Control-theoretic Adaptation Strategies for Autonomic Reconfigurable Parallel Applications on Cloud Environments

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

***Abstract*—Cloud Computing is a paradigm that enables the access to a set of shared networking and computing resources and high-level platforms and services through the exploitation of virtualization technologies. On Clouds, it is of relevant importance to make applications adaptive and reconfigurable, in the sense that the optimal configuration (satisfying desired QoS levels) should be dynamically changed in response to variations in the workload conditions and in the resource availability. Due to this fact, adaptation strategies have gained much attention over the last years. Properties like control optimality (finding proper trade-offs between contrasting QoS goals), reconfiguration stability (expressed as a function of the average time between consecutive reconfigurations) and reconfiguration amplitude (performing sequences of small modifications of the current configuration) are important aspects to consider. In order to meet these needs, we present a control-theoretic approach and we provide a first validation of our proposals, giving an insight about its applicability to Cloud environments.**

Keywords*—*Autonomic Computing, Parallel Computations, Reconfigurations, Model-based Predictive Control, Distributed Cooperative Optimization.

I. Introduction

Cloud Computing is an emerging paradigm encouraging the diffusion of large-scale heterogeneous distributed environments and their efficient and transparent interaction through virtualization technologies. In addition to the maintenance of performance constraints required by the users when executing their applications, economic aspects become increasingly challenging on Clouds [18], where computing and networking resources are dynamically provisioned on-demand to the users against the payment of a monetary charge [7], [4].

For this reason, the dynamic provisioning needs to be automated and integrated through intelligent strategies able to observe the application execution and promptly respond to dynamic modifications of the workload conditions and of execution environment. To do that, applications should be designed as compositions of adaptable parallel components, whose algorithms and parallelizations can be changed by an efficient interaction with Cloud providers.

Adaptation strategies need to be qualitatively and quantitatively compared using specific metrics able to capture their effectiveness. Examples are the *stability of control decisions*, i.e. how long the effects of a reconfiguration last before a new modification of the application configuration should be applied, and their *optimality*, i.e. achieve desired trade-offs between contrasting requirements (e.g. performance, memory usage, storage and power consumption). These properties have gained great importance in Clouds. However, applications and their run-time supports are still not mature enough for supporting adaptiveness in such environments [4]. To improve the actual state-of-the-art, this paper presents formal adaptation strategies based on sound theoretical foundations such as the ones of Control Theory and Distributed Optimization.

This paper is organized as follows. Section II provides a brief review of the existing literature. Section III presents a general description of adaptive parallel applications, and outlines the interesting properties that we require to evaluate and compare adaptation strategies. Section IV introduces our methodology: we present our notion of adaptive components and a control-theoretic adaptation strategy to drive the reconfiguration selection. The cooperation between application controllers is enforced using a distributed cooperative optimization method. Section V presents a first validation of our methodology by discussing results obtained in a simulation environment. Although at the present state our work has not been evaluated in real Cloud platforms, we claim that the methodological work presented in this paper is interesting to improve the existing approaches and will be able to be integrated with real frameworks in the future.

II. Related Work

Providing distributed systems with run-time supports to dynamic reconfigurations has been the subject of researches in different fields like Mobile, Grid and Cloud Computing. On Clouds [17], [12], provisioning mechanisms of virtual machines are developed to accelerate compute-intensive jobs submitted into Cloud platforms. On more general distributed parallel programming frameworks, adaptiveness has been introduced by exploiting dynamic modifications of structured parallelism forms, expressed parametrically w.r.t the parallelism degree. Notable examples are the Behavioral Skeleton approach [1] to the definition of autonomic algorithmic skeletons for Grid applications, and the ASSIST [16] model for adaptive high-performance computations.

Besides efficient run-time supports, emerging computing environments raise critical problems related to when reconfigurations should be executed in order to optimize both performance and economic aspects. Therefore, adaptation strategies

have gained much attention. In addition to classic approaches based on *logic policy rules* [9] (e.g. event-condition-action rules), control-theoretic techniques represent a potential alternative. Over the last years, Control Theory application to computer systems [8] has moved beyond the preliminary stage. Especially the pro-active adaptation to future workload variations [10] seems to be a suitable approach to optimize performance requirements and operating costs by avoiding unnecessary reconfigurations. Along this line Model-based Predictive Control [6] and Adaptive Control [11] are valuable research directions still requiring further investigations especially in fields like distributed parallel computations.

III. Adaptive Applications

In this section we focus on the concept of adaptive applications and the issues related to Cloud environments. The goal of the discussion is to state the fundamental properties that we want to address with advanced adaptation strategies.

