of Structured Parallel Programming and cost models. Section 4 presents the central points of our approach and the contribution of SPP. Section 5 exemplifies the approach on two case-studies. Section 6 gives the conclusion of this work.

2 Related Work

A studied issue for adaptive distributed parallel applications has been represented by the design and development of reconfiguration mechanisms. Reconfigurations are intrusive actions that may induce performance degradation [52,4,22, 44]. Therefore, they must be designed according to highly optimized protocols with a null or a little impact on the computation performance. The literature has demonstrated that the SPP vision [17,23] is a key factor to greatly simplify the process of run-time modification of parallel components [19,55,2]. This paper focuses on a further positive aspect of SPP, i.e. parallelism patterns and their cost models are amenable to be integrated into advanced optimal control techniques, which represent interesting and attractive adaptation strategies.

Reconfigurations must be taken if the resulting configuration improves the achievement of QoS goals. A solution consists in providing the correspondence between reconfigurations and resulting QoS through *logic policy rules* [29, 46], which specify a declarative mapping between events

and corrective actions on the system. Frameworks adopting this vision are described in [36,41] for emergency management systems, and in [55,2] for generic distributed applications and structured parallel computations.

Rule-based strategies belong to a more general approach to develop adaptation strategies based on *heuristics* [37]. Heuristic solutions start from an initial guess about optimal application configuration and adjust this guess using intuitive rules often without relying on a mathematical model of the system. An example is the work presented in [7], in which power management of data centers has been studied using resource allocation policies taking into account QoS expectations of using different system configurations and the power usage characteristics of available physical resources. In [38] a heuristic strategy has been proposed to enable energy conservation for clusters that run Map-Reduce jobs. Scaling up/down activities of cluster nodes are triggered when the average utilization exceeds or becomes lower than specified thresholds.

Despite their cost in terms of accuracy and precision, heuristics are simple and with low complexity. However, as discussed in [37] the use of *model-free* strategies makes more difficult to prove the convergence to optimal solutions and to reach advanced properties such as the *stability* of control decisions and their *optimality*. As thoroughly explained in [37], standard and advanced control-based strategies may reach these goals, but their applicability to parallel computations is still an open issue.

Standard control-based techniques like PID controllers and Admission Control have been applied to the control of physical computing infrastructures, e.g. data centers, clusters and Clouds [53,42,27,34]. A seminal work describing the issues of applying Control Theory to computing systems has been described in [26] with applications to web servers and enterprise applications. This work confirms that providing a sufficiently accurate model of the target system is a first necessary pre-condition to apply control-based solutions. A common modeling approach is based on *statistical models*, achieved through observation of the system under desired operating conditions. Empirical black-box models have been used in several works like [42,26]. Combinations of different control strategies and models have been developed in [43]. This work proposes a strategy in which the system operating region is partitioned into sub-regions coupled with a model and a corresponding adaptation strategy. The idea, though interesting, it is difficult to be generalized, and the identification of correct and effective operating sub-regions can be a hard problem in presence of a large set of observed variables and a discrete space of manipulable control parameters.

Advanced control-based solutions and MPC have been applied to computing systems with special attention to high-performance resources and virtualized computing environments [53,42]. Predictive control has been preliminary applied in [1,33] for controlling: *(i)* a server farm, by dynamically varying the number of active nodes; *(ii)* a CPU, by dynamically adjusting the clock frequency. Although these past researches have some common points with the approach proposed in this paper (e.g. disturbances are taken into account through statistical forecasting), they are heavily tailored to the target physical platform without exploiting any knowledge about the controlled computations (as in [26,42], where models are extracted using black-box techniques). An idea is to apply MPC to parallel applications by exploiting the knowledge of the application structure to derive simple yet powerful models based on structured parallelizations. This is the goal of this paper, which joins the properties of MPC and the features of SPP.

3 Structured Parallel Programming and Cost Models

In this section we recall some basic concepts of SPP. SPP [17,23] is based on the utilization and composition of a limited set of parallel programming *paradigms*, also called *parallelism patterns* or *forms*, with a well-defined semantics. Basic typical paradigms are:

- *task-farm* (master-worker), in which a stream of tasks is scheduled for the execution by a set of identical workers, thus exploiting replication of stateless functions. Task scheduling is often implemented according to a load balancing strategy;
- *data-parallel*, in which a computation (stateless or with internal state) operating on complex data structures (uni- or multi-dimensional arrays) is replicated into a set of workers, each one operating on a partition of the input/output data (in addition, replication is often applied to a subset of data structures). Data-parallelism is a very general paradigm that can be specialized into several variants, in particular *map* (workers are independent during every execution step) or *stencil-based* (data dependences exist among the computation of the various workers, according to a cooperation scheme that may be *static and fixed, static and variable*, or even *dynamic*, passing from an execution step to the next ones), Moreover, different strategies for scheduling, data scattering, gathering, multicasting, as well as for reducing, prefixing and windowing, are adopted for different classes of algorithms or application domains.

Data-parallelism can be applied to both stream-based computations and to computations operating on a single set of data, while farming is meaningful for task streaming only. Task-farm and data-parallel variants of the same stream-based computation have different features in terms of load balancing (easier to achieve in farming), service time, latency and

memory capacity per node (better in data-parallelism). Alternatively, different paradigms have different optimal parallelism degrees for achieving similar performance values. Other parallel paradigms have been studied [17,23], such as divide-and-conquer, multi-pipeline, specific instances of data-flow computations, which often can be expressed in terms of farming, data-parallelism or their combinations.

A lot of experience has been done in the last twenty years by many research groups (including our group) in academic and industrial contexts about SPP. Accordingly, a rich set of *static* and *dynamic implementations* is known for each parallelism paradigm and its variants with respect to different parallel architectures with shared and/or distributed memory. The clear semantics of each paradigm, thus the knowledge of their machine-dependent implementation, renders it possible to define performance models, or more generally *cost models*, which is a distinguishable feature of SPP. Cost models concern configuration and evaluation parameters and metrics of a computation, such as the *optimal parallelism degree*, the *service time*, *latency*, completion time, *efficiency*, *scalability*, as well as the *memory requirements* and the *power consumption*. If a computation acts as a server with respect to a set of clients according to a request-reply interaction, then the *response time* is of main interest, i.e. in such case proper optimizations of the server service time and of the server latency are critical for the client performance.

Cost models are functions of *application-dependent parameters* (sequential calculation time, data size, inter-arrival times, and their statistical distributions, possibly determined through profiling or simulation) and of *architecture-dependent parameters* (process/thread management mechanisms, interconnection structures, memories, interfaces, protocols, and so on). An important feature of SPP is that all the performance metrics are *parametric in the parallelism degree*.

Furthermore, it is possible that, according to the complexity and/or to the performance unpredictability of the underlying architecture, an *analytical or numerical approach* to the resolution of the cost model reveals of unacceptable complexity. In these cases, a *simulation-based* or a mixed approach is the best one. In all cases (analytical/numerical resolution or simulation) the cost model application has to be integrated, and can be integrated, into the set of tools and strategies for the dynamic reconfiguration control.

4 Adaptive Parallel Computations and Predictive Control

In this section we describe our approach to apply MPC to distributed parallel applications expressed according to SPP. Applications can be represented as graphs of cooperating computational *modules* (Figure 1). We use the term module as a synonym of component. Modules can exchange single values or streams of elements. Each module implements a computation activated by receiving values from a set of input streams selected in a *non-deterministic* or in a *data-flow* fashion. According to SPP, *intra-module parallelizations are instances of a limited set of parallelism patterns*.

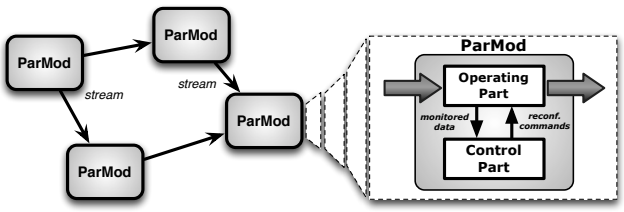


Fig. 1: Computation graphs and internal structure of an Adaptive Parallel Module (ParMod).

The core part of our approach is the concept of *Adaptive Parallel Module* (shortly referred to as ***ParMod***), an independent and active unit featuring a reconfigurable structured parallel computation and an adaptation strategy to maintain the desired QoS in the face of dynamic execution conditions.

4.1 Adaptive Parallel Modules

A ParMod is structured as two cooperating parts (see Figure 1) interconnected in a closed-loop fashion:

- the **Operating Part** executes a parallel computation instantiating a structured parallelism form;
- the **Control Part** represents the ParMod *controller*, an entity able to modify the Operating Part behavior through the execution of reconfiguration activities.

The Operating Part computation can be executed according to different alternative *configurations*. A configuration consists in a specific selection of features such as: *(i)* the current variant in terms of parallelism form and/or sequential algorithm, *(ii)* the actual parallelism degree (number of implementation processes or threads), and, *(iii)* the execution platform on which the computation is currently deployed and executed. Reconfigurations can be distinguished into:

- *functional reconfigurations* replace the parallel implementation of the Operating Part. The possibility to solve the same problem with different parallel variants is a powerful feature of SPP. As stated in Section 3, parallel variants based on distinct paradigms can have different performance, memory usage, and resource consumption. Furthermore, the replacement of the used variant can be executed transparently w.r.t the other parts of the application. This is possible if alternative variants preserve the interfaces of the module, i.e. the interconnections and the types of input/output streams;

– *non-functional reconfigurations* change the values of parameters of the current parallel implementation, e.g. the parallelism degree, and/or the currently used resources. Migration of a parallel computation (or of some of its parts) on different resources is referred to as a *geometric change* in [52]. It does not modify the computation structure (number of implementation processes/threads, their interconnections and support data-structures used for cooperation). Differently, parallelism degree variations are *structural changes* [52] whose cost depends on the characteristics of the parallelism form (e.g. when reconfigurations are executed during the execution flow and if an internal state needs to be collected and re-distributed before/after the reconfiguration).

The implementation of reconfiguration mechanisms and the way in which adaptive applications can be defined is an interesting research aspect. Variants of the same component can be provided by different implementations of the same class in object-oriented frameworks, or by using emerging concepts such as *elastic functions* [56] capturing multiple implementations with meta-information used by the run-time system to choose the best one. In the case of coordination languages for distributed parallel computing, proper programming constructs can be used to provide multiple instantiations of the same component [51, 19, 10]. A discussion about the better way to express such multi-modal nature of adaptive systems is out of the scope of this paper, which explores at a more abstract layer how multiple implementations based on SPP are useful in the context of adaptive parallel applications and predictive control strategies.

