Reconfiguration Stability of Adaptive Distributed Parallel Applications through a Cooperative Predictive Control Approach

Gabriele Mencagli, Marco Vanneschi and Emanuele Vespa

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

Abstract. Distributed parallel applications executed on heterogeneous and dynamic environments need to adapt their configuration (in terms of parallelism degree and parallelism form for each component) in response to unpredictable factors related to the physical platform and the application semantics. On emerging Cloud computing scenarios, reconfigurations induce economic costs and performance degradations on the execution. In this context, it is of paramount importance to define smart adaptation strategies able to achieve properties like control optimality (optimizing the application global QoS) and reconfiguration stability, expressed in terms of number of reconfigurations and the average time for which a configuration is not modified. In this paper we introduce a methodology to address this issue, based on Control Theory and Optimal Control foundations. We present a first validation of our approach in a simulation environment, outlining its effectiveness and feasibility.

Keywords: Distributed Parallel Computations, Reconfigurations, Autonomic Computing, Model-based Predictive Control.

1 Introduction

With the emergence of computational paradigms like Grid and Cloud Computing, properties like reconfigurability and ***adaptiveness*** have gained more importance [1, 2]. In scenarios characterized by variable workload conditions and dynamic execution environments the achievement of the desired *Quality of Service* (QoS) requires to adapt the application configuration expressed in terms of component identification, mapping onto physical resources, and proper selection of a parallelism degree and a parallelism form (e.g. farming and data-parallelism paradigms) for each component [3]. Such choices need to be applied to computations by performing run-time *reconfiguration activities*.

Reconfiguration processes can induce costs on the execution [3]. During a switching from a configuration to another, it is important to take into account several factors, such as the cost of the newly selected configuration (e.g. in a pay-per-use execution environment dependent on the classes of dynamically

provisioned computing resources [1]). The reconfiguration cost can also be proportional to the "*amplitude*" of the switch [2], i.e. a monetary charge and/or a performance overhead proportional to the amount of allocated/deallocated computing resources. Consequently, *it is clear that reconfigurations do not come for free, but they should be executed only when real benefits in terms of QoS can be achieved.*

Over the last years, many studies [3, 4] about adaptiveness for distributed parallel applications have focused on providing run-time supports to dynamic reconfigurations with the minimum impact on the computation performance (often assuming dedicated execution environments). On the other hand, the problem of defining powerful *Adaptation Strategies* is still an open research issue which requires innovative approaches to performance modeling, to the achievement of agreements between control decisions of different controllers, and to ensure the minimization of operating costs related to the application execution.

In this paper we propose a novel approach based on a control-theoretic strategy known as *Model-based Predictive Control* [5]. We control the parallelism degree of distributed parallel computations organized as graphs of parallel components. Each component features a local control sub-problem and cooperates with the others in order to reach the optimal application QoS (expressed in terms of effective performance and resource utilization cost). The cooperation is enforced using the *Distributed Subgradient Method* [6]. We propose different formulations of the adaptation strategy and we show interesting results in terms of *reconfiguration stability*. This property is expressed as the amount of performed reconfigurations and the average time for which a newly selected configuration is not modified.

This paper is organized as follows. The next section reviews research work on adaptiveness for distributed parallel computations. Section 3 presents our methodology. Section 4 shows a first evaluation of our approach developed in a simulation environment. Finally, Section 5 gives the conclusion of this work.

2 Related Work

Providing computing systems with run-time supports to dynamic reconfigurations has been the subject of several researches in different fields like Mobile, Grid and Cloud Computing. On such environments, it is of great importance to dynamically provide computing resources to applications featuring variable QoS requirements and characterized by irregular workload conditions. Examples are described in [7], in which provisioning mechanisms of virtual machines are provided to accelerate compute-intensive jobs submitted into Cloud platforms.

Besides efficient run-time supports [3, 4], emerging computing environments raise critical problems related to when reconfigurations should be executed in order to optimize performance and economic aspects. Therefore, adaptation strategies have gained much attention. The pro-active adaptation to future workload variations goes in the direction of defining powerful strategies able to optimize performance requirements and operating costs by avoiding unnecessary recon-

figurations. Despite the existence of first activities in this area [8], this research direction is open and still lacks from systematic approaches.

Along this line, this paper introduces an approach based on the application of Control Theory and Optimal Control foundations.

