Università di Pisa
Dipartimento di Informatica

Technical Report: TR-11-14

QoS-control of Structured Parallel Computations: a Predictive Control Approach

Gabriele Mencagli Marco Vanneschi

September 13, 2011

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

QoS-control of Structured Parallel Computations: a Predictive Control Approach

Gabriele Mencagli Marco Vanneschi

September 13, 2011

Abstract

A central issue for parallel applications executed on heterogeneous distributed platforms (e.g. Grids and Clouds) is assuring that performance and cost parameters are optimized throughout the execution. A solution is based on providing application components with adaptation strategies able to select at run-time the best component configuration. In this paper we will introduce a preliminary work concerning the exploitation of control-theoretic techniques for controlling the Quality of Service of parallel computations. In particular we will demonstrate how the model-based predictive control strategy can be used based on first-principle performance models of structured parallelism schemes. We will also evaluate the viability of our approach on a first experimental scenario.

1 Introduction

The last years have been characterized by the arising of highly distributed computing platforms composed of a heterogeneity of computing and communication resources including centralized high-performance computing architectures (e.g. clusters or large shared-memory machines), as well as multi-/many-core components also integrated into mobile nodes and network facilities. The emerging of computational paradigms such as *Cloud Computing*, provides potential solutions to integrate such platforms with data systems, natural phenomena simulations, knowledge discovery and decision support systems responding to a dynamic demand of remote computing and communication resources and services.

One of the main issues for customers that use Cloud environments is the necessity to optimize the utilization of the infrastructure layer, through a proper dynamic selection of resources and services (e.g. optimizing operational costs), and the application layer, that may require to meet precise *Quality of Service* (QoS) constraints (e.g. in terms of performance at which computing results are provided to users). These objectives may be in opposition to each other: e.g. optimizing the application performance often requires a more powerful (and more expensive) configuration of the Cloud infrastructure. This is especially

true in the case of parallel computations, where the efficient platform utilization plays a decisive role.

In this context it is of great importance the definition of *adaptation strategies* for distributed applications featuring proper levels of self-adaptation, self-management and self-optimization. Nowadays research works face with this problem following different methodologies. A widely used approach is based on defining policy rules, usually based on logic languages, that express system adaptation actions in response to events about current QoS measurements. An example of this approach for parallel programs is explained in[1]. Other works try to exploit control-theoretic techniques for controlling computing systems. In these works (as in[2]), the most challenging problem is the definition of proper mathematical models of the controlled systems, used for applying classical control techniques (as PID controllers as in[2]) but also more advanced predictive approaches (as in[3]).

Nevertheless the research results are far from being mature and further research efforts are required, especially with the emerging of Cloud environments in which the problem of adaptation and optimization is much more stressed. In this paper we will describe our preliminary work for applying control-theoretic techniques to adapt distributed parallel systems. Our approach is based on a fundamental basic point: in our vision high-performance computations are instances of well-known *structured parallelism schemes*[4] (e.g. data- and task-parallelism schemes) for which a formal modeling of their QoS behavior can be studied analytically. Starting out from this we will present the application of a formal predictive control technique and we will discuss the exploitation of this approach for Cloud environments.

This paper is organized as follows. In Section 2 we will provide a brief overview about existing research works focusing on run-time adaptation of parallel computations. In Section 3 the basic description of our approach will be presented, discussing the concept of adaptive parallel module and introducing a control approach that is practical for our purposes. In Section 4 we will show a first real experiment concerning a distributed emergency management system on which we have evaluated the viability of our work.

2 Related Works

Adaptivity for high-performance applications is mainly intended as the dynamic reconfiguration of parallel programs (e.g. run-time modification of their parallelism degree). Structured parallel programming[4] has lead to optimized solutions[5] w.r.t. approaches based on general parallel programming models (e.g. MPI and OpenMP). Although several research works[5, 6] focus on efficient and highly optimized implementation of run-time support for autonomic high-performance applications (e.g. minimizing the reconfiguration overhead), they do not pay sufficient attention to the decision process (adaptation strategy): i.e. how reconfiguration decisions are taken by the application control logic.

In[1] a distributed hierarchical control of parallel components based on algorithmic skeletons[4] has been introduced, focusing on the possibility to modify the parallelism degree according to a reactive strategy expressed as *event-condition-action* policy rules. Adaptivity for compute-intensive applications has been also targeted in[7], but it is only discussed for adapting the computational load in large-scale simulations, and the possibility to express customized strategies is very limited. We claim that the knowledge of the structure of parallel computations should be used in a better way in order to define more powerful adaptation strategies, featuring several properties as the *predictability* of adaptation cost, the *optimization* of the entire system execution and the *stability degree* of a reconfiguration, i.e. selecting an adaptation action based on a reasonable expectation of how long this choice will be useful for the execution.