Besides triggering reconfigurations, Control Part is in charge of evaluating the adaptation strategy, which is the main focus of this paper. The evaluation is performed periodically, according to two different approaches:

– in ***time-driven controllers*** the adaptation strategy evaluation is performed at equally spaced time instants. We call the time interval between two subsequent decision points as *control step*, i.e. the execution is discretized in control steps of fixed length;
– in ***event-driven controllers*** the occurrence of an event rather than the passing of time states the occurrence of the next decision point. Therefore the adaptation strategy is evaluated at not equally spaced time points.

The main advantage of event-driven controllers is the possibility to reduce the resource utilization of the controller. Nevertheless, as stated in [49] and [25], *"although there are various benefits of using event-driven control like reducing resource utilization (e.g., processor and communication load), their application in practice is hampered by the lack of a system theory for event-driven control systems"*. For this reason in this paper we adopt a time-driven vision, although the integration of our approach with event-driven controllers deserves a future investigation.

The discrete-time closed-loop interaction between Operating Part and Control Part consists in the following information exchange performed at the beginning of each control step:

– *monitoring data*, from Operating Part to Control Part, are measurements that describe the current computation behavior;
– *reconfiguration commands*, from Control Part to Operating Part, are messages triggering the execution of functional and/or non-functional reconfigurations.

In the next subsections we focus in more detail on the MPC strategy and the different elements to instantiate this technique to the ParMod control.

4.2 Application of Model Predictive Control

The application of MPC consists in four elements as shown in Figure 2: *prediction*, *optimization*, *evaluation* and *selection*.

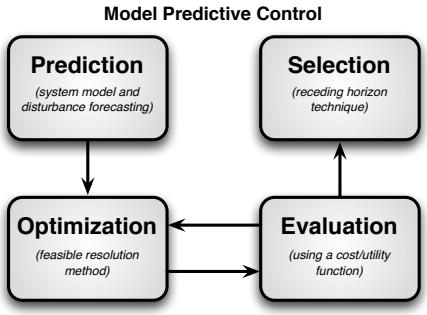


Fig. 2: Elements of Model Predictive Control.

In this section we will present a systematic approach to apply MPC control strategies to adapt the behavior of adaptive parallel modules expressed according to the Structured Parallel Programming paradigms.

4.2.1 ParMod model

The *prediction* element makes it possible to predict the future behavior of the system. To enable that, a pre-condition is the existence of a mathematical model stating the relationship between the following set of variables:

– ***QoS variables*** (denoted by $\mathbf{x}(k) \in \mathbb{R}^n$) represent information that characterizes the quality of the execution at the beginning of control step k;

- ***control inputs*** $\mathbf{u}(k) \in \mathcal{U}$ are the reconfiguration choices for step k, which belong to a discrete set of possibilities $\mathcal{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_v\}$ (e.g. all the feasible combinations of parallelism degrees and existing parallel variants);
- ***disturbance inputs*** ($\mathbf{d}(k) \in \mathbb{R}^m$) model exogenous events affecting the relationship between control inputs and QoS variables. They can be related to the application semantics (e.g. irregular workload, size and complexity of received tasks) or to reasons related to the execution platform (e.g. varying communication latency, or the execution time of sequential parts of parallel computation).

As previously highlighted, a key aspect of ParMods is the presence of multiple alternative configurations:

Definition 1 (Multi-modal behavior of Operating Part). We denote by C the set of alternative Operating Part configurations (*operating modes*):

$$\mathrm{C} = \{C_1, C_2, \ldots, C_v\}$$

Each C_i is represented by the pair $C_i = \langle \mathbf{u}_i, \phi_i \rangle$, where $\mathbf{u}_i$ is the control input vector corresponding to the i-th ParMod configuration, and ϕ_i is a model describing the system behavior when configuration C_i is active.

Accordingly, each operating mode is governed by its own characteristic model:

$$\mathbf{x}(k+1) = \phi_i\big(\mathbf{x}(k), \mathbf{d}(k)\big) \; i = 1, 2, \ldots, v \tag{1}$$

When QoS variables correspond to stateful information, their value at the beginning of the next control step can be expressed as a function of the present value, as in Expression 1. In this case we speak about a *dynamical model* described by a set of difference equations. Contrarily, if future QoS values only depend on the current control inputs and disturbances (as in steady-state performance models), we speak about a *static model* expressible through a set of algebraic equations.

Overall, the future value of QoS variables at step $k+1$ depends on the control inputs decided by Control Part for step k, i.e. which configuration is currently used by the module:

$$\mathbf{x}(k+1) = \Phi\Big(\mathbf{x}(k), \mathbf{d}(k), \mathbf{u}(k)\Big) = \begin{cases} \phi_1\big(\mathbf{x}(k), \mathbf{d}(k)\big) \text{ if } \mathbf{u}(k) = \mathbf{u}_1 \\ \phi_2\big(\mathbf{x}(k), \mathbf{d}(k)\big) \text{ if } \mathbf{u}(k) = \mathbf{u}_2 \\ \vdots \\ \phi_v\big(\mathbf{x}(k), \mathbf{d}(k)\big) \text{ if } \mathbf{u}(k) = \mathbf{u}_v \end{cases}$$

In conclusion, the Operating Part model can be summarized as follows:

Definition 2 (Operating Part Model). The Operating Part model is defined as a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{U}, \Phi \rangle$ where: $\mathcal{X} \subseteq \mathbb{R}^n$ is the space of the possible QoS states; $\mathcal{D} \subseteq \mathbb{R}^m$ is the set of disturbance variables; $\mathcal{U}$ is the finite and discrete set of admissible control inputs. The model consists in a function $\Phi : \mathcal{U} \times \mathcal{X} \times \mathcal{D} \to \mathbb{R}^n$ that maps a discrete-time model $\mathbf{x}(k+1) = \phi_i(\mathbf{x}(k), \mathbf{d}(k))$ onto a configuration $C_i \in C$ corresponding to the actual control input vector $\mathbf{u}(k)$.

Predictability is a key point to use MPC. As described in Section 3, a result of SPP is that each parallelism pattern is composed of a limited set of functionalities (collectors, distributors, workers) with a *precise* behavior: e.g. workers perform the same computation on different data, the distributors/collectors perform a specific policy (scatter, gather, on-demand scheduling). These knowledge is of great help in defining cost models of QoS metrics.

Cost models of SPP play a decisive role in the definition of model Φ. The importance of cost models in *static supports* and *compilers* for structured parallel libraries and languages has been recognized in the past [5,51]. Cost models make it possible to allocate statically the resources needed to run a given program, by determining the best parallelism degree in order to minimize service time or to maximize speedup. Such process can be performed at *compile-time*, or, by exploiting the parametric implementation of parallelism paradigms, at *loading time*. Furthermore, a more extensive use of cost models permits to statically choose the most efficient implementation of run-time support mechanisms (e.g. distribution and collective operations), by evaluating their impact on the underlying architecture. The idea of such a ***static use*** of cost models is that they can be automatically instantiated by recognizing how parallelism patterns have been composed and nested in the application, and by using profiled values for the independent variables of the models.

In the context of our approach, *a static application of cost models is useful at least to establish an initial configuration of parallel components*. However, a high variability and uncertainty of input parameters exists such that a ***dynamic use*** of cost models is necessary to keep updated the application configuration to the actual execution conditions. This is possible by:

- periodically instantiating and re-evaluating the cost models to drive adaptation strategies;
- using current measurements of the execution to obtain up-to-date values of model input variables that enact an effective comparison between alternative configurations.

Conceptually, the use of cost models to drive the adaptation process could be integrated at the *application level*, i.e. the programmer can directly express the cost models and their dynamic evaluation in the application logic. However, to face the complexity of parallel programming, we envision the presence of a high-level programming environment in which the instantiation of cost models, and their dynamic use to drive adaptation, is encapsulated into mechanisms provided by the *run-time support level* (i.e. at a lower level of abstraction). In this way programmer is only requested

to properly use high-level constructs or annotations to express structured parallel variants of ParMods and identify the model variables, while cost models can be automatically derived by the run-time support.

4.2.2 Disturbance forecasting

The current configuration of a parallel module can become ineffective to achieve the desired QoS. This can be due to a change in the environment in which the computation is executed, or when workload and load balancing conditions change w.r.t the initial situation. In cost models this is captured by the presence of uncontrollable *disturbances*.

For the dynamic use of cost models, future disturbance estimation is a crucial point. *Time-series models* are one of the methods to make statistical forecasts of *measurable* disturbances, by using past data to estimate the future behavior along a prediction horizon. They are based on the assumption that disturbance values are autocorrelated and characterized by non-stationarities such as *trends* and *seasonal variations*. A generic prediction *filter* is defined as follows:

$$\mathbf{d}\big(k+l|k\big) = \Psi\Big(\mathbf{d}(k-s),\ldots,\mathbf{d}(k),\{\alpha_1,\ldots,\alpha_t\}\Big) \tag{2}$$

where $\mathbf{d}(k+l|k)$ denotes the predicted disturbances for step $k+l$ using the knowledge available at step k, with $l=1,\ldots,h$ and h the length of the horizon. We use the last $s+1$ past values to predict future values of the process; $\{\alpha_i\}$ denotes a set of fixed or automatically tunable parameters of the filter (e.g. smoothing factors and gains).

In Section 5, devoted to experiments, we will exemplify the use of time-series filters.

4.2.3 Optimization and receding horizon

The second element of MPC is the definition and online resolution of an optimization problem over a prediction horizon of h control steps, constrained by the modeled system dynamics:

$$\min J(k) = \sum_{i=k}^{k+h-1} L\big(\mathbf{x}(i+i),\mathbf{u}(i)\big) \tag{3}$$

such that:

$$\mathbf{x}(i+1) = \Phi\big(\mathbf{x}(i),\mathbf{d}(i),\mathbf{u}(i)\big) \ \text{ for } i=k,\ldots,k+h-1$$

$$\mathbf{u}(i) \in \mathcal{U} \ \text{ for } i=k,\ldots,k+h-1$$

L is a step-wise *cost* notion accounting for the future values of QoS variables and the sequence of control inputs used along the prediction horizon. The solution of the optimization problem is a *trajectory* (sequence) of control inputs, $\overline{U}(k) = \{\mathbf{u}(k),\mathbf{u}(k+1),\ldots,\mathbf{u}(k+h-1)\}$, such that J is minimized based on given disturbance predictions. The cost function is used for the *evaluation* element of MPC (see Figure 2), in which alternative control trajectories are compared and evaluated according to the used cost metrics.

The *selection* element of MPC combines the advantages of a long-term planning with the advantages of feedback control. The optimization problem is solved at each control step. Instead of applying the optimal control trajectory in an open-loop fashion, only the first control input (reconfiguration) of the sequence is really applied to the Operating Part. The optimization problem is then re-evaluated at the beginning of the next control step using the new observations from the system (disturbances and QoS variables). The effect is that the prediction horizon is displaced towards the future; this explains the reason because MPC is often referred to as *receding horizon control* in the literature [20,47].