3 Methodology and Problem Statement

The control problem of distributed parallel applications can be decomposed into a set of sub-problems associated with each application module (we use module as a synonym of component). The distributed control logic should be organized in order to state how individual solutions of local control problems are combined and directed towards a solution that optimizes the application global QoS.

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

- the **Operating Part** performs the *functional logic* of the module, i.e. a parallel computation that instantiates a structured parallelism pattern (e.g. task-farm and data-parallel schemes) [3, 4]. The computation is activated by receiving tasks from input data streams. Results are transmitted onto output data streams directed to specific destinations;
- the **Control Part** (controller) observes the Operating Part execution and performs reconfiguration activities. The Control Part implements the *adaptation strategy* that drives the reconfiguration selection.

At discrete time intervals (called *control steps*), the Operating Part exchanges measurements representing the actual behavior of the parallel computation (e.g. memory usage, resource utilization, service time and computation latency) with the Control Part. In order to take effective reconfiguration decisions, at each control step Control Parts of different ParMods (belonging to the same application) exchange *control information* in order to reach specific agreements between control decisions. The result is a set of reconfigurations able to change the parallel computation, e.g. modifications of the current parallelism degree (number of threads/processes of the current implementation).

3.1 Distributed Model-based Predictive Control

An important precondition to apply control-theoretic techniques is the existence of a mathematical model of the controlled system. For each application module M_i we identify a *local model* involving the following set of variables:

- ***QoS variables*** ($\mathbf{x}_i(k) \in \mathbb{R}^n$) are metrics describing the current behavior of the parallel computation, such as the performance, memory usage and resource consumption assumed at the beginning of control step k;
- ***control variables*** ($\mathbf{u}_i(k) \in \mathcal{U}_i$) are parameters that identify the ParMod configuration used throughout the k-th control step;

– ***disturbances*** ($\mathbf{d}_i(k) \in \mathbb{R}^m$) model exogenous uncontrollable factors influencing the relationship between control and QoS variables (e.g. the arrival rate from external sources).

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

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

The model allows Control Part to predict future values assumed by local QoS variables as a function of local control inputs, disturbances and (in dynamic models) present values of QoS variables. Furthermore, the next QoS of each subsystem is still related to the remaining control variables of the other sub-systems (or a sub-set of them). Therefore *the control problem of the whole application can be viewed as a set of coupled sub-problems.*

In this paper we present adaptation strategies based on a control-theoretic technique named Model-based Predictive Control (shortly **MPC**) [5] . MPC is a method in which the current reconfiguration decision is taken by solving, at the beginning of each control step, a finite-horizon optimal control problem using the current value of QoS variables and statistical multiple-step ahead predictions of future disturbances. To be robust in dynamic and uncertain environments, only the first element of the optimal reconfiguration sequence (trajectory) is passed to the Operating Part, and the same procedure is repeated at the next control step.

Distributed MPC schemes can be applied to control large-scale systems such as distributed computations. In this case the optimization problem is composed of a set of coupled sub-problems each one formed by a local objective function, a local model and a set of local constraints. In a *cooperative scenario*, the goal of the decomposition is to reach a sequence of globally optimal control decisions, i.e. the solutions of the sub-problems should optimize the following global problem:

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

where J_G is the global objective function, defined as the weighted sum of local objectives (w_i is a positive weight), and an uppercase overlined letter represents a trajectory over a prediction horizon of h future control steps.

3.2 Addressing the Stability of control decisions

In dynamic execution contexts, distributed parallel applications should adapt the amount of used resources to provide acceptable levels of performance and a

reasonable resource utilization cost. For each ParMod M_i, the configuration parameter (control input) is the current parallelism degree $n_i(k) \in \mathcal{U}_i$ (number of used computing nodes), where $\mathcal{U}_i$ is the closed interval $[1,\, n_i^{max}]$ of integers. Disturbances are parameters that may change due to environmental or application-dependent reasons. Examples are the mean calculation time per task $T_{calc\text{-}i}(k)$ and the probabilities of task transmission between modules, i.e. $p_{i,j}(k)$ is probability to transmit a task from ParMod M_i to M_j during control step k. The mean *ideal service time* of a module must be defined as a function of its parallelism degree, e.g. $T_{S_i}(k) = T_{calc\text{-}i}(k)/n_i(k)$ (*perfect scalability assumption*).