Distributed parallel applications (e.g. scientific computations, emergency management and intelligent surveillance systems) are composed of several interacting components, exchanging messages representing tasks and (partial) results. Each component applies a computation on each received element: to remove or reduce bottleneck conditions, each component can be internally parallelized in order to sustain the current arrival rate and/or to provide acceptable levels of computation latency.

In our approach we model distributed parallel applications as directed graphs (*work-flow*) of components. We can distinguish between two different levels of parallelism:

- ***inter-component parallelism***: the decomposition of a complex application in several phases, each one associated with a distinct software component, is a first way to decompose the problem and solve it by a set of multiple processing entities distributed over the computing environment;
- ***intra-component parallelism***: each component can implement a parallel computation, activated by receiving messages from a set (or a sub-set) of sources. In our approach we suppose that *intra-component parallelizations are instances of structured parallelism forms* [3], [5] (e.g. task-farm, data-parallel and divide-and-conquer schemes), which can be expressed parametrically w.r.t the parallelism degree (number of threads/processes of the current implementation).

An application *configuration* indicates a precise decision about the way in which components perform the distributed processing. For example a configuration can indicate that some of the application components should be internally parallelized using a specific parallelism form and a parallelism degree.

Due to unexpected execution conditions, a static configuration may not be able to achieve the desired QoS goals throughout the execution. Such variability can be caused by the application semantics, e.g. in presence of time-varying workload conditions due to irregular parallel problems. In dynamic environments also the execution platform can play a decisive role, by making the QoS achieved by the same configuration variable over the execution. As an example, the dynamic availability of computing and networking resources can lead to variable calculation times to process similar tasks as well as high-variance communication latencies.

A reconfiguration implies a dynamic modification of the application configuration: components need to stop the current computation and reach a consistent state (if it is required by the reconfiguration granularity or by the computation semantics - e.g. if it operates on an internal state). Then, a new parallel version can be activated or the same parallelization can be instantiated using a different number of processing nodes (*parallelism degree variations*).

On Cloud environments the problem of dynamic reconfigurations introduces new issues peculiar to this computing paradigm. In elastic Cloud infrastructures, computing resources are usually delivered on-demand to the users in terms of virtual machines deployed in remote provider data centers. When executing adaptive applications, the modification of the application configuration leads to significant changes in the used infrastructure by frequently creating and shutting down virtualized resources (Figure 1 shows a representation of this problem). Creating virtual machines through traditional technologies takes from tens of seconds up to several minutes to complete [18], [15], and parallel components could be blocked waiting for the reconfiguration process to complete. As an example a component should reach a consistent computation state before re-distributing data on the newly allocated resources and then restart the execution [16]. Therefore, *reconfigurations induce performance overhead on the computation, and should be executed only when they bring real benefits in terms of QoS.*

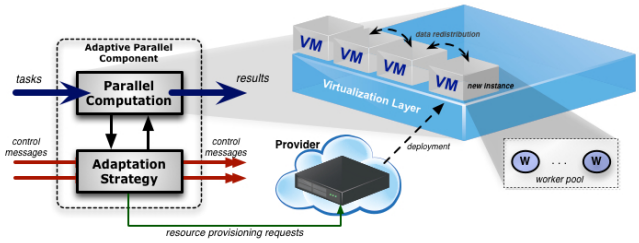


Fig. 1: Example of dynamic deployment of a parallel component on a Cloud environment.

Moreover, Cloud providers manage the infrastructures and the platforms making them available to the users on a pay-per-use basis. Therefore, during a switching from a configuration to another, *it is important to take into account the monetary cost of the newly selected configuration*. For instance the cost can be dependent on the classes of dynamically provisioned computing resources [4], but it can also depend on the "*amplitude*" of the switch [7], e.g. a monetary charge proportional to the amount of allocated/deallocated virtualized computing resources. For these reasons, we need to define adaptation strategies able to target the desired QoS goals by reducing reconfigurations and operating costs.

This paper introduces formal strategies to dynamic adaptation and evaluates them according to three important properties:

- ***control optimality***: the capability of a strategy to achieve the desired trade-off between different QoS objectives over the entire execution (e.g. performance and resource consumption);
- ***reconfiguration stability***: we express the stability of an adaptation strategy in terms of the total amount of performed reconfigurations, and the average time between two consecutive reconfigurations of the same component (informally how long the effects of a reconfiguration last);
- ***little reconfiguration amplitude***: it could be useful to compare adaptation strategies in terms of *reconfiguration amplitude*. Between strategies able to meet the desired QoS goals, we could prefer a strategy that avoids a large "difference" between consecutive configurations, e.g. in the case of parallelism degree variations, a little amplitude means that few resources are involved in the reconfiguration.