The *optimization* element deserves a thoroughly discussion. To be implementable, MPC requires to complete the optimization process within the temporal constraints dictated by the control step of the system. Therefore, *computational efficiency* is a critical issue. In the case of ParMod control this aspect is even more important, since the combinatorial optimization problem theoretically implies an exhaustive search by testing all the possible feasible combinations of reconfiguration decisions. This process can be represented as a tree structure (named *Evolution Tree* as in see [45]) representing the ParMod behavior over a h-step horizon (see Figure 3).

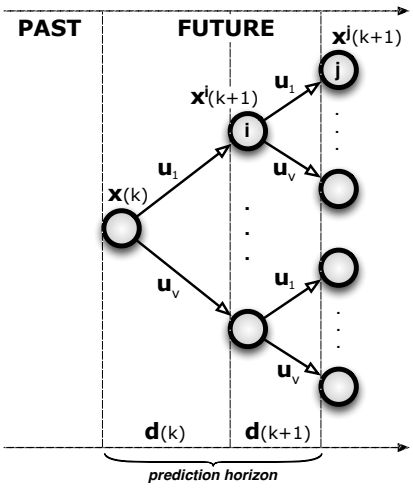


Fig. 3: Example of ParMod Evolution Tree with a prediction horizon of two control steps, $h=2$.

Definition 3 (Evolution Tree). We denote by $\mathcal{T}^k$ the evolution tree at step k characterized by the following features:

- $\mathcal{T}^k$ is a *full tree*, i.e. every node is a leaf or has exactly ν children, where ν corresponds to the number of ParMod configurations;
- the *height* of $\mathcal{T}^k$ is equal to the length h of the prediction horizon;

– each level $l = 0, 1, \ldots, h-1$ of the tree is assigned to a disturbance input $\mathbf{d}(k+l)$;
– each node i at level $l = 0, 1, \ldots, h$ corresponds to a reachable QoS state $\mathbf{x}^i(k+l)$, where $i \in \{1, 2, \ldots\}$ is an index marking reachable states;
– each arc e is associated with an admissible discrete control input $\mathbf{u}^e \in \mathcal{U}$ causing the transition;
– an arc e labeled with $\mathbf{u}^e$ connects a node i at level l with a child j at level $l+1$ *iff* the QoS state $\mathbf{x}^j(k+l+1)$ is directly reachable from state $\mathbf{x}^i(k+l)$ applying control input $\mathbf{u}^e$, i.e. $\mathbf{x}^j(k+l+1) = \Phi\left(\mathbf{x}^i(k+l), \mathbf{d}(k+l), \mathbf{u}(k+l)\right)$ where $\mathbf{u}(k+l) = \mathbf{u}^e$.

The number of tree nodes grows exponentially with the length of the prediction horizon. Therefore, the optimization problem is computationally prohibitive except for short horizons and with a small number of control choices. Interesting and general approaches to improve the feasibility of predictive control consist in using *Branch and Bound* (B&B) methods [21,45].

Branching is done by following one of the possible v alternative control decisions at each node of the tree, which generate v distinct sub-spaces. *Bounding* and *pruning* are done by checking specific criteria in order to decide whether a branch needs to be examined or not. This operation makes it possible to discard a sub-tree rooted at a specific node with the certainty (or the reasonable confidence) that the optimal solution can not be found exploring that sub-space. A general procedure, depicted in Figure 4, can be defined as follows:

– we assign to each node i of the tree a variable C^i that represents the total cost (according to cost function J) spent to reach that node from the root. The cost of the root is initialized to zero;
– for each node i we consider v potential branches. The branch corresponding to a control input $\mathbf{u}$ is followed *iff* the following inequality holds:

  $C^i + L_w(j, \mathbf{u}) < C^{max}$

  $L_w(j, \mathbf{u})$ is a lower bound for the cumulative cost of all the paths starting from node j and reaching a leaf of the tree, where j is the node directly reached from i after applying control input $\mathbf{u}$, and C^{max} is an upper bound to the cost of the optimal solution. *If the inequality does not hold, the sub-tree rooted at j can be discarded.*

We will give a precise description of B&B techniques in Section 5.2, when a specific scheme is instantiated to enable the feasible execution of MPC strategy.

4.2.4 Summary of the approach

MPC can be applied to control ParMods by applying the following sequence of phases depicted in Figure 5:

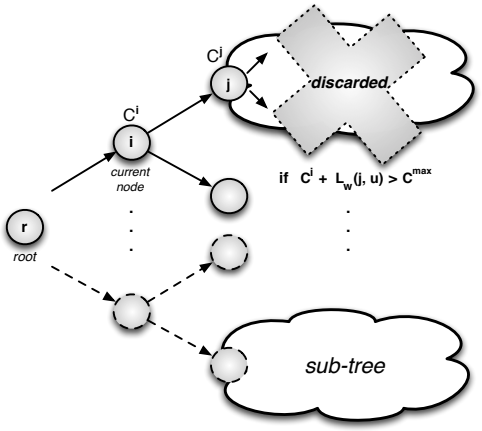


Fig. 4: Branch & Bound approach to the Evolution Tree exploration.

1. ParMods must be provided in multiple structured parallel variants. Each variant is described by a corresponding cost model;
2. the future evolution of QoS variables is predicted by a model exploiting cost models of the parallel variants;
3. the control goals and their trade-off is described in terms of an optimization problem constrained by the dynamics captured by the model;
4. measured disturbances must analyzed in typical working conditions, in order to tune a set of statistical predictive tools, e.g. auto-regressive moving-average filters, neural networks or Kalman filters;
5. the adaptation strategy needs to be evaluated during the system execution taking corrective actions if it is not sufficiently effective to achieve the desired goals (e.g. by modifying model parameters or changing the formulation of the optimization problem).

This approach should be considered a *research methodology* to apply MPC. The first two points inherit from past researches on run-time supports and programming models for SPP [3,32,35,14,6,5]. The *design methodology* of SPP is sufficiently mature to enable the *automatic* derivation of static and dynamic implementations of parallelism paradigms and the instantiation of their cost models [58]. On the other hand the other elements of our approach, mainly the definition of the optimization problem and the use and tuning of statistical filters, are phases still needing an important *human-intervention*, and deserve future investigations to flexibly provide them in new programming models and tools with the sufficient degree of abstraction.

5 Experimental case-studies

This section is aimed at providing an exemplification of our approach by presenting two case-studies of real-world

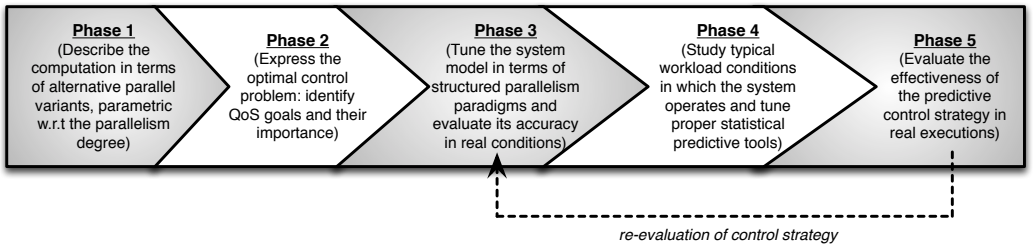


Fig. 5: Phases of the approach: apply MPC to parallel components expressed using multiple structured parallel variants and their cost models.

distributed parallel applications needing adaptive behavior. The examples are useful to understand how MPC can be applied to parallel computations, and the effectiveness of this techniques in terms of control optimality and stability, which, as stated in Section 1, are important properties of adaptation strategies.

5.1 First scenario: an Emergency Management System

The first case-study represents the core component of an *Emergency Management System* [19,8,13] (EMS). EMSs support civil protection during the management of natural or man-made disasters (e.g. floods and earthquakes) through the online and real-time computation of short-term forecasts and the execution of decision support models. We consider a simplified view of the system for detecting flood emergencies studied in [11,10,9,12,39]. The application is composed of three classes of components:

- a set of *Generator* modules receive data from sensors deployed along and around an observed environmental scenario (e.g. a river basin), with the aim at monitoring measurements such as punctual precipitation, water depth and speed;
- a flood forecasting *ParMod* (namely *Solver*) takes environmental data from Generators and returns short-term forecasts for a specific requested area;
- results are disseminated to a set of *Clients* components, applying the results to further phases such as the temporal-spatial analysis and automatic tools for decision support.

The Solver component adopts a bi-dimensional hydrodynamic model [15] solved using a finite difference method. The resolution consists in a set of tri-diagonal linear systems for each point of the environment discretization. A parallelization of this problem operating on streams (described in more details in [10]), can be done by a *task-farm* structured parallelism pattern, in which each new system is scheduled to an available worker. In this application we can identify two distinct control goals:

1. *maximize throughput*: in order to reduce the completion time to perform the flood simulation, the Solver ParMod must be able to process and calculate the highest number of tasks as possible;
2. *minimize operating cost*: we suppose that the Solver ParMod is executed on a Cloud provider offering cluster-on-demand/ HPCaaS services. Customers pay a cost for each reserved computing unit and for the effective utilization time [54,59]. Additional units can be elastically allocated/deallocated to accommodate users' requests.

The fulfillment of such goals requires to adapt the Solver configuration in terms of used computing resources (i.e. non-functional reconfigurations - *dynamic modifications of the parallelism degree*).

5.1.1 Model definition and predictive control

A meaningful parameter for the Solver performance is the current length of its input task queue. We model the future evolution of the queue length using a *dynamic model*. The next queue length is expressed as a function of its current value $\mathcal{Q}(k)$ (QoS variable), mean inter-arrival time $T_A(k)$ (disturbance), and mean service time $T_S(k)$. The service time is expressed as a function of the current parallelism degree $n(k)$ of the Solver (control input):

$$\mathcal{Q}(k+1) = \max\left\{0,\ \mathcal{Q}(k) + \left(\frac{\mathrm{T}(k)}{T_A(k)} - \frac{\mathrm{T}(k)}{T_S(k)}\right)\right\} \tag{4}$$

$\mathrm{T}(k)$, as it will be discussed in the sequel, represents the time length of effective working phase during the last control step k. The next queue length is given by the last queue length plus the difference between the number of arrivals and the number of served requests during the last control step. This concept is schematized in Figure 6.