The interaction between distributed modules usually follows the message-passing paradigm. Communications resort on *blocking mechanisms* to address the finiteness of the input buffers. If a message attempts to enter a full capacity destination queue upon the completion of a service at M, it is forced to wait in that component until the destination has a free position. We call the mean *inter-departure time*, the steady-state average time between two successive result departures. We denote with $T_{D_i}(k)$ the QoS variable representing the mean inter-departure time of M_i at the beginning of control step k (it refers to the average value assumed during the last step $k-1$).

To exemplify our approach, we adopt a simple yet powerful performance model already discussed in [9]. The method is valid for a large class of computation graphs, i.e. *acyclic graphs with a single source module*. The main result is summarized by the following theorem (the proof can be found in [9]):

Theorem 1 (Steady-State Analysis) *Given a single source acyclic graph G of N modules, the inter-departure time T_{D_i} from M_i can be expressed as:*

$$T_{D_i}(k+1) = \max\Big\{ f_{i,1}\big(T_{S_1}(k)\big),\, f_{i,2}\big(T_{S_2}(k)\big), \ldots, f_{i,N}\big(T_{S_N}(k)\big) \Big\} \tag{3}$$

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

$$f_{i,j}\big(T_{S_j}(k)\big) = T_{S_j}(k)\, \frac{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_1 \to M_j)} \left(\prod_{\forall (s,d) \in \pi} p_{s,d}(k) \right)}{\displaystyle\sum_{\forall \pi \in \mathcal{P}(M_1 \to M_i)} \left(\prod_{\forall (s,d) \in \pi} p_{s,d}(k) \right)} \tag{4}$$

where M_1 denotes the source, $\mathcal{P}(M_1 \to M_i)$ is the set of all the paths starting from M_1 and reaching M_i, and (s,d) is an edge of the path π. Since we do not know which module will be the bottleneck, the inter-departure time is calculated by taking the maximum between the functions $f_{i,j}$ for $j = 1, \ldots, N$.

We study two different formulations of the MPC strategy. In the first one we do not model any abstract term related to the reconfiguration cost (we refer to this as *Non-Switching Cost Formulation* for brevity):

Definition 1 (Non-Switching Cost Formulation). *Each parallel module has a local cost function defined over a horizon of one future step:*

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

The performance cost discourages configurations that compromise the capability to process incoming tasks. The second part expresses a cost proportional to the number of used nodes. α_i and β_i are two positive coefficients establishing the desired trade-off between the two contrasting aspects of the cost function.

In Grid and Cloud environments the reconfiguration process can induce costs on the computation, both in terms of a performance degradation (e.g. parallel modules could be blocked waiting for the reconfiguration process to complete) as well as in terms of a monetary charge due to the dynamic provisioning of resources. In the second formulation we account for an abstract cost term:

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

$$J_i(k) = \underbrace{\sum_{q=k}^{k+h-1} \alpha_i \cdot T_{D_i}(q+1)}_{performance\ cost} + \underbrace{\sum_{q=k}^{k+h-1} \beta_i \cdot n_i(q)}_{resource\ cost} + \underbrace{\sum_{q=k}^{k+h-1} \gamma_i \cdot \Delta_i(q)^2}_{switching\ cost} \tag{6}$$

where $\Delta_i(k) = n_i(k) - n_i(k-1)$. The switching cost term is defined as a function of the square of parallelism degree variations over the horizon (γ_i is a positive coefficient), and binds control decisions between consecutive steps allowing to express formulations with a parametric horizon length.

This formulation is aimed at improving the reconfiguration stability by discouraging reconfigurations with large amplitude and avoiding fluctuating behaviors due to disturbances with high variance and featuring trend patterns.