In the next section we will present and evaluate our methodology and different adaptation strategies.

IV. Methodology

The adaptation strategy of large-scale distributed parallel applications is executed by a set of control entities associated with each application component. In our approach each parallel component is structured in terms of two interconnected parts:

- the **Operating Part** performs a parallel computation instantiating a structured parallelism pattern [16]. The computation is activated whenever an input element is received from one of the input streams. According to the computation semantics, the result of each activation is transmitted onto one of the output streams directed to a specific destination;
- the **Control Part** (controller) observes the Operating Part execution and implements an *adaptation strategy* that drives the reconfiguration selection. Reconfigurations can change implementation aspects of the parallel computation, notably the number of used processing nodes - i.e. the parallelism degree of the component.

Our approach follows a control-theoretic modeling in which controllers are *time-driven*, i.e. adaptation strategy is executed every fixed time interval. Therefore, we model the passage of time in discrete time intervals of a fixed length (each interval is called *control step*). At each control step the controllers perform the following sequence of actions:

- they acquire the updated monitoring information from their Operating Part (e.g. actual performance levels);
- they evaluate the adaptation strategy possibly exchanging *control messages* with other controllers of the application;
- each Control Part performs the reconfiguration actions on the corresponding Operating Part, changing the current configuration of the component.

A. Distributed Model-based Predictive Control

To apply control-theoretic strategies, we need to define a local model of each controlled component able to capture the effects of reconfigurations on the QoS. This model formalizes the relationship between the following set of variables:

- a ***local QoS variable*** describes the steady-state average time between two successive result departures from a component. We call this measure the *mean inter-departure time*. We denote with $T_{D_i}(k)$ the inter- departure time of the i-th component at the beginning of control step k (it refers to the average value assumed during the last step $k-1$);
- a ***local control variable*** identifies the parallelism degree $n_i(k) \in \mathcal{U}_i$ used by the i-th component during step k, where $\mathcal{U}_i$ is the closed interval $[1,\, n_i^{max}]$;
- ***local disturbance variables*** model exogenous uncontrollable factors influencing the relationship between control and QoS variables. Examples are the mean calculation time per task $T_{calc\text{-}i}(k)$ and the probabilities of task transmission, i.e. $p_{i,j}(k)$ denotes the probability to transmit a task from component i to component j during control step k.

The performance of a parallel component can be measured without considering the interaction with other components of the application. This ideal behavior is modeled by the concept of *mean service time*, which depends on the performance model of structured parallelism schemes [16]. For the sake of generality, we initially express the service time of a component as a simple function of its parallelism degree, i.e. $T_{S_i}(k) = T_{calc\text{-}i}(k)/n_i(k)$ (*perfect scalability assumption*).

Since components are interconnected in a graph structure, their inter-departure time depends on their local configuration but it is also influenced by the presence of other components that can act as a bottleneck. To model this fact, we use an approach presented and demonstrated in [13] (Chapter 4 from page 47). The method is valid for a large class of computation graphs, i.e. *acyclic graphs with a single source component*. The main result is summarized by the following theorem:

Theorem 1 (Performance Modeling): Given a single source acyclic graph G of N components, the inter-departure time T_{D_i} from component C_i can be expressed in the following way:

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

Each $f_{i,j}$ with $j = 1, 2, \ldots, N$ expresses the inter-departure time of C_i if component j is currently the bottleneck. $f_{i,j}$ is defined as a function of the service time of C_j :

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

where C_s denotes the source component, $\mathcal{P}(C_s \to C_i)$ is the set of all the paths starting from C_s and reaching C_i, and

(u, v) is an edge of the path π. Since we do not know which component will be the bottleneck, the inter-departure time is calculated as the maximum between the functions $f_{i,j}$ for $j = 1, \ldots, N$.

In this paper we apply this performance modeling with a strategy based on a control-theoretic technique named *Model-based Predictive Control* (shortly **MPC**) [6] . MPC is a method in which reconfiguration decisions are taken by solving, at the beginning of each control step, a finite-horizon optimization problem using 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 (following the so-called *receding horizon principle*).

Besides a model of QoS variables, we need to introduce a proper set of *local objective functions* associated with each application component. 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 component has a local cost function defined over a horizon of *one future control step*:

$$J_i(k) = \underbrace{\alpha_i \, T_{D_i}\big(k+1\big)}_{\textit{performance cost}} + \underbrace{\beta_i \, n_i(k)}_{\textit{resource cost}} \tag{3}$$

The first part 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 positive coefficients establishing the desired trade-off between the two contrasting aspects of the cost function. $T_{D_i}(k+1)$ is calculated using Theorem 1.