The ideal service time $T_S(k)$ can be determined by applying the *task-farm performance model*, expressed as a function of the parallelism degree $n(k)$:

$$T_S(k) = \max\left\{T_{distr}, \frac{T_{worker}}{n(k)}, T_{coll}\right\} \tag{5}$$

where T_{distr} and T_{coll} are the mean distribution (scheduling) and collecting times, and T_{worker} is the average time to apply

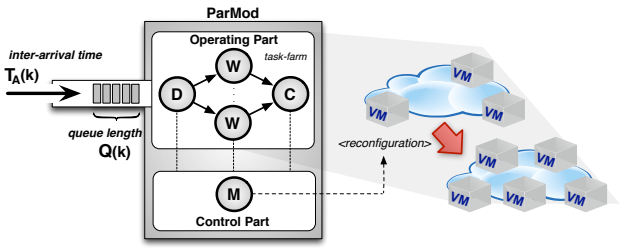


Fig. 6: The Operating Part of the Solver ParMod performs a task-farm pattern, with a distributor D, a collector C and a dynamically reconfigurable set of workers W.

the whole computation on a single stream element. Such parameters can be dynamically profiled during the system execution and are considered as *constant* terms in this application (they model random variables with negligible variance). As we can observe, by increasing the number of workers this parallelism form is able to reduce the service time as long as the distributor or the collector functionalities become a bottleneck, i.e. the task-farm service time is the maximum between the distribution and collection times and the overall service time of the set of $n(k)$ workers.

Reconfigurations may cause significant modifications in the used infrastructure by frequently changing the configuration in terms of virtual machines (VMs) and, consequently, number of real processing nodes. During phases in which the application can sustain the current workload using fewer resources, it is economically useful to shrink the set of used VMs (if the computation is deployed on several VMs hosting pools of task-farm workers), or by migrating it on a VM configured with a smaller set of resources (e.g. with fewer cores per CPU). In both the cases, creating and shutting down VMs may take tens of seconds up to several minutes to complete [59,50], and the computation could be blocked waiting for the reconfiguration process to complete [52]. Such *reconfiguration delay* has been similarly considered in past works such as in [53], in which a provisioning model of virtual appliances has been proposed for the dynamic scaling of virtualized data centers.

To capture this effect, $\mathrm{T}(k)$ represents the portion of the control step length during which the ParMod is currently processing input tasks. We model it in the following way:

$$\mathrm{T}(k) = \begin{cases} \tau \text{ if } n(k) = n(k-1) \\ \tau - T_{deploy} \text{ otherwise} \end{cases}$$

where τ is the control step length, and T_{deploy} is equal to the average time-to-deploy of virtual machines on the used Cloud infrastructure.

In the MPC strategy we consider a prediction horizon of $h \geq 1$ control steps. At the beginning of the current step k, the Control Part monitors: (1) the actual number of received tasks that are waiting to be scheduled; (2) the mean inter-arrival time experienced during the last control step. The operating cost can be defined as follows:

$$\mathcal{C}(k) = C_{node}\, n(k) + C_{fix}\, \Delta_N(k) \tag{6}$$

This definition captures several aspects of existing billing models used in Cloud environments [59]. Resources are provisioned and reserved on-demand to the users by a service provider for how long they are needed. In order to discourage too many resource re-organizations, providers can apply fixed costs to each reconfiguration. For this reason in Expression 6 we account for two cost terms: C_{node} and C_{fix}. $\Delta_N(k)$ denotes the presence of a reconfiguration at control step k and it is defined as follows:

$$\Delta_N(k) = \begin{cases} 1 \text{ if } n(k) \neq n(k-1) \\ 0 \text{ otherwise} \end{cases}$$

By applying statistical predictions of the future inter-arrival time along the prediction horizon, Control Part finds the optimal trajectory of reconfigurations that minimizes the cost function defined below:

$$J(k) = \sum_{i=k+1}^{k+h} \Big[w_1\, \mathcal{Q}(i) + w_2\, \mathcal{C}(i-1) \Big] \tag{7}$$

where w_1 and w_2 are two positive weight coefficients. By assuming $w_1 >> w_2$, we are interested in a control trajectory maximizing the number of completed tasks at the end of the prediction horizon (i.e. by keeping the input queue as empty as possible). If multiple trajectories exist that satisfy this goal, Control Part selects the sequence with the minimum operating cost. Finally, according to the receding horizon principle, only the first element of the optimal trajectory is applied while the rest is discarded.

From a qualitative viewpoint, MPC may give an effective outcome in the the following cases:

- if a persistent drop in the mean inter-arrival time is predicted, the control strategy can modify in advance the ParMod configuration in order to minimize the number of reconfigurations and adapt the parallelism degree directly to the optimal value;
- if the inter-arrival time is expected to be higher in the future, Control Part may decide to release a proper amount of computing nodes. If we are able to estimate how long this condition holds, the Control Part can evaluate if the release of a certain set of resources is effectively useful (e.g. avoiding to re-acquire them nearly in the future);
- in the case of a temporary fluctuation in the mean inter-arrival time, the Solver ParMod can avoid to acquire or release computing resources repeatedly, avoiding to make the operating cost higher without a real outcome from the performance standpoint.

5.1.2 Estimating disturbances through statistical forecasting

EMSs are executed on geographically distributed environments characterized by interconnection networks whose availability and reliability is extremely varying. We execute the application on a test-bed platform composed of two workstations (for the Generator and Client) and an AMD magnycours architecture featuring 24 homogeneous cores for the Solver ParMod. In the experiments we do not use a real Cloud platform, but we reproduce the relevant aspects of the model described in the previous section, notably reasonable reconfiguration delays and a realistic network behavior.

In our test-bed architecture, the information exchange between Generator and Solver is performed through a TCP connection over a network characterized by a time-varying available bandwidth, packet loss probability and network latency. The average time needed to transfer the data-structure representing a task (from 4 to 32 MB depending on the space discretization [19,10] used by the application) is influenced by the behavior of the network resources along the path through the application components.

To simulate dynamic network conditions, we use NCTUNS [40], a network emulator/simulator which allows the integration of a simulated environment with real hosts running application components (see Figure 7). NCTUNS is executed on a workstation where we simulate a network topology composed of two routers and a *WAN object* reproducing wide-area network delays and packet loss probability.

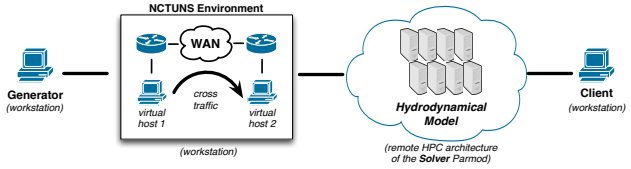


Fig. 7: Test-bed platform of the first experiment.

In the simulation environment we execute two virtual hosts generating TCP/UDP traffic. To this purpose we use the D-ITG [18] traffic generator able to reproduce phenomena such as *level shifts* (sudden changes in the network load), and *trends* corresponding to time periods in which the network is under-loaded or, alternatively, progressively congested. An example of trace-file of the inter-arrival time of tasks to the Solver ParMod is depicted Figure 8, sampling the results using time intervals (control steps) of 240 seconds.

To predict future values of the inter-arrival time, we use the *Holt-Winters* time-series filter able to capture trend non-stationarities on the underlying time-series. This filter has been already applied in [24] to predict the throughput of TCP connections. We denote with $\hat{T}_A(k)$ the predicted mean inter-arrival time at control step k. The Holt-Winters predictor is composed of two exponential (EWMA) filters for the smooth (mean level) and the trend components. The predicted value at step $k+h$ with $h \geq 1$ is calculated as follows:

$$\hat{T}_A(k+h) = T_A^s(k) + h\,T_A^t(k) \tag{8a}$$

$$T_A^s(k) = a\,T_A(k) + (1-a)\,\left(T_A^s(k-1) + T_A^t(k-1)\right) \tag{8b}$$

$$T_A^t(k) = b\left(T_A^s(k) - T_A^s(k-1)\right) + (1-b)\,T_A^t(k-1) \tag{8c}$$

where Expression 8b describes the smoothing component and 8c the trend one. Expression 8a makes h-step ahead predictions by extending the time-series into the future w.r.t the trend. Parameters a and b are the smoothing factors and range between zero to one. The best values are calculated using a fitting initial period of observations by minimizing the sum of the squared one-step ahead forecast errors.

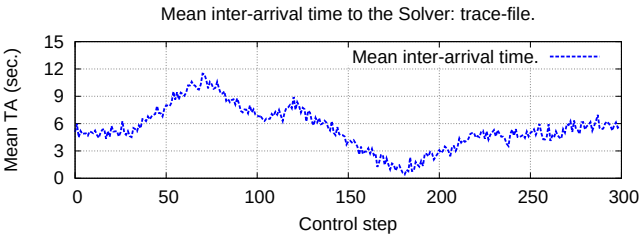


Fig. 8: Time-series of the mean inter-arrival time to the Solver ParMod (using tasks of 32 MB).

We apply the filter to the time-series of Figure 8. We are interested in evaluating the percentage error between the real and the predicted trajectories at each control step. To this end, we calculate the *Mean Absolute Percentage Error* (MAPE) over the entire execution. Table 1 reports the numerical values of the errors. As we can observe the increase is limited (lower than 5% with 4-step ahead predictions).

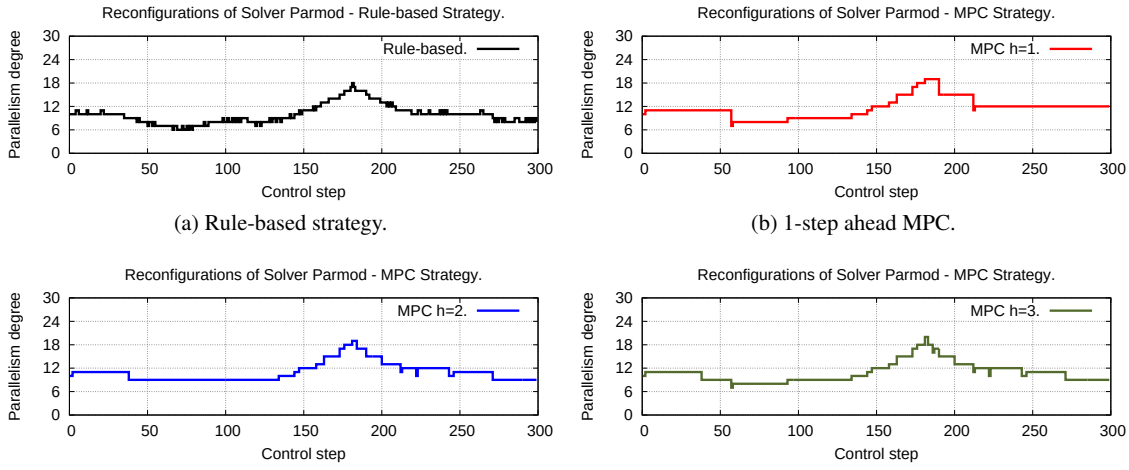
	Horizon 1	**Horizon 2**	**Horizon 3**	**Horizon 4**
MAPE	3.56%	3.81%	4.01%	4.23%

Table 1: Global MAPE over the entire execution.