To this end we have investigated control-theoretic techniques for controlling the QoS of structured parallel computations. The exploitation of Control Theory foundations for controlling computing systems is rarely used. In[8] the problem of managing resource utilization for web servers has been studied providing queueing models and PID controllers used to regulate the system response time. In[9] the control of multiple QoS measurements has been presented for the IBM Lotus Domino Server: both CPU consumption and memory requirements are simultaneously controlled by exploiting a statistical system model and a PID controller. Furthermore an interesting approach has been introduced in[10]. In this work the performance level and the power consumption of a CPU have been adjusted by tuning its clock rate. The proposed approach exploits a well-known predictive control technique which has also been used in[3] for optimizing the power consumption of a server farm. Although this predictive control approach is a valuable starting point also for our work, in these researches its application is limited to ad-hoc systems (e.g. queueing models modeling the CPU behavior). Inheriting from these past experiences, in this paper we will introduce our formalization for controlling structured parallel computations.

3 Formalization and Control of a Parallel Module

The basic element of our approach is the concept of *adaptive parallel module*, shortly **ParMod**, an independent and active unit executing a parallel computation and an adaptation strategy for responding to changing execution conditions. ParMods, interconnected through data streams[1], can be composed into directed graphs representing distributed applications.

From an abstract standpoint a ParMod can be structured in two interconnected parts (see Figure 1), following the general closed-loop (i.e. feedback) interaction scheme:

- **Operating Part**: this part performs a parallel computation that instantiates a certain structured parallelism scheme. The computation is activated

[1]For stream we intend a sequence, possibly of unlimited length, of typed elements.

at each task reception from input data streams (input interfaces), and the results transmission is exploited onto output data streams (output interfaces) to other parallel modules. From a control-theoretic viewpoint the operating part is the observed *plant* of the closed-loop architecture;

- **Control Part**: this part represents the *controller*, i.e. an entity able to observe the operating part execution and modify its behavior exploiting reconfiguration activities.

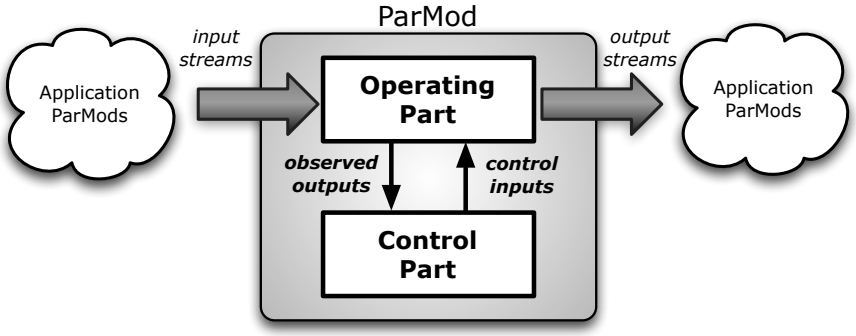

Figure 1: Operating Part and Control Part structure of a ParMod.

The information exchange between plant and controller in the two directions is exploited through a pair of:

- *observed outputs* from operating part to control part, i.e. all the interesting measurements that describe the computation behavior: e.g. the average throughput in terms of completed tasks per time unit, the mean computation latency and the current memory usage;

- *control inputs* from control part to operating part, i.e. commands that correspond to run-time reconfiguration activities of the current operating part configuration.

For structured parallel computations we can classify the set of adaptation processes in two categories namely non-functional and functional reconfigurations. ***Non-functional reconfigurations*** are activities involving the modification of some implementation aspects of a parallel computation, such as its current degree of parallelism, e.g. increasing the number of parallel activities (e.g. processes or threads) in such a way as to achieve a better performance level.

Moreover, in[11, 12] ***functional reconfigurations*** have also been introduced as an effective approach for dealing with heterogeneous computing environments. In real scenarios (e.g. Emergency Management Systems[11]), it is worth noting to be able to alternatively deploy the ParMod computation at run-time onto very different classes of resources, such as distributed-memory

architectures as clusters, or multi-core components or next-generation mobile nodes. In this case it is of great importance to provide distinct *versions* of the ParMod computation, featuring different sequential algorithms (e.g. optimized for the efficient exploitation of specific memory hierarchies) or different parallelism schemes (e.g. ensuring a better scalability on a particular class of architectures). Nevertheless these versions need to preserve the input and output interfaces of the ParMod: i.e. without modifying the global behavior of the application graph.

The design and the development of an adaptive ParMod requires to study two important and correlated aspects. The first one deals with the efficient implementation of run-time support mechanisms for applying adaptation actions with a minimum reconfiguration cost. In our past works (see[11]) we have discussed optimized reconfiguration protocols for structured parallel computations. The second aspect is instead an opened and challenging research issue that concerns how reconfiguration actions are decided by the control part. In[12] we have proposed a reactive control approach based on a control automaton semantics. In this paper, in order to improve the outcome of an adaptive execution, we will discuss the exploitation of more advance techniques starting from a novel modeling of the operating part behavior.

3.1 A Hybrid modeling of the Operating Part behavior

We need to model the QoS temporal evolution of a parallel computation. The final aim of a *system model* is to determine a mathematical relationship between reconfigurations and their impact on QoS variables. System models can be expressed empirically, based on statistical techniques over experimental data, or by exploiting first-principle relations. Although the model definition is often the most critical issue to apply control-theoretic approaches for controlling computing systems[2], we will show how the adoption of the structured parallel programming methodology plays a central role for simplifying such modeling effort. In our vision a ParMod features a multi-modal behavior:

Definition 3.1. (Multi-modal behavior of ParMod Operating Part). At each point of time the operating part behaves adopting a certain active configuration C_i, belonging to a finite set C of alternative operating modes:

$$C = \{C_0, C_1, \ldots, C_{\nu-1}\}$$

We have a ***finite and discrete set of statically known alternative configurations***, corresponding to a precise choice of: computational version (i.e. parallelism scheme and parallelized algorithm), parallelism degree and execution platform.