As said, in Cloud environments the reconfiguration process can induce costs on the computation, both in terms of a performance degradation 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 component C_i is defined over a prediction horizon of h control steps (with $h \geq 1$):

$$J_i(k) = \underbrace{\sum_{q=k}^{k+h-1} \alpha_i \cdot T_{D_i}(q+1)}_{\textit{performance cost}} + \underbrace{\sum_{q=k}^{k+h-1} \beta_i \cdot n_i(q)}_{\textit{resource cost}} + \underbrace{\sum_{q=k}^{k+h-1} \gamma_i \cdot \Delta_i(q)^2}_{\textit{switching cost}} \tag{4}$$

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 (γ_i is a positive coefficient). Its goal is to discourage reconfigurations with large amplitude and avoid fluctuating behaviors due to disturbances with high variance.

B. A Cooperative Interaction between Controllers

As we have seen, the control problem of distributed parallel applications consists in several coupled sub-problems each one formed by a local objective function and a local model. In a *cooperative scenario*, instead of optimizing their local cost functions selfishly, controllers cooperate in order to select optimal control actions in a system-wide sense, i.e. reconfigurations that optimize a notion of ***global objective function***, e.g. defined as the (weighted) sum of the local cost functions of each component.

We solve the cooperative distributed MPC problem using the ***Distributed Subgradient Method***, originally proposed in [14] for multi-agent environments. The method addresses the problem of optimizing the sum $J_G(k) = \sum J_i(k)$ of non-smooth convex functions known only by their agents. This method 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 (the inter-departure time is defined as the point-wise maximum of a set of convex functions $f_{i,j}$);
- for scalability and feasibility reasons, Control Parts are directly interconnected only between neighbors.

Each Control Part computes and maintains an estimate of the optimum ***strategy profile matrix*** $\mathcal{S}(k) \in \mathbb{R}^{h \times N}$, where each column i corresponds to the parallelism degree trajectory of component C_i (parallelism degrees are considered real values for feasibility reasons) and h is the horizon length. Neighboring controllers iteratively exchange their local estimates (control messages) and compute the next estimate using the following rule:

$$\mathcal{S}_{[i]}^{(q+1)}(k) = \mathcal{P}_{\mathcal{U}_f} \left[\sum_{j=1}^{N} \Big(\mathcal{W}[i,j] \, \mathcal{S}_{[j]}^{(q)}(k) \Big) \; - \; a^{(q)} \, \mathcal{G}_i \right] \tag{5}$$

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 [14] 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 Subgradient 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;

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

- at each iteration, controllers receive the local estimates from their neighbors, apply the update rule (5) 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 components, providing that the mean 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.

V. Experiments

To provide a first evaluation of our methodology, we have developed a simulation environment based on the OmNeT++ discrete event simulator[2]. An adaptive parallel component is simulated by two simulation modules which implement the Operating Part and the Control Part. OmNeT makes it possible to define the behavior of a simulation module following an event-driven programming style. A module receives different classes of messages. Every time a new message is received, the `handlemessage()` handler routine is called. Modules exchange messages through communication ports (see Figure 2). The reception of a message invokes a corresponding event handler function defined by the programmer. To reproduce a blocking communication semantics, we have implemented a communication protocol based on the exchange 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 can be processed with an execution time equal to the calculation time divided by the parallelism degree.

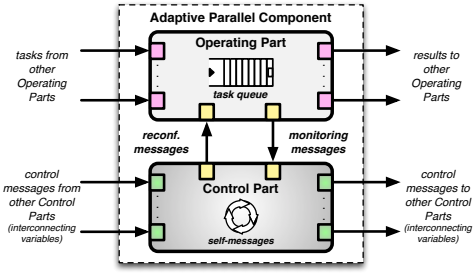


Fig. 2: Simulation of an Adaptive Parallel Component.

As a benchmark case, we consider an abstract computation composed of five distributed components organized in the computation graph depicted in Figure 3. The source component implements a sequential computation (i.e. its parallelism degree is fixed to 1 throughout the execution) and transmits tasks to components C_1 and C_2 according to a discrete probability distribution.

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

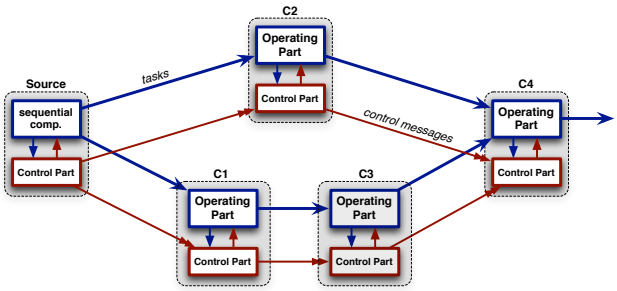


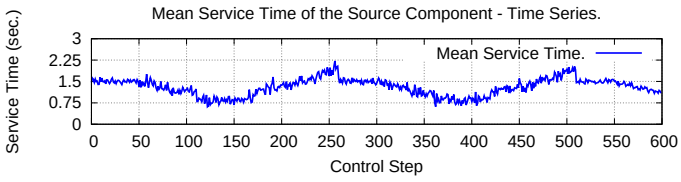
Fig. 3: Computation graph of the experiment.