For this example the Holt-Winters filter is sufficiently accurate to show the effectiveness of the predictive control strategy.

5.1.3 Results and comparison with a heuristic strategy

The application has been implemented using the MPI message-passing support. The Solver Operating Part is composed of an emitter, a collector and a dynamic set of worker



(a) Rule-based strategy.

(b) 1-step ahead MPC.

(c) 2-step ahead MPC.

(d) 3-step ahead MPC.

Fig. 9: Reconfiguration sequences using different adaptation strategies and lengths of the prediction horizon.

processes communicating through MPI send/receive primitives. The emitter is responsible for scheduling each received task to an available worker by performing an on-demand distribution. The processes of the Operating Part periodically transmit to a *manager* process in the Control Part (denoted by M in Figure 6) measurements of the current computation behavior (e.g. last sampled inter-arrival time and queue length). Reconfigurations are triggered by the manager by instantiating/removing workers using the MPI library function MPI_COMM_SPAWN. In order to emulate a realistic time-of-deploy of Cloud environments, we apply a variable delay after each reconfiguration, modeled as a normally distributed random variable with mean equal to 10% the control step length (equal to 24 seconds), which represents a realistic delay of modern Cloud architectures featuring pre-defined memory templates of virtual machines and caching systems [50].

In order to demonstrate the effectiveness of MPC, we compare the results with two strategies:

- a MAX configuration, in which we fix the parallelism degree to the maximum value throughout the execution;
- a *rule-based strategy* already applied to the control of parallel computations in [55,2]. Figure 10 shows two condition-action rules to adjust the parallelism degree of the Solver ParMod.

The rule-based strategy observes the current *utilization factor* $\rho(k)$ calculated as the ratio between the ideal service time and the last sampled inter-arrival time. The smaller the utilization factor is the more the parallelism degree is oversized, resulting in a waste of computing resources. In constrast, a high utilization factor corresponds to situations in which the Solver is a performance bottleneck. We apply a reconfiguration if the utilization factor exceeds two thresholds T_{min} and T_{max}.

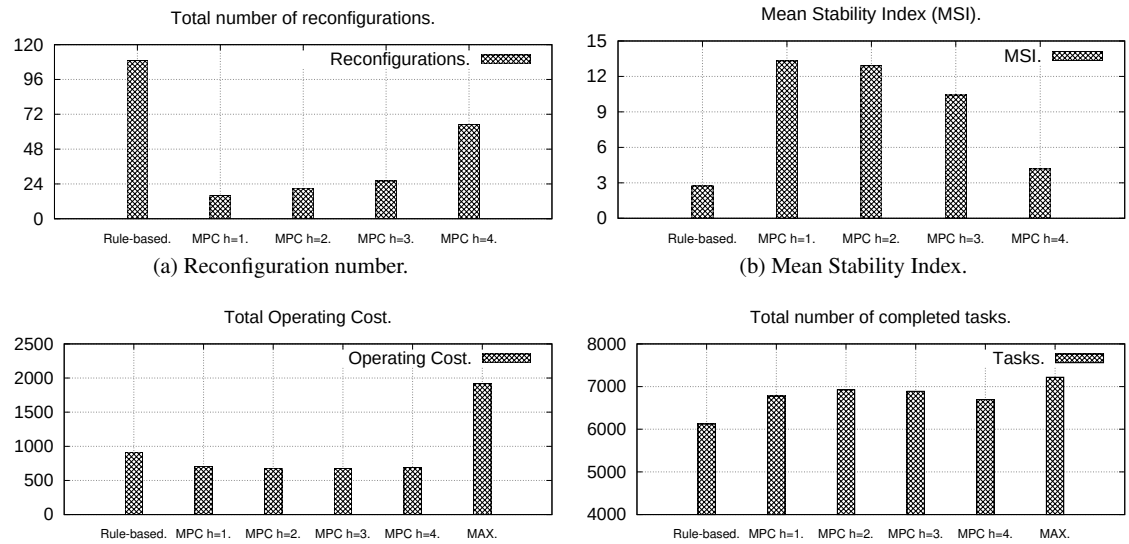
```
if (ρ(k-1) > Tmax) then n(k) := min (MAX_DEGREE, n(k-1) + 1);

if (ρ(k-1) < Tmin) then n(k) := max (1, n(k-1) - 1);
```

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

Figure 9 shows the reconfiguration sequence using the MPC strategy with three prediction horizons and the rule-based strategy (with $T_{min} = 0.95$ and $T_{max} = 1.05$). As we can observe, the rule-based strategy accurately follows the inter-arrival time time-series. It performs a high number of reconfigurations influenced by the variance of the time-series. The MPC strategy produces a significant stabilization by reducing the amount of reconfigurations. This effect is due to the exploitation of smooth predictions of the inter-arrival time and by accounting for the reconfiguration delay associated with parallelism degree variations (captured by the Operating Part model, as described in Section 5.1.1).

Figure 11a shows the number of reconfigurations. As we can observe, by increasing the controller foresight (up to a horizon of 4 steps) reconfigurations are taken in advance. This fact, coupled with the exploitation of linear trend predictions using the Holt-Winters filter, tends to produce a higher number reconfigurations with longer prediction horizons. However, *with every horizon length the MPC strategy performs a smaller number of reconfigurations compared with the rule-based strategy* (in the worst case we achieve a reduction of 40%). This aspect, near to the concept of *reconfiguration stability* introduced in Section 1, can be formalized by introducing the following metric:

Definition 4 (MSI). The *Mean Stability Index* (MSI) of an adaptation strategy is the average number of control steps between successive reconfigurations performed by the strategy.



(a) Reconfiguration number.

(b) Mean Stability Index.

(c) Operating Cost ($C_{node} = 0.2$ and $C_{fix} = 0.8$).

(d) Completed tasks.

Fig. 11: Comparison between adaptation strategies (MPC with different horizons and the rule-based strategy) and the static MAX configuration.

Figure 11b shows the `MSI` achieved by different adaptation strategies. As expected, the best `MSI` corresponds to the strategy achieving fewer reconfigurations (i.e. 1-step ahead MPC). In this case a reconfiguration is applied every 13 steps in the average case (5 times greater than the rule-based approach). In terms of control optimality, we use a fixed cost C_{fix} four times greater than C_{node}. The total operating cost is defined as the sum of the step-wise cost $\mathcal{C}(k)$ over the entire execution. As Figure 11c outlines, MPC strategies are able to reduce the total cost of even 65% compared with the `MAX` strategy, and they produce a further improvement w.r.t the rule-based approach of 27% with a prediction horizon of 3 steps (the best strategy in terms of cost minimization).

In terms of completed tasks (Figure 11d), the best result is achieved using a 2-step ahead horizon. In this case we are able to complete the highest number of tasks (12% greater than using the rule-based strategy). The performance loss w.r.t the static `MAX` configuration is only of 4%, but as we have seen with a significant reduction of operating cost.

We conclude this example by considering the computational burden of MPC. In this experiment the best horizon length in terms of performance consists in using 2-step ahead predictions. The resolution of the 2-step ahead MPC problem consists in exploring an evolution tree composed of 601 states. In our test-bed platform the optimization process has a negligible completion time compared with the control step length (less than 1 millisecond). This consideration is not valid anymore if we use architectures with higher parallelism degrees or if we use longer prediction horizons. In that case search-space reduction techniques must be applied. This aspect will emerge in the second case-study.

5.2 Second scenario: a Video Surveillance System

The second case-study consists in a *video surveillance system* based on a client-server architecture [31,48]. The application scheme is depicted in Figure 12. The application is composed of a set of localized cameras which can automatically change frame resolution and frame rate when motion is detected in the monitored locations. *Client* components receive video feeds from cameras and transmit them to a *Server* component which applies noise-elimination filters, feature extraction techniques and object classification algorithms. In this application we consider two control goals:

1. *maintaining acceptable levels of response time*: surveillance systems process large sequences of data in real-time, applying time-consuming algorithms in order to identify dangerous events and send alert messages to users. For this reason the Server configuration needs to be adapted to minimize the number of *QoS violations*, i.e. when the response time exceeds thresholds established by the users;
2. *minimize reconfigurations*: in order limit system failures or malfunctions, it is requested to minimize the number of reconfigurations performed by the Server.

For the sake of simplicity, in this example we limit the Server activity to the application of noise-elimination filters, which represent the preliminary phase of more complex image processing algorithms. We consider two algorithms: *(i)* a low-pass linear *mean filter*, which consists in replacing each image pixel with the weighted average value of all the neighbors in a square window surrounding the target pixel; *(ii)* a *median filter*, which replaces each pixel with the median of its neighboring entries.

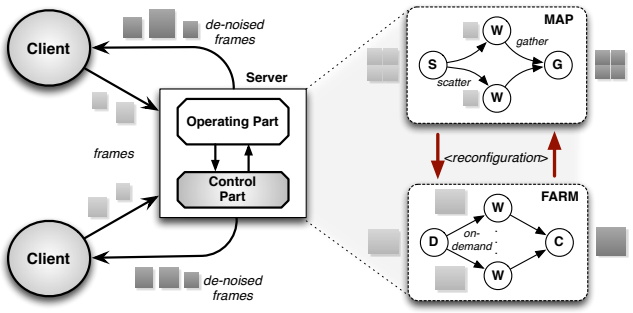


Fig. 12: Client-Server graph of the second case-study.

Although the median filter is preferable in general (it is able to preserve image edges while removing noise), the Server component can switch to the simplest mean filter whenever this choice is essential to avoid QoS violations. We consider two parallel variants:

- a ***task-farm*** parallelization of the mean filter, in which each image is scheduled by a distributor process to an available worker that performs the filtering on the whole image. Results are collected by a collector process;
- a ***map*** (data-parallel) parallelization of the median filter. The current frame is sliced by a scatter process into sub-parts (with small overlapping regions for borders), which are independently calculated by a set of worker processes. The filtered image is reconstructed by a gather process which transmits it to the requesting Client.

In this application reconfigurations belong to the two classes described in Section 4.1: non-functional reconfigurations of the Server parallelism degree, and functional reconfigurations of the used parallel variant.

5.2.1 Modeling the response time using structured parallelism paradigms

In this section we show how to apply cost models of structured parallelism patterns when modeling the response time in computation graphs exhibiting request-reply interactions. We apply a *static steady-state queueing model* described by the following system of equations:

$$\begin{cases} T_C = T_G + R_Q \\ T_A = \dfrac{T_C}{N_{client}} \\ \rho_s = \dfrac{T_S}{T_A} \\ R_Q = W_Q(\rho_s, T_S, T_A) + L_S \end{cases} \tag{9}$$

Let T_S be the Server service time and T_C the Client effective service time (we assume all clients identical), calculated as the sum between T_G (the Client ideal service time) and the Server mean response time R_Q. The second and the third equation express the inter-arrival time T_A to the Server and its utilization factor ρ_s. The last equation defines R_Q as a function of the *mean waiting time* W_Q and the mean latency of the Server L_S. The solution of the system is subject to the constraint $\rho_s < 1$, since at steady-state the Server service time can not be greater than its inter-arrival time.