We solve the cooperative distributed MPC problem using the ***Distributed Subgradient Method***, originally proposed in [6] for multi-agent environments. The method addresses the problem of optimizing in a distributed fashion the sum $J_G(k) = \sum J_i(k)$ of non-smooth convex functions known only by their agents. This method suits particularly well our needs, since:

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

Each Control Part computes and maintains a local estimate of the optimum ***strategy profile matrix*** $\mathcal{S}(k) \in \mathbb{R}^{h \times N}$, where each column i corresponds to

the reconfiguration trajectory of ParMod M_i (parallelism degrees are considered real values for feasibility reasons). Neighboring controllers iteratively exchange their local estimates and compute the next estimate using the following rule:

$$\mathcal{S}_{[i]}^{(q+1)}(k) = \mathcal{P}_{\mathcal{U}_f}\left[\sum_{j=1}^{N}\Big(\mathcal{W}[i,j]\,\mathcal{S}_{[j]}^{(q)}(k)\Big) - a^{(q)}\,\mathcal{G}_i\right] \tag{7}$$

where q is the current iteration, $a^{(q)} > 0$ is the *step-size* and $\mathcal{G}_i$ is a *subgradient* of J_i at point $\mathcal{S}_{[i]}^{(q)}(k)$[1]. $\mathcal{P}_{\mathcal{U}_f}$ is the Euclidean projection onto the convex set of admissible strategy profiles defined by: $\mathcal{U}_f = \mathcal{U}_1^h \times \mathcal{U}_2^h \times \ldots \times \mathcal{U}_N^h$.

Each controller maintains a set of weights representing the importance given to the estimates received by the controllers (zero is assigned to non-neighbor controllers). To prove the convergence to the global optimum, in [6] the authors state a condition about how the weights should be assigned: the weight matrices $\mathcal{W} \in \mathbb{R}^{N\times N}$ should be *doubly stochastic*, i.e. all the columns and rows sum to 1.

The MPC strategy based on the Distributed Sub-gradient Method consists in a sequence of actions performed by the controllers at each control step k:

- each controller acquires monitoring information from its Operating Part and calculates statistical predictions of disturbances over the prediction horizon;
- each controller uses a specific initial estimate of the strategy profile matrix and applies the iterative protocol for a fixed number of iterations;
- at each iteration, controllers receive the local estimates from their neighbors, apply the update rule (7) and transmit the next estimate;
- after the last iteration, each controller knows its optimal reconfiguration trajectory and applies the first element of that trajectory (properly rounded to the nearest integer) as the new parallelism degree for control step k.

This method allows us to consider also *non-ideal performance behaviors* of parallel modules, providing that the ideal service time is modeled as a convex function of the parallelism degree. An example is when the service time stops to decrease or even increases using parallelism degrees larger than a specific value.

4 Evaluation of the approach

We have developed a ParMod simulation environment based on the OmNeT++ discrete event simulator. A ParMod is simulated by two OmNeT components modeling the Operating Part and the Control Part. The Operating Part implements a queue logic in which buffered elements represent input tasks. To reproduce a blocking semantics, we have implemented a communication protocol based on the transmission of send and ack messages. The Operating Part can adopt two working logics: (i) a *task-farm semantics*, in which at most p tasks in parallel can be executed, where p is the current parallelism degree; (ii) a *data-parallel semantics*, in which only one task at a time is processed with an execution time equal to the calculation time divided by the parallelism degree.

[1] the subscript $[i]$ denotes that $\mathcal{S}_{[i]}^{(q)}(k)$ is the estimate of the i-th controller.

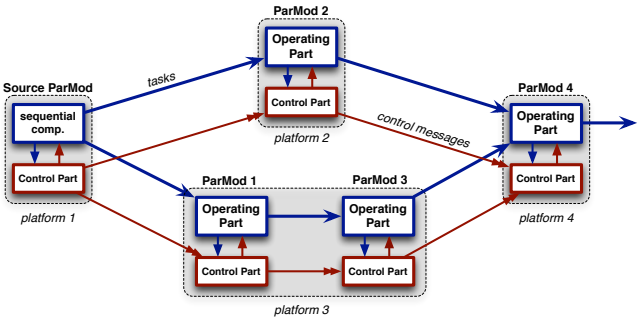


Fig. 1: Computation graph of the experiment.

We consider the computation graph depicted in Figure 1. The source module, implementing a sequential computation, transmits tasks with a variable rate to ParMod 1 and 2 according to the same probability. ParMods need to dynamically adapt their parallelism degree in order to sustain the current arrival rate and to avoid using computing nodes unnecessarily. To exemplify a dynamic situation, Figure 2 shows a time-series of the ideal service time (the inverse of the service rate) of the source module, which is modeled as a measured disturbance.