A. Multi-disturbance Scenario

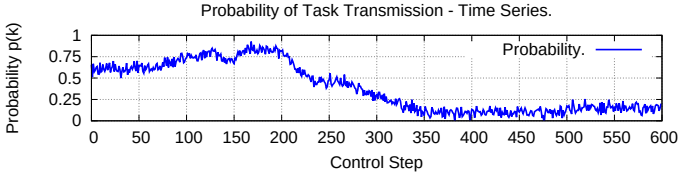
The application components need to dynamically adapt their parallelism degree in order to sustain the current arrival rate and to avoid using computing nodes unnecessarily. We exemplify a dynamic situation characterized by a *multi-disturbance scenario*: during the execution, the optimal application configuration changes in response to the following sources of variability:

- there are time intervals in which the source component generates tasks faster/slower than other periods. In other words, *the mean service time of the source may change significantly*. Since the source component is sequential, its service time coincides with its calculation time, i.e. $T_{calc_{src}}(k) = T_{S_{src}}(k)$;
- components C_1, C_3 and C_2 correspond to different sub-systems able to process tasks of different types. The source transmits tasks to C_1 with probability $p(k)$ and to C_2 with probability $1 - p(k)$. Due to the application semantics, probability $p(k)$ changes during the execution, and influences the arrival rates to the application components.

We simulate a scenario in which the source's service time and the transmission probability change following the time-series depicted in Figure 5. The execution consists of 600 control steps each one of 60 seconds.



(a) Service time.



(b) Probability $p(k)$.

Fig. 5: Multi-disturbance scenario of the experiment.

To apply the MPC strategy, we need to predict the values assumed by disturbances over a limited horizon of few control steps in the future. To do that, history-based forecasting techniques can be applied to our time-series. We exploit the well-known *Holt-Winters* filtering technique [2], an effective method accounting for time-series featuring non-stationarities such as trends and seasonal components. For the probability time-series, we adopt a classic Holt-Winters scheme able to estimate the trend component using two EWMA filters, one for the smooth component and the other for the trend. For the service time we adopt a seasonal Holt-Winters scheme in which we use a third EWMA filter to estimate the seasonal behavior of the time-series. Table I shows the global relative errors between the real trajectories and the predicted ones averaging between all the control steps of the execution and considering a horizon length from 1 to 3 control steps (multiple-step ahead predictions). As we can see, this forecasting method is effective for our time-series, leading to a global error bounded by 12%.

	Hor. 1	**Hor. 2**	**Hor. 3**
Probability Series	10.17%	10.66%	11.18%
Service Time Series	8.57%	9.27%	9.83%

TABLE I: Global relative errors over the entire execution.

B. Simulation Results

In this section we compare the Non-Switching Cost and the Switching Cost Formulations using different horizon lengths. The cost parameters and the mean calculation times are summarized in Table II. For all the components (except the source), the calculation time is modeled as a normal random variable with a given mean and a small variance. User-defined cost parameters for the resource cost and the switching cost term are taken in order to simulate a heterogeneous scenario in which our application is executed on a federation of Cloud providers, applying different billing models.