We can identify two classes of transitions that characterize the operating part execution:

- *continuous transitions*: when a configuration has been fixed, the evolution of continuous-valued QoS parameters can be predicted applying a specific model corresponding to the currently used configuration;

- *discrete transitions*: according to the adopted control law the current configuration can be changed passing from a configuration C_i to a different configuration C_j.

The presence of discrete and continuous transitions suggests to model the operating part behavior as a special class of *hybrid systems*[13] in which these two dynamics are modeled in a unique and refined mathematical structure.

Although the term hybrid may also be associated to the time-domain of system evolution, in this paper we refer to the domain in which model variables take their values and we fix the time domain to be discrete: i.e. we use a discretized time notion of *control step* of length τ. The beginning of each step represents a decision point, that is the plant-controller interaction and the adaptation strategy evaluation are performed periodically, at equally spaced time points (we speak about *time-driven controller*). With this assumption the ParMod model is expressed through the definition of three classes of variables:

- ***internal state variables***: $\mathbf{x}(k)$ is the value of state variables at the beginning of control step k. They represent state-ful measurements, e.g. the number of queued tasks and the number of completed input elements;

- ***measured disturbance inputs***: disturbance inputs $\mathbf{d}(k)$ are uncontrolled exogenous signals that can affect the relationship between control inputs and state variables. Examples are platform-dependent parameters (e.g. network behavior, CPU usage) and application-dependent parameters as the mean computational grain of tasks;

- ***control inputs***: control part decides the actual configuration that should be used for the entire duration of each control step. Thus each configuration is uniquely identified by a proper control input $\mathbf{u}(k)$ taking discrete values.

As we have seen input-state relationship is strictly coupled with the ParMod configuration. We can have multiple models that describe the internal state evolution according to the currently active configuration. For each of these a proper set of difference equations is provided in a state-space form:

$$\mathbf{x}(k+1) = \phi_i\Big(\mathbf{x}(k), \mathbf{d}(k)\Big) \;\; i = 0, 1, \ldots, \nu - 1 \tag{1}$$

This multi-modal modeling is a peculiar property of a class of hybrid systems, namely ***Switched Hybrid Systems***[13], featuring a limited set of alternative operating modes each one coupled with a corresponding model, and a switching law between them. In our case this law is completely controlled by an external entity, i.e. the ParMod control part (see Figure 2). The entire system modeling is given by:

$$\begin{aligned} &\mathbf{x}(k+1) = \Phi\Big(\mathbf{x}(k), \mathbf{d}(k), \mathbf{u}(k)\Big) \\ &= \text{ if } \Big(\pi\big(\mathbf{u}(k)\big) = C_i\Big) \text{ then } \phi_i\big(\mathbf{x}(k), \mathbf{d}(k)\big) \end{aligned} \tag{2}$$

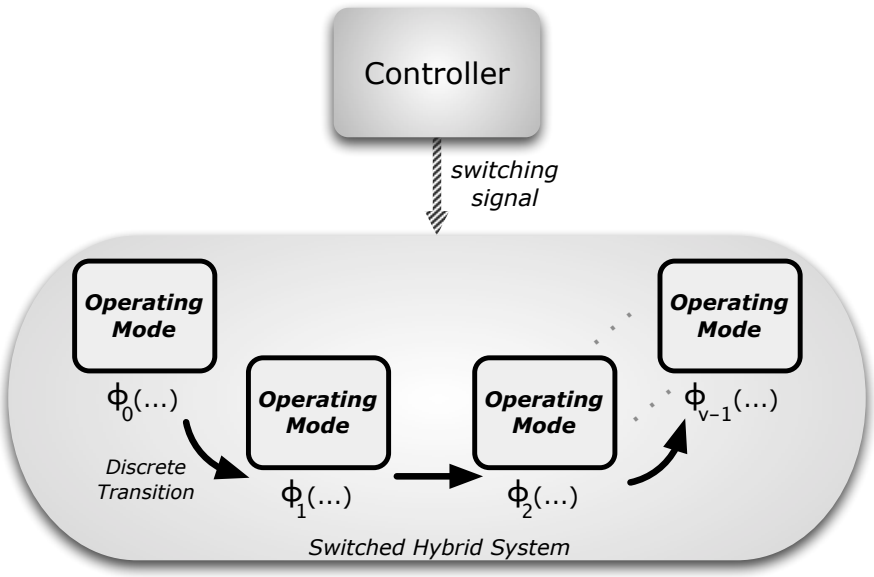

Figure 2: Switched Hybrid System with controlled switching law.

where π is a bijective function that maps the discrete set of control inputs onto the corresponding configuration indices. Formally speaking the hybrid modeling of the ParMod operating part can be stated as follows:

Definition 3.2 (Operating Part Model)**.** In our approach the operating part of an adaptive parallel module is modeled as a switched hybrid system with controlled switching law defined by the tuple $(\mathbf{U}, \mathbf{X}, \mathbf{D}, \Phi)$ where: $\mathbf{U}$ is the finite and discrete set of admissible control inputs corresponding to the possible operating part configurations, $\mathbf{X} \subseteq \mathbb{R}^s$ is the continuous-valued space of the internal states and $\mathbf{D} \subseteq \mathbb{R}^m$ is the continuous-valued set of exogenous inputs. The model that describes the next state expression is provided by the following function: $\Phi : \mathbf{U} \times \mathbf{X} \times \mathbf{D} \rightarrow \mathbb{R}^n$ maps a specific discrete-time model $\mathbf{x}(k+1) = \phi_i(\mathbf{x}(k), \mathbf{d}(k))$ onto a configuration $C_i \in C$ such that $\pi(\mathbf{u}(k)) = C_i$.

3.2 ParMod Optimal Control: Predictive Strategy

In this paper we are interested in describing predictive adaptation strategies based on control-theoretic foundations. Predictive approaches are control methods where a controller tries to estimate future in some way thinking ahead of corrective actions. A typical formulation is based on an optimal control problem, in which the controller exploits the system model in order to determine the best control trajectory (i.e. a sequence of reconfigurations) to optimize a properly defined objective function. Due to the static unpredictability of disturbances and the possible perturbations and unmodeled dynamics of the system

model, instead of applying the optimal reconfiguration trajectory in an open-loop fashion the optimization process is repeated iteratively at each sampling interval, based on the current monitoring data provided by the system.

A practical method, widely used over the last decades for controlling chemical and industrial plants, is the so-called *Model-based Predictive Control* (**MPC**)[14]. At the beginning of each control step k the model state $\mathbf{x}(k)$ of the operating part is measured by the controller. Then the controller predicts the QoS behavior through the system model and solves an optimization problem in order to find an optimal sequence of reconfiguration decisions (i.e. a *reconfiguration plane*) for each control step of a short prediction horizon of h control steps. A plant objective function needs to be provided (e.g. an utility function as in (3)):

$$\max\ U(k) = \sum_{i=k}^{k+h-1} L\Big(\mathbf{x}(i+1), \mathbf{u}(i)\Big) \tag{3}$$

To determine the optimal plane, the controller needs to predict the values of disturbances over the prediction horizon by applying proper statistical techniques (e.g. autoregressive models and smoothing filters). After that, instead of applying the optimal reconfiguration plane step-by-step, only the first decision is applied and afterwards the complete procedure is repeated at the next control step. Since the prediction horizon has a fixed length but it is moved towards the future by one control step each time, this approach is also known as *receding horizon* technique.

Despite its large applicability MPC is considered a compute-intensive approach especially for hybrid systems (as a ParMod) where the set of control inputs is discrete. In this case the on-line optimization problem, if not properly addressed with specific techniques (e.g. branch&bound approaches) or heuristics, implies an exhaustive search by testing among all the feasible combinations of reconfigurations, thus potentially limiting its viability to systems with long sampling intervals. Therefore the exploitation of this technique for controlling adaptive parallel computations may require relatively short prediction horizons and few possible reconfiguration alternatives, unless effective search space reduction techniques are used.

4 A test-bed application and Experiments

We have applied our approach to an existing distributed system for flood risk forecasting. Emergency management systems (EMS) are developed for addressing the critical demand of computation and communication during natural or man-made disasters (e.g. floods or earthquakes). In oder to facilitate the decision making process of civil protection personnel, an ICT infrastructure periodically executes computationally intensive simulations and promptly spreads the results to the user end-point devices (e.g. PDA and smart-phones).

In[11, 12] we have described an system for flood emergency prediction. This application is composed of a ***Generator*** module, that periodically produces

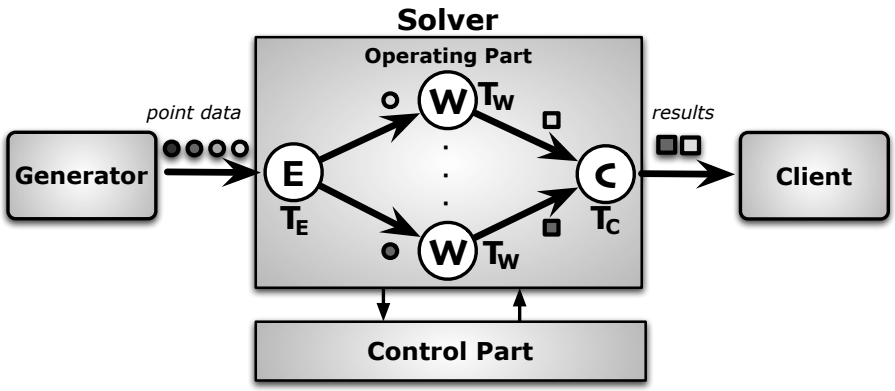

Figure 3: Task-Farm implementation of the Forecasting Model.

sensor data (e.g. water surface elevation and water speed) for each point of a 2D discretization of the emergency scenario (e.g. a river basin). Each point is considered as an independent task by a ***Solver*** ParMod, that numerically solves a system of differential equations describing the flow behavior (i.e. a hydrodynamical model). Although the calculation of each point can also be parallelized through data-parallel techniques (see[12]), in this example we have considered a single parallel version of the computation based on a *task-farm* parallelization, as described in[11].