In order to apply the distributed MPC strategy, we need multiple-step ahead predictions of disturbances. We exploit the well-known *Holt-Winters* filtering technique [10], an effective method accounting for time-series featuring non-stationarities such as trends and seasonal patterns. In this example we achieve accurate predictions: over the entire execution the mean relative error between the real trajectories and the predicted ones at each control step is of 8.83%, 9.38%, 10.06% and 10.77% with a horizon length equal to 1, 2, 3 and 4 steps.

We compare the Non-Switching Cost Formulation with the strategy in which we consider a switching term in the local cost functions. Moreover, in order to have a performance upper-bound, we consider the static case (named `MAX`) in which ParMods do not perform any reconfiguration, but they are configured to use their maximum parallelism degree for the entire execution. Table 1 shows

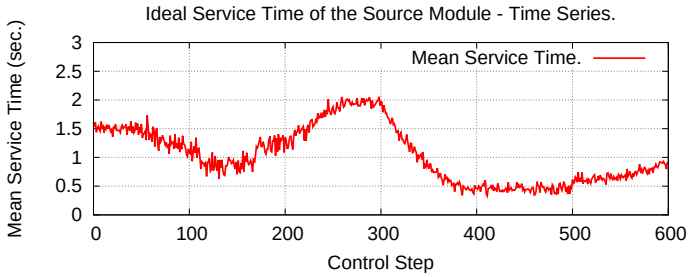


Fig. 2: Ideal service time of the source: 600 control steps each one of 240 seconds.

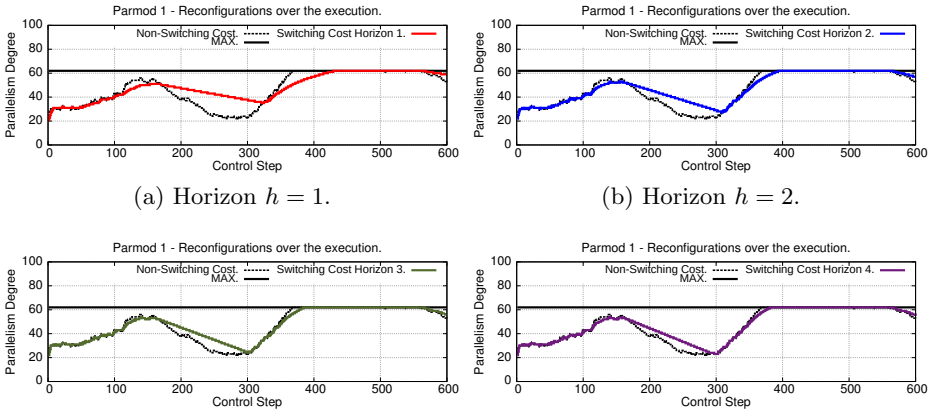
a set of user-defined parameters representing the importance of the different control objectives (performance vs. resources), the mean calculation times and maximum parallelism degrees.

	Source	**ParMod 1**	**ParMod 2**	**ParMod 3**	**ParMod 4**
$\boldsymbol{T_{calc}}$	Fig. 2	90 sec.	25 sec.	70 sec.	35 sec.
$\boldsymbol{\alpha}$	20	20	20	20	20
$\boldsymbol{\beta}$	0.5	0.3	0.8	0.3	0.4
$\boldsymbol{\gamma}$	-	1.5	1.2	1.5	1.2
$\boldsymbol{n_i^{max}}$	1	64	48	64	48

Table 1: Configuration parameters of the experiment.

Figure 3 shows the reconfiguration sequence of ParMod 1. For the sake of space we omit the results of the other modules (which are qualitatively similar). The reconfiguration sequence with the Non-Switching Cost Formulation follows the behavior of the source module. Execution phases in which the arrival rate to ParMod 1 decreases (i.e. from step 150 to 300) correspond to time intervals in which the module releases computing resources (its parallelism degree is oversized). The opposite behavior can be noticed after control step 300: due to a decreasing trend of the source service time, ParMod 1 starts to increase its parallelism degree. After control step 370 it reaches n_1^{max} (64 nodes), i.e. it can not acquire any other resource and becomes the graph bottleneck.

As we consider the switching cost, we achieve smoother reconfiguration sequences. *The switching cost acts as a disincentive to reconfigurations*: i.e. during execution phases in which the workload is lighter, it slows down the release of



(a) Horizon $h = 1$.