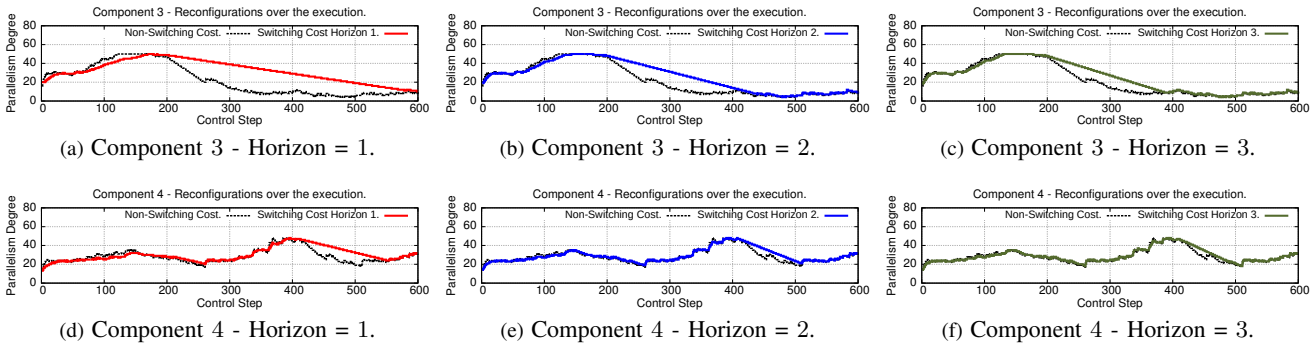
1) Reconfigurations over Executions: Figure 4 shows the reconfiguration sequence of component C_3 and C_4 (the results for the other components are qualitatively similar). The reconfiguration sequence of C_3 depends on the combined effect of the source's service time variability and the probability $p(k)$. In fact, both of them influence the arrival rate to the subsystem composed of components C_1 and C_3. During phases in which the arrival rate increases, we expect that the parallelism degree increases too. The contrary happens during phases characterized by a lower pressure of incoming tasks to such components. As we can observe from Figures 4a, 4b and 4c, the parallelism degree of the third component follows the general trend of the probability (Figure 5b). In the first part of the execution (from step 0 to 200) tasks are transmitted more frequently to C_1 and C_3, while from step 350 to 600 the arrival rate to C_2 becomes much more higher than the other components. Therefore, significant changes in the probability $p(k)$ dominates the variability of the arrival rates to C_1, C_2 and C_3 which adapt by modifying their parallelism degree correspondently.

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$
T_{calc}	90 sec.	100 sec.	70 sec.	35 sec.
α	10	10	10	10
β	0.3	0.8	0.3	0.4
γ	1.5	1.2	1.5	1.2
n_i^{max}	64	32	48	48

TABLE II: Configuration parameters of the experiment.

C_4 is characterized by a different behavior. Here, the reconfiguration sequence follows the variability of the source's service time and it is not influenced by the probability $p(k)$. However this is an expected phenomenon. As we can observe from Figure 3, the last component is responsible to receive tasks from C_2 and C_3. The semantics is that whenever a task is received from the input queue, component C_4 starts its parallel computation on the current stream element (received either from C_3 or from C_2). Therefore, the two input streams are selected non-deterministically, based on the presence of input elements. The total arrival rate to C_4 is given by the sum of the two arrival rates from C_2 and C_3. Provided that the other components are not bottlenecks, this means that the arrival rate to C_4 corresponds to the inverse of the source's service time.

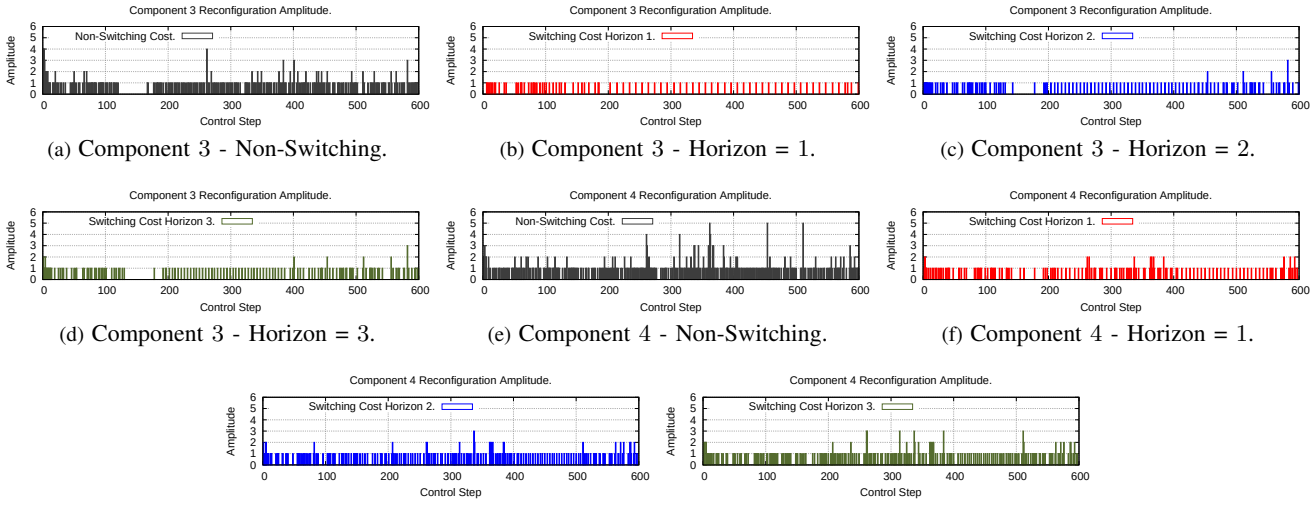
We compare the reconfigurations with the Switching Cost Formulation and the sequence using the Non-Switching Cost Formulation. As we can observe *the switching cost term acts as a disincentive to parallelism degree variations*. During phases



(a) Component 3 - Horizon = 1. (b) Component 3 - Horizon = 2. (c) Component 3 - Horizon = 3.