Task-farm is a structured parallelism scheme which exploits the independence among a large sequence of input tasks. For this scheme three classes of parallel functionalities are defined (Figure 3): (i) an *emitter* (E) is responsible for scheduling each task (represented as a data-structure of 16 MB in the solver configuration that we have used) according to a load-balanced distribution (e.g. tasks are scheduled only to available workers, following an on-demand policy); (ii) the hydrodynamical model is replicated among a set of parallel *workers* (W), each one applying it on different scheduled points; (iii) a *collector* (C) collects the results and transmits them to a ***Client*** module performing post-processing and visualization activities.

To provide results in real-time, as required in emergency contexts, the Solver computation needs to be executed on sufficiently powerful computational resources. In this sense Cloud Computing is a promising paradigm, since the provision of resources as CPUs and storage is remotely provided on-demand by a service provider (i.e. *Infrastructure as a Service*). Moreover we can also consider a fluctuating demand of such resources, due to changing environmental conditions (e.g. an emergency detection can lead to tighter QoS constraints) or caused by the dynamic availability of the underlying computing/communication infrastructures.

4.1 QoS Modeling of the Solver ParMod

For this experimental application the notion of QoS has a two-fold nature:

1. *Performance*: in this application context it is of extreme importance to complete the forecasting computation for the interested area of the emergency scenario until a requested time. In this way forecasting results can be used by the system users in order to effectively plan proper response actions in advance to potentially dangerous events. In order to do that, *a general constraint requires to complete the forecasting processing in the minimum completion time as possible*;

2. *Operational Cost*: we suppose a centralized cloud infrastructure that remotely hosts the Solver computation. Customers do not own this physical infrastructure, but they pay a cost proportional to the amount of resources (e.g. CPUs) they use. Additional processing elements can be allocated by adding new CPUs to existing virtual machines, and/or by switching on real computing nodes on-demand. In order to discourage too many resource re-organizations, we suppose a business model in which a fixed cost should be paid each time a new resource request will be submitted to the cloud infrastructure.

In order to address these two requirements, the Solver ParMod is provided with an adaptation logic able to express non-functional reconfigurations, i.e. modifications of the current used parallelism degree. Therefore in this example we instantiate our general model in an application context in which different configurations of the Solver ParMod are uniquely identified by the parallelism degree parameter (i.e. the parallel version and the execution platform are fixed throughout the execution). Parallelism degree modifications are exploited by the control part in order to adapt the resource utilization in response to a dynamic workload condition: i.e. we suppose a time-varying mean arrival rate of tasks from the Generator, due to a dynamic behavior of the underlying interconnection network among the application modules.

To this end we will apply the predictive control approach to the Solver ParMod and we will compare its results with other notable adaptation strategies. First of all the main requirement that we need to meet is the performance objective: we will dynamically select parallelism degree modifications in order to optimize the number of computed tasks throughout the execution. Furthermore, among the set of strategies able to target this requirement, we need to select that which produces the lower long-term operational cost as possible.

4.1.1 System Model and Cost Function

According to our approach we can model the behavior of the Solver ParMod in terms of: a discrete control input $n(k)$ that indicates the parallelism degree currently used at control step k; an internal state variable $T(k)$ expressing the number of tasks completed up to the beginning of control step k; a non-

controllable disturbance $T_A(k)$ modeling the average inter-arrival time[2] of tasks experienced during the k-th control step of the execution.

The task-farm scheme is a structured parallelism pattern for which an analytical ***performance model*** can be expressed through basic notions of Queueing Network Theory. Let us analyze the task-farm *in isolation*, i.e. supposing that the external arrival rate of tasks is hypothetically infinite. We denote the mean service time (calculation time) of the emitter, worker and collector with T_E, T_W and T_C respectively, that are assumed to be fixed throughout the execution. The most meaningful parameter is given by the mean inter-departure[3] time from the collector: i.e. its inverse indicates the average number of tasks that the task-farm will complete in a time-unit.

Since the emitter exploits an on-demand distribution, we can suppose an uniform distribution of probability that tasks are transmitted to any worker. Therefore, if N indicates the current parallelism degree, the inter-arrival time to any worker is equal to $T_E \cdot N$ and the inter-departure time T_{d_W} from any worker can be calculated as the maximum between its inter-arrival time and its service time:

$$T_{d_W} = \max\Big\{T_E \cdot N,\, T_W\Big\}$$

The total inter-arrival time T_{A-c} to the collector can be calculated by summing all the arrival rates from each worker:

$$T_{A-c} = \left(\sum_{i=1}^{N} \frac{1}{T_{d_W}}\right)^{-1} = \frac{\max\{T_E \cdot N,\, T_W\}}{N}$$

We can formally define the mean inter-departure time T_{d_C} from the collector (which is also the global service time of the task-farm T_{farm}) as the maximum between the total average inter-arrival time from the worker set and the collector service time. We can express the following relationship assuming a parallelism degree $n(k)$ for the k-th control step:

$$T_{farm}(k) = T_{d_C}(k) = \max\left\{T_E, \frac{T_W}{n(k)}, T_C\right\} \tag{4}$$

This analytical description proves an intuitive behavior: i.e. by increasing the parallelism degree the number of tasks that a task-farm scheme can compute in a time unit increases until the emitter or the collector become computation bottleneck.