This model accounts for the nature of different parallel variants of the Server. R_Q depends on the server service time and latency. Independently from the used parallelism degree, the task-farm variant is not able to improve latency, since each worker applies the computation on each received task sequentially. Contrarily, the map variant, based on data partitioning, has effects both on the service time and the latency aspects of the Server performance.

We identify the following meaningful variables of the Operating Part model: *(i)* a QoS variable $R_Q(k)$ represents the mean response time of the Server during the last control step $k-1$; *(ii)* $M(k)$ models a disturbance identifying the average size of video frames received during control step k; *(iii)* the control variables are $n(k) \in [1, n_i^{max}]$ for the parallelism degree, and a binary value $op(k) \in \{0,1\}$ for the parallel variant (conventionally 0 for the task-farm and 1 for the map). The cost models are shown in Table 2.

In Table 2, the mean calculation time $T_{calc}(k)$ is given by the frame resolution $M(k)$ (in pixel) multiplied by the per-pixel cost for applying the two filters (T_H and T_F respectively). It is worth noting that by increasing the parallelism degree we are able to improve the service time as long as the scheduling (for the task-farm) or the scattering/gathering functionalities (for the map) become a bottleneck. This is captured by the *max* in the service time expressions of Table 2. The scattering/gathering phases are implemented as a linear sequence of send/receive operations to/from workers. The term L_{com} corresponds to the *communication latency* for transmitting/receiving images (or their parts). This parameter is modeled as a linear function of the message size σ:

$$L_{com}(\sigma) = t_{startup} + \sigma \cdot t_{transm} \tag{10}$$

As shown in several examples in the literature [57], if $t_{startup}$ and t_{transm} are properly estimated in typical working conditions of the network, this model gives reasonably accurate approximations particularly for sufficiently large messages.

The response time is influenced by the mean waiting time W_Q in the Server queue. By using Queueing Theory results, we have many degrees of freedom associated with the queue size, the population type, and the probability distribution that fits better with the real behavior of the system. In this example we use a `M/M/1` queueing model which assumes exponential distributions for the service time and the inter-arrival time. As it is known, for this queue type the waiting time can be estimated as follows:

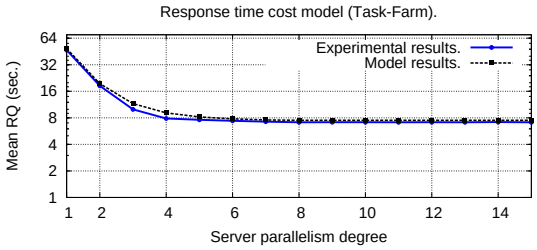
$$W_Q(k) = \frac{T_S(k)^2}{T_A(k) - T_S(k)} \tag{11}$$

Task-Farm ($op(k)=0$)	**MAP - Data-Parallel** ($op(k)=1$)
$Tcalc(k) = T_H \cdot M(k)$	$Tcalc(k) = T_F \cdot M(k)$
$T_S(k) \simeq \max\left\{2L_{com}\big(M(k)\big), \dfrac{Tcalc(k)}{n(k)}\right\}$	$T_S(k) \simeq \max\left\{L_{com}\big(M(k)\big) + T_{scatter}(k), \dfrac{Tcalc(k)}{n(k)}\right\}$
$L_S(k) \simeq 4L_{com}\big(M(k)\big) + T_{calc}(k)$	$L_S(k) \simeq 2L_{com}\big(M(k)\big) + 2T_{scatter}(k) + \dfrac{T_{calc}(k)}{n(k)}$
	$T_{scatter}(k) = n(k) \cdot L_{com}\left(\dfrac{M(k)}{n(k)}\right)$

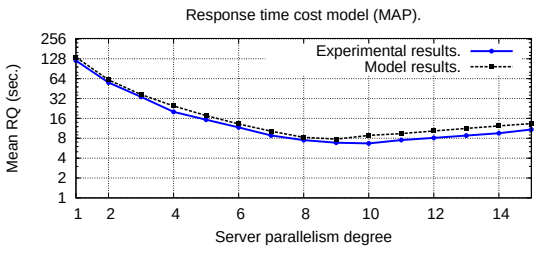
Table 2: Performance models of the two variants of the Server Operating Part.

With this definition, the system of equations admits one and only one positive solution satisfying $\rho_s < 1$. We use this solution to estimate the mean response time, i.e. $R_Q(k+1)$.

We have performed a preliminary experiment on a platform composed of 10 Clients interconnected to a homogeneous cluster of production workstations hosting the Server execution. The run-time support of the Operating Part (executing the task-farm or the map parallelism form) has been implemented using the `MPI` library. In the experiment we use `FullHD` frames (~ 5.93 MB per frame). The mean frame rate from each Client (the inverse of T_G) is equal to 30 frames per seconds. The parameters of the communication cost model have been estimated through a linear regression. Figure 13 shows the experimental results using up to 15 cluster nodes.



(a) Task-Farm variant.



(b) MAP (Data-Parallel) variant.

Fig. 13: Response time using different parallelism variants.

In the task-farm, by increasing the parallelism degree we observe a monotonically decreasing behavior of the response time (Figure 13a). For parallelism degrees greater than the number of Clients, each new task (frame) can be immediately scheduled to an available worker. In constrast, the map variant is characterized by a non-monotonic behavior (Figure 13b). With high parallelism degrees the scattering/-gathering phases may become a bottleneck that negatively influences the Server service time and the waiting time in the input queue. With a parallelism degree greater than 9 in the model (10 in the experiment), the negative effect on the service time dominates the latency reduction giving worse response times. With the `M/M/1` model we achieve slightly over-estimated results (the mean relative error is $\sim 10\%$). A better precision can be obtained by adopting other queueing models: e.g. using different distributions or considering finite populations.

5.2.2 Application of Model Predictive Control

In this section we study the application of MPC to adapt the configuration of the Server ParMod. We introduce two variables defined as a function of response time and control inputs: $P(k)$ and $\Delta_U(k)$. The former is a measure of how much the response time constraint is satisfied. To provide a real-time execution of this application, the response time needs to be maintained within a desired region established with the users. If the mean response time during step k is greater than $R_{Q_{max}}$ or less than $R_{Q_{min}}$ (two user-defined maximum and minimum thresholds), $P(k)$ is equal to the corresponding relative error between the actual value and the nearest threshold. Otherwise, it is zero for any value within the desired region:

$$P(k) = \begin{cases} 0 \quad \text{if } R_{Q_{min}} \leq R_Q(k) \leq R_{Q_{max}} \\ \dfrac{R_Q(k) - R_{Q_{max}}}{R_{Q_{max}}} \quad R_Q(k) > R_{Q_{max}} \\ \dfrac{R_{Q_{min}} - R_Q(k)}{R_{Q_{min}}} \quad R_Q(k) < R_{Q_{min}} \end{cases}$$

The second variable $\Delta_U(k)$ indicates the presence of a reconfiguration at control step k, where the control input vector is composed of two elements for the parallelism degree and the parallel variant, i.e. $\mathbf{u}(k) = [n(k), op(k)]^T$:

$$\Delta_U(k) = \begin{cases} 0 & \text{if } \mathbf{u}(k) = \mathbf{u}(k-1) \\ 1 & \text{otherwise} \end{cases}$$

The optimal control problem consists in the minimization of the following cost function, defined over a prediction horizon of $h \geq 1$ control steps:

$$J(k) = w_1 \cdot \sum_{i=k+1}^{k+h} P(i) + w_2 \cdot \sum_{i=k}^{k+h-1} \Delta u(k) \tag{12}$$

where w_1 and w_2 are two positive weights. If $w_1 >> w_2$, the Control Part explores the horizon in order to find the best sequence of reconfigurations to respect the response time constraint. If more than one reconfiguration sequence satisfies the given thresholds, the controller selects the trajectory with the minimum number of control input changes.

5.2.3 Dynamic workload and statistical forecasting

Automated or human users can dynamically variate the frame resolution in order to analyze smaller details with a higher degree of accuracy (e.g. when motion is detected). Variable resolutions correspond to a different computational burden to the Server that needs to select the most effective configuration to respect the QoS constraints.

In this example we study a scenario in which the frame size can be modeled as a non-stationary process exhibiting seasonal patterns [48] (e.g. periodical execution phases of different length and amplitude characterized by a higher frame resolution). Figure 14 shows an example of time-series composed of 300 steps each one of 240 seconds.

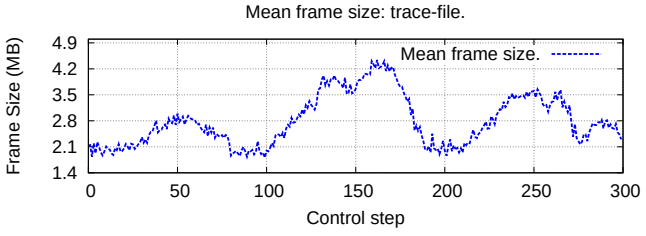


Fig. 14: Time-series of the mean frame size.

Such kind of seasonal workload can be predicted using an improved definition of the Holt-Winters filter (the details about this method can be found in [16]). We exploit a triple exponential smoothing using an additional exponential (EWMA) filter for estimating the seasonal component. This method provides accurate multiple-step ahead predictions of the time-series. Global errors for each considered length of the horizon are shown in Table 3.

	Horizon 1	**Horizon 2**	**Horizon 3**	**Horizon 4**
MAPE	4.60%	4.97%	5.39%	5.82%

Table 3: Global MAPE over the entire execution.