(d) Component 4 - Horizon = 1. (e) Component 4 - Horizon = 2. (f) Component 4 - Horizon = 3.

Fig. 4: Reconfiguration sequence of the third and the fourth component.



(a) Component 3 - Non-Switching. (b) Component 3 - Horizon = 1. (c) Component 3 - Horizon = 2.

(d) Component 3 - Horizon = 3. (e) Component 4 - Non-Switching. (f) Component 4 - Horizon = 1.

(g) Component 4 - Horizon = 2. (h) Component 4 - Horizon = 3.

Fig. 6: Reconfiguration amplitude of the third and the fourth component.

in which the arrival rate is lower, it slows down the release of computing resources, while in phases of higher rates it slows down the allocation of new resources. As we consider longer horizons, controllers have a better degree of foresight and can more precisely evaluate if the release/acquisition of a certain set of resources is effectively useful (e.g. avoiding to re-acquire/release them nearly in the future). Using a sufficiently long horizon, the sequence of parallelism degrees tends to the one obtained optimizing the cost functions without the switching cost, but avoiding many reconfigurations of little amplitude due to high-variance disturbances.

2) Reconfigurations Amplitude and Stability: Figure 6 depicts the histograms of the reconfiguration amplitude, i.e. the absolute value of the difference between consecutive parallelism degree variations. In the histograms related to the Non-Switching Cost Formulation, reconfigurations are almost equally distributed throughout the execution and reach peaks of amplitude 4 and 5 for the third and the fourth component. The Switching Cost Formulation produces a drastic reduction in terms of number of reconfigurations (parallelism degree variations are less frequent) but also a smaller amplitude (using a one-step horizon all the reconfigurations of the third component are unitary increase or decrease in the parallelism degree). With longer horizons the strategy is more responsive to disturbances: as the horizon becomes longer as the performance part of the cost functions becomes more dominant giving more reconfigurations with a slightly larger amplitude.

The concept of reconfiguration frequency can be more formally studied by introducing the following metric:

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

Table III shows the total number of reconfigurations performed by the application components and the global `MSI`. The Non-Switching Cost Formulation accurately adapts the parallelism degrees to disturbance variations without any constraint in releasing or allocating resources. In this way the amount of reconfigurations is higher than the other strategies. By introducing the switching cost, and therefore a break in the resource allocation/deallocation, we can drastically reduce the reconfigurations performed over the execution. The reduction is of 56%, 40% and 37% using a horizon of 1, 2 and 3 steps respectively. The `MSI` gives also a clear insight about how long a configuration remains active in the average case. Using a horizon of 3 steps, the effects of a reconfiguration last for more than three steps on average (better than using the Non-Switching Cost Formulation, in which we have a reconfiguration every two steps).

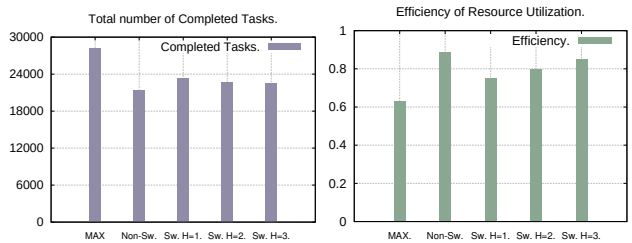
Strategy	Reconfigurations	MSI
Non-Switching Cost	1238	1.97
Switch. Cost $h = 1$	544	5.08
Switch. Cost $h = 2$	741	3.55
Switch. Cost $h = 3$	784	3.32

TABLE III: Reconfigurations and Mean Stability Index.

3) Performance and Efficiency evaluations: Besides the number of reconfigurations and their amplitude, an adaptation strategy should be evaluated by considering the performance achieved by the distributed parallel application. In order to have a quantitative result, we compare different strategies using the number of tasks that leave the system - i.e. tasks completely processed by the application components - as a measure of the global achieved performance.