(b) Horizon $h = 2$.

(c) Horizon $h = 3$.

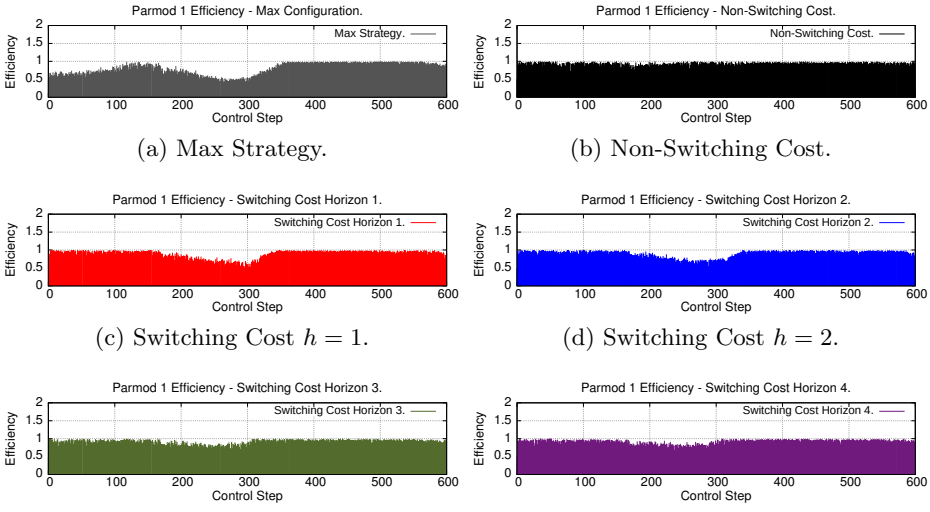
(d) Horizon $h = 4$.

Fig. 3: Parallelism degree variations of ParMod 1.

computing resources, while in phases of heavy workload it slows down the allocation of new resources. With longer horizons, controllers have a better degree of foresight and can more precisely evaluate if the acquisition/release of resources is effectively useful (e.g. avoiding to release/re-acquire them nearly in the future). This justifies a more rapid capability of longer-horizon strategies to respond to a pronounced change in the disturbance trend. Moreover, in execution phases characterized by a high level of uncertainty, the Switching Cost Formulation avoids many small reconfigurations of little amplitude (as happens from step 0 to 300). Therefore, *we can say that the switching cost acts also as a stabilizer in presence of disturbances with a significant variance.*

The horizon length has also important consequences on the *efficiency* of resource utilization, measured as the ratio between the ideal service time of a ParMod and its inter-departure time. An efficiency smaller than 1 means that the parallelism degree is over-sized. Figure 4 outlines the efficiency of ParMod 1. The `MAX` configuration (Figure 4a) suffers from a severe degradation from step 0 to 370. After step 370 the efficiency rises to 1 because the ParMod becomes the application bottleneck and it begins to fully exploit its maximum parallelism degree. Extremely interesting is the behavior of the Non-Switching Cost Formulation. In this case the efficiency is near to 1 throughout the execution. The reason is given by the structure of the local cost function (Definition 1): if a module is adopting an over-sized parallelism degree, it can release some computing resources without affecting its effective performance, but improving the value of its local cost without making the other cost functions worse off.

By introducing the switching cost we have a break that causes a slower release of computing resources. This induces a slight degradation of the efficiency



(a) Max Strategy.

(b) Non-Switching Cost.

(c) Switching Cost $h = 1$.

(d) Switching Cost $h = 2$.

(e) Switching Cost $h = 3$.

(f) Switching Cost $h = 4$.

Fig. 4: Efficiency of ParMod 1 with the different strategies.

from step 150 to 370. Using longer horizons (providing that the disturbance predictions are sufficiently accurate) we are able to mitigate this effect, and the efficiency tends to 1 again as Figure 4f depicts. To compare different strategies in terms of their reconfiguration stability, we introduce the following metric:

Definition 3. *We denote as* ***Mean Stability Index*** *(shortly* `MSI`*) the average number of control steps for which a reconfiguration remains active.*