5.2.4 Effectiveness and feasibility of MPC

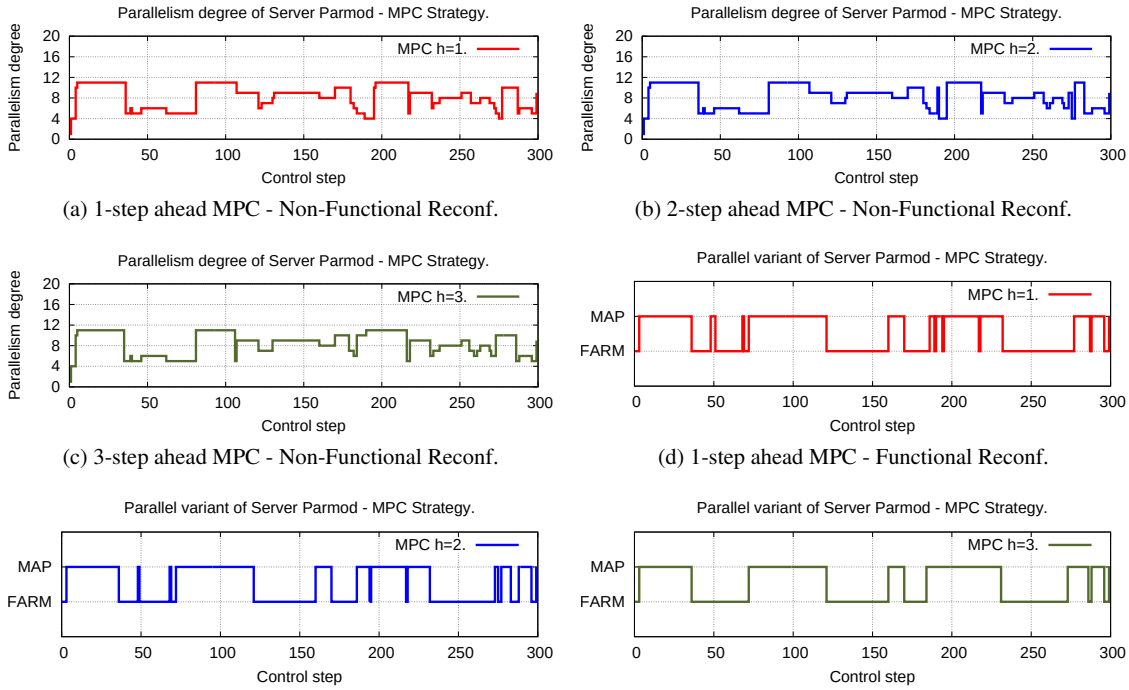
The application has been executed using 10 Clients and setting the maximum and the minimum threshold to 4.5 and 4 seconds respectively. The initial configuration of the Server is the task-farm with one worker. Server ParMod is executed on a homogeneous cluster of 32 workstations. Figure 15 shows the reconfiguration sequences with different horizons of 1, 2 and 3 control steps.

We note that *a long horizon improves the stability degree of the control decisions*, since the Control Part is able to determine the minimum set of reconfigurations that allows the response time to be within the thresholds for the entire duration of the prediction horizon. With a horizon of 4 steps (not shown in Figure 15 for space reasons), we are able to slightly reduce the number of reconfigurations of 12.24% compared to 1-step ahead MPC strategy (the `MSI` passes from 6.10 to 6.95).

Besides the reconfiguration number, a long horizon (provided that predictions are still sufficiently accurate) is effective in reducing the number of QoS violations, i.e. situations in which the mean response time measured is higher or lower than the maximum/minimum threshold. Figure 16 depicts the mean response time sampled for each step of the execution. The number of violations is 88, 86, 75 and 58 using a horizon of 1, 2, 3 and 4 steps. With the 4-step ahead MPC strategy we achieve a 34% reduction of QoS violations compared with 1-step ahead MPC.

Unfortunately, *the prediction horizon can not be arbitrary long*. The first reason is because predictions become less accurate going deeper in the horizon. Secondly, the exploration of the evolution tree (Section 4.2.3) must be completed within the control step duration. In this example the Operating Part has 64 possible configurations (two parallel variants each one with 32 parallelism degrees) resulting in a large state-space if we consider long horizons. With a horizon of 4 steps the state-space is formed by 17M states, thus prohibitive to explore exhaustively. Therefore, we need to apply techniques able to make the resolution *feasible* and even *negligible* w.r.t the control step length.

We instantiate the Branch & Bound scheme described in Section 4.2.3. During the exploration of the evolution tree, we denote by C^{opt} the minimum cost of all root-to-leaf paths



(a) 1-step ahead MPC - Non-Functional Reconf.

(b) 2-step ahead MPC - Non-Functional Reconf.

(c) 3-step ahead MPC - Non-Functional Reconf.

(d) 1-step ahead MPC - Functional Reconf.

(e) 2-step ahead MPC - Functional Reconf.

(f) 3-step ahead MPC - Functional Reconf.

Fig. 15: Functional and Non-functional reconfigurations of the Server ParMod.

currently explored during the search process and by C^j the cost associated with the node j of the tree, i.e. it represents the cost of the path from the root to j. The search space can be reduced by applying the pruning condition stated by the following proposition:

Proposition 1 *(Pruning condition). Let be i the currently explored node of the evolution tree with a cost C^i. If the cost function J is monotonically increasing with the steps of the horizon, we can apply the following condition:*

- *if i is not a leaf and $C^i \geq C^{opt}$, we can discard the entire sub-tree rooted at node i.*

Otherwise, if the previous condition is not verified and i is a leaf with $C^i < C^{opt}$, then we set $C^{opt} = C^i$.

Proof (sketch)
The idea of the proof can be straightforwardly derived from the monotonicity of cost function J. If node i is not a leaf and its cost C^i is higher than the best cost C_{opt}, we can certainly state that all the sub-paths starting from i and reaching a leaf will have a total cost higher than C^{opt}, since by going deeper in the horizon the cost can only increase and never decrease. Therefore, it is possible to discard the sub-tree rooted at i without losing the optimal solution.

It is trivial to verify that our cost function is monotonically increasing with the steps of the horizon. Therefore we can apply the pruning condition to this problem. Table 4 compares the theoretical number of states with the average number explored by the B&B approach.

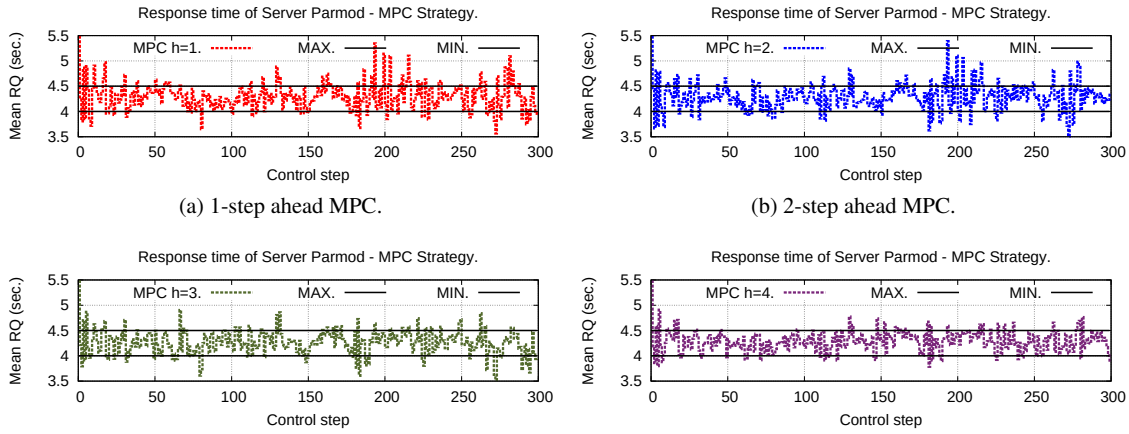
Strategy	Total States	Expl. States (B&B)	Time (B&B)
MPC h=1	65	65	$<$ 1 ms
MPC h=2	4,161	253	$<$ 1 ms
MPC h=3	266,305	1,038	$<$ 1 ms
MPC h=4	17,043,521	5,884	14.53 ms

Table 4: Explored states with the Branch & Bound approach.

As we can see the reduction is of one or more orders of magnitude. This makes it possible to complete the calculation of the optimal control trajectory of the 4-step ahead MPC strategy with an average time near to 14 milliseconds, which is negligible w.r.t. the control step length.

6 Conclusions and Future Work

This paper describes the application of MPC to distributed parallel computations. We show how the knowledge of the computation structure, expressed using structured parallelism paradigms, and the definition of their cost models can be exploited to instantiate the control of distributed parallel computations using MPC. To exhaustively evaluate our approach, in the future we plan to provide further examples using patterns not discussed in this paper, such as stencil data-parallel programs and divide-and-conquer parallelizations. Further-



(a) 1-step ahead MPC.

(b) 2-step ahead MPC.

(c) 3-step ahead MPC.

(d) 4-step ahead MPC.

Fig. 16: Mean response time R_Q of the Server during the execution.

more, it can be interesting to explore the use of a *discounting factor* in the formulation of optimization problems. Discounting factors could be an effective way to deal with uncertain operating environments in which the future system state predictions become increasingly inaccurate as we go deeper into the prediction horizon.

In the paper we give an insight into how high-level programming models should incorporate the different elements of MPC by placing proper mechanisms in the run-time support level. This aspect of the research, only hinted in this paper actually, needs further investigation in the future.

From the experimental viewpoint, the presented case-studies show that MPC, besides being applicable to structured parallel computations, maintains the desired properties studied in the literature [37]. In the first case-study we compare MPC with a heuristic strategy. The results show that by using MPC we are able to reduce reconfigurations of 40% w.r.t a simple strategy based on condition-action rules (applied for a similar problem in [55,2]). Operating cost can be reduced of 24% with an improvement in the number of completed tasks of 12% using the same execution conditions. In the second case-study we focus on the effects of the horizon length on the effectiveness of the adaptation process. By using sufficiently long horizons (feasible managed using Branch and Bound schemes), we are able to reduce QoS violations of 35% with a further reduction of reconfigurations of 12% compared with the strategy using a one-step horizon.

Finally, the paper discusses the problem of making the optimization process feasible with the used sampling interval. We introduce a generic B&B scheme that should be considered one of the possible methods to deal with this issue. The scheme has been instantiated in the second case-study exploiting the specific properties of the optimization problem. When larger platforms featuring hundreds or thousands of processors are considered, the computational cost could become impractical even using B&B. In that case other interesting techniques are based on *evolutionary algorithms* and specialized heuristics. As a future work we plan to evaluate the integration of such techniques in our approach.