In order to simulate a scenario in which reconfigurations affect on the application performance, we have simulated a performance overhead correlated with parallelism degree variations. To complete the reconfiguration process consistently with the computation's semantics, we suppose that the Operating Part of a parallel component suspends to receive and process incoming tasks for an amount of time modeled by a random variable *delay* with a normal distribution and a given mean and variance. For the sake of simplicity, in this

experiment we apply the same mean reconfiguration delay for all the components, equal to 15 seconds (which corresponds to 25% of the control step length). The results in terms of completed tasks are outlined in Figure 7a. In order to compare our strategies with a performance upper-bound, we consider the static case (named MAX) in which parallel components do not perform any reconfiguration (their parallelism degree is fixed to the maximum value n_i^{max} throughout the execution). As we can observe from Figure 7a, the MAX configuration is the best one in terms of completed tasks. The Non-Switching Cost Formulation, which accurately follows the disturbance predictions, provides the worst results since it applies a large number of reconfigurations. Using the Switching Cost we are able to achieve a better performance behavior avoiding unnecessarily reconfigurations. The best result is achieved with a horizon of one step (the performance loss is of 15% w.r.t the MAX configuration).



(a) Completed Tasks. (b) Efficiency.

Fig. 7: Completed tasks and efficiency of resource utilization.

As we use longer horizons, although this implies more reconfigurations and a lesser MSI, we can improve the *efficiency* of resource utilization, measured as the ratio between the ideal service time of a component and its inter-departure time (see Figure 7b). An efficiency lower than 1 means that the parallelism degree is over-sized. In this case the MAX configuration provides the worst results in terms of resource utilization (in the average case only 60% of the allocated resources are effectively utilized). On the other hand the best efficiency is achieved using the Non-Switching Cost Formulation, in which components quickly adapt to the current disturbances and avoid to use over-sized or under-sized parallelism degrees. With a short horizon, the Switching Cost Formulation is penalized during the releasing phases of computing resources. Using longer horizons we mitigate this effect, reaching acceptable levels of efficiency also with the Switching Cost Formulation.

VI. Conclusion

The efficient exploitation of heterogeneous platforms like Grids and Clouds poses serious problems of adaptiveness and reconfigurability of applications. In this paper we presented a theoretical work based on formal adaptation strategies inherited from Control Theory and Optimal Control. We proposed the application of a Distributed Model-based Predictive Control strategy and different formulations to address a switching cost due to changes in the application configuration. Optimal control problems were solved in a cooperative fashion enforcing the Distributed Subgradient Method. First experiments performed in a simulation environment provided encouraging results in terms of control optimality, reconfiguration stability and little reconfiguration amplitude, that we claim are important properties in Cloud environments.

References

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

[2] C. Chatfield and M. Yar, "Holt-winters forecasting: Some practical issues," *Journal of the Royal Statistical Society. Series D (The Statistician)*, vol. 37, no. 2, pp. pp. 129–140, 1988.

[3] M. Cole, "Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming," *Parallel Comput.*, vol. 30, no. 3, pp. 389–406, 2004.

[4] R. Costa, F. Brasileiro, G. Lemos, and D. Sousa, "Analyzing the impact of elasticity on the profit of cloud computing providers," *Future Generation Computer Systems*, no. 0, 2013.

[5] J. Darlington, Y.-k. Guo, H. W. To, and J. Yang, "Parallel skeletons for structured composition," in *PPOPP '95: Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 1995, pp. 19–28.

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

[7] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems*, no. 0, 2012.

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

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

[10] D. Kusic and N. Kandasamy, "Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems," *Cluster Computing*, vol. 10, 2007.

[11] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva, "Comparison of decision-making strategies for self-optimization in autonomic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 4, pp. 36:1–36:32, Dec. 2012.

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

[13] G. Mencagli, *A Control-Theoretic Methodology for Controlling Adaptive Structured Parallel Computations*. Ph.D Thesis, University of Pisa, Italy, 2012. [Online]. Available: http://etd.adm.unipi.it/t/etd-05242012-212444/

[14] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *Automatic Control, IEEE Transactions on*, vol. 54, no. 1, pp. 48 –61, jan. 2009.

[15] X. Shi, C. Liu, S. Wu, H. Jin, X. Wu, and L. Deng, "A cloud service cache system based on memory template of virtual machine," in *Proceedings of the 2011 Sixth Annual ChinaGrid Conference*, ser. CHINAGRID '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 168–173. [Online]. Available: http://dx.doi.org/10.1109/ChinaGrid.2011.20

[16] M. Vanneschi and L. Veraldi, "Dynamicity in distributed applications: issues, problems and the assist approach," *Parallel Comput.*, vol. 33, no. 12, pp. 822–845, 2007.

[17] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, 2011.

[18] Q. Yuan, Z. Liu, J. Peng, X. Wu, J. Li, F. Han, Q. Li, W. Zhang, X. Fan, and S. Kong, "A leasing instances based billing model for cloud computing," in *Proceedings of the 6th international conference on Advances in grid and pervasive computing*, ser. GPC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 33–41.