At this point the task-farm performance model can be exploited in order to predict how the number of computed tasks evolves during the execution. If we suppose a limited buffer size for incoming tasks from the Generator (e.g. one or two buffered elements, which is a practical situation if we use existing message-passing parallel programming frameworks as MPI[15]), the number of completed

[2]For mean inter-arrival time we intend the average time between the reception of two subsequent input tasks from the Generator.

[3]For average inter-departure time we intend the average time between two subsequent result transmissions.

tasks at the beginning of the next control step $k+1$ can be approximated by:

$$T(k+1) = T(k) + \left\lfloor \min\left\{\frac{1}{T_A(k)}, \frac{1}{T_{farm}(k)}\right\} \cdot \tau \right\rfloor \tag{5}$$

The increase in the number of tasks can be estimated as the minimum between the arrival and the service rates multiplied by the length τ of the control step. In fact, though the theoretical service rate of the task-farm can be greater than the arrival rate from the Generator, the upper bound of the number of tasks that will be completed in a control step can be approximated by the arrival rate from the Generator multiplied by the control step length.

It is worth noting that this example is a starting case that treats one of the most straightforward modeling for a ParMod. In fact in this case alternative ParMod configurations are only identified by the used degree of parallelism and the system model can be expressed parametrically w.r.t this parameter. In most general cases, when multiple parallel versions are provided for the same parallel module, each different configuration can be coupled with a proper model that can have a different analytical formulation than the ones of the other configurations (e.g. as in the case of performance models of task-parallel and data-parallel schemes).

In order to apply the predictive control approach we introduce the following utility function that needs to be maximized over a prediction horizon of h steps.

$$\max U(k) = w_1\, T(k+h) - w_2 \sum_{i=k}^{k+h-1} \Big[C_n\, n(i) + C_f\, \gamma(i)\Big] \tag{6}$$

The coefficients w_1 and w_2 indicate a trade-off among performance and operational cost. Assuming that $w_1 >> w_2$, we select a plane with the minimal operational cost such that the number of completed tasks at the end of the prediction horizon is maximized. C_n and C_f represent the cost for using a node for a single a control step and the fixed cost for each parallelism degree variation. $\gamma(i)$ is a variable equal to 1 iff the control input selected at the i-th control step is different from the one decided at the previous step, 0 otherwise.

4.2 Implementation details and Experimental results

The entire graph depicted in Figure 3 has been implemented by a network of distributed processes. Each ParMod consists in a separate MPI program executed on the underlying computing architectures. Inter-ParMod communications have been exploited by using the standard POSIX socket library TCP connections, whereas MPI send/receive primitives have been used for intra-ParMod communications.

The most important implementation issues have been addressed for developing the Solver ParMod. The emitter and the collector processes are responsible for monitoring the task inter-arrival time and the number of completed tasks, providing at each sampling period to the control part the average values of

these two measurements. The adaptation strategy is executed by a dedicated *manager* process that constitutes the control part.

In order to request the reservation of a specific set of computing nodes on a remote server architecture, the manager communicates with a remote process that simulates the cloud provider and manages the availability of a set of homogeneous computing nodes. Once allowed, parallelism degree variations are exploited by the manager through the MPI library function `MPI_COMM_SPAWN`, that instantiates a new set of processes executing the same MPI program (in our case the worker program).

In order to simulate a dynamic execution workload, we have represented a situation in which the task inter-arrival time changes significantly due to a time-varying network availability. In particular we suppose a non-dedicated interconnection network among application modules (see Figure 4). The Gener-

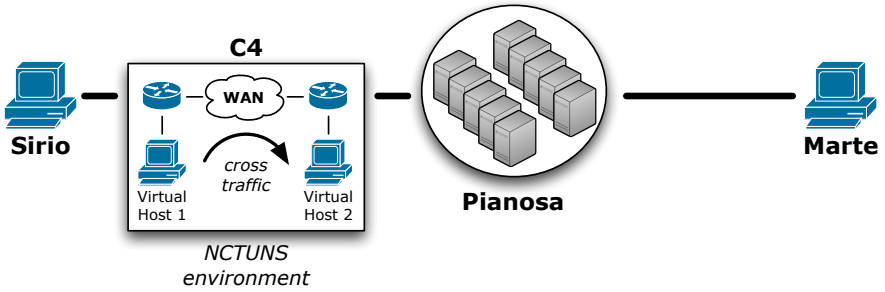

Figure 4: Execution platform of the experiment.

ator and the Client modules are executed on two workstations (Sirio and Marte) whereas the Solver is executed on a cluster (Pianosa) of 15 homogeneous production workstations simulating a cloud architecture. In order to reproduce a realistic network, the NCTUNS[16] network emulator/simulator is executed on a workstation (C4), and the task traffic from Generator to Solver ParMod is routed to this node. Inside C_4 a network topology is simulated, composed of two routers and a WAN object reproducing classical WAN delays and packet loss probability. Furthermore, inside the simulation environment two NCTUNS virtual hosts generate cross network traffic alternating unloaded periods and network congestion phases.