References

1. Abdelwahed, S., Kandasamy, N., Neema, S.: Online control for self-management in computing systems. In: Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE, pp. 368 – 375 (2004)
2. Aldinucci, M., Danelutto, M., Kilpatrick, P.: Towards hierarchical management of au- tonomic components: A case study. In: Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, pp. 3–10 (2009). DOI 10.1109/PDP.2009.48
3. Aldinucci, M., Danelutto, M., Teti, P.: An advanced environment supporting structured parallel programming in java. Future Generation Computer Systems **19**(5), 611 – 626 (2003). ¡ce:title¿Tools for Program Development and Analysis. Best papers from two Technical Sessions, at ICCS2001, San Francisco, CA, USA, and ICCS2002, Amsterdam, The Netherlands¡/ce:title¿
4. Arshad, N., Heimbigner, D., Wolf, A.L.: Deployment and dynamic reconfiguration planning for distributed software systems. Software Quality Control **15**(3), 265–281 (2007)
5. 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 **7**(3), 225–255 (1995)
6. Bacci, B., Danelutto, M., Pelagatti, S., Vanneschi, M.: A heterogeneous environment for hpc applications. Parallel Computing **25**(13-14), 1827–1852 (1999)
7. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems **28**(5), 755 – 768 (2012). ¡ce:title¿Special Section: Energy efficiency in large-scale distributed systems¡/ce:title¿
8. Bernini, D., Micucci, D., Tisato, F.: Space integration services: a platform for space-aware communication. In: IWCMC '10: Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, pp. 489–493. ACM, New York, NY, USA (2010). DOI http://doi.acm.org/10.1145/1815396.1815510

9. Bertolli, C., Buono, D., Lametti, S., Mencagli, G., Meneghin, M., Pascucci, A., Vanneschi, M.: A programming model for high-performance adaptive applications on pervasive mobile grids. In: Proceeding of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems, pp. 38–54 (2009)
10. Bertolli, C., Buono, D., Mencagli, G., Vanneschi, M.: Expressing adaptivity and context-awareness in the assistant programming model. In: Proceedings of the Third International ICST Conference on Autonomic Computing and Communication Systems, pp. 38–54 (2009)
11. Bertolli, C., Mencagli, G., Vanneschi, M.: Adaptivity in risk and emergency management applications on pervasive grids. In: Proceeding of the 10th International Symposium on Pervasive Systems, Algorithms, and Networks, p. to appear (2009)
12. Bertolli, C., Mencagli, G., Vanneschi, M.: Analyzing memory requirements for pervasive grid applications. Parallel, Distributed, and Network-Based Processing, Euromicro Conference on **0**, 297–301 (2010)
13. Bhavanishankar, R., Subramaniam, C., Kumar, M., Dugar, D.: A context aware approach to emergency management systems. In: IWCMC '09: Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing, pp. 1350–1354. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1582379.1582675
14. Botorog, G.H., Kuchen, H.: Efficient high-level parallel programming. Theoretical Computer Science **196**(12), 71 – 107 (1998)
15. Charteris, A., Syme, W., Walden, W.: Urban flood modelling and mapping 2d or not 2d. In: Proceedings of the 6th Conference on Hydraulics in Civil Engineering: The State of Hydraulics, pp. 355–363. Barton A.C.T: Institution of Engineers, Barton, Australia (2001)
16. Chatfield, C., Yar, M.: Holt-winters forecasting: Some practical issues. Journal of the Royal Statistical Society. Series D (The Statistician) **37**(2), pp. 129–140 (1988)
17. Cole, M.: Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel Comput. **30**(3), 389–406 (2004)
18. D-ITG: Distributed Internet Traffic Generator. `http://www.grid.unina.it/software/ITG/papers.php`
19. 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. **9**(4), 445–467 (2009)
20. Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: theory and practice a survey. Automatica **25**, 335–348 (1989)
21. Gholami, M., Salahshoor, K., Tabatabaei-pour, M., Shaker, H., Alizadeh, T.: Improved model predictive control of discrete-time hybrid systems with mixed inputs. In: Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE, pp. 744 –749 (2007)
22. Gomes, A.T.A., Batista, T.V., Joolia, A., Coulson, G.: Architecting dynamic reconfiguration in dependable systems pp. 237–261 (2007)
23. González-Vélez, H., Leyton, M.: A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers. Software–Practice & Experience **40**(12), 1135–1160 (2010). DOI 10.1002/spe.1026. 2010 JCR Impact Factor 1-year: 0.573 5-year: 0.786
24. He, Q., Dovrolis, C., Ammar, M.: On the predictability of large transfer tcp throughput. Comput. Netw. **51**, 3959–3977 (2007)
25. Heemels, W.P.M.H., Sandee, J.H., Van Den Bosch, P.P.J.: Analysis of event-driven controllers for linear systems. International Journal of Control **81**(4), 571–590 (2008)
26. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
27. Horvath, T., Abdelzaher, T., Skadron, K., Liu, X.: Dynamic voltage scaling in multitier web servers with end-to-end delay control. IEEE Trans. Comput. **56**(4), 444–458 (2007). DOI 10.1109/TC.2007.1003
28. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**(3), 1–28 (2008)
29. Kephart, J., Walsh, W.: An artificial intelligence perspective on autonomic computing policies. In: Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on, pp. 3–12 (2004)
30. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
31. Kieran, D., Yan, W.: A framework for an event driven video surveillance system. In: Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on, pp. 97–102 (2010). DOI 10.1109/AVSS.2010.57
32. Kuchen, H., Ernsting, S.: Data parallel skeletons in java. Procedia Computer Science **9**(0), 1817 – 1826 (2012). ¡ce:title¿Proceedings of the International Conference on Computational Science, {ICCS} 2012¡/ce:title¿
33. Kusic, D., Kandasamy, N.: Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. Cluster Computing **10**(4), 395–408 (2007). DOI 10.1007/s10586-007-0022-y. URL `http://dx.doi.org/10.1007/s10586-007-0022-y`
34. 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 **8**(3), 245–258 (2011). DOI 10.1109/TNSM.2011.0726.100045
35. Legaux, J., Loulergue, F., Jubertie, S.: Osl: An algorithmic skeleton library with exceptions. Procedia Computer Science **18**(0), 260 – 269 (2013). ¡ce:title¿2013 International Conference on Computational Science¡/ce:title¿
36. Liu, H., Parashar, M.: Accord: a programming framework for autonomic applications. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **36**(3), 341 – 352 (2006). DOI 10.1109/TSMCC.2006.871577
37. Maggio, M., Hoffmann, H., Papadopoulos, A.V., Panerati, J., Santambrogio, M.D., Agarwal, A., Leva, A.: Comparison of decision-making strategies for self-optimization in autonomic computing systems. ACM Trans. Auton. Adapt. Syst. **7**(4), 36:1–36:32 (2012). DOI 10.1145/2382570.2382572
38. Maheshwari, N., Nanduri, R., Varma, V.: Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. Future Gener. Comput. Syst. **28**(1), 119–127 (2012)
39. Mencagli, G., Vanneschi, M.: Qos-control of structured parallel computations: A predictive control approach. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pp. 296–303 (2011). DOI 10.1109/CloudCom.2011.47
40. NCTUNS: Network Simulator and Emulator. `http://nsl.csie.nctu.edu.tw/nctuns.html/` (2011)
41. Parashar, M., Liu, H., Li, Z., Matossian, V., Schmidt, C., Zhang, G., Hariri, S.: Automate: Enabling autonomic applications on the grid. Cluster Computing **9**, 161–174 (2006). DOI 10.1007/s10586-006-7561-5. URL `http://dl.acm.org/citation.cfm?id=1127683.1127688`
42. Park, S.M., Humphrey, M.: Predictable high-performance computing using feedback control and admission control. Parallel and Distributed Systems, IEEE Transactions on **22**(3), 396–411 (2011). DOI 10.1109/TPDS.2010.100
43. Patikirikorala, T., Colman, A., Han, J., Wang, L.: An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems. Journal of Systems and Software **85**(12), 2678 – 2696 (2012). ¡ce:title¿Self-Adaptive Systems¡/ce:title¿

44. Pellegrini, M.C., Riveill, M.: Component management in a dynamic architecture. J. Supercomput. **24**(2), 151–159 (2003). DOI http://dx.doi.org/10.1023/A:1021798709301
45. Potocnik, B., Music, G., Zupancic, B.: Model predictive control systems with discrete inputs. In: Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean, vol. 1, pp. 383 – 386 Vol.1 (2004)
46. Reiff-Marganiec, S., Turner, K.J.: Feature interaction in policies. Comput. Netw. **45**, 569–584 (2004). DOI 10.1016/j.comnet.2004.03.004. URL http://portal.acm.org/citation.cfm?id=1031816.1031818
47. Rossiter, J.A.: Model-based predictive control: a practical approach. Control series. CRC Press, pub-CRC:adr (2003)
48. Saini, M., Xiangyu, W., Atrey, P., Kankanhalli, M.: Dynamic workload assignment in video surveillance systems. In: Multimedia and Expo (ICME), 2011 IEEE International Conference on, pp. 1–6 (2011). DOI 10.1109/ICME.2011.6012076
49. Sandee, J.H., Heemels, W.P.M.H., Van Den Bosch, P.P.J.: Case studies in event-driven control. In: Proceedings of the 10th international conference on Hybrid systems: computation and control, HSCC'07, pp. 762–765. Springer-Verlag, Berlin, Heidelberg (2007)
50. Shi, X., Liu, C., Wu, S., Jin, H., Wu, X., Deng, L.: A cloud service cache system based on memory template of virtual machine. In: Proceedings of the 2011 Sixth Annual ChinaGrid Conference, CHINAGRID '11, pp. 168–173. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/ChinaGrid.2011.20. URL http://dx.doi.org/10.1109/ChinaGrid.2011.20
51. Vanneschi, M.: The programming model of assist, an environment for parallel and distributed portable applications. Parallel Comput. **28**(12), 1709–1732 (2002)
52. Vanneschi, M., Veraldi, L.: Dynamicity in distributed applications: issues, problems and the assist approach. Parallel Comput. **33**(12), 822–845 (2007)
53. 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 **11**(3), 229–245 (2008). DOI 10.1007/s10586-008-0053-z. URL http://dx.doi.org/10.1007/s10586-008-0053-z
54. Warneke, D., Kao, O.: Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. IEEE Trans. Parallel Distrib. Syst. **22**(6) (2011). DOI 10.1109/TPDS.2011.65
55. Weigold, T., Aldinucci, M., Danelutto, M., Getov, V.: Process-driven biometric identification by means of autonomic grid components. Int. J. Auton. Adapt. Commun. Syst. **5**(3), 274–291 (2012). DOI 10.1504/IJAACS.2012.047659. URL http://dx.doi.org/10.1504/IJAACS.2012.047659
56. Wernsing, J.R., Stitt, G.: Elastic computing: a framework for transparent, portable, and adaptive multi-core heterogeneous computing. SIGPLAN Not. **45**(4), 115–124 (2010)
57. Wilkinson, B., Allen, M.: Parallel programming: techniques and applications using networked workstations and parallel computers. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1999)
58. Yaikhom, G., Cole, M., Gilmore, S., Hillston, J.: A structural approach for modelling performance of systems using skeletons. Electronic Notes in Theoretical Computer Science **190**(3), 167 – 183 (2007). ¡ce:title¿Proceedings of the Fifth Workshop on Quantitative Aspects of Programming Languages (QAPL 2007)¡/ce:title¿
59. 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. In: Proceedings of the 6th international conference on Advances in grid and pervasive computing, GPC'11, pp. 33–41. Springer-Verlag, Berlin, Heidelberg (2011)