Table 2 shows the total number of reconfigurations and the number of tasks that leave the system. The global `MSI` is the average of the individual indices of each module. With the `MAX` configuration we are able to maximize the number of completed tasks at the expense of a big waste of computing resources during the first part of the computation. With the Non-Switching Cost strategy the difference in completed tasks (the "no delay" column) is only of 8.25% but with a significant benefit in terms of efficiency. The Switching Cost Formulation has a large degree of configurability: with short horizons it heavily reduces the number of reconfigurations and the stability index with a small loss in terms of completed tasks. As we use longer horizons, the performance difference becomes negligible (only 0.5% of tasks reduction) but with a great improvement (34%) in terms of reconfigurations with a horizon of four steps. We conclude that the Switching Cost strategy is extremely powerful: *it makes it possible to significantly reduce the number of reconfigurations with a negligible performance reduction.*

Strategy	**Reconf.**	**MSI**	**Compl. Tasks (no delay)**	**Compl. Tasks (with delay)**
MAX	-	-	144,403	144,403
Non-Switch. Cost	870	2.77	132,482	123,643
Switch. Cost $h = 1$	389	6.28	129,560	124,898
Switch. Cost $h = 2$	524	4.65	131,176	124,911
Switch. Cost $h = 3$	559	4.34	131,641	125,030
Switch. Cost $h = 4$	574	4.25	131,808	125,823

Table 2: Number of reconfigurations, completed tasks and Mean Stability Index.

When a performance overhead is introduced, performing fewer reconfigurations could be useful also from the performance viewpoint. To prove this insight we modify our simulator: every time a ParMod applies a reconfiguration, it suspends to process incoming tasks for an amount of time modeled by a random variable *delay*. In order to reproduce a Cloud scenario, in which the time-to-deploy of a virtual machine can reach tens of seconds [1], we repeat the simulations using a delay of 30 seconds. Now we have a different tendency in terms of completed tasks (the "with delay" column): *using the Switching Cost Formulation we achieve better performance saving a consistent number of reconfigurations.*

We conclude by pointing out the feasibility of our approach. Although the subgradient method can be rather slow, we can limit the number of iterations (we

used 125 iterations per step). This can be done by considering two aspects: (i) firstly each Control Part applies an integer rounding of the parallelism degree, thus a high precision is not necessary actually; (ii) since between consecutive control steps optimal solutions are likely close, we use as starting estimate the optimal strategy profile matrix calculated at the previous step. In this way we can drastically reduce the number of iterations maintaining an acceptable precision.

5 Conclusion

This paper provides a description of our approach. The control logic of each module consists of a performance model and a local cost function. Reconfigurations are applied following the receding horizon principle and the MPC strategy. Controllers cooperate to reach globally optimal decisions using the Distributed Subgradient Method. In order to enforce the stability of control decisions, and measuring the impact of stability w.r.t QoS goals, we evaluate different MPC formulations. Simulation results show the effectiveness of our approach. In the future we plan to apply our techniques in real-world distributed environments.

References

1. Costa, R., Brasileiro, F., Lemos, G., Sousa, D.: Analyzing the impact of elasticity on the profit of cloud computing providers. Future Generation Computer Systems (0) (2013)
2. Han, R., Ghanem, M.M., Guo, L., Guo, Y., Osmond, M.: Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. Future Generation Computer Systems (0) (2012)
3. Vanneschi, M., Veraldi, L.: Dynamicity in distributed applications: issues, problems and the assist approach. Parallel Comput. **33**(12) (2007) 822–845
4. Aldinucci, M., Campa, S., Danelutto, M., Vanneschi, M.: Behavioural skeletons in gcm: Autonomic management of grid components. In: Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. (Feb. 2008) 54–63
5. Garcia, C.E., Prett, D.M., Morari, M.: Model predictive control: theory and practice a survey. Automatica **25** (May 1989) 335–348
6. Nedic, A., Ozdaglar, A.: Distributed subgradient methods for multi-agent optimization. Automatic Control, IEEE Transactions on **54**(1) (jan. 2009) 48 –61
7. Warneke, D., Kao, O.: Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. IEEE Trans. Parallel Distrib. Syst. **22**(6) (2011)
8. Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems **28**(1) (2012) 155 – 162
9. Mencagli, G.: A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations. Ph.D Thesis, University of Pisa, Italy (2012)
10. Chatfield, C., Yar, M.: Holt-winters forecasting: Some practical issues. Journal of the Royal Statistical Society. Series D (The Statistician) **37**(2) (1988) pp. 129–140