In order to perform our simulation, we have generated many different inter-arrival time traces, each one made up of ~ 900 samples during a total execution time of 130 minutes; each trace simulates a variable inter-arrival time due to a realistic cross traffic generated by the D-ITG[17] traffic generator executed on the two virtual hosts. A relevant trace-file example is depicted in Figure 5. In Figure 6 inter-arrival time measurements have been averaged on a control step of size 120 sec. The corresponding time-series exhibits two important classes of non-stationarities: (i) *Level shifts*, i.e. sudden changes in the mean of the observed values; (ii) *Trends*, i.e. relatively slow long-term movements in the

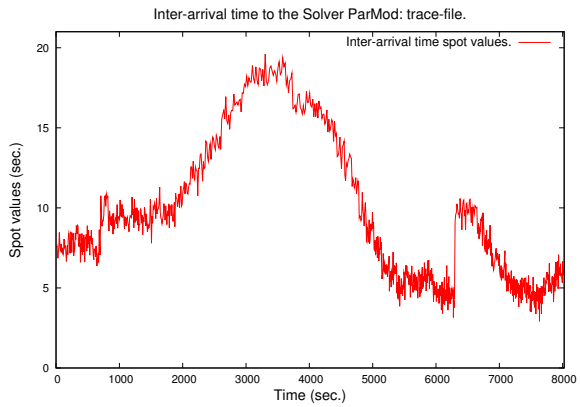

Figure 5: Example of inter-arrival time trace-file.

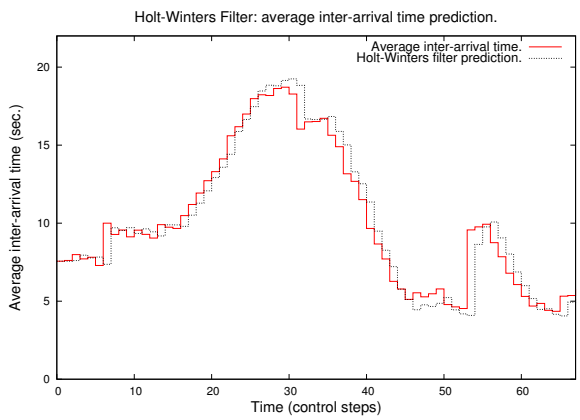

Figure 6: Average inter-arrival time predictions.

time-series. In order to predict the mean values of the inter-arrival time over a limited horizon of few control steps, a filtering technique based on a *Holt-Winters* (HW) filter has been applied.

Non-seasonal Holt-Winters is a filtering technique based on a simple EWMA (*exponentially-weighted moving average*) model that attempts to capture the trend in the underlying time series. Two different EWMA filters are used, the first one for estimating the smooth component s of the predicted value, and the second one for predicting the trend component t.

$$\begin{aligned}
\hat{T}_A(k) &= \hat{s}(k) + \hat{t}(k) \\
\hat{s}(k+1) &= a \cdot T_A(k) + (1-a) \cdot \hat{T}_A(k) \\
\hat{t}(k+1) &= b \cdot \Big[\hat{s}(k) - \hat{s}(k-1)\Big] + (1-b) \cdot \hat{t}(k-1)
\end{aligned}$$

The static gains a and b have been fixed to 0.9 and 0.2 respectively, that give

the best prediction results for this experiment. In Figure 6 is depicted the mean inter-arrival time of tasks (solid line) for each control step of the execution against the corresponding predicted value (dashed line). We can observe that predicted values are quite accurate for time-series exhibiting trends and level shifts, providing a relative mean square relative error of about 10% for this experiment.

The predictive control approach has been applied considering three possible lengths of the prediction horizon (i.e. 1, 2 and 3 control steps). Hence in this case a reconfiguration plane is a trajectory of parallelism degree variations for each step of the considered prediction horizon. This approach has been compared with two other reconfiguration strategies: the *MAX strategy*, in which we fix the parallelism degree to maximum value (15) for the entire execution, and a purely *reactive strategy* in which the parallelism degree is simply increased or decreased by one unit if the current utilization factor (i.e. ratio between the average task-farm service time and the average inter-arrival time) measured at the beginning of the k-th control step is greater or less than two pre-defined thresholds (i.e. 1.2 and 0.8 respectively).

Adaptation Strategy	**Completed Tasks**
MAX	946
Reactive	874
MPC 1-step ahead	939
MPC 2-step ahead	936
MPC 3-step ahead	938

Table 1: Completed tasks with different adaptation strategies.

Table 1 compares the number a completed tasks by exploiting the different strategies. As we can see the predictive approach is able to complete more tasks than the reactive one during the same execution time and with an identical mean inter-arrival time fluctuation. In this case the number of completed tasks is similar to the theoretically optimal one achieved by the MAX strategy, in which we always use the maximum number of computing resources. Moreover, the predictive approach has also a positive impact on the stability degree of a configuration: i.e. for every prediction horizon length, the MPC strategy always features a lower number of reconfigurations than the reactive approach (see Figure 7).

In Figure 8 is depicted the long-term operating cost throughout the execution. The importance of having a run-time parallelism degree adaptation is clearly highlighted. Experiments have been exploited assuming a fixed cost per reconfiguration (C_f) twice as much the unitary cost for using a node for each control step (C_n). W.r.t the MAX strategy, which is the most expensive one, the other adaptation strategies reduce the operating cost at least of the 40%. Moreover it emerges that the predictive strategy is able to produce a further reduction compared to the reactive adaptation of even $\sim$ 16% with the MPC

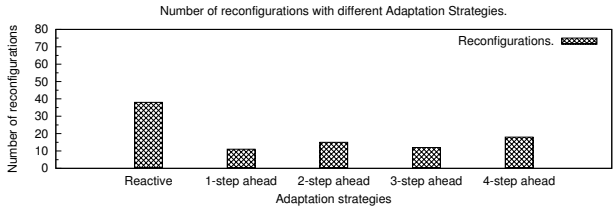

Figure 7: Number of reconfigurations.

3-step ahead, thus demonstrating how taking reconfigurations in advance to future workload predictions is an effective adaptation technique. In this example having a horizon length greater than three steps is not convenient, since the disturbance prediction accuracy degrades with longer prediction horizons. In fact we have verified that with a 4-step ahead strategy the operational cost is higher than with 3 steps, without any additional completed task.

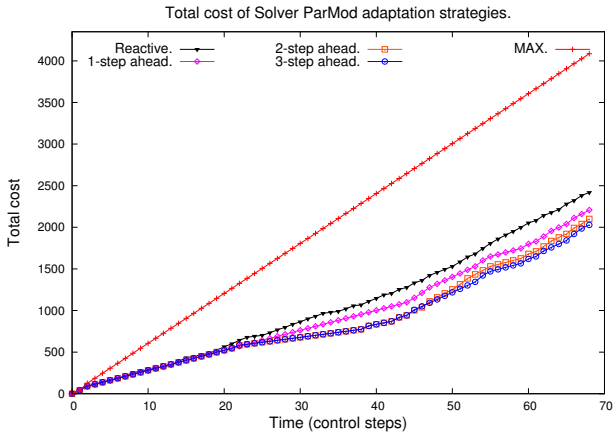

Figure 8: Total long-term operating cost.

As a final consideration we discuss the approach feasibility in terms of computational complexity. As stated in Section 3.2 predictive control for hybrid systems may be difficult to be applied, especially when the search space is large. The space can be represented as a complete tree with a depth equal to the prediction horizon length and an arity that coincides with the number of ParMod configurations. Nevertheless in practical scenarios, since the number of configurations is sufficiently limited and prediction horizons are normally short due to disturbance prediction errors, this approach can be exploited without requiring complex search space reduction techniques. For instance in this example with 15

References

[1] M. Aldinucci, M. Danelutto, and P. Kilpatrick, "Autonomic management of non-functional concerns in distributed & parallel application programming," in *Proceedings of the 2009 IEEE International Symposium on Parallel& Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587401

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

[3] N. Kandasamy, S. Abdelwahed, and J. Hayes, "Self-optimization in computer systems via on-line control: application to power management," in *Autonomic Computing, 2004. Proceedings. International Conference on*, May 2004, pp. 54 – 61.

[4] 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.

[5] 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.

[6] M. Aldinucci, M. Danelutto, and M. Vanneschi, "Autonomic qos in assist grid-aware components," in *PDP '06: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 221–230.

[7] S. Diwan and D. Gannon, "Adaptive resource utilization and remote access capabilities in high-performance distributed systems: The open hpc++ approach," *Cluster Computing*, vol. 3, pp. 1–14, January 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=592890.592937

[8] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury, "Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, 2002, pp. 219 – 234.

[9] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management," *Real-Time Syst.*, vol. 23, pp. 127–141, July 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1015350520175

[10] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online control for self-management in computing systems," in *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, May 2004, pp. 368 – 375.

[11] C. Bertolli, G. Mencagli, and M. Vanneschi, *High-Performance Pervasive Computing*, ser. Computer Science, Technology and Applications. Nova Publisher, 2011, vol. 1.

[12] C. Bertolli, D. Buono, G. Mencagli, and M. Vanneschi, "Expressing adaptivity and context awareness in the assistant programming model," in *Intl. Conf. on Autonomic Comp. and Comm. Syst.*, ser. LNICS, vol. 23. Springer, 2009.

[13] S. A. J. v. der and J. M. Schumacher, *Introduction to Hybrid Dynamical Systems*. London, UK: Springer-Verlag, 1999.

[14] 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. [Online]. Available: http://portal.acm.org/citation.cfm?id=72068.72069

[15] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: The Complete Reference*. Cambridge, MA, USA: MIT Press, 1995.

[16] NCTUNS, "Network Simulator and Emulator," http://nsl.csie.nctu.edu.tw/nctuns.html/, 2011.

[17] D-ITG, "Distributed Internet Traffic Generator," http://www.grid.unina.it/software/ITG/papers.php.